Enabling User-Driven Rule Management

in Event Data Analysis

Weisi Chen

A thesis in fulfilment of the requirements for the degree of

Doctor of Philosophy

School of Computer Science and Engineering

Faculty of Engineering

January 2015

Contents

Abstract

Increasingly in the information age, overwhelming quantities of available data has brought about opportunities as well as difficulties. Data analysis is of considerable importance to finding interesting patterns, discovering useful information and making decisions accordingly. Event data has unique characteristics including temporal dependencies, high flow rate and huge volume, which makes it more difficult to analyse than other data types. Unlike data analysts working in large companies that have IT staff and expensive software infrastructure, those working in the research sector find it more difficult to efficiently manage event data analysis by themselves. User-driven rule management is a particular challenge especially when analysis rules increase in size and complexity over time. This thesis addresses these problems by proposing a new architecture called EP-RDR aimed at enabling data analysts, with no IT experience, to manage their event data analysis.

EP-RDR enables complex event processing and facilitates user-driven rule set evolution according to changing requirements. The architecture leverages event processing system (EPS) technology with a rule-based method called Ripple-Down Rules (RDR). EP-RDR has two main components: an RDR component playing the role of managing the event processing logic and of supporting incremental rule insertion that enables data analysts to define and add rules by themselves, and an EPDaaS (Event Pattern Detection as a Service) component that can invoke any EPS so that data analysts are able to conduct event processing without concern about which EPS to use. To facilitate the interoperability between components in EP-RDR, this thesis also proposes an Event Data Modelling Framework (EDMF) to assist in building data models for any application of EP-RDR. EDMF consists of an Event Data Meta-Model and its associated Operational Guidelines. Any data model built based on EDMF allows event pattern types to be defined, abstracting existing event and event pattern occurrence representation formats in a consistent manner. Finally, to evaluate the proposed new method, a prototype has been implemented and applied on real-life scenarios involving financial market data pre-processing. This case study shows that the proposed method effectively satisfies requirements of event data analysis, namely feasibility and interoperability, the capability of complex event processing, and the capability of user-driven rule set evolution.

Acknowledgments

First and foremost, I would like to express my sincere gratitude to the service-computing research group at the School of Computer Science and Engineering in University of New South Wales, especially my supervisor Prof. Fethi Rabhi who has continuously given judicious advice and timely guidance during the last four years, and has been encouraging me to keep the research work on track and make progress as expected.

I would also like to thank my co-supervisor Prof. Paul Compton for his expertise and valuable advice on the RDR approach, and Zoran Milosevic and Andrew Berry for their expertise on complex event processing and for allowing me access to a commercial event processing system called EventSwarm. I wish to acknowledge with gratitude Dr. Lawrence Yao for his kind guidance when dealing with technical issues and for answering some of my research-related questions, and Prof. Boualem Benatallah for his valuable suggestions.

Furthermore, I would like to thank the Smart Services Cooperative Research Centre in Australia for sponsoring this research project and Sirca for providing the financial market data used in the case study.

Last but not least, I would also like to thank my family for their continuous support and encouragement throughout these years of PhD study.

List of Publications

Below is a list of publications related to this thesis.

Published Papers:

1. CHEN, W. & RABHI, F. A. 2014. Validating an Incremental Rule Management Approach for Financial Market Data Pre-Processing. In *Proceedings of Workshop on Enterprise Applications, Markets and Services in the Finance Industry* (**FinanceCom 2014**).
2. CHEN, W. & RABHI, F. A. 2013. An RDR-Based Approach for Event Data Analysis. In *Proceedings of 3rd Australasian Symposium on Services Research and Innovation* (**ASSRI'13**).
3. CHEN, W. 2013. E-Research Event Data Quality. In *Proceedings of Workshop in 29th IEEE International Conference on Data Engineering* (**ICDE'13**).
4. CHEN, W. & RABHI, F. A. 2011. Incrementally Defining Analysis Processes Using Services and Business Processes. In *Proceedings of 13th International Conference on Enterprise Information Systems* (**ICEIS'11**). Beijing, China.

Special Reports:

1. RABHI, F. A., CHEN, W. & QUDAH, I.A. 2014. Evaluation of Event Study Workbench for Large Scale Event Studies. *New Financial Services Report – Smart Services CRC.*
2. RABHI, F. A., LATTAB, K., LANGE, B. & CHEN, W. 2011. The News Processing Portal An SOA for News Search. *Service Delivery and Aggregation Project Progress Report – Smart Services CRC.*
3. RABHI, F. A., SIMMONDS, D., PEAT, M., ZHAI, J., CHEN, W. & TEO, D., 2011. Event Study Workbench (ESW). *Final Report – Smart Services CRC.*

List of Figures

List of Tables

List of Abbreviations

API Application Programming Interface

APPD Approved

ASX Australian Securities Exchange

BP Bespoke Program

CEP Complex Event Processing

CSV Comma Separated Values

DBMS Database Management System

Div Dividend

DPPT Data Pre-Processing Tool

DSP Data Stream Processing

ECA Event-Condition-Action

EDMF Event Data Modelling Framework

EOD End Of Day

EPDaaS Event Pattern Detection as a Service

EPL Event Processing Language

EP-RDR Event-Processing RDR

EPS Event Processing System

ESP Event Stream Processing

GMT Greenwich Mean Time

GREL Google Refine Expression Language

GUI Graphical User Interface

HTML HyperText Markup Language

IPL Impact Policy Language

IT Information Technology

JSON JavaScript Object Notation

MCRDR Multiple Classification RDR

NoSQL Not only SQL

OP OpenRefine

PC Personal Computer

P-DAG Pattern Directed Acyclic Graph

PROP Proposed

RBS Rule-Based System

RDR Ripple-Down Rules

REST Representational State Transfer

RIC Reuters Instrument Code

SAS Statistical Analysis System

SDAT Statistical Data Analysis Tool

SCRDR Single Classification RDR

SDLC Software Development Life Cycle

Sirca Securities Industry Research Centre of Asia-Pacific

SME Small and Medium Enterprises

SPSS Statistical Package for the Social Sciences

TRTH Thomson Reuters Tick History

TSV Tab Separated Value

UML Unified Modelling Language

URSE User-Driven Rule Set Evolution

UTC Coordinated Universal Time

WADL Web Application Description Language

XML eXtensible Markup Language

Chapter 1 Introduction

This chapter provides an introduction to the research area addressed in this thesis. It starts by introducing the data deluge and some related concepts in Section 1.1. Section 1.2 discusses the importance of event data analysis. Section 1.3 provides an overview of the research problem, and Section 1.4 presents the thesis objectives. Finally, Section 1.5 outlines the organisation of the thesis.

1.1 The Data Deluge Problem

In the contemporary world, the quantity of information is soaring. This has contributed to a situation where a vast volume of new data is being generated every day, which overwhelms the development of infrastructures and tools to manage and make use of the data, and in turn leads to information overload. This situation is referred to as data deluge [1], data flooding [2], or information explosion [3]. Literature often notes the ever increasing growth of data available and the need for technology to manage the data deluge [1, 4].

As opposed to its literal meaning, data deluge has brought about opportunities as well as problems. On the one hand, data analysts have more chance to find more subtle information hidden inside the data; and with more input from a wide range of sources, decision making can be more rational than it used to be. On the other hand, despite the potential opportunities brought about by data deluge, the major problem that arises is how to take advantage of such opportunities. In other words, current tools are less than sufficient in efficiently handling the exploding amounts of data.

In order to address this issue, some interrelated concepts have emerged, including big data and eResearch. These concepts together form a fourth paradigm of science - data-intensive science, following on from the three former paradigms: empirical science (descriptions of natural phenomena), theoretical science (using models and generalisations to explain phenomena, e.g. Kepler’s Laws, Newton’s Laws of Motion) and computational science (simulating complex phenomena when modelling is too complicated) [4]. Specifically,

- *Big Data*: Big Data is characterised by several dimensions [5] – Volume (exponential growth rate), Velocity (increasing frequency of data generation), Variety (complexity and diversity of data types) and Veracity (uncertainty of data). These characteristics pose varying challenges to researchers conducting data analysis. The concept of big data is a result of data deluge, demonstrating the opportunities and challenges brought about by huge quantities of data for both enterprises and researchers.

- *eResearch* [6]: Researchers are using many different methods to collect or generate data, e.g. recording web logs, network traffic messages, periodic sampling from sensors, charge-coupled devices, social media feeds, and financial reports sensors. However, due to the increasing quantities of data, it is a challenge for researchers with no IT experience to know what to do with the data after gathering it. This is the purpose of eResearch, which can be informally defined as "where IT meets researchers". eResearch is multi-disciplinary research that is conducted with a heavy reliance on IT. It commonly involves data-intensive analysis tasks and is mainly performed by non-IT experts from different application domains (e.g. finance, biology and health). These experts usually use data, libraries and packages from a variety of sources and manage the analysis process by themselves. With more than 15 years of history, there are currently a number of national eResearch initiatives taking place in different countries, including the UK (with a different term "eScience"), the US (under the name "Cyberinfrastructure") and Australia. eResearch is now being adopted worldwide across all research disciplines, harnessing high-capacity and collaborative data from a large range of data sources and communication technology to improve and enable research that cannot be conducted otherwise. From an IT-focused perspective, the concept of eResearch offers solutions to researchers to enable them to handle enormous amounts of data, and provides the technologies to do this.

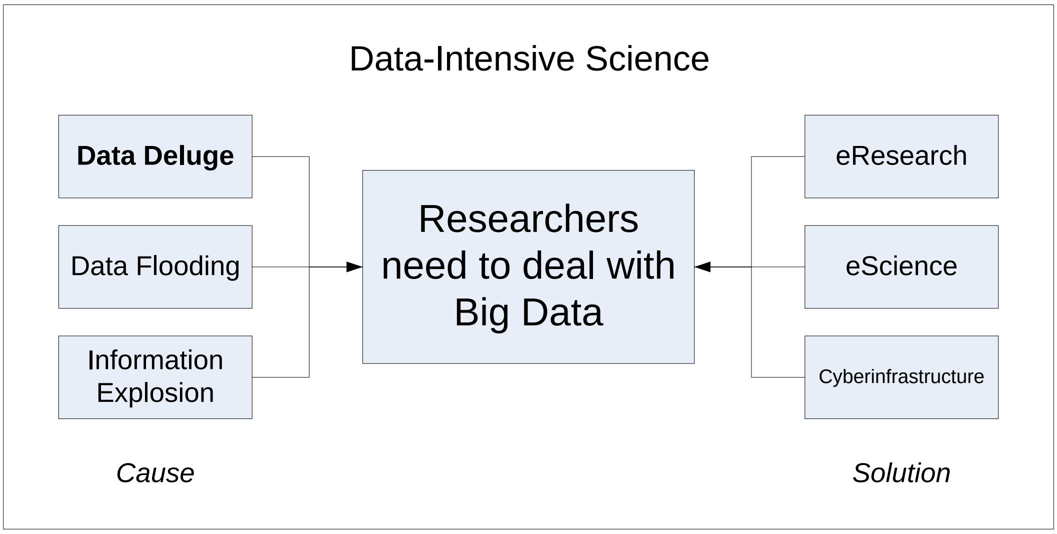


Figure 1.1 Interrelated concepts regarding data deluge introduced in Section 1.1.

The relationship of all the concepts mentioned above is demonstrated in Figure 1.1.

1.2 Introduction to Event Data Analysis

Data analysis is a means to process the data so that useful information can be discovered and decisions or conclusions can be made based on the data available. It is the key method for taking advantage of data deluge. In some cases, data analysis is as simple as manipulating datasets in an ordinary software tool like Microsoft Excel [7]. This is not difficult when the data is simple in nature. However, some types of data are complex and hard to analyse, and generic tools like Microsoft Excel do not suffice. In particular, event data (or event-based data) deserves special focus because it can be generated on a considerably massive scale with a high level of complexity. It is thus more difficult to analyse than other types of data. Since raw event data often presents as rather unorganised, event data in itself is useless unless data analysts make sense of it via data analysis. 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. The benefits of data analysis include but are not limited to:

- Re-organising the information from event data collection;
- Acquiring meaningful insights (e.g. the identification of mission-critical trends) from the event datasets;
- Making important decisions based on the insights.

However, unlike data analysts working in large companies that have access to IT staff and expensive software infrastructure, data analysts in research sections find it harder to efficiently manage their event data analysis by themselves. Sometimes, they need to conduct the data analysis manually. A typical process is as follows [8]. First, a data analyst manually collects data from multiple sources, such as databases, spreadsheets, and reports. An IT expert might be needed to request the required data, which may delay the data analysis process. Second, the data analyst will import the data into a PC application to conduct data analysis. The application can be a dedicated program developed by a programmer, or a generic data analysis tool (either free or commercial). The third step is to conduct the actual analysis, which is not a one-off task and often requires iterative modifications on the analysis settings and logic. Finally, the data analyst needs to inspect the result of the analysis and draw a conclusion based on it. This manual process is time-consuming and requires great effort.

1.3 Research Problem

Manual data analysis requires data analysts to use certain tools. There are many tools available. Some of them are designed for data analysts with no IT experience, for instance, Microsoft Excel and OpenRefine [9]. These generic tools usually feature user-friendly interface so that data analysts are able to use them without too much difficulty. To utilise them for a particular data analysis task, the data analyst needs to learn the functions provided by these tools, and select suitable functions to perform the analysis. However, since these tools are for generic data analysis purpose, they require mostly manual work and they lack capabilities or efficiency in dealing with issues of particular data types. There are more professional tools which are specialised in different aspects of data analysis, for instance, database management systems like MySQL [10], statistical data analysis tools like SAS [11], event processing systems like Esper [12]. Unlike generic data analysis tools, they provide specialised capabilities and are more efficient in particular circumstances, but require a steep learning curve or certain level of IT expertise.

In order to acquire valuable insights and make rational decisions from event data, both domain knowledge and IT expertise are needed. In real-life scenarios, data analysts from the research sector are normally domain experts with very limited or even no IT experience, while IT experts lack domain knowledge. Thus, neither data analysts alone nor IT experts alone are able to conduct event data analysis easily. This is the main reason why event data analysis is a challenge. Data analysts usually employ a programmer to develop a dedicated tool for a particular data analysis task, or take advantage of existing data analysis tools with the assistance of the IT expert. Due to different capabilities of different applications, the data analyst may utilise multiple applications to fulfil the analysis requirements, which may require the intervention of IT experts.

Consequently, the research problem of this thesis is how data analysts with no IT experience can take advantage of existing tools, and have control over their own data analysis process.

1.4 Thesis Objectives

The primary aim of this thesis is to provide a new method that enables data analysts to manage data analysis on event data with minimal IT expert intervention. First and most importantly, the new method addresses the issues outlined in the research problem (see the previous section). In order to achieve this, this thesis looks into event data characteristics and the definition and norm of event data analysis. Then, existing solutions are explored and compared in terms of various capabilities and limitations. It may be tempting to design new tools to overcome the limitations of existing tools. However, this is time-consuming and labour-intensive as it involves tool development, testing, installation, staff training and license fees. Therefore, a reasonable and more cost-efficient approach is to reuse existing technologies.

The proposed new method takes advantage of existing technologies and approaches, and leverages the most suitable ones to integrate. The thesis focuses on the system design aspects, including component specification and interaction, so as to provide guidance for developers to build systems according to the needs of data analysts in a particular domain.

In addition, it is necessary to instantiate the new method by implementing a prototype based on it, so as to validate the feasibility of the proposed method. To assess the capabilities of the new method, an appropriate case study involving event data analysis has been conducted. This enables the evaluation of the new method in relation to the research objective described above.

1.5 Thesis Outline

This chapter provides a general introduction to data-intensive science and eResearch. It also presents an overview of the research question, research objective and research contribution of this thesis. The remainder of this PhD thesis is organised into the following chapters:

- *Chapter 2 Literature Review*: provides some background research material and discussions on the topics related to the thesis. This includes what role event data analysis plays in data-driven research, why rule management in event data analysis is important, major categories of techniques used in data analysis and rule management techniques.
- *Chapter 3 Research Plan*: discusses the research questions, objectives and the methodology used in carrying out this research.
- *Chapter 4 Proposed Architecture*: describes the first design artefact proposed in this research, which is an architecture that facilitates rule management in event data analysis for data analysts.
- *Chapter 5 Proposed Event Data Model*: provides the details of the second design artefact proposed in this research, which is an event data modelling framework that facilitates event data modelling and thus enables the exchange of data between components in the EP-RDR architecture.
- *Chapter 6 Prototype Implementation*: describes the details of a prototype implementation of the proposed artefacts described in Chapters 4 and 5 for evaluation purpose.

- *Chapter 7 Evaluation and Case Study - Financial Market Data Pre-Processing*: uses a case study in the finance domain, i.e. financial market data pre-processing to evaluate the proposed work. This chapter defines the evaluation criteria and corresponding evaluation metrics based on the research questions raised in Chapter 3, and provides the details of experiments and results.
- *Chapter 8 Conclusion and Future Work*: concludes this thesis with an outline of the thesis contributions, and a discussion of the limitations and future work directions.

Chapter 2 Literature Review

In this chapter, we provide some background research material for the thesis. Firstly, Section 2.1 explains the role of event data analysis and the importance of rule management in event data analysis. Then Section 2.2 discusses three generic approaches that can be used in event data analysis in. A discussion on systems specialised in event data processing is provided in Section 2.3. Section 2.4 explores the rule-based systems and the role they can play in event data analysis. Finally, Section 2.5 compares all the mentioned techniques and discusses their suitability in conducting event data analysis.

2.1 Event Data Analysis

In this section, we start with an introduction to data analysis in general. Section 2.1.2 summarises the unique characteristics of event data. Section 2.1.3 discusses analysis conducted specifically for event data. Section 2.1.4 raises the problem of rule management faced in event data analysis.

2.1.1 Data Analysis in General

As mentioned in Chapter 1, the purpose of data analysis is to answer questions, to suggest conclusions, and to support decision-making for data analysts. It is a process of converting raw data into useful information by applying statistical and/or logical techniques systematically [13]. Data analysis is a key stage of eResearch, which is frequently conducted by data analysts in various research domains. It is also essential for enterprises that need to gain specific information from the data they collect.

The following distinguished phases are normally included in data analysis:

- *Data collection*: the process of gathering data of interest from certain data sources that leads to answering research questions. [14]
- *Data cleansing*: also referred to as the core part of "data pre-processing". The common method is to split the process into several steps including auditing, parsing, standardisation, scrubbing, de-duplicating, integration, etc. [15, 16]
- *Data transformation*: also belongs to data pre-processing, which denotes preparing data for further analysis via standardisation or normalisation. It is needed since in many applications of data analysis, the raw data or cleansed data cannot be used directly [17] due to formatting issues. The formatting of values into consistent layouts is performed

based on various elements, including industry standards, local standards, business rules and domain knowledge bases. Some literature indicates that this functionality often uses a rule library or can automatically be derived from schema matching tools. [18]

- *Statistical analysis*: The goal of this phase is to find interesting patterns so as to identify trends. This phase involves various algorithms and can be broken down into five discrete steps, namely "data nature description", "data relation exploration", "data modelling on how the data relates to the underlying population", "validation of the model", and "predictive analytics". Statistical analysis is often accompanied with or followed by data visualisation to interpret the results of the analysis. [19, 20]

Note that these phases are akin to the phases in Knowledge Discovery in Databases (KDD) [21], which focuses on data analysis in databases. However, these phases are also standard in data analysis in general.

2.1.2 Characteristics of Event Data

An event is “anything that happens, or is contemplated as happening” [22] 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, and 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”, which is sometimes cited as time series data.

Event data has some characteristics [23] that make it more difficult to analyse than other data types:

- *Temporal dependencies*: Event data flows in time-streams. Compared with other data types, 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, the analysis of event data involves a great deal of time handling (i.e. complex event processing).
- *High flow rate*: In an event-based system, new events are continuously coming in to guarantee the timeliness of the data. The flow rate is high as opposed to other data types. Event data with very high flow rate is also referred to as high-frequency data.
- *Huge volume*: Event data records are normally generated and stored in huge volumes, sometimes containing data for years. Therefore, it is way beyond human capability to manually process a huge amount of data.

- *Immutability:* On account of the high flow rate of event data, each record that comes in will never be modified.
- *Referability:* 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, e.g. several days before or after the current event.
- *Influence:* Any 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.

All these characteristics give rise to drastic difficulties in processing event data.

2.1.3 Event Data Analysis

Currently, event data analysis is being conducted widely (e.g. using data mining techniques) to explore “the characteristics of object evolution, or the trend of changes for objects in the database” [21]. For many years, event data analysis has been conducted by the business sector for various purposes such as studying market trends, improving the efficiency of operational processes and gathering business intelligence. We view event data analysis as a process of primarily detecting patterns in the data and taking a number of actions accordingly so as to interpret the meaning of information collected from a particular event data source.

To conduct event data analysis tasks, data analysts have to rely on IT experts either to implement a bespoke program/service or to customise an existing tool such as an event processing system (EPS) according to their needs. Because of constant changes in business needs and the environment, data analysts need to communicate their new requirements to IT experts all the time to update and maintain the event data analysis business logic - event processing rules. In existing event processing systems, rules are normally executed separately or very small sets of rules can be executed together. When it comes to rule sets, especially large and complex rule sets, their management is much more difficult for a data analyst. A typical way to support such an event data analysis process requires assistance from IT experts and knowledge engineers, as illustrated in Figure 2.1. On the one hand, the knowledge engineer manages the rule set and on the other hand, the IT expert implements the rules according to the underlying software infrastructure. There could be a multitude of data analysts 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.

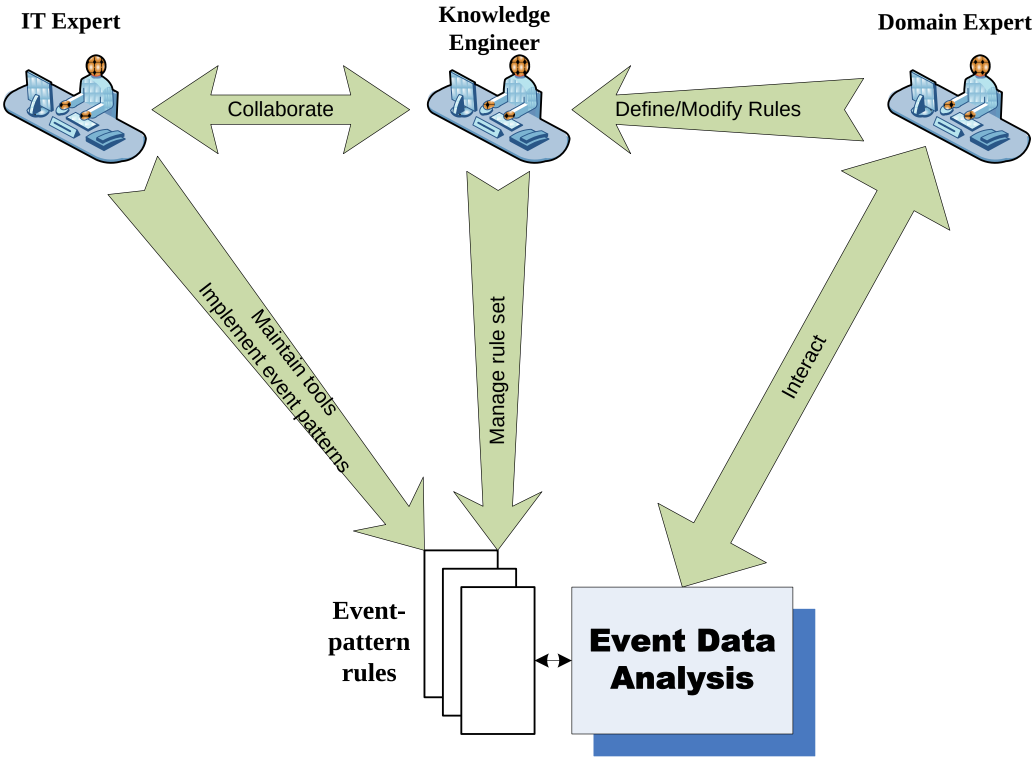


Figure 2.1 Evolution of an event data analysis process.

More recently, event data analysis is becoming of increasing interest to academic researchers looking for patterns in data. This is contributing to the emergence and popularity of "data-intensive science" [24]. Unlike data analysts working in large companies and having access to IT staff and expensive software infrastructures, those working in the research sector tend to conduct the analysis mostly on historical event datasets by themselves using a range of data processing and statistical tools. This creates a need to enable analysis of event data by data analysts 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.

2.1.4 An Example of Rule Management in Event Data Analysis

In this section, we raise the problem of rule management in event data analysis by presenting a real-world example that involves the analysis of Sirca's daily data [25] by data analysts in the finance domain.

Financial market data analysis often requires the computation of 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 causes a fall in the stock price by the dividend amount on the ex-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 2.1 Different cases in eliminating duplicate dividends.

(a) Case 1 – simple duplicate dividend records.

#RIC[1]	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) and they have the same Div Ex Date and the same Div Amt, 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	11/09/2012	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 dividends 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

Table 2.1 illustrates different cases of duplicate dividend issues. In most cases like Table 2.1(a), duplicate events just record the same dividend announcement multiple times. The initial logic of the rule is:

[1] RIC is short for "Reuters Instrument Code", which is a unique identification code for a particular company.

If there are two Dividend events issued at the same event time (Date), the same "Div Amt." and the same "Div Ex Date"
Then it is a duplicate so delete the first one

Over time, the data analyst may find this logic leads to incorrect decisions and actions in a new case (Table 2.1(b)), which requires changing the rule to:

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

Again, due to another new case (Table 2.1(c)), the rule must be modified to:

If there are two Dividend events with the same event time, the same "Div Amt.", the same "Div Ex Date" and 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

Generally, in real-world event processing, rules are never "perfect" as there are always exceptions to existing rules. As an extension of the example above, the defined data cleansing rules built for Australian stock data may be applied to another country's stock data, e.g. German stock data, with similar but not exactly the same logic. This requires additional modifications to the rules which can be even more complicated than the example above. It 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.

2.2 Generic Approaches for Event Data Analysis

This section explores the role of generic methods to conduct event data analysis. Data processing tools are often utilised to conduct general data analysis tasks but many researchers and enterprises also apply these tools to event data analysis. The tools include three categories: bespoke programs, commercial tools and open-source tools.

Bespoke programs are developed according to specific needs of data analysts and thus designed for a particular purpose. This method is not flexible. Every time the data analyst wants to change the processing logic, he/she needs to ask an IT expert to modify the program.

In regard to commercial and open-source tools, there are quite a few products available on the market that facilitate one or more phases of data analysis. Commercial tools may be unaffordable for data analysts working in the research sector as most of them are very expensive and not open to the public.

The following sub-sections explore more examples of different categories of data processing tools, namely data pre-processing tools (Section 2.2.1), database management systems (Section 2.2.2), and statistical data analysis tools (Section 2.2.3).

2.2.1 Data Pre-Processing Tools

Data pre-processing is an important phase in data mining and there are many tools available for this purpose. In this section, we choose two of the most popular tools (OpenRefine and Data Wrangler) and illustrate their capabilities.

OpenRefine [9] is a free desktop application previously funded by Google, which enables data analysts to import, cleanse, manipulate, enrich data and finally export it for further analysis. OpenRefine has the following key features [26, 27]:

- *Importing*: Files in a wide range of formats (e.g. CSV, TSV, Excel, JSON, XML and Google Docs documents) can be imported into OpenRefine. Data analysts can upload a local file, import data from web pages or XML documents, paste data from the clipboard, or retrieve a document from Google Docs.
- *Editing*: Akin to Microsoft Excel, data analysts are able to edit cells, columns and rows manually. It provides a number of pre-defined operations such as eliminating consecutive white spaces in texts, escaping or un-escaping HTML entities, changing letter case and converting data formats.
- *Transformation*: Performed via GREL (Google Refine Expression Language) code. GREL enables the splitting of columns, the creation of new columns based on values of other columns, and the combining of cells to create new columns.
- *Extending*: New columns may be created based on existing columns, or by calling remote APIs exposed by external services to enrich data.
- *Filtering/Faceting*: A subset of the data set can be displayed according to certain filtering criteria. This is also a function provided by Microsoft Excel.
- *Clustering*: Similar entries in the data set can be detected based on a number of algorithms. Data analysts can decide manually whether these detected similar entries should be merged or ignored. This is to enable data analysts to handle inconsistencies in data. This is one of the unique OpenRefine's features.
- *Exporting*: After performing data analysis tasks, data analysts can export the processed data as CSV, TSV, Excel or HTML files. In addition, data analysts can also export their

manual processing steps to a file in JSON format, and apply them to similar datasets to avoid repetition in the future. Thus, data analysts can customise the tool to their own need as long as the steps are not very complicated.

Data Wrangler [28] is an interactive web application for data pre-processing. Similar to Microsoft Excel and OpenRefine, it allows direct manipulation of data. Besides, it has the intelligence to automatically suggest applicable operations of data transformation [26]. At any stage, data analysts are able to repeatedly scan all the suggested options, to preview the corresponding effects, and to decide which option should be taken into effect. In addition, validation and type conversion are supported by integrating semantic data types (e.g. dates, geographical locations, classification codes).

One obvious disadvantage of these data pre-processing tools is that they significantly limit the size of input data. The reason is that they have graphical interfaces for data analysts to interact with and all data needs to be displayed on the screen so data analysts are able to navigate and manipulate the displayed data files. Thus, there is normally a limit of the size of the input data to restrict the memory usage and for optimisation purposes. Another disadvantage is that the analysis process is not automated in these tools so a great deal of manual effort needs to be made by data analysts.

2.2.2 Database Management Systems (DBMSs)

Database Management Systems (DBMSs) enable the user to manage a database. Depending on the privilege, the user may be able to store new data, to modify existing data, or to search and extract data from the database. In most cases, some data processing functions are available in DBMSs, which make it possible for the user to extract aggregate information according to specific needs. DBMSs are generally very efficient in storing, organising and searching for information in a database. However, data processing functions are more difficult to use, so merely competent users are able to take advantage of them to generate useful information.

The most commonly used type of DBMS is a relational database [29], which stores data in two-dimensional tables and applies SQL queries to manage data. Some user-friendly and powerful relational DBMSs include Microsoft Office’s ACCESS [30], Apache OpenOffice BASE [31], MySQL [10] and PostgreSQL [32]. Relational databases are competent for general data management purposes; however, the scalability of relational databases is poor in a cloud environment when they need to support an ever-increasing rate of transactions per second without compromising on speed [33].

As an alternative to relational databases, NoSQL (often interpreted as Not Only SQL) databases [34] emerged for the purpose of facilitating the processing of big data and addressing the needs of real-time web applications. They are designed to efficiently cope with heavy read/write loads and have better scalability than relational databases. Therefore, they are more suitable to be run on cloud platforms. However, NoSQL has not yet been used in most current applications, because they often require rewriting the code of these applications. Some examples of NoSQL databases include CouchDB [35], Hadoop [36], Cassandra [37], Neo4J [38], and MongoDB [39].

There is another particular type of database named "temporal database" [40] that is worth mentioning. The research on temporal databases has been motivated by the need to assign a validity interval to relational data. Despite the fact that a time interval can be assigned to tuples in relational databases easily, it is complicated to query such temporal data. In essence, temporal databases operate on historical data that has a temporal dimension. Generally, the expressiveness of temporal databases is similar to that of relational databases but for the support of the correlation of temporal data. Thus, temporal databases can also be viewed as a generalisation of relational databases [41] in which operators for the temporal dimension are supported, e.g. [42]. It has been decades since temporal databases first became a research topic but this technology is still immature [43].

2.2.3 Statistical Data Analysis Tools

In this section, we will explore and compare four commonly used statistical data analysis tools/packages, namely R, SAS, Stata and SPSS. These tools have their advantages compared with general-purpose programming languages and applications, as they permit data analysts to concentrate more on the information contained in the data and the way to analyse it than technical details of the data and how it is stored. This appeals to data analysts who have a limited knowledge of IT or computer science. Each of these tools offers its own unique strengths and weaknesses. Thus, many data analysts need to learn multiple tools and combine them to complete their data analysis. In addition, data analysts are required to learn some programming concepts so that they can utilise these tools to an optimal extent.

2.2.3.1 R Programming Language

R [44] is a free software environment for statistical computing and graphics, which compiles and runs on most common operating systems. It is often regarded as the most comprehensive statistical analysis package available.

The advantages of R include [45]:

- A comprehensive language for managing, manipulating, and analysing data
- Capability to generate high-quality graphs and a fully-programmable graphics language
- Open source and free, permitting anyone to use and to modify it

The disadvantages of R include [45]:

- As a programming language, R is not easy to learn
- Documentation is sometimes too simplified and not very useful for beginners
- The quality of some packages needs to be evolved over time
- Many R commands give little thought to memory management, so R may consume all available memory very quickly

2.2.3.2 Statistical Analysis System (SAS)

SAS (Statistical Analysis System) [11] is a statistical software package that enables data analysts to conduct statistical analysis on price data interpolated with certain events. It can run on most of the common operating systems.

Data in SAS is stored as tables and researchers can write scripts via a GUI to perform sequences of operations on them. SAS also exposes APIs to different components so researchers can invoke SAS components from external programs.

The advantages of SAS include [46]:

- It is very powerful and programmable
- SQL queries are enabled so users can manage the data as in traditional databases
- It can work with multiple files at once
- SAS can handle huge files depending on the size of the hard disk

The disadvantages of SAS include [46]:

- If you make a mistake in an SAS program, it can be hard to see where the error occurred or how to correct it
- With more complicated functions, SAS has a steep learning curve

2.2.3.3 Stata

Stata [47] is another popular statistical package. With smart data-management facilities and a wide range of latest statistical techniques integrated, it provides reliable and powerful statistical analysis functionality. In addition, it can also produce high-quality data visualisation graphs. Stata is fast and easy to use and is available across common operating systems.

The advantages of Stata include [46]:

- It is easy to learn but still powerful, which is good for the novice as well as advanced users
- Mistakes in commands/programs can be easily diagnosed and corrected
- It provides simple syntax of graph commands to create graphs for the data analysis

The disadvantages of Stata include [46]:

- Data cannot be easily managed
- It works with only one data file at a time, so to analyse multiple datasets at once can be cumbersome

2.2.3.4 Statistical Package for the Social Sciences (SPSS)

SPSS (Statistical Package for the Social Sciences) [48] is an IBM software application, a Windows-based program used to perform data entry and analysis and to create tables and graphs. It is capable of handling large amounts of data and is commonly used in various domains, including business and social sciences. The graphical interface is similar to Excel Spreadsheet with far superior functions for completing complex data manipulation and analysis. Note that SPSS can be integrated with some IBM add-ons to conduct more complicated analysis. For instance, the SPSS Modeler [49] is an application that can be integrated with SPSS to allow data mining and text analytics.

The advantages of SPSS include [46]:

- Basic functions are easy to learn due to its user-friendly user interface
- It generates high-quality graphs for data analysis

The disadvantages of SPSS include [46]:

- Data cannot be easily managed
- It works with only one data file at a time, so analysis of multiple datasets at once can be cumbersome
- For academic use, SPSS lags notably behind SAS and R. Its menu offers the most basic analysis functions. To take advantage of more advanced functions, data analysis needs to purchase licences for add-ons, and these additional functions have a steep learning curve.

2.2.4 Discussion

Up until now, we have explored three generic methods to conduct event data analysis, namely data pre-processing tools, database management systems, and statistical data analysis tools.

Data pre-processing tools normally have a user-friendly interface allowing data analysts to interact with the tool easily. The first limitation of data pre-processing tools is that they normally do not support complex event processing. Also, data analysts must interact with the tools to manually specify commands for each step of the analysis as the function is not provided for users to specify a sequence of processing rules, i.e. the business logic of the analysis. In other words, analysis processes can hardly be automated.

Database management systems are powerful but, on the other hand, provide very limited interface to users; i.e. they require users to be competent in managing databases via queries, etc. NoSQL databases are for the purpose of cloud computing and thus more difficult to use. Temporal databases are emerging to address issues in handling temporal data but this technology is still immature.

Data analysts find statistical data analysis tools appealing due to their statistical data processing capabilities. With technical details hidden from the user, these tools are easier to learn, and concentrate more on the information in data than general-purpose programming languages. Each of these tools has advantages and disadvantages, provide similar functionality and have some limitations. Firstly, they are not specifically designed for event processing, so to utilise these tools to analyse event data is challenging. Secondly, while these tools are easier than normal programming languages, data analysts must still take a fair amount of time to learn some programming concepts before they can benefit from conducting event data analysis. Thirdly, these tools assume that datasets have already been cleansed prior to processing them, so in many circumstances, data analysts need to combine data pre-processing tools with the statistical tools to produce reliable results.

2.3 Event Processing Systems (EPSs)

2.3.1 Basic Concepts

Originating from both the area of active database systems [50] and distributed event-based systems [51], event processing is "computing that performs operations on events", including filtering, transforming and finding occurrences of given event patterns [23, 52]. Luckham [22] and Bacon et al. [53] pioneered research into event processing in the 1990s. The former mainly focused on simulation and pattern matching, whereas the latter focused on the construction of distributed applications using event-driven approaches. There are some distinct but related concepts regarding event processing:

- Complex Event Processing (CEP) [22, 54-58]: combines data from multiple sources and it typically focuses on detecting patterns in the order of occurrence of the incoming events
- Data Stream Processing (DSP) [59, 60]: focuses on filtering and aggregation of stream data
- Event Stream Processing (ESP) [22]: practically used as a near-synonym for Complex Event Processing in the context of event streams arriving over a network.
- Event Processing: A more general term covering all of the three concepts above. An event processing system (EPS) does some or all tasks associated with these three concepts

2.3.2 Overview of Event Processing Systems (EPSs)

An EPS 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 (Figure 2.2).

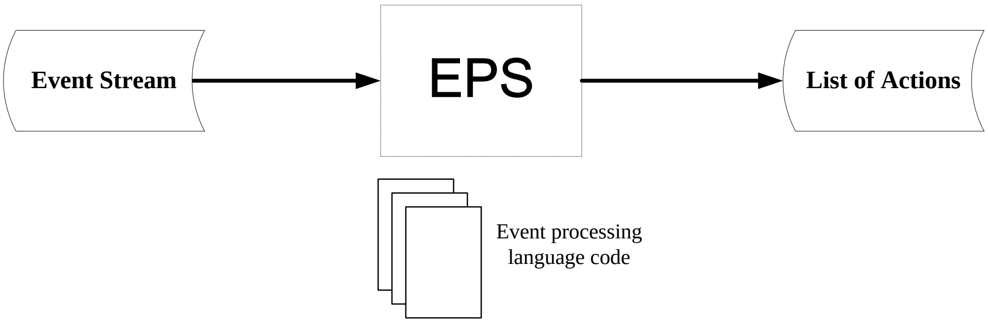


Figure 2.2 An event data analysis process using an EPS.

Based on several research papers, the components of an EPS platform can be categorised into:

- Event Metadata: contains specifications of event data and event processing rules. Currently, there is no common standard existing for defining and representing event data [61]

- Event Processing Language (EPL): the language used to express event pattern types or detect occurrences of event pattern types. Event pattern types must be written in EPL before they can be detected by the Event Processing Engine in the right way [56]
- EPL Compiler: translates an event pattern description from EPL into machine code that the Event Processing Engine can execute [62]
- Event Processing Engine: is at the core of an EPS. It matches event pattern types specified in the EPL code against the incoming event data and produces occurrences of these event pattern types once they are detected [63]
- Event Development and Management Tools: event development tools allow a user to define event data and processing rules, and event management tools are used for managing event data, event generation, event processing infrastructure, etc.
- Enterprise Integration Components: provides an interface for the system to connect to required external services. Examples of common Enterprise Integration Components include event pre-processing, publishing and subscribing, and business process invocation [64]
- Sources and Targets: specify the sources of incoming event data and the targets on which event-driven actions are performed [65]
- Event Database: data storage used for storing the event data that are processed by the Event Processing Engine [63]

Over the past few years many papers have been written on development of EPSs and their applications in various fields. Table 2.2 lists different types of event processing languages and their corresponding products. There are some common event operations (key features) in these products [23]:

- Event expressions: "the fundamental building block of an EPS, allowing the specification of matching criteria for a single event" [66]
- Filter: reducing the overall set of events to be processed by an EPS to a subset of events that are actually relevant for the given processing task, e.g., removing erroneous or incomplete data. The complexity of filtering mechanism in EPSs can vary depending on the expressiveness of the system, from basic static inspection of event attributes to comparisons against complex computed abstractions
- Transformation: changing event data from one form to another, including translation, splitting, aggregation, composition, etc.

- Translation takes each incoming event and operates on it independently of preceding or subsequent events. It performs a "single event in, single event out" operation.
- Splitting takes a single incoming event and emits a stream of multiple events, performing a "single event in, multiple events out" operation.
- Aggregation takes a stream of incoming events and produces one output event that is a function of the incoming events. It performs a "multiple events in, one event out" operation.
- Composition takes two streams of incoming events and operates on them to produce a stream of output events. This is akin to the join operator in relational algebra, except that it joins streams of events rather than tables of data.

- Event Pattern Detection: recognising high-level patterns by examining a collection of events (using correlation, aggregation, abstraction, etc.), which is the core mechanism in an EPS. This operation can be further broken down into three steps:

- Pre-detection: Event pattern types are validated (e.g. syntax and grammar errors) and then compiled into executable EPL code by the EPL Compiler.
- Detection: The Event Processing Engine executes the EPL code of the selected event pattern type generated from the previous step. Input event data is generated by a particular source and pushed to the Event Processing Engine. The Event Processing Engine then monitors the incoming event stream, matches the selected event pattern type in data, and produces an output of detected occurrences of this event pattern type.
- Post-detection: The Event Processing Engine stores all occurrences of the selected event pattern type into the Event Database, and notify relevant targets to perform corresponding actions.

Table 2.2 Types and examples of event processing languages.
(* indicates open-source products)

Language Type	Language / Product
Query-Based	CCL/Aleri [67] *CQL [68] *Esper [12] StreamSQL [69]
General Purpose	Netcool/Impact Policy [70] MonitorScript/Apama [71]

		EventSwarm [72]
Rule-oriented	Production rules	*Drools Fusion [73] BusinessEvents [74]
	Event-Condition-Action rules (ECA rules)	Amit [75] InfoSphere Streams [76]
	Other (Logic programming)	*ETALIS [77] *Prova [78]

In the following sub-sections, we provide further details about each category of EPL/EPS listed in Table 2.2.

2.3.3 Query-Based EPS

Query-based EPSs typically support an EPL extended from the ubiquitous relational database language SQL to query event data. The queries expressed in a query-based EPL are often referred to as continuous/continual queries [79]. In contrast to traditional non-persisting queries that work on persistent data, continuous queries are stored persistently in the database and applied to event streams. The processing paradigm in such systems is:

- Define queries in an SQL-like language
- Process queries
- Results of the processing step are only selectively stored in the database

In order to handle unbounded input streams, a common feature among these query-based languages is "the extensive operations on sliding windows" [80]. Sliding windows are used to divide the event stream into segments so these segments can then be manipulated and analysed without the system running into unlimited wait time and memory usage. There are different ways to calculate sliding windows. Examples include [81]:

- Time-driven model: The window is re-evaluated only at the end of each time step. CQL [68] adopts this model.
- Tuple-driven model: The window is re-evaluated every time a new tuple arrives. StreamSQL [69] adopts this model.

Since CQL and StreamSQL adopt different sliding window models, not all queries that can be expressed in CQL can also be expressed in StreamSQL, and vice versa [81]. In any one particular query-based language, it is important to stick to the consistent semantics so that all implementations using this EPL work in a consistent manner and generate expected results.

Query-based EPLs are considered as good at defining patterns of "low-level aggregation views according to event types defined as nested queries" [82]. However, any of these languages has shortcomings when expressing event pattern types. For example, CQL does not have the ability of expressing windows with a variable slide parameter [81]. Additionally, when detecting occurrences of the same event pattern type, different query-based EPLs may generate different results, but the user does not have the power to control which result should be generated [83].

2.3.4 General Purpose EPS

General purpose EPSs often have designed EPLs in the imperative style, e.g. Netcool/Impact Policy [70]. This apparently renders it hard for data analysts with limited IT expertise to learn and make full use of the EPS. In Netcool/Impact Policy, a policy consists of "a set of statements written in either the Impact Policy Language (IPL) or JavaScript" [70]. Event handling is enabled but limited to parsing incoming events, sending events from one source to another, updating events and deleting events. Event pattern detection is relatively restrictedly supported. Based on real-life experience, building a set of policies that work on event data analysis is equivalent to building a bespoke program in a common programming language, e.g. Java.

Another mature general purpose EPS that focuses on pattern matching is the EventSwarm framework. Its conceptual model [72] is shown in Figure 2.3. It provides a programming framework based on Java. EventSwarm offers "a range of pre-defined abstractions and pattern components implemented in the Java programming language" [66]. There are two typical styles of application built using this framework, namely applications built for specific, pre-defined patterns or abstractions, and domain-specific applications that allow end users to define new patterns. For the latter, a graphical interface can be developed and provided for the user. It also has the following outstanding features [66]:

- "the ability to filter on any computed abstraction that matches a single event, including statistical analysis"
- the capability of "using causal precedence in sequence patterns"
- It is advantageous in managing time and ordering issues of events due to event timestamps, buffering, flexible relationships between events, time skew allowance and causal ordering
- the capability of statistical calculation on time-driven sliding windows and use the calculated results in the EPL code

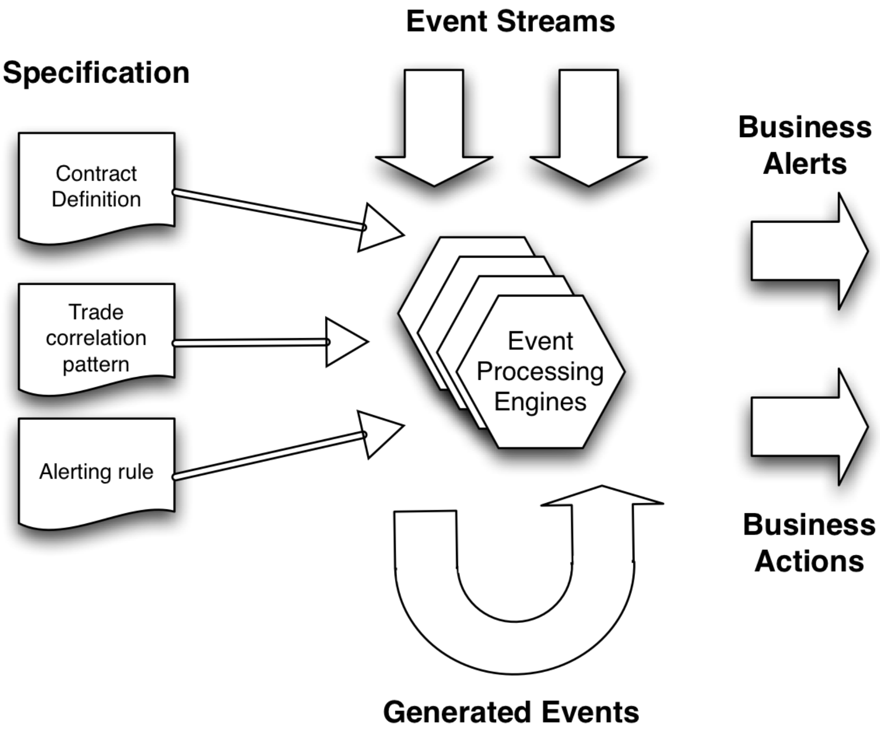


Figure 2.3 EventSwarm architecture.

2.3.5 Rule-Oriented EPS

Compared with query-based EPSs and general purpose EPSs, rule-oriented EPSs work best on describing higher-level conditional business event pattern types.

2.3.5.1 Production Rules

In the area of expert systems (or knowledge-based systems), production rules originated in the 1980s. Since then they have been investigated comprehensively, become very popular, and has been successfully applied commercially in various domains such as medicine, troubleshooting in telecommunication networks, and computer configuration systems [82].

Most production rule systems as well as database implementations of production rules adopt forward-chaining operational or execution semantics [82]. The rules are in the structure of "if Condition then assert Conclusion/Action". Whenever the rule base manager needs to modify the condition, the conclusion/action needs to be updated accordingly as well.

While production rules in nature "react to condition state changes and have no connections to event data" [82], recent work has attempted to extend production rule systems with new features like object models and fact base to make CEP possible. Specifically, in the declarations of production rules, event types are defined. Incoming events are initialised as instances of the declared event types and are dynamically added to the fact base. In the rule condition, some operations such as filters and event pattern detection may be applied to the instances of events. If the condition is fulfilled, the action is triggered. Representative tools include Drools [73] and BusinessEvents [74].

However, contemporary production rule systems still lack expressiveness as "they do not have a clear declarative semantics, suffer from termination and confluence problems of their execution sequences and typically do not support expressive non-monotonic features such as classical or negation-as-finite failure or preferences" [82]. Therefore, certain event pattern types can hardly be expressed in such systems. Some research in this area focuses on extending the core production systems to enhance their expressiveness [84], but this results in more complication of the usage of the system.

2.3.5.2 ECA Rules

The advent of Event-Condition-Action (ECA) rules [85] was due to the necessity to react to different kinds of events occurring in active databases [86]. There are three components in an ECA rule:

- Event: specifies the event that triggers the invocation of the rule. The event itself can be a composition of different event types, in which case it is called a composite event.
- Condition: consists of the conditions that need to be satisfied, in order to carry out the specified action. The condition is only checked upon occurrence of the specified event.
- Action: specifies the actions to be taken on the data.

Examples of active database systems using ECA rules include ACCOOD [87], Chimera [88], COMPOSE [89], NAOS [90], HiPac [91]. Expert system techniques are sometimes integrated to allow rule triggering to be automatic. Apart from active databases, ECA rules have also been applied in conventional databases, where the condition is a traditional query to the local database; and in memory-based rule engines, where the condition is a test on the local data.

Event processing systems applying ECA rules in the corresponding EPL include Amit [92] and InfoSphere Streams [76]. These systems support event algebra operators analogous to those provided by active database event algebras, e.g. Snoop [50] and SAMOS [93], where complex expressions can be created using operators like And, Or, Sequence, etc. to describe event pattern types that can be applied over the event streams in real time.

Two typical usages of ECA rules in event data analysis include detecting and reacting to occurrences of particular event pattern types in the database that may undermine the data integrity in real time, and executing some business logic on incoming event streams.

2.3.6 Discussion

Overall, event processing technology is broadly applied in many application domains but it is still not a mature technology [94] that is actively researched. Among all the EPSs available, many commonalities exist but the underlying EPL and features available in products vary according to their targeted markets and applications. Each of the languages/products listed in Table 2.2 has its advantages and disadvantages that reflect the usual tradeoffs between simplicity and expressiveness. Therefore, the performance of an event data analysis greatly depends on the selected EPL/EPS, and whatever EPL/EPS is used, there are always limitations. Switching to a different EPS is by any means troublesome and costly.

Another issue is that rule management in EPS is yet to be researched. Luckham initially claimed event processing rules as "the foundation for applications of CEP" [22]. In a later publication, he claimed that managing large sets of event processing rules is a challenge which has not yet been effectively tackled [95]. The example illustrated in Section 2.1.4 indicates the challenge of managing rules to react to the evolving needs of researchers.

In the next section, we will consider technologies specifically designed to facilitate defining and managing rules and look into their possible role in managing event data analysis rules.

2.4 Role of Rule-Based Systems in Event Data Analysis

Rule-Based Systems (RBSs) are types of knowledge-based systems in which knowledge is represented as rules stored in a database. Due to the obvious advantages over conventional business rules expressed in conventional programs or designed languages resembling natural ones, RBSs have gradually been chosen to store business rules to assist in data processing.

This section will review RBSs as this is an area that was found to be lacking when discussing EPSs in Section 2.3 and there are actually a number of attempts to apply RBSs to data processing. Section 2.4.1 illustrates conventional RBSs and raises some problems with these systems. Section 2.4.2 illustrates another type of RBSs called ripple-down rules (RDR), which can address the issues existing in conventional RBSs.

2.4.1 Conventional Rule-Based Systems

Conventional rule-based systems apply production rules introduced in Section 2.3.5.1. These systems acquire new knowledge by inserting, deleting, modifying existing knowledge.

A production rule system consists of:

- A set of rules

- Working memory that stores temporary data
- A forward-chaining inference engine (an example of forward chaining decision making is shown in Figure 2.4)

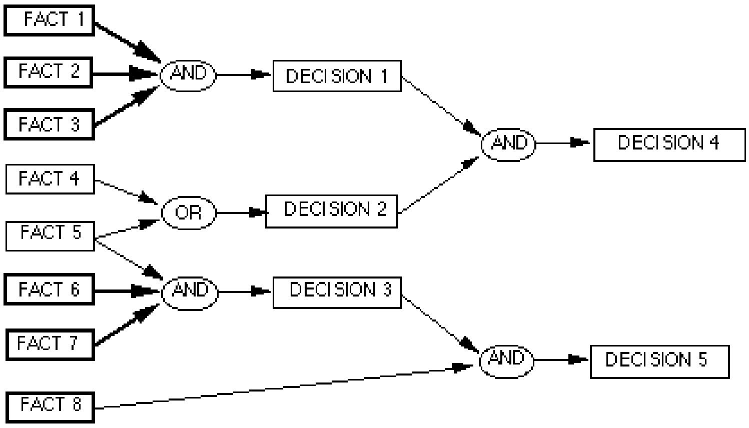


Figure 2.4 Forward-chaining in production rule systems.

Examples of rule-based systems for data processing include Eprentise Data Quality Software [96], which offers a rule-based data quality engine for standardisation and matching (merging and de-duplication), enabling business users to check or change rules directly, with minimal IT involvement. In this type of system, a rule base contains integrity rules to test integrity of the data and cleansing rules to take action when violations are encountered [97].

Two acknowledged problems in attempts at commercial use of conventional rule-based systems in data analysis are [98]:

- Unexpected interaction: It is hard to avoid unexpected interaction between rules as the size of rule base becomes huge
- Maintenance: In real-world data processing, data analysts constantly need to update rules incrementally to correct undesired behaviour (or deal with unusual cases). However, it makes it considerably difficult to manage the rule base, especially when the rule base becomes huge, as modifying or adding rules may collapse the behaviour of the rest of the system

For the first problem, a partial solution is to utilise partitioned production systems [99]. In these systems, only a subset of the rules are active at any time, so the knowledge engineer can concentrate on just a small number of rules from time to time, and the interactions between rules can be managed much more easily. As for the second problem, the best known approach is to use Ripple Down Rules (RDR) [100], which also deal with the first problem. We will comprehensively explore RDR in the following section.

2.4.2 Ripple-Down Rules (RDR)

The emergence of RDR is inspired by an insight for domain experts' acquisition pattern: "experts can never explain how they reach a conclusion, rather they justify that a conclusion is correct, and provide this justification in a particular context" [100]. This insight suggests that knowledge acquisition should be incremental and captured within context. Therefore, unlike other rule management systems, RDR is an error-driven, case-based, incremental rule acquisition framework. RDR learns incrementally with the domain expert adding rules when a false conclusion occurs, but never corrupting the existing knowledge base. There are two types of false conclusions: false positives and false negatives [101]. False positives happen when some false conclusions are included because of over-simplified rules, and false negatives happen when some true conclusions are excluded because of some over-precise rules. After summarising these false conclusions, domain experts then adjust the rule base only by adding new rules. 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 compare new cases by the domain experts.

As a 20-year old knowledge acquisition technique, RDR has been used by a number of companies due to the following advantages [102, 103]:

- Rule bases can be more easily constructed and maintained by domain experts rather than knowledge engineers
- High accuracy
- Providing the validation of conclusions
- Requires extremely short time to add new knowledge (It has been proved that the whole processing of adding a rule including checking cornerstone cases takes only a couple of minutes [104])
- High efficiency

On account of these advantages, RDR can successfully eliminate the limitation of normal RBSs. There are quite a few successful examples of RDR applications in numerous areas such as email management. For example, "EMMA" - an E-Mail Management Assistant [105] uses a type of RDR to manage e-mails. Users are enabled to define their rules to deal with coming emails by selecting and filling in fields.

There have been some attempts to apply RDR to data analysis in general. For example, RDR has been used as the basis of existing linkage techniques to detect duplicate invoices [106]. This system is called "Duplicate Classifier with Ripple Down Rules", which enables domain experts to incrementally specify relevant business rules referring to concrete examples; meanwhile, the rule base (knowledge base) is updated over time. A rule example is as follows:

If: Invoice Number [j] = Invoice Number [i]
& Invoice Date [j] <u>*Similar with*</u> Invoice Date [i]
& Amount [j] = Amount [i]
& Vendor Name [j] = Vendor Name [i]
& Recurring Vendor = true
Then: Not a duplicate invoice

Figure 2.5 depicts how the "Duplicate Classifier with Ripple Down Rules" system works.

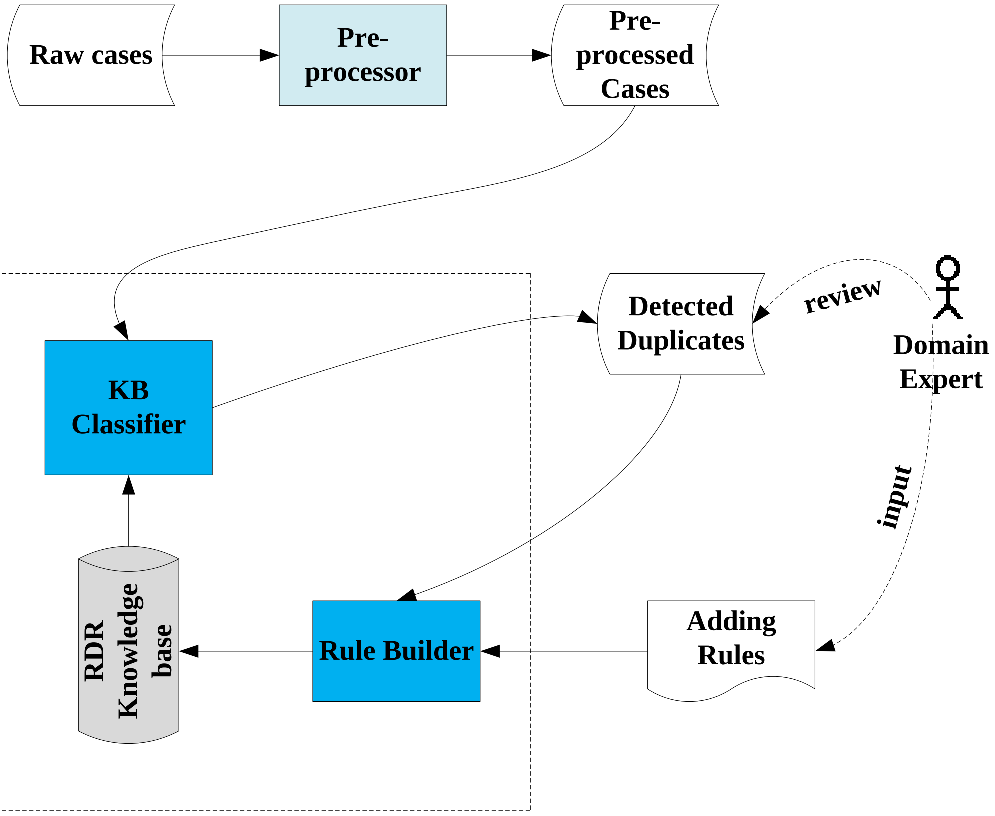


Figure 2.5 The architecture of an RDR application on duplicate invoice detection.

Another successful RDR application on data analysis is to standardise address information, one major step of address cleansing in enterprises [107], which is the second stage of an IBM product – IBM Quality Stage [108]. Apart from RDR, the system also adopts variants finder and synonyms finder in cooperation with RDR framework, focusing on one data field – address in this case, by splitting the address into several subfields and classify them according to the context, that is, using RDR in a classification way [109]. Also, they have a matching tool to identify similar or duplicate records in the third stage, which is optional in their proposed system. To sum up, they are using RDR to deal with one single domain (customer data), focusing on semantics (parsing and standardisation), and using additional techniques in other parts of the tool to address issues that RDR cannot handle.

One limitation of RDR is that it does not provide an appropriate solution for issues associated with the unique characteristics of event data. In other words, complex event processing is not supported by RDR.

2.4.3 Using Rule-Based Systems for Managing Event Data Analysis

We now perform a comparison between the capabilities of data processing, event processing and rule-based systems concepts in order to examine their role in event data analysis. We have selected three capabilities to be used in this comparison.

- Automation (A) – Can the system complete tasks automatically?
- Complex Event Processing (CEP) – Is the system capable and efficient when operating on temporal data? In other words, does the system support complex event processing?
- User-Driven Rule Set Evolution (URSE) – Does the data analyst have the control over the business logic of the data analysis? In other words, can the system be evolved by the data analyst alone without IT experts involved?

Table 2.3 How do existing potential technologies/approaches meet the criteria.

√ : Yes, ×: No, √ *: Possibly yes but not mature or efficient

DPPT: data pre-processing tool; DBMS: database management system; SDAT: statistical data analysis tool; EPS: event processing system; RBS: rule-based system; RDR: ripple-down rules.

Category	Technologies / Approaches	A	CEP	URSE
Generic	*DPPT* (Section 2.2.1)	×	×	×
	DBMS (Section 2.2.2)	×	√ *	×
	SDAT (Section 2.2.3)	√	×	×
EPS	*EPS* (Section 2.3)	√	√	×
RBS	*Conventional RBS* (Section 2.4.1)	√	×	×
	RDR (Section 2.4.2)	√	×	√

The results of the comparison are displayed in Table 2.3. None of these technologies or approaches has all capabilities required for conducting event data analysis. Specifically:

- Generic Approaches: They can hardly handle time-related issues. In other words, they do not have the capability of complex event processing. Besides, data analysts cannot evolve the system by themselves.
  - DPPT (Data Pre-Processing Tool): Due to the niche market of this type of tools, these tools are only capable in generic data pre-processing on a rather fundamental level. Thus, in many cases they play a supplementary role for mainstream tools like Microsoft Excel. Users need to give commands step by step so they are not automatic.
  - DBMS (Database Management System): Competent users can define their own queries to manage the database according to their needs. However, DBMSs are deemed a high-level technology to data analysts who have limited IT expertise, so they can hardly use the system by themselves. Except temporal database, which is still an immature technology that is not commercialised, DBMS cannot handle time issues efficiently.
  - SDAT (Statistical Data Analysis Tool): These tools are powerful provided that the user can write specific programs/scripts. This requires a certain level of programming skills. Once the program is written, it can automate the data analysis process.

- EPS (Event Processing System): The obvious advantage of EPS is the capability of complex event processing. Learning any specific EPL is time-consuming so data analysts themselves cannot use the system easily by themselves. Regardless of the type of EPL, once EPL code is modified, the previous data analysis can no longer be repeated. Whilst an overwhelming body of research work focuses on the operational issues, e.g. event processing language expressiveness and performance [110], there is a lack of research on event processing rule management. Another important criterion for EPS is giving data analysts the ability to customise the system according to their needs [111], which is not given sufficient emphasis, as most work on EPS does not support user-driven rule set evolution in event data analysis no matter how good the language expressiveness is. Among the very limited discussion on user-driven rule set evolution, there are two important insights: reuse of existing event patterns is of great importance for efficiency [112]; the idea of rule templates for EPS for completing rules as well as decoupled "building blocks" of rule logic is promising [113]. 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 (or rule base) is needed to re-build the entire rule set. Besides, most literature on event processing rule management tends to focus on building each single rule in isolation and disregard the management of the rule set as a whole. It has been proved in the knowledge acquisition community that when the size of the rule set gets huge, it becomes very difficult to maintain the rule set, as any modification of the rule set may cause the system to collapse [103]. Thus in most existing EPSs, as rules may be closely associated with each other, it is difficult to keep track of changes effectively.

- RBS (Rule-Based System): Rule-based systems are not designed for data analysis purpose, but attempts have been made to apply them to assist data analysis processes. Thus, they do not have obvious advantages on data analysis; however, RDR, a sub-type of RBS, has been applied successfully to some certain types of data analysis, e.g. duplicate invoice detection and address standardisation. It proves to be successful in user-driven rule set evolution of the system. However, RDR has never been used for event data analysis.

Based on all the discussion above, we can find out that the research gap is:

There is no existing technology or approach for event data analysis that enables complex event processing and facilitates user-driven rule set evolution.

2.5 Conclusion

This chapter firstly provided some background regarding event data analysis, and then reviewed literature regarding approaches for event data analysis. It explored three major categories of technologies or approaches: generic approaches, event processing systems (EPSs) and rule-based systems (RBS). Examples of each category have been given to illustrate their capabilities.

Finally, this chapter compared all these technologies or approaches using three capabilities: automation, complex event processing and user-driven rule set evolution, and found that no single approach has all these capabilities. To be more specific, capability A (automation) can be easily met as long as the event data analysis logic can be defined as a runnable process. Capability CEP (complex event processing) can only be satisfied by EPS, which however does not facilitate user evolution of the system. Capability URSE (user-driven rule set evolution) can only be satisfied by RDR, which in nature does not support complex event processing and thus has not yet been applied to event data analysis. The research gap is "*There is no existing technology or approach for event data analysis that enables event processing and facilitates user-driven rule set evolution.*"

Next chapter will resume the discussion on the research gap, raise research questions and illustrate the methodology of conducting this research.

Chapter 3 Research Plan

This chapter provides an overview of the research plan used in the thesis. Section 3.1 summarises the research gap identified in Section 2.5 and expresses the corresponding research questions addressed by this thesis. Section 3.2 describes the research approach. Finally, Section 3.3 provides the details of the methodology used in this research.

3.1 Problem Statement

Based on the literature review, we find that event data analysis is best conducted using a combination of multiple technologies or approaches providing different capabilities. We have investigated possible technologies or approaches in three different categories (data processing tools, event processing systems and rule-based systems) and found that none of them can satisfy all the following capabilities (first introduced in Section 2.5):

- Automation (A) – Can the system complete tasks automatically?
- Complex Event Processing (CEP) – Is the system capable and efficient when operating on temporal data? In other words, does the system support complex event processing?
- User-Driven Rule Set Evolution (URSE) – Does the data analyst have the control over the business logic of the data analysis? In other words, can the system be evolved by the data analyst alone without IT experts involved?

Our first research question is:

How can we provide a new method
that enables complex event processing and facilitates user-driven rule set evolution?

As discussed in Chapter 2, complex event processing can be enabled by some existing technologies (e.g. EPS), and user-driven rule set evolution can be facilitated by some approaches (e.g. RDR). One possible new method is to leverage the EPS technology with the RDR approach in one system. Thus, the second research question is:

How can we leverage existing Event Processing Systems to support the new method?

Once we find out how to leverage existing event processing systems, another problem emerges. All these techniques including Event Processing Systems diverge from each other in many aspects, e.g. the data model they use, the availability of application programming interfaces, etc. This renders it extremely hard to accommodate these techniques in one system so that they can interoperate with each other. As a result, our last research question is:

How can we facilitate the interoperability of event data in any proposed system?

3.2 Research Approach

To answer the first research question, this thesis proposes a new method that enables complex event processing and facilitates user-driven rule set evolution, by leveraging the capability of the EPS technology with the RDR approach. The reason why this new method can answer the first research question is that the RDR approach brings in the capability of user-driven rule set evolution (see Section 2.4.2), and the EPS technology is specialised in complex event processing (see Section 2.3).

For the second research question, this thesis proposes to develop an architecture that meets the following requirements:

- The RDR approach is adapted to play the role of routing the event processing logic, while still supporting incremental acquisition that enables data analysts to define and add rules by themselves;
- Any EPS can be integrated into the system so that data analysts are allowed to conduct event data analysis without any concern about which event processing language/engine to use.

For the third research question, this thesis proposes a framework that facilitates event data modelling. This can enable the interoperability between components of an event data analysis system. Specifically, the new event data modelling framework should:

- Allow event pattern types and occurrences to be defined;
- Abstract existing event and event pattern occurrence representation formats in a consistent manner;
- Be easily extended to different types of datasets and domains.

The architecture that integrates EPS with RDR, and the event data modelling framework that facilitates the interaction between components of an event data analysis system are the two main artefacts that will be developed as part of this research. The following section will illustrate our research methodology to realise these specific objectives for the research.

3.3 Research Methodology

Figure 3.1 demonstrates the research methodology used in this thesis, which is adopted from the Software Development Life Cycle (SDLC) [114]. The main reasons why we use the iterative SDLC methodology in this research include [115]:

- This process is easy to control and the project is easily monitored;
- It is a highly structured systematic process which allows the capture of all evolving requirements throughout the development of the proposed work;
- The artefacts in this research that are dependent on each other can be incrementally refined;
- The product is easy to maintain.

The two artefacts described in the previous section (the architecture that integrates EPS technology with an RDR approach, and the event data modelling framework) will be developed iteratively using the SDLC in four phases. These include Analysis, Design, Implementation and Evaluation. The weight and importance of each phase is varied in each cycle. In early iterations, we focus more on the Analysis phase than the other phases. We concentrate more on the Design and Implementation phases in the middle stage of the research. The Evaluation phase is emphasised when we move toward the end of the research. Figure 3.2 illustrates the research methodology used in our research. The rest of this section provides the details of each SDLC phase used in our research methodology.

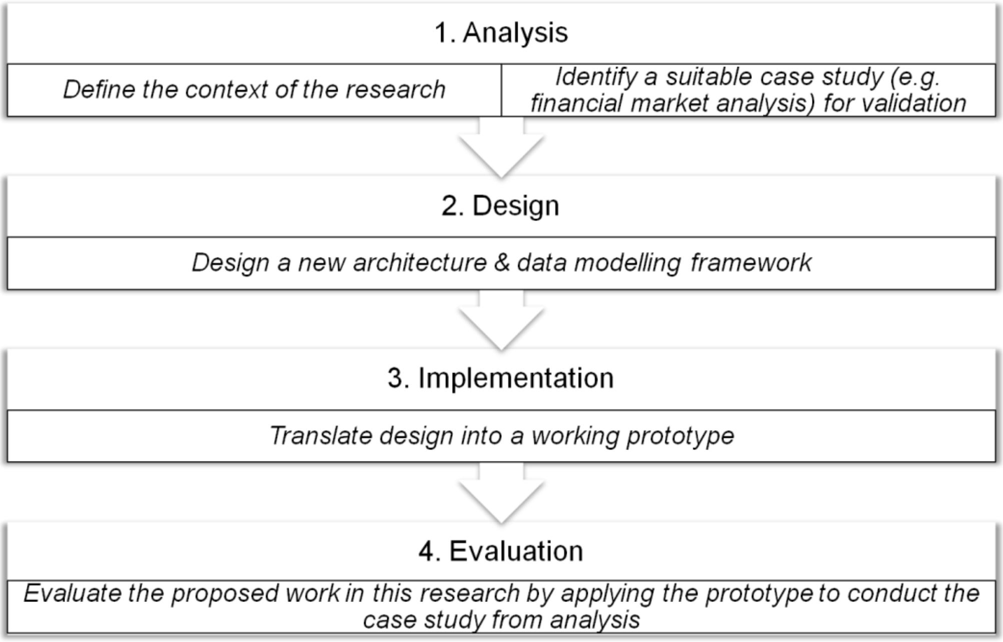

1. Analysis
Define the context of the research
Identify a suitable case study (e.g. financial market analysis) for validation
2. Design
Design a new architecture & data modelling framework
3. Implementation
Translate design into a working prototype
4. Evaluation
Evaluate the proposed work in this research by applying the prototype to conduct the case study from analysis

Figure 3.1. Research methodology.

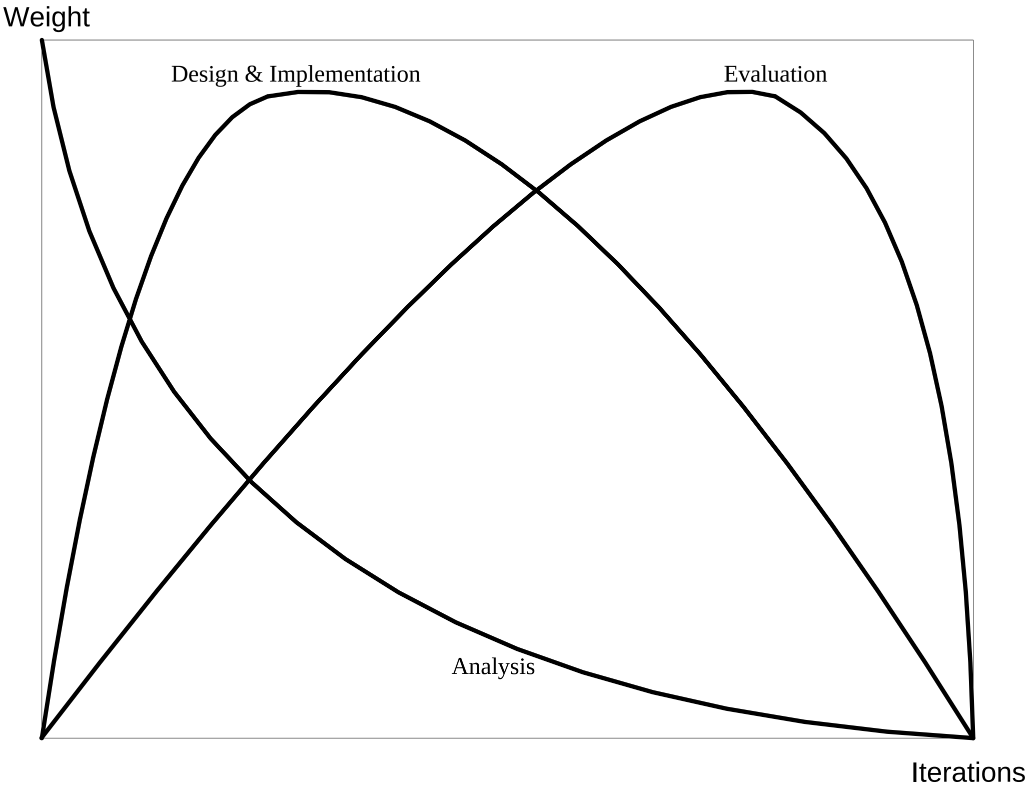


Figure 3.2 Weights of each deliverable throughout the research.

- Analysis

In this phase, an investigation is carried out on the background of the research problem (or from previous iterations if applicable). Advantages and disadvantages of the existing solutions are determined. A thorough study on event data analysis was conducted in Chapter 2 of this thesis, which raised the challenge of rule management in event data analysis (see Section 2.1.4). Different types of possible solutions have been discussed and we have identified the EPS technology and the RDR approach that have the most potential for inclusion in our design.

- Design

In this phase, we carry out the design of the two artefacts, namely the architecture and the event data modelling framework. The first half of the research concentrated on the architecture design, defining the components of the architecture, and the facility to permit communication between components. The description of the proposed architecture is provided in Chapter 4, and we elaborate on the proposed event data modelling framework in Chapter 5.

- Implementation

For each iteration of the SDLC cycle, a prototype is developed or refined based on the proposed architecture in an incremental manner. The details of the prototype implementation can be found in Chapter 6.

- Evaluation

A case study regarding financial market data pre-processing will be used to evaluate the proposed work in this research. Use cases and event processing rules are selected based on real-life experience of finance experts. Further details of the evaluation process and results are provided in Chapter 7.

3.4 Conclusion

Following the discussion of related work in Chapter 2, we have further discussed the possible approach to resolving the limitations of existing solutions to event data analysis in this chapter. We have broken down the research problem into three research questions and then proposed a specific research approach to answering all the research questions. Two artefacts need to be developed, namely a software architecture in which event processing systems (EPSs) are integrated with ripple-down rules (RDR); and a new event data modelling framework facilitating the construction of event data models that defines a standard representation of event data and event pattern occurrences. We also describe the methodology used to carry out this research as well as provide an overview of the two research artefacts in this chapter. We will elaborate on these two artefacts and our implementation respectively in the next three chapters.

Chapter 4 Proposed Architecture

This chapter illustrates the first design artefact of the research called the EP-RDR architecture. We start with an overview of the proposed architecture in Section 4.1. Sections 4.2, 4.3 and 4.4 describe in detail components in the business layer of the architecture. The data layer and the user layer are illustrated in Sections 4.5 and 4.6 respectively. In Section 4.7, two use cases of the system are demonstrated using sequence diagrams. The material presented in this chapter has been published in [116-118].

4.1 Overview

As mentioned in Section 3.2, the proposed architecture should meet the following requirements:

- The RDR approach is adapted to play the role of routing the event processing logic, while still supporting incremental acquisition that enables data analysts to define and add rules by themselves;
- Any EPS can be integrated into the system so that data analysts are allowed to conduct event data analysis without any concern about which event processing language/engine to use.

The proposed architecture is referred to as "Event-Processing RDR (EP-RDR)" to reflect the integration of event processing system technology with an RDR approach. Basically, EP-RDR is a novel approach to enable incremental user-driven rule set evolution in event data analysis. From the perspective of rule management, we aim to adopt the RDR approach for event processing purposes. The advantages are eliminating event processing rule rebuilding, enhancing the reuse of existing event processing rules, keeping track of event processing changes, simplifying rule management and thus avoiding rule base collapse. From the perspective of complex event processing, we aim to take advantage of existing EPS technologies by allowing them to be integrated into the system so that data analysts are able to conduct their event data analysis without any concern about which event processing language/engine to use.

The key feature of the architecture is that the event processing rules in the system can be incrementally enhanced by data analysts, meanwhile eliminating the risk of corrupting the rule base. They can execute existing rules, inspect results, and evolve the rule base according to the results. IT experts will only play the role of defining and deploying event pattern types. Specifically, the purpose of EP-RDR is to support the following three use cases:

For data analysts:

- Rule Addition: This is the case where the event data analyst defines a rule and adds it into the rule base.
- Rule Execution: This is the case where a defined rule base is executed upon event datasets.

For IT experts:

- Event Pattern Type Definition: This is the case where the IT expert defines and deploys event pattern types.

The EP-RDR architecture is shown in Figure 4.1. Conceptually, it is organised into three layers according to the widespread three-layer architecture [119, 120]:

- *Data Layer*: This layer includes data persistence mechanisms, e.g. databases (often relational databases), and the facility called "data access components" for encapsulating data persistence mechanisms and exposing data stored in databases to the business layer so that applications can access the data. In the data layer of the EP-RDR architecture, the data persistence mechanisms include an Event Pattern Definition Table, an Event Processing Rule Base, an Event Pattern Occurrences Stack repository and a List of Actions repository. All these data persistence mechanisms will be elaborated in Section 4.5.
- *Business Layer*: This layer is sometimes cited as "logic layer" or "application layer". It includes core components and interaction between these components in the application and essentially it controls the application's functionality. The two core components in the business layer are the Ripple-Down Rule System (RDR) and an Event Pattern Detection as a Service (EPDaaS). The RDR component has two parts: the Rule Builder that manages the business logic of the event data analysis, i.e. event processing rules, and the Engine that executes the rules. The EPDaaS component invokes one or more EPS and detects occurrences of event pattern types defined in the rules. More details of each of these two components and their interaction will be provided in Sections 4.2 and 4.3. Underlying EPSs that can be invoked by EPDaaS will be illustrated in Section 4.4.
- *User Layer*: This layer can also be called "user interface layer" or "presentation layer". It is the topmost level of the architecture. It displays information related to the business functionality and allows the user to interact with the Business Layer via Graphical User Interfaces (GUIs) or web pages. The user interfaces in the user layer include an Event

Processing Rule Manager GUI and an Event Processing Rule Processor GUI. Both these two user interfaces will be discussed in Section 4.6.

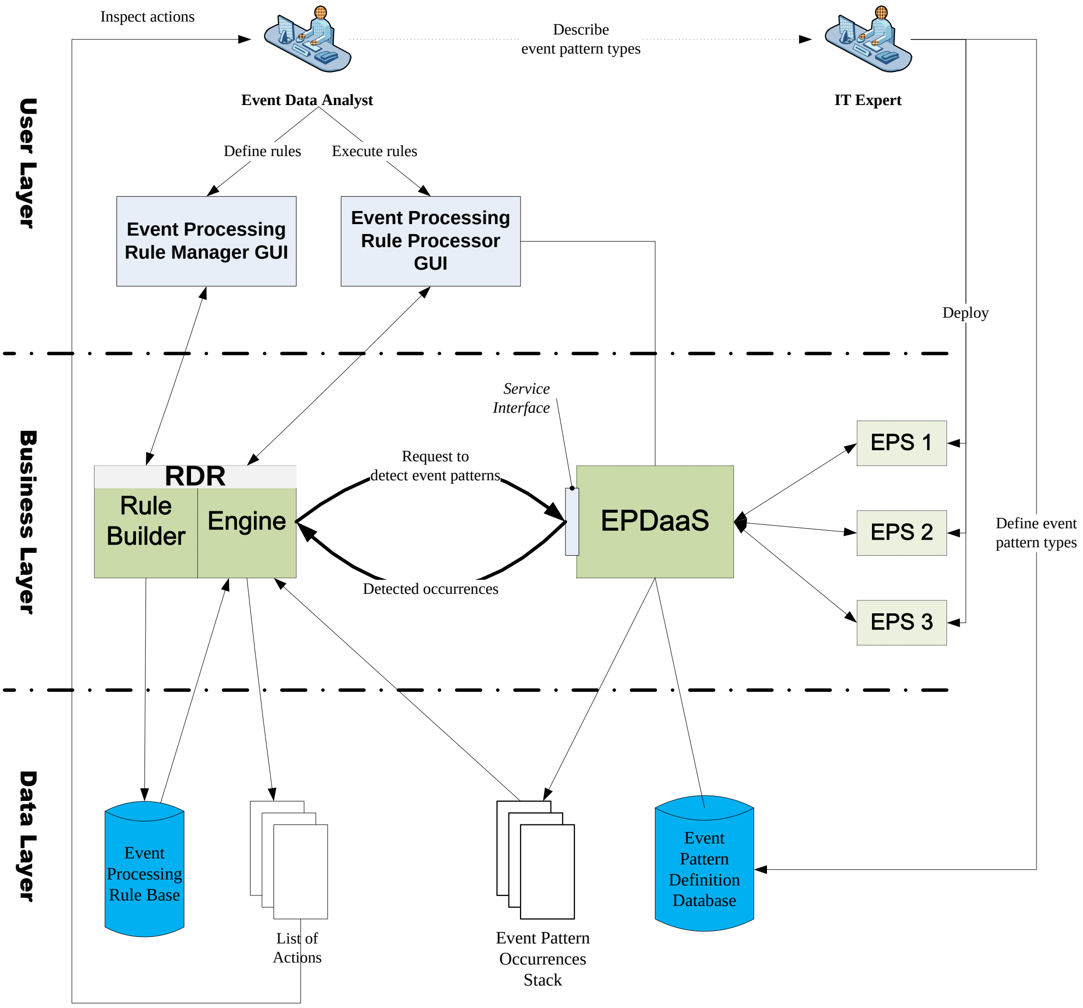

Inspect actions
Describe event pattern types
Event Data Analyst
IT Expert
Define rules
Execute rules
Event Processing Rule Manager GUI
Event Processing Rule Processor GUI
Deploy
User Layer
Business Layer
Data Layer
Service Interface
Request to detect event patterns
RDR
Rule Builder
Engine
EPDaaS
EPS 1
EPS 2
EPS 3
Define event pattern types
Detected occurrences
Event Processing Rule Base
List of Actions
Event Pattern Occurrences Stack
Event Pattern Definition Database

Figure 4.1 Proposed EP-RDR architecture.

4.2 The RDR Component

We now describe the RDR component in more detail. Firstly, we delve more into different RDR algorithms and inference techniques, and determine which algorithm and inference technique is the most appropriate to be applied to the EP-RDR architecture. After that, we illustrate the two sub-components of the RDR component, namely the RDR Rule Builder and the RDR Engine.

4.2.1 Selection of RDR Algorithms and Inference Techniques

There are a number of types of RDR that use different algorithms and inference techniques. In this section, we explore these algorithms and inference techniques and determine the most appropriate algorithm and inference technique of RDR for the EP-RDR architecture.

4.2.1.1 Algorithm Selection

Many RDR algorithms exist. The most frequently used ones include the following: Single Classification Ripple Down Rules (SCRDR), Flat Ripple Down Rules (Flat RDR), Multiple Classification Ripple Down Rules (MCRDR).

- Single Classification RDR (SCRDR)

An SCRDR [121] knowledge base is a finite binary tree with two edges labelled with "except" and "if-not" respectively and nodes labelled with primary rules. SCRDR works from the root of the tree passing a data case all the way to the end when there are no more nodes to evaluate. If the case entails the condition of the rule, records the conclusion temporarily, overrides the previous conclusion, and then passes it through the except branch to the next node. Otherwise, the case will be passed through the if-not branch to the next node. Figure 4.2 is an example pattern of SCRDR. The rule number signifies the order in which rules are added into the system. It can be easily seen from the examples that the newly added rules revise the knowledge without collapsing the original knowledge base. This process agrees with the pattern of human knowledge acquisition where any knowledge people acquire at a stage may have limitations and exceptions and it may not always be true. RDR enables users to at any stage modify what they have already learned before.

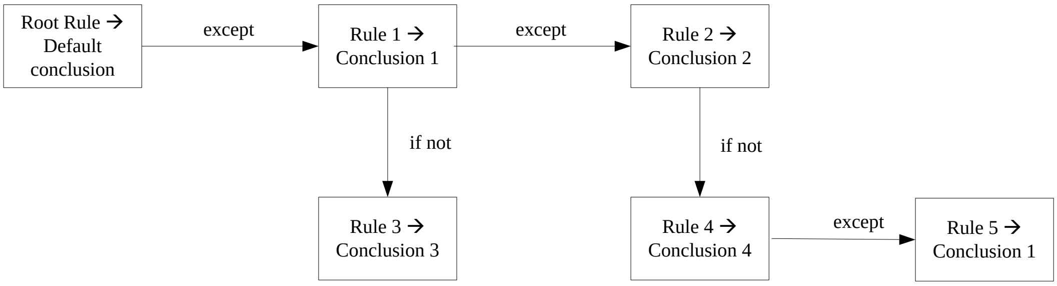


Figure 4.2 SCRDR example pattern [101].

SCRDR has been successfully adopted in different domains such as PEIRS on pathology (the first RDR system) and an address standardisation system [107].

The strengths of SCRDR is that rule bases can be easily revised [103]. However, SCRDR does not clearly link which attributes result in which conclusion [103]. Also, SCRDR is not suitable for multiple independent classifications (exponential redundancy increase) [122]. Most importantly, when using SCRDR, the rule base may be ill-structured and considerable repetition of knowledge may result [122].

- Flat RDR

A Flat RDR is different from SCRDR in that it is an n-ary tree of depth two with each node labelled with a primary rule. The root is a default rule leading to a dummy conclusion. Depth 1 contains classification rules and Depth 2 contains deletion rules to refine Depth 1 conclusions. All the classification rules on Depth 1 are evaluated and the system records the conclusions, overriding the dummy conclusion. New deletion rules are added when the classification is an over-generalisation. A new classification should be added when the classification is an over-specialisation. It is mainly designed for multiple classifications. So it does not function as well as SCRDR in single classifications [101].

Figure 4.3 is a simple example pattern of Flat RDR. A complete Flat RDR classification contains classification and deletion. The users have to add new classification rules continuously but not add exceptions on existing rules. This may result in difficulties when managing the rule base.

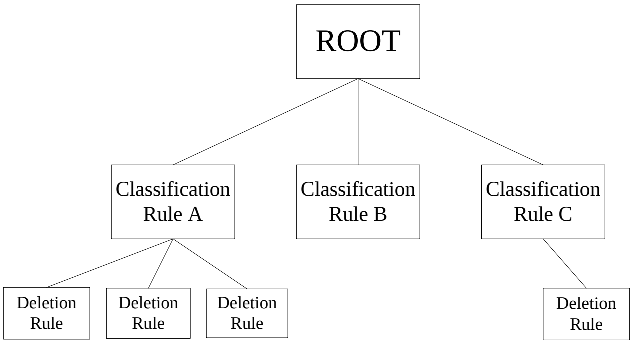


Figure 4.3 Flat RDR example pattern [101].

- Multiple Classification RDR (MCRDR)

MCRDR [122], as an extension of SCRDR, allows multiple independent classifications. An MCRDR is an n-ary tree with only except branches. If a node is true, all its children are evaluated. Figure 4.4 is an example pattern of MCRDR. As can be seen from these examples, MCRDR can lead to multiple conclusions.

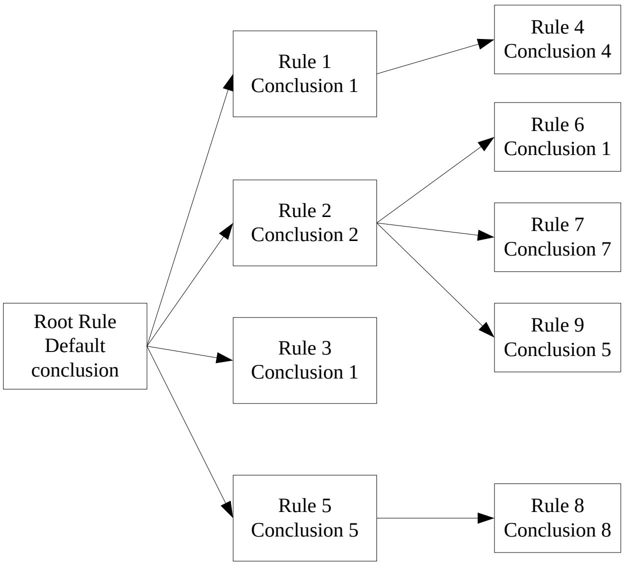


Figure 4.4 MCRDR example pattern [101].

While MCRDR is not efficient for single classifications [103], it has some advantages over other algorithms. Firstly, MCRDR clearly links attributes to corresponding conclusions. Secondly, in MCRDR all conclusions are independent so it outperforms other algorithms when a single subject may lead to multiple conclusions.

In event data analysis, each single event may be involved in many multiple situations (event patterns), and thus leads to a number of different actions on the raw data. Under this circumstance, SCRDR may be ill-structured and results in exponential redundancy increase when the size of the rule base gets larger; Flat RDR is hard to manage; and MCRDR allows multiple conclusions without the repetition issue. Thus, MCRDR is the most suitable existing RDR algorithm that can be applied.

4.2.1.2 Selected Inference Technique

There are primarily two inference techniques, both of which can be applied to any of the algorithms discussed above:

- Refinement rules: Refinement rules are stored and processed as a tree. All children rules are reached only if their parent is fulfilled. Data analysts can add rules after any leaf rule of the tree.
- Linked-production rules [123]: 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.

Linked-production rules have obvious advantages over refinement rules in regard to redundancies and maintenance, so we adopt linked-production rules as the inference technique in implementing the RDR component.

4.2.2 RDR Rule Builder Sub-Component

An event processing RDR rule is defined as:

```
If
        /* an event pattern occurs*/
        the number of occurrences of a particular event pattern type >= 0;
Then
        case action;
        inference action: go to rule a;
Else
        inference action: go to rule b;
```

The role of the RDR Rule Builder is to permit incremental definition of rules based on the presence of event patterns – each event "situation" is represented as an event pattern type in a rule. The RDR Rule Builder is used in the Rule Addition use case, where the Event Processing Rule Manager GUI directly interacts with the RDR Rule Builder. Specifically, when the event data analyst defines a rule and adds it into the rule base, the Event Processing Rule Manager GUI will invoke the RDR Rule Builder to access the Event Processing Rule Base as well as the Event Pattern Definition Table. In regard to the event pattern type specified in each rule, if it is an existing one, the system will simply associate the existing event pattern type to the new rule; if it is a new one, an IT expert will assist to implement its detection using underlying EPSs and provide links to the implemented applicable EPS service; the links will be saved in the Event Pattern Definition Table. If the rule is defined without a problem, the rule will be added to the rule base according to the right schema.

In order to ensure that previous processing logic can be repeated by data analysts even after rule addition, the platform needs to keep the set of inference actions at any stage of Rule Addition use case. Therefore, an additional feature is included: before adding any new rule, the old inference actions are copied and stored along with the rule. If the event data analyst later repeats whatever they have done using old rule sets at any point of the rule base evolution, they are able to select previous rule sets by simply selecting the right rule size. For instance, if the event data analyst selects to run the rule set that has a size of 5, the system will use the set of inference actions saved for the rule set of that particular size (i.e. the rule set when the size was 5). The algorithm to add a new rule is shown in Figure 4.5.

```
BEGIN ADD_RULE
  /** new_rule_id: the ID of the new rule to be added
   ** trigger_rule_id: the ID of the existing rule that is related to the inspected
      incorrect action
   ** trigger_inf: the predicate to denote whether the true inf. action
      or the false inf. action triggered the addition of the new rule
   ** rule_size: the current size of the rule set
  */
  new_rule_id = rule_size + 1;
  Add inf. action for rule_size (true) & inf. action for rule_size (false) as a column
    in the rule base;
  Copy and store existing inf. action (true) in inf. action for rule_size (true);
  Copy and store existing inf. action (false) in inf. action for rule_size (false);
  Select the event pattern type, get pattern_id from the database;
  Rules[new_rule_id].pattern_type = pattern_id;
  Define Rules[new_rule_id].case_action;
  Save the cornerstone case (JSON) in Rules[new_rule_id].cornerstone_case;
  IF trigger_rule_id < rule_size
    IF trigger_inf == 'true' THEN
      Rules[new_rule_id].inf_action_true
          = Rules[trigger_rule_id].inf_action_true;
      Rules[new_rule_id].inf_action_false
          = Rules[trigger_rule_id].inf_action_true;
      Rules[trigger_rule_id].inf_action_true = new_rule_id;
    ELSE IF trigger_inf == 'false' THEN
      Rules[new_rule_id].inf_action_true
          = Rules[trigger_rule_id].inf_action_false;
      Rules[new_rule_id].inf_action_false
          = Rules[trigger_rule_id].inf_action_false;
      Rules[trigger_rule_id].inf_action_false = new_rule_id;
    END IF
  ELSE
    Rules[new_rule_id].inf_action_true = -1;
    Rules[new_rule_id].inf_action_false = -1;
    IF trigger_inf == 'true' THEN
      Rules[trigger_rule_id].inf_action_true = new_rule_id;
    ELSE IF trigger_inf == 'false' THEN
      Rules[trigger_rule_id].inf_action_false = new_rule_id;
    END IF
  END IF
END
```

Figure 4.5 The algorithm of rule addition.

4.2.3 RDR Engine Sub-Component

The RDR Engine is used in the Rule Execution use case. The Event Processing Rule Processor GUI directly interacts with the RDR Engine in this process. Specifically, when the event data analyst initiates processing, the Event Processing Rule Processor GUI will call the RDR Engine to access the Event Processing Rule Base as well as the Event Pattern Definition Table and execute all the rules one after another according to the inference logic. For each rule, the corresponding action will be saved into the list of actions for the event data analyst to inspect after rule execution.

4.3 The EPDaaS Component

The EPDaaS component exposes a service interface that has the ability to invoke any underlying EPS (using the corresponding EPL) to detect event pattern occurrences. The reason why the EPDaaS component exposes a service interface rather than using an EPS directly for the event pattern detection is to allow multiple EPSs that diverge in terms of expressiveness to be integrated.

The way EPDaaS works is that when invoked with an event pattern type and a reference to an event dataset, it will perform the following tasks:

- Accessing the Event Pattern Definition Table to find the specified event pattern type.
- Selection of one suitable underlying EPS. If there is merely one EPS, it will naturally be the selected EPS. If there are multiple EPSs available, the selection can be based on various algorithms. An example of such algorithms can be:
  - First code available: The first EPS that has a proper code to detect the specified event pattern type is selected, regardless of the priority of the EPSs.
- Invoking the selected EPS for the detection of the event pattern type.
- Storing the results of running the selected EPS. The results will be the occurrences of the specified event pattern type in this event data set, or abstractions or aggregations of these occurrences. They are stored in the Event Pattern Occurrences Stack repository.

In principle the RDR component sends a request to detect an event pattern type specified in a particular rule, and the EPDaaS responds with event pattern occurrences. Rather than sending a request to the EPDaaS for each event and each rule, all event pattern occurrences of a particular event pattern type will be detected and returned in one invocation, with all occurrences saved in the event pattern occurrences stack repository. Then for each event and each rule, the RDR

Engine will process each occurrence in the repository in turn to check the "condition" and assert "action" as specified in the rule. This way, the efficiency of the platform can be guaranteed. The RDR Engine executes the rule base and generates a list of actions on the original event dataset, which will be inspected by the event data analyst.

Figure 4.6 shows the service interface of EPDaaS in Web Application Description Language (WADL) [124]. Specifically, for each single event pattern being detected, the RDR engine sends a request to the EPDaaS with 2 inputs and 1 output:

- Input (1): the key of the event pattern type to be detected, referred to as "Event Pattern ID" in Figure 4.6, which is used as an index in the Event Pattern Definition Table.
- Input (2): a reference to the event dataset to be processed (referred to as "Event Data Set" in Figure 4.6).
- Output: a stack of corresponding event pattern occurrences (referred to as "Detected Occurrences" in Figure 4.6).

```
<method name="GET" id="DetectPatternOccurrences">
 <request>
  <param name="EventPatternID" style="query" type="xsd:string"
required="true"/>
  <param name="EventDataSet" style="query" type="xsd:string"
required="true"/>
  </param>
 </request>
 <response>
  <representation mediaType="text/json"
   element="DetectedOccurrences"/>
 </response>
</method>
```

Figure 4.6 The WADL specification of the EPDaaS interface.

4.4 Underlying EPS

At least one EPS can be invoked by EPDaaS. For each underlying EPS, the IT expert writes the EPL code to detect occurrences of each existing event pattern type, deploys the code as a service and exposes the service via a link. At least one EPS link should exist in the database.

Multiple attributes can co-exist for all available EPSs. Due to the expressiveness issues in each EPS, some event pattern types cannot be expressed in particular EPSs.

4.5 Data Layer

4.5.1 Event Pattern Definition Table

As discussed in Section 2.3, there are many EPSs available on the market but unfortunately none are expressive enough to capture all event pattern types. Thus, it is always hard for data analysts to select the most appropriate one for their event data analysis. This is the motivation for the Event Pattern Definition Table, which stores event pattern types and links to each of the applicable EPSs invoked by the EPDaaS component. This database table is accessed in both the Rule Addition and the Rule Execution use cases (see Section 4.7). Table 4.1 displays an example of three rows in the Event Pattern Definition Table. In this table, at least the following attributes should exist:

- ID: the unique identification number for a particular event pattern type.
- Name: the name of the particular event pattern type.
- Description: the description of the event pattern type. This information could be used for IT experts to implement this event pattern type using the underlying EPS(s).
- EPS 1 Link: the link to the API of one of the underlying EPS(s) that detects occurrences of the particular event pattern type. The link could contain some variables. In the example, each link has a <file> variable, which denotes a reference ID of the data file to be processed.

In this example, there is only one underlying EPS, so only "EPS 1 Link" is present. If multiple EPSs are integrated, there will be more attributes like "EPS 2 Link" in the Event Pattern Definition Table. When a particular EPS is not capable of expressing a particular event pattern type, the link attribute for this EPS can be left empty.

Table 4.1 An example of the Event Pattern Definition Table.

ID	Name	Description	EPS 1 link
1	Dividend event	An event is a "Dividend" event.	https://unsw.eventswarm.com/data_files/<file>/eventswarm?rule=Dividend%20e

			vent
2	Duplicate Dividends	Two events with Type "Dividend" has the same timestamp, the same "Div Amt." and the same "Div Ex Date".	https://unsw.eventswarm.com/data_files/<file>/eventswarm?rule=Dividend%20dividend
3	Missing an EOD event on DED	No "End Of Day" event exists with "Div Ex Date" of a "Dividend" event as the timestamp.	https://unsw.eventswarm.com/data_files/<file>/eventswarm?rule=Missing%20EOD%20on%20DED

4.5.2 Event Processing Rule Base

The Event Processing Rule Base is a relational database that stores event processing rules defined via the Event Processing Rule Manager. It is also accessed in both the Rule Addition and the Rule Execution use cases. Table 4.2 displays an example of an event processing rule base. In each table of the rule base, there should be at least the following attributes:

- Rule ID: a unique number that identifies the particular rule.
- Event pattern ID: the unique number that identifies the event pattern type to be detected in the particular rule. This ID is linked to the event pattern ID in the Event Pattern Definition Table.
- Action: the action to be taken on the data if occurrences of the particular event pattern type are detected.
- Inference action (true): the ID of the rule to be directed to if occurrences of the particular event pattern type are detected.
- Inference action (false): the ID of the rule to be directed to if occurrences of the particular event pattern type are *not* detected.
- Cornerstone case: the case that prompted the addition of a rule. It is stored along with the rule and is used in comparison to new cases by the data analysts.

Inference action (true) and inference action (false) together form the mechanism to route the event processing logic.

Table 4.2 An example of an Event Processing Rule Base.

Rule No.	Event pattern ID	Action	Inf. action (true)	Inf. action (false)	Cornerstone Case

1	1	Action1	2	3	Event a
2	2	Action2	exit	3	Event b
3	3	Action3	exit	exit	Event c

4.5.3 Event Pattern Occurrences Stack Repository

The Event Pattern Occurrences Stack repository is accessed only in the Rule Execution use case. It is used to temporarily store detected occurrences of each event pattern type associated with rules. The purpose of this repository is to eliminate the repetition of detecting the same event pattern type. Specifically, during rule execution, the rule set is executed on each event in the dataset; thus each rule in the rule set may be executed multiple times and the event pattern type associated with each rule may be detected multiple times by the EPDaaS component, which is inefficient. With the Event Pattern Occurrences Stack repository, for each single rule and its associating event pattern type, only one invocation of EPDaaS is needed and all occurrences of the particular event pattern type are stored in the repository for use throughout the Rule Execution use case. Once a particular event pattern type is detected, the same event pattern type will never be sent to EPDaaS to detect again. When processing other events in the dataset, if the same rule/event pattern type is called, the RDR Engine will simply process the occurrences already detected before. More details on how this repository works are provided in Section 4.7.2.

This repository can be a relational database or simply a file folder, i.e. anything that can store event pattern occurrences detected by EPDaaS. As there is no standard for representing event pattern occurrences, a new event data modelling framework will be introduced in Chapter 5 to address this issue.

4.5.4 List of Actions Repository

A list of actions is the final result generated by the system after processing all the rules. The list of actions can be stored in a relational database or simply in files. The List of Action repository stores each action asserted by each rule, i.e. what to do with the original event dataset. The list of actions should be accessible throughout the Rule Execution use case and available to the data analysts via the Event Processing Rule Processor GUI so that they can inspect the result after rule execution.

An important feature is that a track of decision making, i.e. how this conclusion (action) is achieved, is stored along with each asserted action in the list. This, together with the cornerstone case saved along each rule, will make it easier for data analysts to inspect the result and find anomalies.

4.6 User Layer

There are two parts in the user layer. The Event Processing Rule Manager GUI provides two main functions for the event data analyst to manage the event processing rules, namely defining event pattern types and defining rules. The Event Processing Rule Processor GUI allows the event data analyst to commence processing a defined rule set in the Event Processing Rule Base, and directs the event data analyst to inspect the result, i.e. a List of Actions generated by RDR, after Rule Execution. Figure 4.7 shows that after Rule Execution the event data analyst has to inspect the result, and if an incorrect action is found due to a new cornerstone case, the event data analyst can add a new rule, where the cornerstone case is stored.

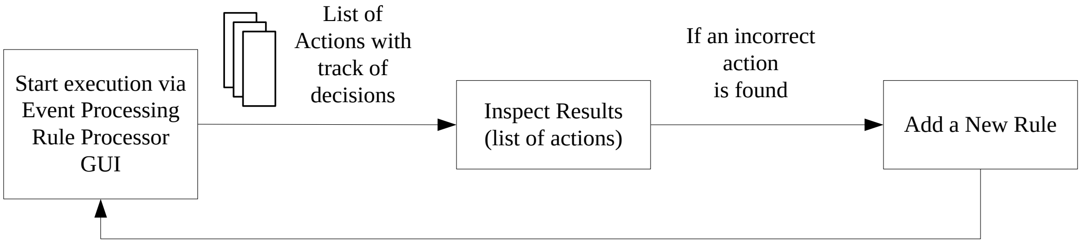

Start execution via Event Processing Rule Processor GUI
List of Actions with track of decisions
Inspect Results (list of actions)
If an incorrect action is found
Add a New Rule

Figure 4.7 Rule set evolution.

4.7 Sequence Diagrams

4.7.1 Rule Addition

Figure 4.8 shows a sequence diagram for the Rule Addition use case. To define a new rule, the event data analyst needs to specify the following information through the Event Processing Rule Manager GUI:

- Rule set: Which rule set (which table of the Event Processing Rule Database) will this new rule be added into?
- Rule inference information: This includes the ID of the rule and the inference action in the rule that has resulted in the incorrect action.
- Cornerstone case: The case that has fired the addition of this new rule.
- Event pattern type: An event pattern type is selected from the pre-defined event pattern types in the Event Pattern Definition Table. Note that every time a new event pattern

type is defined by an IT expert, it is saved so that all pre-defined event pattern types can be selected when building new rules.

- Action: The action to be taken on the data set if an occurrence of the selected event pattern type is found.

All these parameters will be passed on to the RDR Builder in the business layer. It will then associate the selected event pattern type with the new rule, and add it into the selected table (Rule set) of the Event Processing Rule Base. The algorithm of the RDR Builder addition of a new rule has been illustrated in Section 4.2.2.

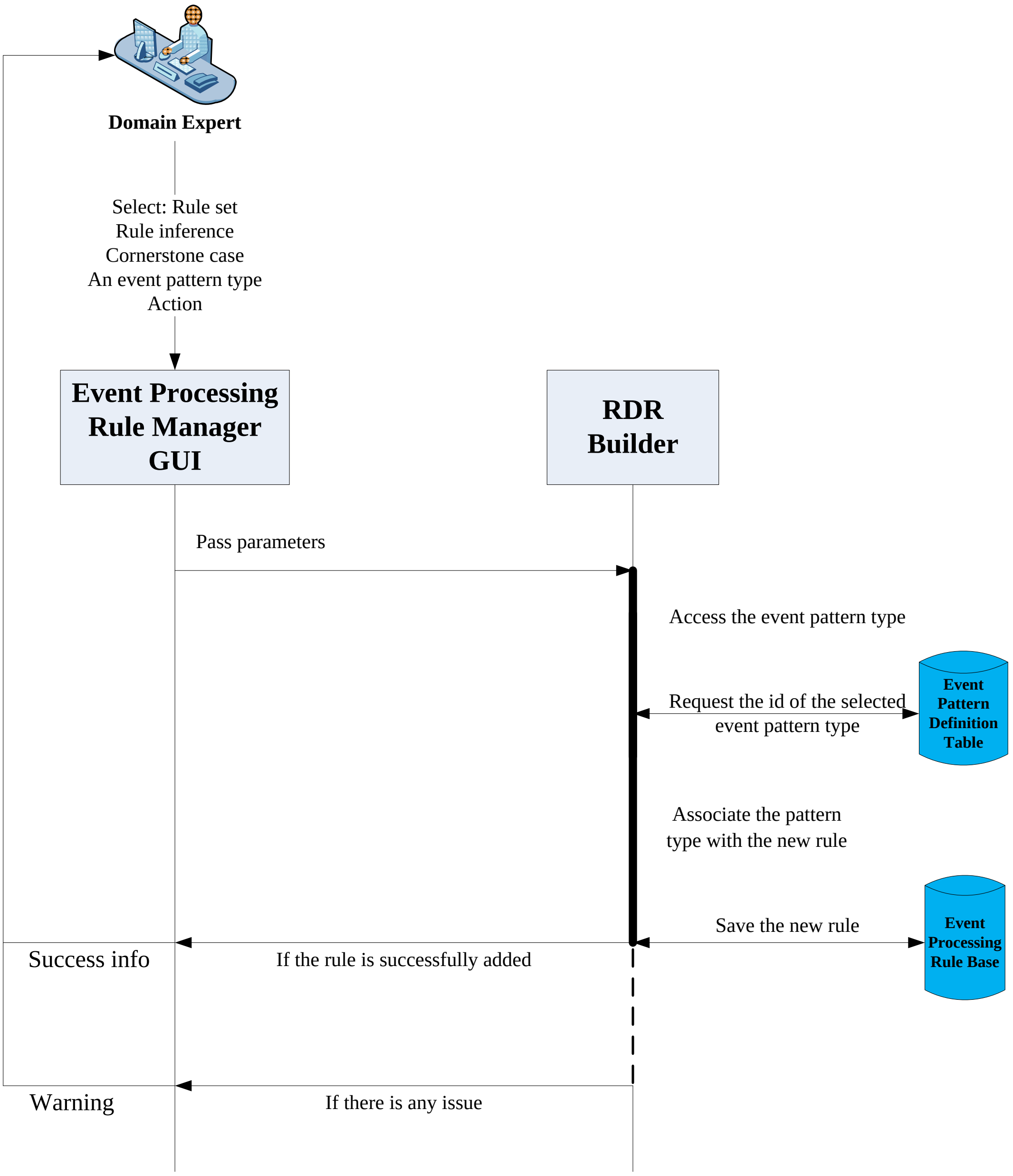

Domain Expert
Select: Rule set
Rule inference
Cornerstone case
An event pattern type
Action
Event Processing Rule Manager GUI
RDR Builder
Pass parameters
Access the event pattern type
Request the id of the selected event pattern type
Event Pattern Definition Table
Associate the pattern type with the new rule
Save the new rule
Event Processing Rule Base
Success info
If the rule is successfully added
Warning
If there is any issue

Figure 4.8 Rule Addition sequence diagram.

4.7.2 Rule Execution

Figure 4.9 shows a sequence diagram for the Rule Execution use case. First, the event data analyst invokes the Event Processing Rule Processor GUI. The rule set is then applied on each event in the dataset. For each rule being processed, the RDR Engine either sends a request (a service call) to the EPDaaS component if the detection of event pattern occurrences for the event pattern type specified in the rule is not yet done by the EPDaaS, or otherwise processes the existing Event Pattern Occurrences Stack repository that stores occurrences detected before.

Whenever the EPDaaS component receives a request, it invokes one of the underlying EPS according to the information stored in the Event Pattern Definition Table. The selected EPS then returns a stack of corresponding event pattern occurrences and stores them in the Event Pattern Occurrences Stack repository. For each iteration, the RDR Engine processes the stack, asserts the action according to the rule and goes to the next rule according to the "inference action" defined in the rule. After processing all rules on all events, a list of actions and a track of all "decisions" made during execution will be generated for the event data analyst's inspection.

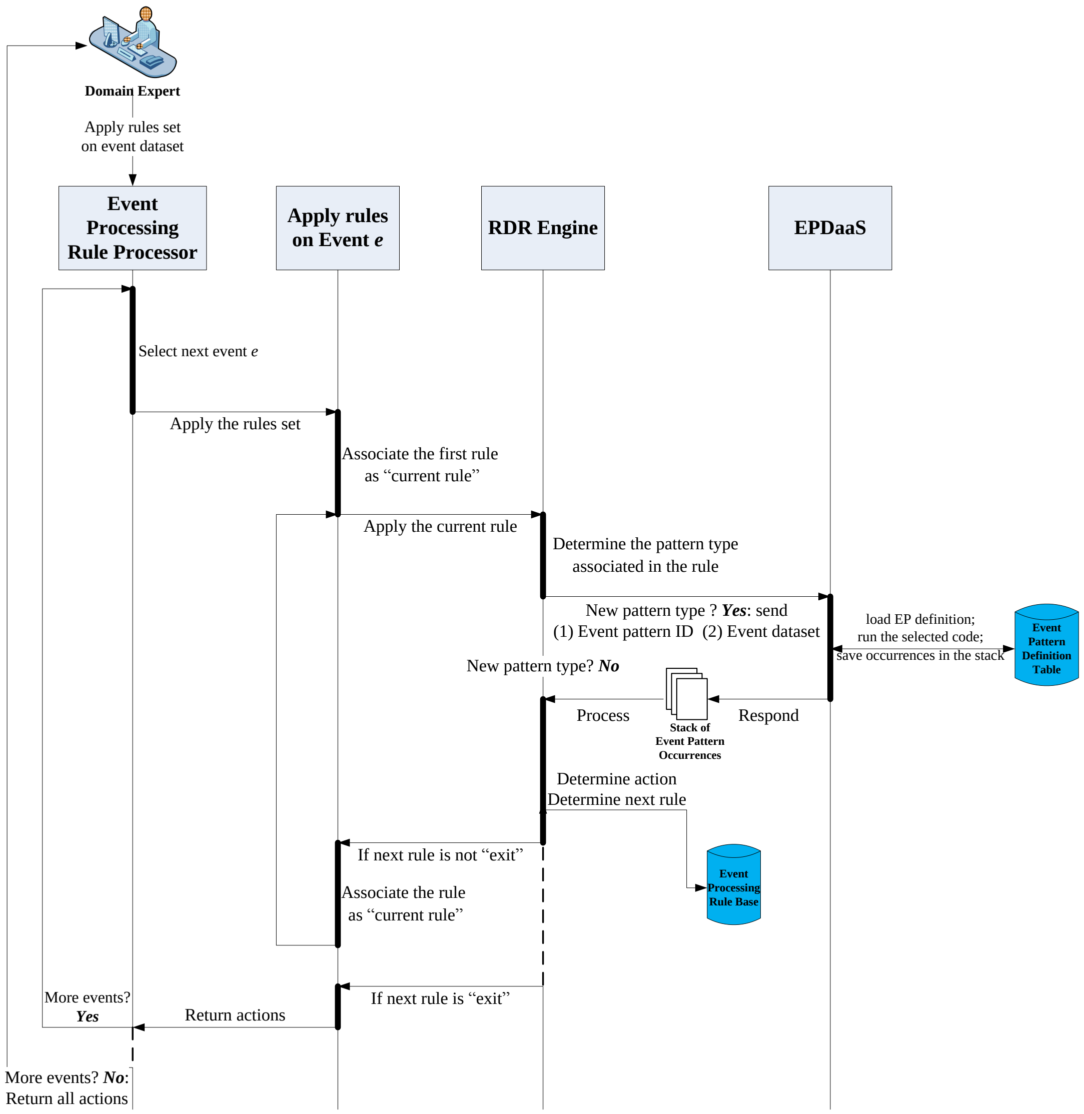


Figure 4.9 Rule Execution sequence diagram.

4.8 Conclusion

In this chapter, we introduced the proposed architecture, called event processing RDR (EP-RDR), for building and conducting user-driven event data analysis. Its two main components are a Ripple-Down Rule (RDR) system and an event pattern detection as a service (EPDaaS) which can interface with any event processing system (EPS). The EP-RDR architecture is designed to enable rule management by data analysts from the perspective of managing the whole rule base, and to leverage the functionality of existing Event Processing Systems.

Compared with other event processing systems, the advantages of EP-RDR include:

- Thanks to the RDR Component, the architecture allows for incremental, error-triggered rule management, i.e. adding a new rule when an incorrect action is inspected without corrupting the existing rule base.
- The EPDaaS component provides a service interface that can invoke multiple EPSs. For the execution of each event processing rule, the choice of the EPS depends on the capability of expressing particular event pattern types. Thus, data analysts do not have to select the EPS by themselves.
- For different types of event data analysis, data analysts can simply modify the rule set.
- IT experts are not involved in rule management but solely in managing the links to EPSs in the Event Pattern Definition Table.

However, in the interaction between the two main components - the RDR component and the EPDaaS component, occurrences of event pattern types detected by EPDaaS need to be stored in an Event Pattern Occurrences Stack repository so that the RDR Engine can further process the occurrences and assert the corresponding actions. Due to a lack of standard format for representing event pattern occurrences, it is still difficult to integrate any EPS easily into the system. This issue will be addressed by proposing a new event data modelling framework that can accurately represent event data and event pattern occurrences and can be used by both the RDR component and any underlying EPS behind EPDaaS. We will illustrate the new event data modelling framework in the following chapter.

Chapter 5 Proposed Event Data Modelling Framework

This chapter illustrates in detail the proposed Event Data Modelling Framework (EDMF) that facilitates event data modelling and thus enables the exchange of data between components in the EP-RDR architecture described in Chapter 4. First, the chapter illustrates the motivation of this work and the approach adopted. Second, some assumptions that form the basis of the proposed event data modelling framework are listed in Section 5.2. The chapter then presents the meta-model and the operational guidelines in Sections 5.3 and 5.4 respectively. Finally, we apply the framework to build an event data model on a real-life event data analysis scenario in Section 5.5.

5.1 Motivation and Approach

The Event Data Modelling Framework (EDMF) proposed aims to facilitate event data modelling for event data analysis systems, particularly for the EP-RDR architecture described in Chapter 4. The two components of the EP-RDR architecture both require event data handling: the RDR Rule Builder should allow the definition of event pattern types in the rules (Rule Addition use case), and the RDR Engine should be able to recognise particular event pattern types to be detected in the rule conditions (Rule Execution use case). As for the EPDaaS component, the service interface should know what event pattern type has to be detected and which EPS is suitable for detecting occurrences for this particular event pattern type.

There have been many attempts at designing data models in diverse domains, for example, [125, 126], but there is no widely used standard for the representation of event data and event pattern occurrences, and each single data model available exclusively focuses on data from a particular domain or data source. The lack of standard in the representation formats for event data and event patterns is a barrier in the exchange of data between components, e.g. the two components of the EP-RDR architecture, as one of the main goals is to allow the reuse of existing EPSs regardless of which data format they use.

Instead of a fixed data model, we propose an Event Data Modelling Framework (EDMF), which allows data model builders to construct an event data model more easily and in a consistent manner.

EDMF consists of an Event Data Meta-Model and its associated Operational Guidelines as shown in Figure 5.1. The Event Data Meta-Model is a high-level model that provides sufficient abstractions from various co-existing event data representation formats so that we can capture the semantics of different approaches in a consistent manner. The Operational Guidelines are a number of steps to be followed by the data model builder.

Data model builders can easily follow the operational guidelines and match the meta-model to build a target event data model for different event data sources or EPSs. Thus, extensions to various domains are facilitated. Any system applying EDMF will be independent from any data source or any EPS.

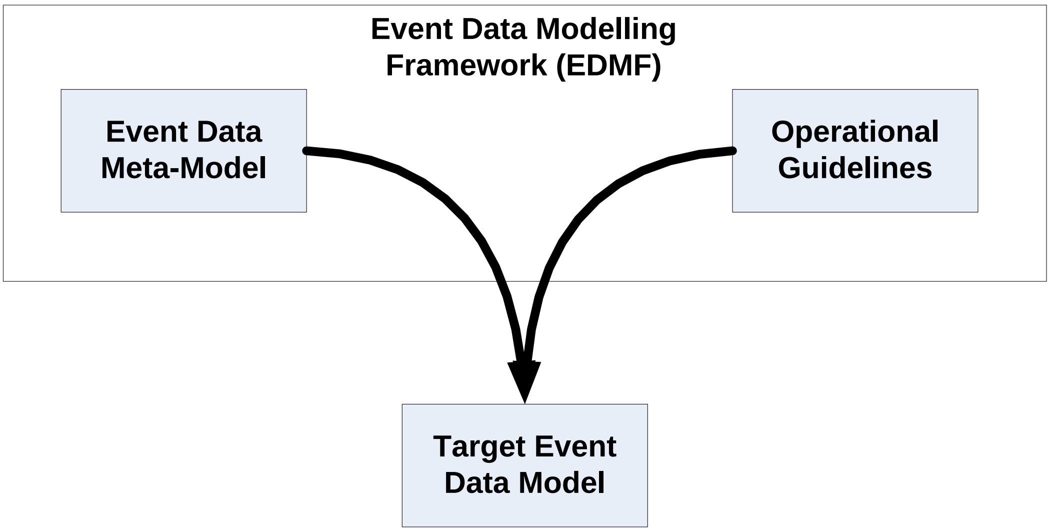


Figure 5.1 The proposed Event Data Modelling Framework (EDMF).

5.2 Basic Assumptions

The proposed EDMF will facilitate the development of a target data model for the representation of the following elements in event data analysis:

- Event: a record in an event dataset; it can be either a simple event or a complex event.
  - Simple event: an event retrieved from the dataset directly.
  - Complex event: an event constructed from an event pattern occurrence that matches a particular event pattern type.
- Event pattern type: a specification of an event pattern, expressing a set of event types and their relationships, used to specify matching conditions related to the values, abstractions or aggregations of the attributes of constituent events.
- Event pattern occurrence: a set of events linked via a number of relationships that match an event pattern type. In other words, event pattern occurrences are real-life cases of

event pattern types; event pattern types are the specification of event pattern occurrences.

Figure 5.2 shows the relationships between simple events, event pattern occurrences, and complex events. Specifically, a complex event is derived from an event pattern occurrence, which is a matched case of a certain event pattern type. An event pattern occurrence consists of a set of events (simple events or complex events or a mix of them) extracted from an event data repository (or an event data stream). Figure 5.3 is a real-life example in the finance domain (duplicate dividends) to demonstrate the relationships described in Figure 5.2. This example is a follow-up of the duplicate dividends cases firstly introduced in Section 2.1.4. A Duplicate Dividends event is derived from an occurrence of Duplicate Dividends pattern type. The Duplicate Dividends event pattern occurrence consists of two simple Duplicate events extracted from Sirca TRTH data.

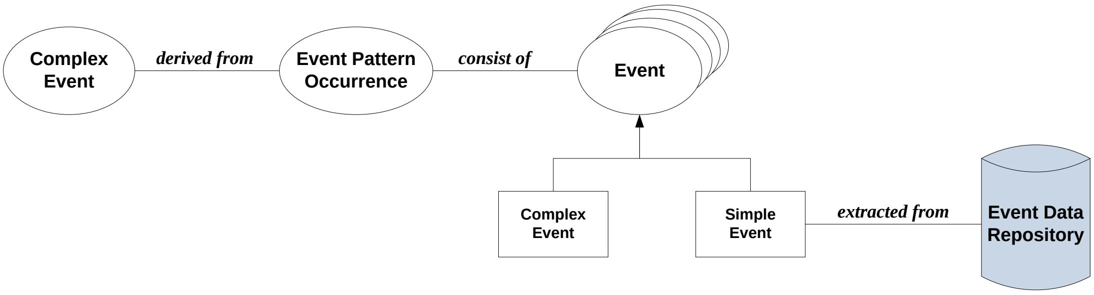


Figure 5.2 The relationship between events and event pattern occurrences.

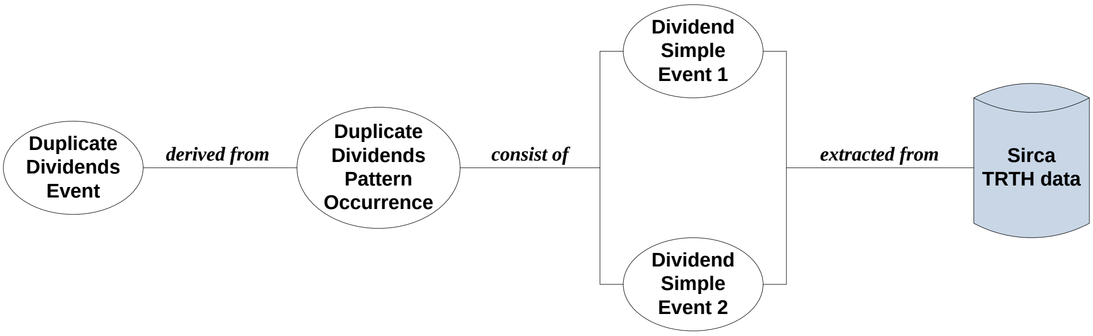


Figure 5.3 Example of simple events, event pattern occurrences and complex events.

In the following sections, we describe the two main components of EDMF: the Event Data Meta-Model will be explained in the next section and the Operational Guidelines will be explained in Section 5.4.

5.3 Event Data Meta-Model

Some Meta-Models that represents event pattern types have been proposed, e.g. [127]. However, it is simple and not powerful enough for this thesis. Thus, this section proposes a new meta-model, introduces the meta-model concepts for events, event patterns and related event data concepts, represented using UML, and thus each of these concepts is represented using a UML element [128]. Note that subsections will provide examples of specific instances of these concepts, e.g. specific events, event patterns and other related concepts.

5.3.1 Overview

Figure 5.4 shows the UML diagram that describes the proposed event pattern meta-model. We separate instances and types (and thus meta-model vs. model delineation). The latter would allow us to identify a minimal event pattern meta-model, which can be instantiated in terms of specific event data models. The meta-model elements and their corresponding model delineations will be illustrated in detail in the following sections. An XML schema for the meta-model is shown in Appendix A (see Table A.1).

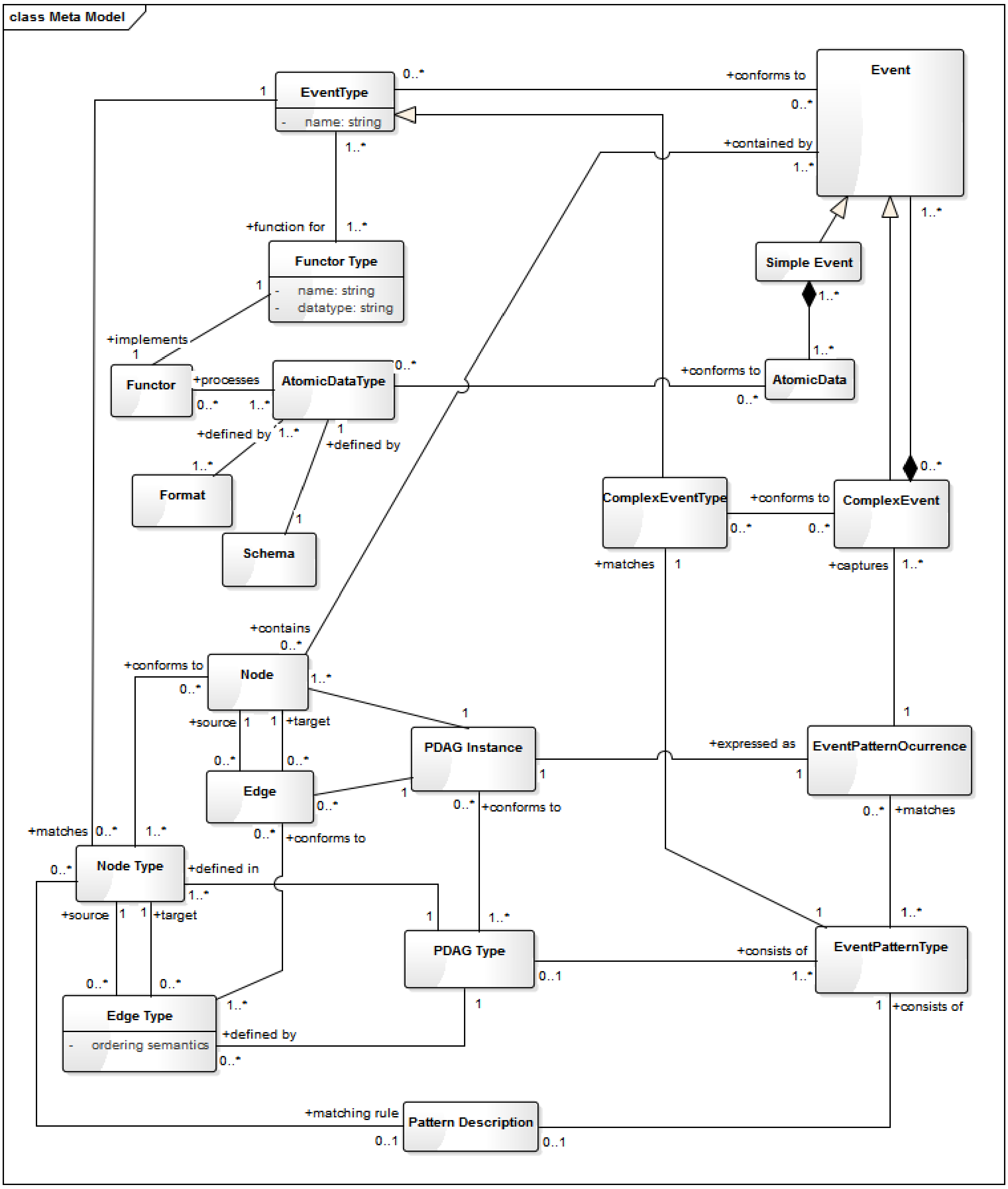


Figure 5.4 Event Data Meta-Model of EDMF.

5.3.2 Event Type

An event can be either a simple event or a complex event. Each event contains a list of values, including the timestamps and other attribute values. In the proposed system, any event from raw data of a particular data source is a simple event. A complex event is generated by an occurrence of a particular event pattern type, which consists of a number of events and the metadata indicating how the complex event is generated. *Event Type* is the model delineation of *Event*. *Event* is an instance of *Event Type*.

Event Type and *Event* are the two most fundamental elements in the meta-model, which should be considered first when matching the meta-model to build an event data model.

5.3.3 Atomic Data Type and Functor Type

Atomic Data is an element to allow more generic access to data and to make it possible to access any data types, e.g. Sirca TRTH data, Twitter data, etc. Each of the *Atomic Data Type* is defined by its *Format* and *Schema*. The *Format* is any format that represents and stores event data, e.g. XML or JSON. The *Schema* describes the structure of the data. Essentially, an *Atomic Data Type* is a generic element of an attribute in the actual data, regardless of the data source and the format of the original data. Through separating *Event Type* and *Atomic Data*, multiple wire formats can co-exist without changing types.

A *Functor* is a function of an event type (*Event Type*) returning a value (*AtomicData*), which is essentially data carried by an event instance (*Event*). Specific *Functor* instances can be defined to access attributes of an event from data sources. Take a Dividend *Event Type* for example (refer to Section 2.1.4 of Chapter 2 for the actual dividend data example). It may have *Functor Types* for accessing *Atomic Data Types* of dividend amount, dividend ex-date and payment status. Each of the *Atomic Data Types* is then defined by its own *Format* and *Schema*. Furthermore, specific *Functor* instances should then be defined to access these attributes, namely dividend amount (field "Div Amt" in Sirca data), dividend ex-date (field "Div Ex Date" in Sirca data) and payment status (field "Payment Status") from the data source, for instance, Sirca TRTH data repository. With the functor concept, extracting attribute values from an event can be more generic. It knows how to extract an attribute value from an event, but it does not care how the value is extracted or what type of data is contained inside the event.

Practically, two fundamental functors are:

- ***Source***: The return value of this functor is a string indicating the source of the event, e.g. "TR_MStream", "ASX_AETH", "Yahoo_Finance", etc. Different data sources use different clocks, which results in differences in the measurement of time. For example, there are two sources containing Australian trading data, namely Thomson Reuters Tick History (source = "TR_MStream"), and Australian Equity Tick History (source = "ASX_AETH"). In "TR_MStream", for a particular company named "BHP", a trading happened at 10-JAN-2014, 11:00:35.211 (local time), at the price of 36.37 and with the volume of 591; for the event in "ASX_AETH" recording the identical trade, however, it has a different time – 11:00:35.011 (local time). This makes it difficult to compare the time between events from various data sources. Thus, a database storing the clock skew for each source is required because the *source* determines the allowance for clock skew for an event. This thesis focuses on events from a single source, so we do not provide more discussion on the clock skew issue.

- ***Timestamp***: The return value of this functor is the recorded time of a particular event, which includes *start time* and *end time*. Both start time and end time are *timestamps*, each of which is represented by a Coordinated Universal Time (UTC) plus a Greenwich Mean Time (GMT) offset for the local time. In the context of Australia, the GMT offset of most of the examples in this thesis is +10 (winter time) or +11 (day light saving time). Start time and end time being identical means the event is an ***instantaneous event*** at a certain level of precision (e.g. millisecond); otherwise, it is an ***interval event***. A consideration of the precision of the time will make this more complicated; in this thesis, we keep the precision as millisecond (the default value). Also note that the start time and the end time of a complex event is derived from the timestamps of the individual events used to construct it.

5.3.4 P-DAG Type

P-DAG denotes "pattern directed acyclic graph", which is used to represent an *event pattern occurrence* of a certain *event pattern type*. Formally, a P-DAG instance is defined as:

- *P-DAG* = < *N, E* >
- *N* = < n_1, n_2, ... >
  - n_i is a set of event(s), n_i = < e_1, e_2, ... >
  - e_i is an event (i = 1, 2, …)
- E = <edge$_1$, edge$_2$, … >
  - *edge$_i$* is an edge between an ordered pair of nodes
  - *edge$_i$* is defined by the ordering semantics (*source*, *target* and the *ordering option*)
  - *source* and *target* are two nodes in the P-DAG instance, which specifies the order of two nodes
  - *ordering options* specifies additional rules of the ordering, e.g. the start time of the source node must be earlier than the start time of target node

To sum up, a *node* depicts a constituent *event* in an *event pattern occurrence*; an *edge* represents the time dependencies between *nodes* (constituent events).

5.3.5 Event Pattern Type and Event Pattern Occurrence

Basically, an event pattern type consists of:

P-DAG + Pattern Description

Pattern description contains additional information to be passed on to the event pattern implementer to build the event pattern type. The information includes but is not limited to:

- **Node referencing sequence**: the order of detecting constituent events, i.e. the first, the second, etc. node to be detected in the pattern detection phase.
- **Temporal Constraints:** information that constrains time, including the time between events in an event pattern type, and the time range to detect the occurrences (e.g. to get all events within a defined period of time).
- **Matching Conditions:** conditions that denote the relationships between the nodes of events in the pattern type in regards to values, abstractions or aggregations of the attributes of constituent events.

The pattern description can be written in a natural language (e.g. English), which will assist the event pattern implementer to implement the pattern type on an EPS accordingly. All this information will also be attached to event pattern occurrences detected.

Each *event pattern occurrence* involves a number of events, which can be *simple events* and/or *complex events*.

5.4 Operational Guidelines

The Operational Guidelines of EDMF help data model builders to apply the Event Data Meta-Model to their target data source and domain in order to derive a target event data model. The Operational Guidelines include the following steps (shown in Figure 5.5):

- **Step 0: Identify types of events in data and event pattern types to be used.** This is essentially a preparation stage to investigate as much as possible the data and the data analysis tasks to be conducted on the data, thereby accurately identifying event types and event pattern types to be used in the event data analysis tasks.

- **Step 1: Define event types.** After identifying event types from the data, we can define event type elements. Each event type element must implement/match the Event Type element in the Event Data Meta-Model.
- **Step 2: Define functor types that match attributes in data.** Functor types are defined to facilitate access to attributes in the data. Each event type has a unique set of attributes in the data; for each attribute, a functor type element should be defined. Each functor type element must implement/match the Functor element in the Event Data Meta-Model.
- **Step 3: Define event pattern types and corresponding complex event types.** After identifying event pattern types to be used in the analysis tasks, we can define event pattern type elements and corresponding complex event types accordingly. Each event pattern type in a specific model, e.g. the duplicate dividends event pattern type[2] element must be an instance of the Event Pattern Type element from the Event Data Meta-Model. Similarly, each complex event type in a specific model element must be an instance of the Complex Event Type element from the Event Data Meta-Model.
- **Step 4: Define atomic data types.** According to the need in implementation, the data model builder can decide to define atomic data type elements to capture different representations of the data (e.g. XML or JSON). Each atomic data type element must implement/match the Atomic Data Type element in the Event Data Meta-Model.

[2] See Section 5.5.2 for further detail of this event pattern type.

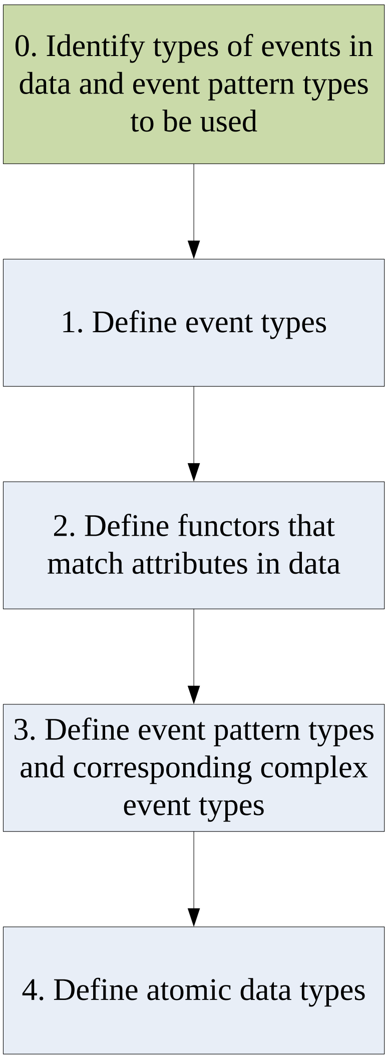

0. Identify types of events in data and event pattern types to be used
1. Define event types
2. Define functors that match attributes in data
3. Define event pattern types and corresponding complex event types
4. Define atomic data types

Figure 5.5 Operational Guidelines of EDMF.

5.5 Example: Building a Data Model for Sirca TRTH Data

Using the Operational Guidelines described in Section 5.4 and the Event Data Meta-Model described in Section 5.3, this section shows how to build a data model for Sirca TRTH data. This will be used in this thesis for validation purposes.

5.5.1 Identification of Event Types

In this example we need at least three types of events in Sirca TRTH data. Examples of all the three categories of events are listed in Table 5.1:

- End Of Day: events summarizing stock prices, and volumes along with some additional information at the end of each trading day. This type of data has a duration of 24 hours and is also called Daily Events.
- Dividend: a type of corporate action event which record actions issued by a corporate, resulting in the change of share ownership and cash flows from or to the shareholder. A dividend is 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 (the attribute "Div Ex Date").
- Capital Change: another type of corporate action event. One most frequent kind of capital change is called stock split, which is an increase or decrease in the number of outstanding shares, decreasing or increasing the per-stock price and earnings per stock.

Table 5.1 Examples of events in the financial domain in Australia.

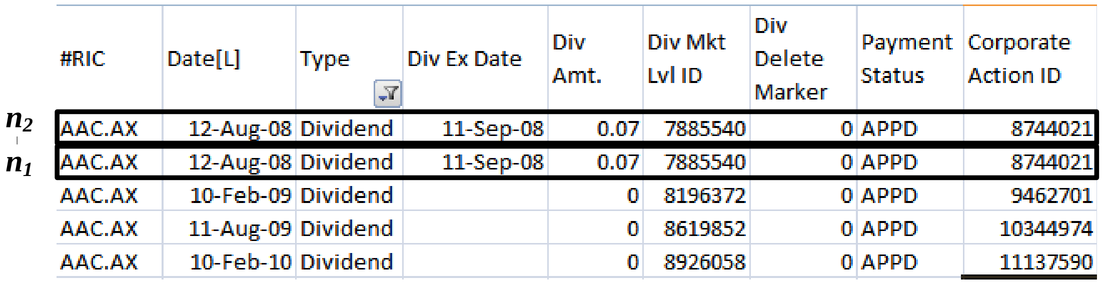
Data Type	Start Time (st) End Time (et)	Some Attribute Values				
End Of Day	st = 26 Oct 2008 14:00:00.000, +10 et = 27 Oct 2008 13:59:59.999, +10	RIC	Open	Last		Volume
		CBA.AX	40.66	40.02		4196278
Dividend	st = 30 Oct 2008 14:00:00.000, +10 et = 31 Oct 2008 13:59:59.999, +10	RIC	Div Ex Date	Div Amt.		Div Mkt Lvl ID
		CBA.AX	19 Jan 2009	2.472		802225
Earnings	st = 5 May 2008 14:00:00.000, +10 et = 6 May 2008 13:59:59.999, +10	RIC	EPS Amount	EPS Period End Date	EPS Scaling Factor	EPS Period Length
		AQA.AX	4990	31 Dec 2007	-4	6

5.5.2 Identification of Event Pattern Types

The event pattern types to be used in the analysis tasks include "Duplicate Dividends" and "Two 6-month earnings before End Of Day". These two event pattern types are illustrated below as examples:

- Duplicate Dividends.

P-DAG	$\boldsymbol{n_2} \longrightarrow \boldsymbol{n_1}$
Pattern Description	The duplicate dividends pattern type involves two nodes of simple events. The event types of both events in this pattern type are identical, i.e. "Dividend". For a real example (Figure 5.6), there are two dividend events on the date of 12-Aug-08. Both of them have the same "Div Ex Date", "Div Ex Date" and "Div Amt." values, which implies that these two events can be detected as an occurrence of the event pattern type. (Note: refer to Section 2.1.4 for the definition of a dividend.)

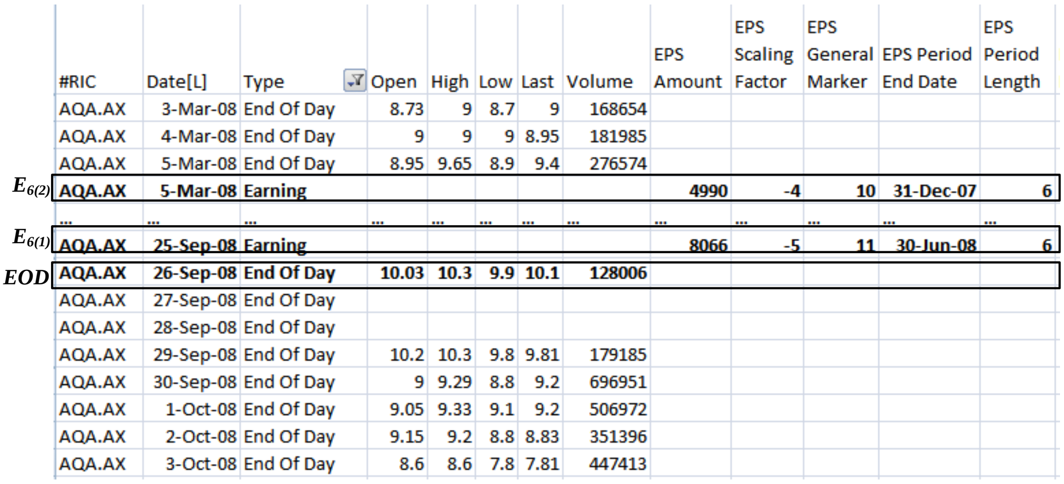


#RIC	Date[L]	Type	Div Ex Date	Div Amt.	Div Mkt Lvl ID	Div Delete Marker	Payment Status	Corporate Action ID
AAC.AX	12-Aug-08	Dividend	11-Sep-08	0.07	7885540	0	APPD	8744021
AAC.AX	12-Aug-08	Dividend	11-Sep-08	0.07	7885540	0	APPD	8744021
AAC.AX	10-Feb-09	Dividend		0	8196372	0	APPD	9462701
AAC.AX	11-Aug-09	Dividend		0	8619852	0	APPD	10344974
AAC.AX	10-Feb-10	Dividend		0	8926058	0	APPD	11137590

Figure 5.6 Pattern occurrence example of "duplicate dividends".

- Two 6-month earnings before End Of Day.

P-DAG	$\boldsymbol{E_{6(2)}} \longrightarrow \boldsymbol{E_{6(1)}} \longrightarrow \boldsymbol{EOD}$
Pattern Description	Two events $E_{6(1)}$ and $E_{6(2)}$ with type "Earning" ($E_{6(2)}$ before $E_{6(1)}$) happen before an event with type "End Of Day" (EOD) with: • The "EPS Period Length" of both $E_{6(1)}$ and $E_{6(2)}$ is 6; • $E_{6(2)}$.epsEndDate + $E_{6(2)}$.epsLength = $E_{6(1)}$.epsEndDate Find only one closest occurrence for each EOD if it exists. Figure 5.7 shows a real example of an occurrence of this event pattern type.



	#RIC	Date[L]	Type	Open	High	Low	Last	Volume	EPS Amount	EPS Scaling Factor	EPS General Marker	EPS Period End Date	EPS Period Length
	AQA.AX	3-Mar-08	End Of Day	8.73	9	8.7	9	168654					
	AQA.AX	4-Mar-08	End Of Day	9	9	9	8.95	181985					
	AQA.AX	5-Mar-08	End Of Day	8.95	9.65	8.9	9.4	276574					
$E_{6(2)}$	**AQA.AX**	**5-Mar-08**	**Earning**						**4990**	**-4**	**10**	**31-Dec-07**	**6**
	...	...	...	...	...	...	...	...	...	...	...	...	...
$E_{6(1)}$	**AQA.AX**	**25-Sep-08**	**Earning**						**8066**	**-5**	**11**	**30-Jun-08**	**6**
EOD	**AQA.AX**	**26-Sep-08**	**End Of Day**	**10.03**	**10.3**	**9.9**	**10.1**	**128006**					
	AQA.AX	27-Sep-08	End Of Day										
	AQA.AX	28-Sep-08	End Of Day										
	AQA.AX	29-Sep-08	End Of Day	10.2	10.3	9.8	9.81	179185					
	AQA.AX	30-Sep-08	End Of Day	9	9.29	8.8	9.2	696951					
	AQA.AX	1-Oct-08	End Of Day	9.05	9.33	9.1	9.2	506972					
	AQA.AX	2-Oct-08	End Of Day	9.15	9.2	8.8	8.83	351396					
	AQA.AX	3-Oct-08	End Of Day	8.6	8.6	7.8	7.81	447413					

Figure 5.7 Pattern occurrence example of "Two 6-month earnings before End Of Day".

The next section will focus on defining data model elements in accordance with Steps 1-4 in the Operational Guidelines illustrated in Section 5.4.

5.5.3 Defining Data Model Elements

After the identification, we define all the event types and event pattern types in the data model. Each type of event implements the Event Type in the meta-model, i.e. each type is an instance of Event Type in the meta-model. Each event pattern type implements the Event Pattern Type in the meta-model. Finally, we can achieve a new model for Sirca TRTH data shown in Figure 5.8 (Note a large number of event pattern types can be identified for Sirca TRTH data but only the duplicate dividends event pattern type is presented). The elements are marked with numbers that indicate the corresponding step in Figure 5.5 when they are built. The details of these steps are as follows:

Steps 1 & 2. According to the investigation on event types needed for Sirca TRTH data in Section 5.5.1, we can define event type elements:

- SircaTRTHEvent -> instance of -> MM:EventType with:
  - id -> instance of -> MM:FunctorType (datatype = string)
  - RIC -> instance of -> MM:FunctorType (datatype = string)
  - timestamp -> instance of -> MM:FunctorType (datatype = timestamp)
  - source -> instance of -> MM:FunctorType (datatype = string)
  - attribute(name) -> instance of -> MM:FunctorType(datatype = string)
- Dividend extends SircaTRTHEvent -> instance of -> MM:EventType with:

- dividend ID -> instance of -> MM:FunctorType (datatype = string)
- dividend amount -> instance of -> MM:FunctorType (datatype = currency)
- dividend ex-date -> instance of -> MM:FunctorType (datatype = date)
- payment status -> instance of -> MM:FunctorType (datatype = string)
- [… and so on]

Other event types can be modelled similarly.

Step 3. We now define event pattern type elements. The event pattern type just captures the need to define Sirca TRTH event pattern types without being too specific about detail:

- SircaTRTHEventPattern -> instance of -> MM:EventPatternType

Once we have the base types for event and event pattern, we can define a complex event:

- SircaTRTHComplexEvent extends SircaTRTHEvent -> instance of -> MM:ComplexEventType with:

  - events -> instance of -> MM:FunctorType (datatype = set(SircaTRTHEvent))
  - pattern -> instance of -> MM:FunctorType (datatype = SircaTRTHEventPattern)

The following elements capture a particular event pattern type (duplicate dividends) and a complex event type for this event pattern type:

- DuplicateDividendPattern extends SircaTRTHEventPattern -> instance of -> MM:EventPatternType with:

  - Pattern Description: ’2 dividend events with same date and stock id number’ -> instance of -> MM:Pattern Description
  - PDAG: instance of -> MM:PDAGType with:
    NodeTypes: first -> instance of -> MM:NodeType captures Dividend
    second -> instance of -> MM:NodeType captures Dividend
    EdgeTypes: from first to second -> instance of -> MM:EdgeType with:
    ordering weak_before

- DuplicateDividendEvent extends SircaTRTHComplexEvent -> instance of > MM:ComplexEventType: matches DuplicateDividendPattern

Step 4. Now we define AtomicDataTypes to capture different representations of SircaTRTHEvent, for example, XML and JSON:

- SircaTRTHXMLEvent-> instance of -> MM:AtomicDataType with:

- SircaTRTHXMLEventSchema -> instance of -> MM:Schema with:
  - id implements SircaTRTHEvent:id -> instance of -> MM:Functor
  - timestamp implements SircaTRTHEvent:timestamp -> instance of -> MM:Functor
  - source implements SircaTRTHEvent:source -> instance of -> MM:Functor
  - attribute(name) implements SircaTRTHEvent:attribute -> instance of -> MM:Functor
- SircaTRTHJSONEvent -> instance of -> MM:AtomicDataType with
- SircaTRTHJSONEventSchema -> instance of -> MM:Schema with:
  - id implements SircaTRTHEvent:id -> instance of -> MM:Functor
  - timestamp implements SircaTRTHEvent:timestamp -> instance of -> MM:Functor
  - [… and so on]

It can be seen that through separating type and atomic data, we have multiple wire formats without changing our types. Defining a complex event atomic data type is similar:

- SircaTRTHXMLComplexEvent extends SircaTRTHXMLEvent -> instance of -> MM:AtomicDataType with:
  - SircaTRTHXMLComplexEventSchema -> instance of -> MM:Schema
  - events implements SircaTRTHComplexEvent:events -> instance of -> MM:Functor
  - pattern implements SircaTRTHComplexEvent:pattern -> instance of -> MM:Functor

An example of using this data model to represent "Duplicate Dividends" event pattern occurrences is shown in Appendix A (see Table A.2).

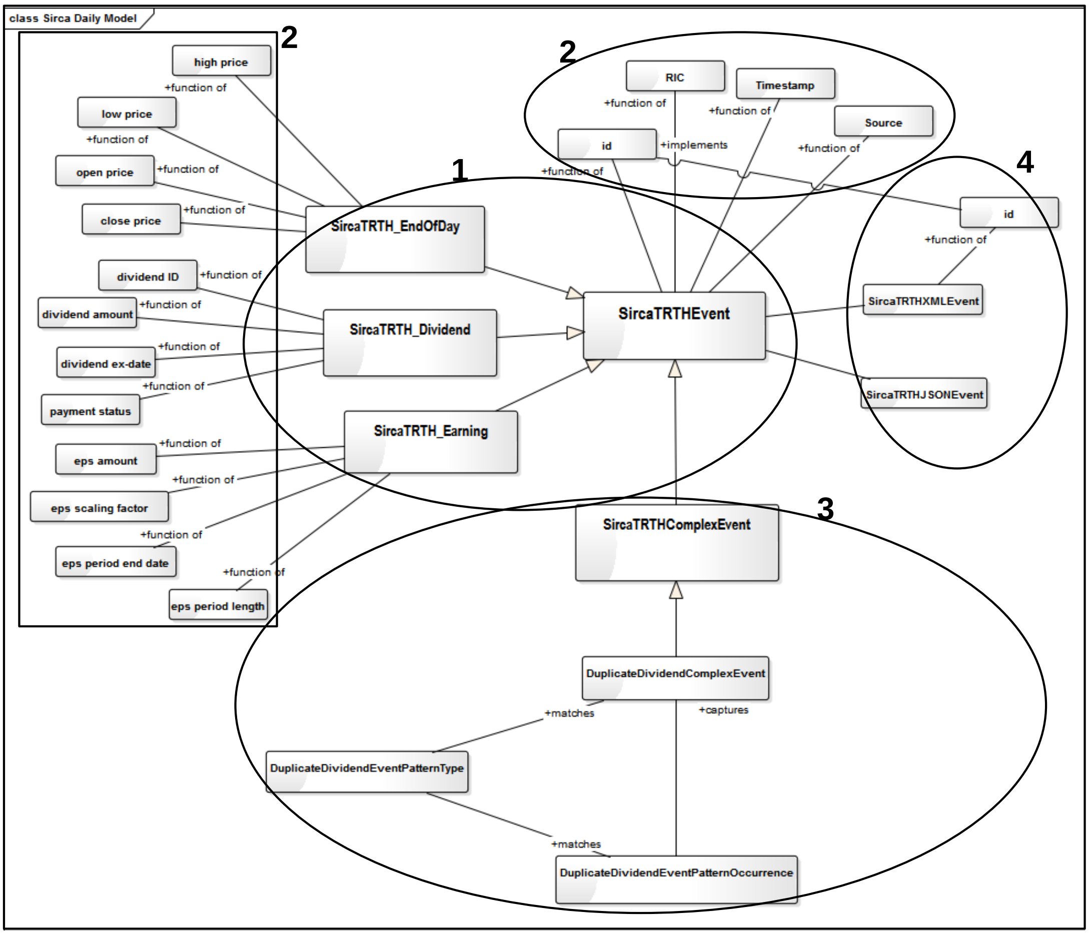


Figure 5.8 Data model for Sirca TRTH data (based on the meta-model).

5.6 Conclusion

In this chapter, we proposed a new Event Data Modelling Framework (EDMF) that facilitates event data modelling regardless of the domain, data source, the EPS to be used in the EP-RDR architecture described in Chapter 4. EDMF consists of an Event Data Meta-Model that abstracts common elements (such as events, event pattern occurrences), and Operational Guidelines to specify how to construct a target event data model.

EDMF brings benefits for both EP-RDR developers and data analysts. For developers, applying EDMF to the EP-RDR architecture, it will be easier to implement generic components that do not depend on how event pattern occurrences are represented. The data model provides the essential link between the two components of EP-RDR, i.e. RDR and EPDaaS, enabling the exchange of data in between. This framework can also be extended to other event data analysis software development. Components of the system may have different data models, but as long as these data models are all built based on EDMF, it is much easier to relate and match the separately developed models. For data analysts, data models constructed using EDMF provide a more informative output, and thus a more detailed tracking of event pattern detection is facilitated.

Chapter 6 Prototype Implementation

This chapter describes an implementation of the EP-RDR architecture and the event data modelling framework (EDMF) proposed in Chapters 4 and 5. The purpose of the prototype implementation is to validate the feasibility of the proposed system. This constitutes the first step in the evaluation. The prototype implementation is also used in the next chapter to conduct experiments to evaluate other aspects of the proposed system.

6.1 Overview of the Prototype Implementation

In this section, we describe how we have implemented a prototype of the proposed EP-RDR architecture and the data model using a number of technologies. The relationship between architectural components and technologies is illustrated in Figure 6.1.

In the business layer, the RDR component (including the RDR Rule Builder and the RDR Engine) and the EPDaaS component are both implemented in Java. The service interface of EPDaaS exposed to the RDR Engine is a RESTful API (Application Programming Interface). More details of the business layer will be provided in Section 6.2.

In the data layer, we have used PostgreSQL to implement the Event Pattern Definition Table and the Event Processing Rule Base. We have used a folder of files in JSON format to implement the Event Pattern Occurrences Stack repository, and a folder of files in CSV format to implement the List of Actions repository. More details of the data layer will be provided in Section 6.3.

In the user layer, the Event Processing Rule Manager GUI and the Event Processing Rule Processor GUI have been implemented using Java Swing. More details of the user layer will be provided in Section 6.4.

As the purpose of the EP-RDR architecture is to enable the use of an EPS back-end, we have selected a commercial system called EventSwarm. EventSwarm has proven effective and efficient in real-time analytics for data streams in the health domain [66]. It supports a range of predefined abstractions and pattern components implemented in Java programming language. EventSwarm can be invoked via two typical styles. The first style provides users with a GUI to conduct event pattern detection and inspect the detected occurrences. The second style enables EventSwarm to be invoked by other applications via a RESTful API, and this is the one used in this prototype implementation.

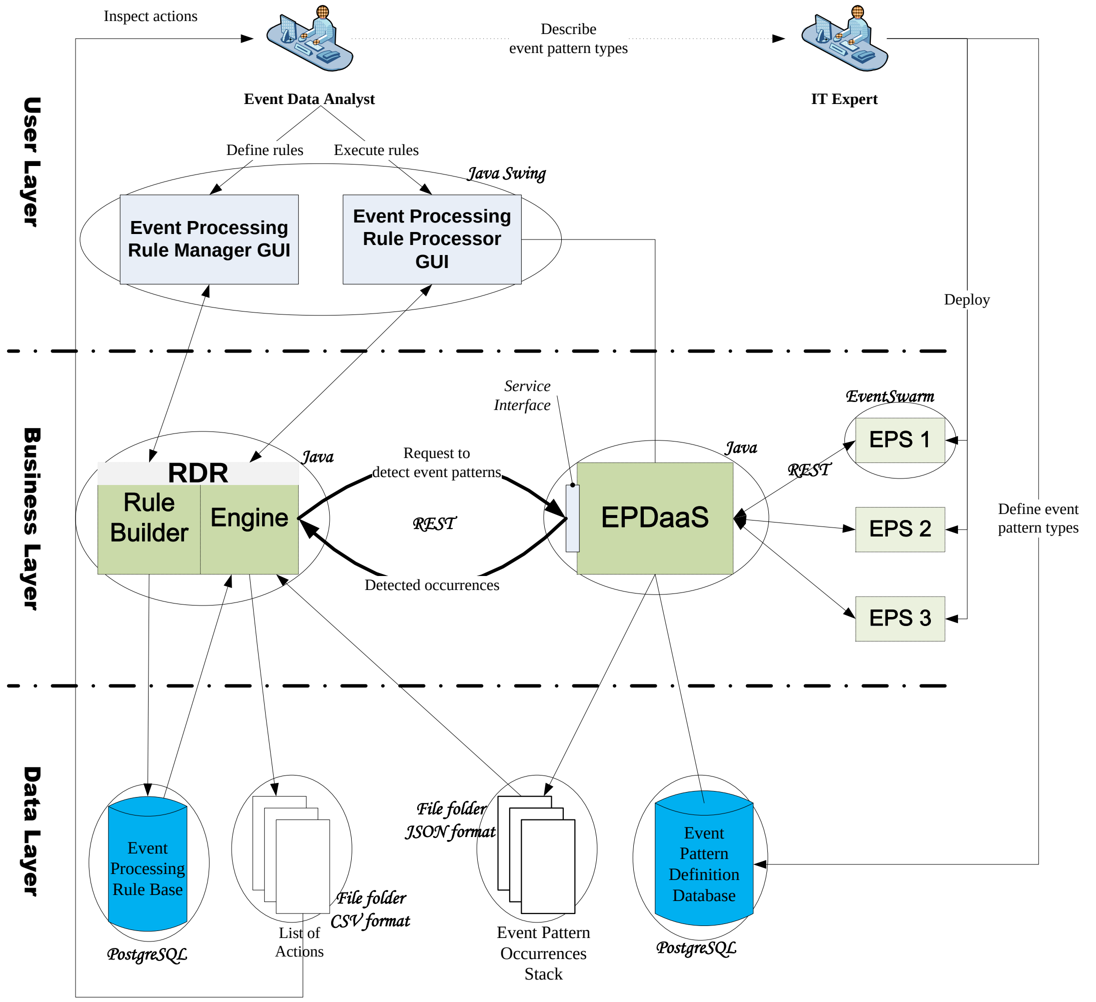


Figure 6.1 Technologies used in the prototype implementation.

Prior to implementing the components of the business layer, we have first designed the data model for implementation and validation purposes according to the Event Data Modelling Framework (EDMF) proposed in Chapter 5.

Following the steps of the Operational Guidelines described in Section 5.4, we identified the types of events and event pattern types to be used. A number of event pattern types related to these types of events are also identified. Then we sequentially defined a number of elements to match the Event Data Meta-Model described in Section 5.4, including all the event types (that implement the Event Type in the Event Data Meta-Model), functor types (that implement the Functor Type in the Event Data Meta-Model), event pattern types (that implement the Event Pattern Type in the Event Data Meta-Model), complex event types (that are instances of the Complex Event Type element from the Event Data Meta-Model), and atomic data types (that implement the Atomic Data Type element in the Event Data Meta-Model). Finally, we successfully constructed a data model, which is the one provided as an example in Section 5.5. The UML diagram describing the main part of this data model can be found in Figure 5.8. Note that there are a large number of event pattern types that can be identified for the data we use in the case study, but only the duplicate dividends pattern type is presented in the diagram. More details of all event pattern types will be provided in Chapter 7.

The details of implementing the business layer, the data layer and the user layer in the system are now provided in Sections 6.2, 6.3 and 6.4 respectively.

6.2 Business Layer Implementation

The business layer is at the core of the proposed architecture and consists of an RDR component and an EPDaaS component whose implementation is described next.

6.2.1 RDR Component

The RDR component consists of two sub-components, namely the RDR Rule Builder and the RDR Engine (see Section 4.2). They have been developed as two Java programs.

The role of the RDR Rule Builder (see Section 4.2.2) is to permit incremental definition of rules based on the presence of event patterns. Each event "situation" is represented as an event pattern type in a rule. The RDR Rule Builder is used in the Rule Addition use case, where the Event Processing Rule Manager GUI directly interacts with the RDR Rule Builder. Figure 4.8 shows a sequence diagram that describes this process. The role of the RDR Engine (see Section 4.2.3) is to execute a rule set on a particular data set. It is used in the Rule Execution use case. The Event Processing Rule Processor GUI directly interacts with the RDR Engine in this process. For each rule, the Rule Engine sends a RESTful service request to the EPDaaS component and waits for response. More details of this process have been shown as a sequence diagram in Figure 4.9 (see Chapter 4).

As discussed in Section 4.2.1, we have selected MCRDR as the RDR algorithm and linked-production rules as the inference technique in implementing the RDR component.

6.2.2 EPDaaS Component

As discussed in Section 4.3, the EPDaaS Component exposes a RESTful service interface that has the ability to invoke any underlying EPS (using the corresponding EPL) to detect event pattern occurrences. The way EPDaaS works is that when invoked with an event pattern type and a reference to an event dataset, it will access the Event Pattern Definition Table to find the specified event pattern type, select one underlying EPS (in this case, there is only one underlying EPS so virtually there is no need to do the selection), invoke the selected EPS for the detection of the event pattern type, and store the results of running the selected EPS.

The EPDaaS component is also implemented as a Java program. As mentioned earlier, we have only selected one underlying EPS to be invoked by EPDaaS - EventSwarm, because it has the following outstanding features [66]:

- It has the ability to filter on any computed abstraction that matches a single event, including statistical analysis;
- It is capable of using causal precedence in sequence patterns.
- It is advantageous in managing time and ordering issues of events due to event timestamps, buffering, flexible relationships between events, time skew allowance and causal ordering.
- It provides the ability to calculate statistics on sliding time windows and use the statistics in expressions.

An IT expert is still required to implement the rules in EventSwarm for detecting all the event pattern types needed. There is a total of thirteen event pattern types that have been implemented for evaluation purposes (details in Chapter 7). The way EventSwarm works in this prototype is shown in Figure 6.2.

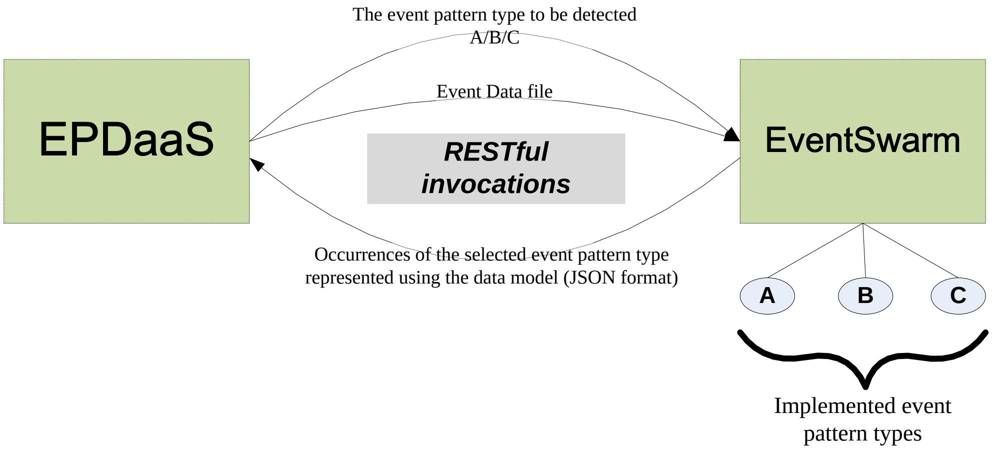

The event pattern type to be detected
A/B/C
Event Data file
EPDaaS
RESTful invocations
EventSwarm
Occurrences of the selected event pattern type represented using the data model (JSON format)
A
B
C
Implemented event pattern types

Figure 6.2 How EventSwarm works in the implementation.

6.3 Data Layer Implementation

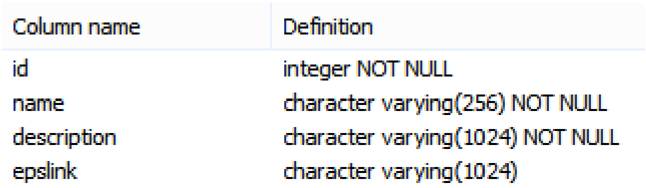
6.3.1 Event Pattern Definition Table

The Event Pattern Definition Table stores event pattern types and links to each the applicable EPS invoked by the EPDaaS component (see Section 4.5.1 for details). It is implemented using a PostgreSQL [32] database. The SQL statements for creating the Event Pattern Definition

Table are shown in Figure 6.3. The columns in the Event Pattern Definition Table are shown in Figure 6.4.

```
CREATE TABLE eventpatterndefinition
(
  id integer NOT NULL, -- The unique identification number for a particular event
pattern type.
  name character varying(256) NOT NULL, -- The name of the particular event pattern
type.
  description character varying(1024) NOT NULL, -- The description of the event pattern
type. This information could be used for IT experts to implement this event pattern type
using the underlying EPS(s).
  epslink character varying(1024), -- The link to the API for an EPS that detects
occurrences of the particular event pattern type.
  CONSTRAINT "ID" PRIMARY KEY (id)
)
WITH (
  OIDS=FALSE
);
ALTER TABLE eventpatterndefinition
  OWNER TO postgres;
COMMENT ON COLUMN eventpatterndefinition.id IS 'The unique identification number for a
particular event pattern type.';
COMMENT ON COLUMN eventpatterndefinition.name IS 'The name of the particular event
pattern type.';
COMMENT ON COLUMN eventpatterndefinition.description IS 'The description of the event
pattern type. This information could be used for IT experts to implement this event
pattern type using the underlying EPS(s).';
COMMENT ON COLUMN eventpatterndefinition.epslink IS 'The link to the API for an EPS that
detects occurrences of the particular event pattern type.';
```

Figure 6.3 SQL for creating the Event Pattern Definition Table.



Column name	Definition
id	integer NOT NULL
name	character varying(256) NOT NULL
description	character varying(1024) NOT NULL
epslink	character varying(1024)

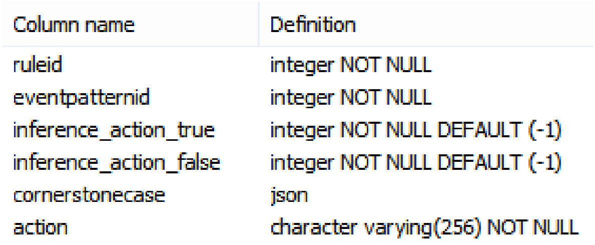
Figure 6.4 Definition of Event Pattern Definition Table columns in PostgreSQL.

6.3.2 Event Processing Rule Base

The Event Processing Rule Base is a relational database that stores event processing rules defined via the RDR Rule Builder (see Section 4.5.2 for details). It is also implemented using a PostgreSQL database. The SQL statements for creating a rule set of Event Processing Rule Base are shown in Figure 6.5. The columns in a rule set of Event Processing Rule Base are shown in Figure 6.6.

```
CREATE TABLE eventprocessingruleset
(
  ruleid integer NOT NULL, -- A unique number that identifies the particular rule.
  eventpatternid integer NOT NULL, -- The unique number that identifies the event
pattern type to be detected in the particular rule.
  inference_action_true integer NOT NULL DEFAULT (-1), -- The ID of the rule to be
directed to if occurrences of the particular event pattern type are detected.
  inference_action_false integer NOT NULL DEFAULT (-1), -- The ID of the rule to be
directed to if occurrences of the particular event pattern type are not detected.
  cornerstonecase json, -- The case that prompted the addition of a rule.
  action character varying(256) NOT NULL, -- The action to be taken on the data if
occurrences of the particular event pattern type are detected.
  CONSTRAINT "Rule ID" PRIMARY KEY (ruleid),
  CONSTRAINT "matching event pattern type ID" FOREIGN KEY (eventpatternid)
      REFERENCES eventpatterndefinition (id) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION
)
WITH (
  OIDS=FALSE
);
ALTER TABLE eventprocessingruleset
  OWNER TO postgres;
COMMENT ON COLUMN eventprocessingruleset.ruleid IS 'A unique number that identifies the
particular rule.';
COMMENT ON COLUMN eventprocessingruleset.eventpatternid IS 'The unique number that
identifies the event pattern type to be detected in the particular rule.';
COMMENT ON COLUMN eventprocessingruleset.inference_action_true IS 'The ID of the rule to
be directed to if occurrences of the particular event pattern type are detected.';
COMMENT ON COLUMN eventprocessingruleset.inference_action_false IS 'The ID of the rule
to be directed to if occurrences of the particular event pattern type are not
detected.';
COMMENT ON COLUMN eventprocessingruleset.cornerstonecase IS 'The case that prompted the
addition of a rule. ';
COMMENT ON COLUMN eventprocessingruleset.action IS 'The action to be taken on the data
if occurrences of the particular event pattern type are detected.';
```

Figure 6.5 SQL for creating a rule set of Event Processing Rule Base.

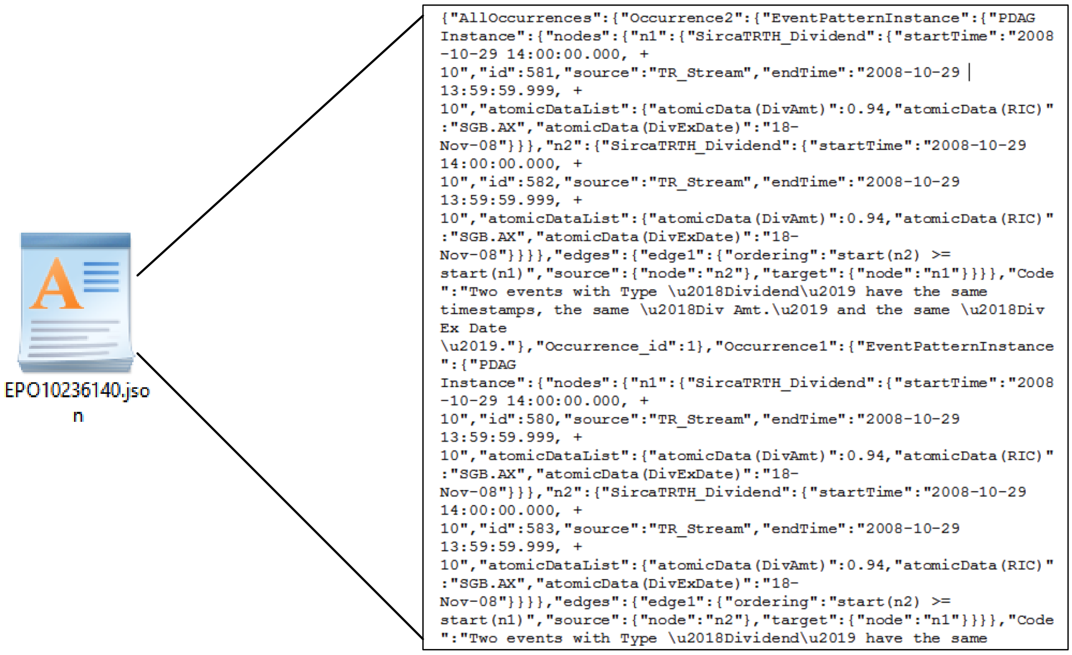


Column name	Definition
ruleid	integer NOT NULL
eventpatternid	integer NOT NULL
inference_action_true	integer NOT NULL DEFAULT (-1)
inference_action_false	integer NOT NULL DEFAULT (-1)
cornerstonecase	json
action	character varying(256) NOT NULL

Figure 6.6 Definition of a rule set of Event Processing Rule Base columns in PostgreSQL.

6.3.3 Event Pattern Occurrences Stack & List of Actions Repositories

The Event Pattern Occurrences Stack repository is used to temporarily store detected occurrences of each event pattern type associated with rules. The purpose of this repository is to eliminate the repetition of detecting the same event pattern type (see Section 4.5.3 for details). In this implementation we use a file folder that contains all event pattern occurrences detected by the EPS. For each request for detecting one particular event pattern type, a JSON file named "EPO######" containing event pattern occurrences detected by the EPS is generated and stored in this folder (see Figure 6.7). The structure of the JSON file is based on the data model built on the EDMF proposed in Chapter 7.



```
{"AllOccurrences":{"Occurrence2":{"EventPatternInstance":{"PDAG
Instance":{"nodes":{"n1":{"SircaTRTH_Dividend":{"startTime":"2008
-10-29 14:00:00.000, +
10","id":581,"source":"TR_Stream","endTime":"2008-10-29
13:59:59.999, +
10","atomicDataList":{"atomicData(DivAmt)":0.94,"atomicData(RIC)"
:"SGB.AX","atomicData(DivExDate)":"18-
Nov-08"}}},"n2":{"SircaTRTH_Dividend":{"startTime":"2008-10-29
14:00:00.000, +
10","id":582,"source":"TR_Stream","endTime":"2008-10-29
13:59:59.999, +
10","atomicDataList":{"atomicData(DivAmt)":0.94,"atomicData(RIC)"
:"SGB.AX","atomicData(DivExDate)":"18-
Nov-08"}}}},"edges":{"edge1":{"ordering":"start(n2) >=
start(n1)","source":{"node":"n2"},"target":{"node":"n1"}}}},"Code
":"Two events with Type \u2018Dividend\u2019 have the same
timestamps, the same \u2018Div Amt.\u2019 and the same \u2018Div
Ex Date
\u2019."},"Occurrence_id":1},"Occurrence1":{"EventPatternInstance
":{"PDAG
Instance":{"nodes":{"n1":{"SircaTRTH_Dividend":{"startTime":"2008
-10-29 14:00:00.000, +
10","id":580,"source":"TR_Stream","endTime":"2008-10-29
13:59:59.999, +
10","atomicDataList":{"atomicData(DivAmt)":0.94,"atomicData(RIC)"
:"SGB.AX","atomicData(DivExDate)":"18-
Nov-08"}}},"n2":{"SircaTRTH_Dividend":{"startTime":"2008-10-29
14:00:00.000, +
10","id":583,"source":"TR_Stream","endTime":"2008-10-29
13:59:59.999, +
10","atomicDataList":{"atomicData(DivAmt)":0.94,"atomicData(RIC)"
:"SGB.AX","atomicData(DivExDate)":"18-
Nov-08"}}}},"edges":{"edge1":{"ordering":"start(n2) >=
start(n1)","source":{"node":"n2"},"target":{"node":"n1"}}}},"Code
":"Two events with Type \u2018Dividend\u2019 have the same
```

Figure 6.7 A sample file in the Event Pattern Occurrences Stack repository.

The List of Actions repository stores the final results generated by the system after processing all the rules, containing each action asserted by each executed rule, i.e. what to do with the original event dataset (see Section 4.5.4 for details). In this implementation, a CSV file is generated for each execution. Each action in the list has an ID, the action content and a decision making trace, i.e. how this conclusion (action) is achieved. A sample of the List of Actions is shown in Table 6.1.

Table 6.1 A sample of the List of Actions.

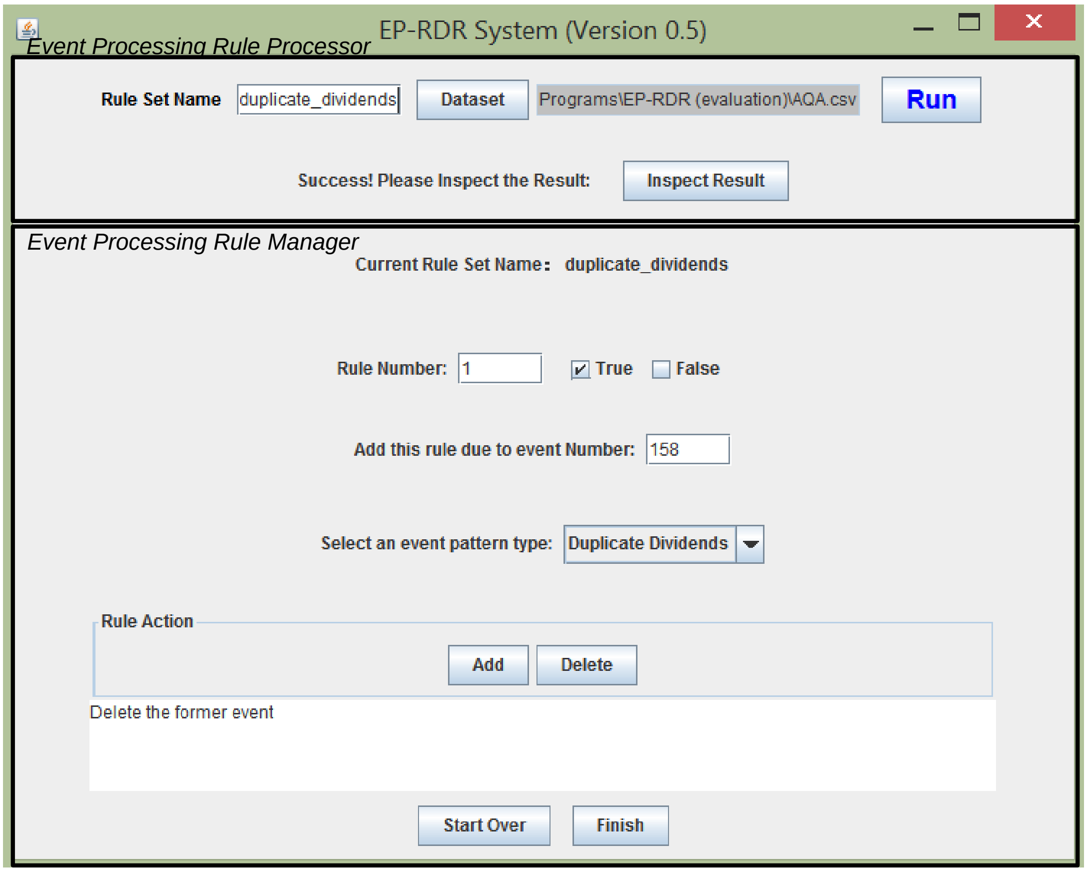
ID	Action Content	Decision Making Trace
1	Remove Event 158	Pattern "Duplicate Dividends": Event 158 and Event 159 ...
2	Remove Event 225	Pattern "Duplicate Dividends": Event 225 and Event 226 ...
...	...	...

6.4 User Layer Implementation

As discussed in Section 4.6, there are two GUIs in the user layer. The Event Processing Rule Manager GUI provides two main functions for the event data analyst to manage the event processing rules, namely defining event pattern types and defining rules. The Event Processing Rule Processor GUI allows the event data analyst to commence processing a defined rule set in the Event Processing Rule Base, and directs the event data analyst to inspect the result, i.e. a List of Actions generated by RDR, after Rule Execution. In this implementation, we have combined these two GUIs in one screen. Figure 6.8 shows a snapshot of this user interface, which has been developed using Java Swing.

The Event Processing Rule Management GUI enables the event data analyst to select pre-defined event pattern types, define event processing rules, and add defined rules into the rule base. Prior to rule definition, the GUI first asks the data analyst to specify the ID of the rule (shown as "Rule Number" in the GUI in Figure 6.8) and inference action (shown as "True" and "False" in the GUI in Figure 6.8) in the rule that has resulted in the incorrect action, along with the ID of the cornerstone case (i.e. the event that have triggered the rule addition, shown as "Add this rule due to event Number" in the GUI in Figure 6.8). When defining a rule, the data analyst needs to select a pre-defined event pattern type. Once a pre-defined event pattern type is selected, the analyst needs to specify the action to be taken on the occurrences of the selected event pattern type. Note that the cornerstone case specified here will be saved together with the new rule, and the inference actions will be changed according to the triggering rule and inference action specified here.

The Event Processing Rule Processor GUI provides the event data analyst with a list of rule sets to select from, and asks the analyst to select a dataset they would like to process. Then they can simply click the "Run" button to start the execution of the selected rule set on the selected dataset. If the execution is completed successfully, the data analyst can inspect the result by clicking the "Inspect Result" button.



EP-RDR System (Version 0.5)

Event Processing Rule Processor

Rule Set Name: duplicate_dividends

Programs\EP-RDR (evaluation)\AQA.csv

Success! Please Inspect the Result:

Event Processing Rule Manager

Current Rule Set Name: duplicate_dividends

Rule Number: 1 ☑ True □ False

Add this rule due to event Number: 158

Select an event pattern type: Duplicate Dividends

Rule Action

Delete the former event

Figure 6.8 An overview of the GUI in the prototype.

6.5 Limitations

As mentioned earlier, the main purpose of developing the prototype is to validate the feasibility of the proposed EP-RDR architecture and the corresponding EDMF via a case study. This case study, described in Chapter 7, is related to financial market data pre-processing and involves thirteen different event pattern types. The limitations of this prototype implementation are now discussed.

Firstly, while data analysts are able to define rules using pre-defined event pattern types, they are still not able to define new event pattern types by themselves. Data analysts need to communicate the details of new event pattern types to the IT expert, who will then define event pattern types separately from the GUI and associate them with corresponding EPS code to detect them. To improve the process of event pattern type definition, we can set up a standard on the visual presentation to define an event pattern type, and add a new component to the GUI. This allows data analysts to define simple event pattern types by drawing an event-dependency graph called P-DAG (see Section 5.3.4), and specifying the description of the event pattern type. Then the IT expert can read this information and implement the new event pattern accordingly. This makes the GUI more user-friendly and simplifies the process of communication between the IT expert and the data analyst. However, even if the GUI is extended in this way, only simple event pattern types can be easily defined via the GUI. As for the definition of complex event pattern types, the process is not simplified much especially when the P-DAG cannot be easily drawn via the GUI. In this case, data analysts still need to communicate the details of new event pattern types with the IT expert in person.

The second limitation is that only one EPS (EventSwarm) can be selected to be invoked by EPDaaS although the architecture is designed to have the ability to invoke multiple EPSs. This is not a big issue as long as the selected EPS can handle all the event pattern types required for the case study used in the thesis. To make this tool more powerful in expressiveness and performance, additional EPSs need to be integrated into the system.

Thirdly, the data model must be constructed according to the data related to a particular domain. This still requires manual work by an IT expert. To minimise the time of data model construction, more sophisticated data modelling tools need to be used.

6.6 Conclusion

This chapter described how we implemented the EP-RDR architecture proposed in Chapter 4 along with the data model proposed in Chapter 5. This implementation will be used to evaluate the suggest approach in the next chapter.

Chapter 7 Evaluation and Case Study: Financial Market Data Pre-Processing

This chapter uses a case study in the finance domain, i.e. financial market data pre-processing to validate the proposed work. In Section 7.1, it reviews the research questions, and defines the evaluation criteria and corresponding evaluation metrics. Then in Section 7.2, it gives some background to financial market data pre-processing and describes two scenarios used in the case study. The detailed experiments and results of the evaluation of the proposed method are provided in Sections 7.3 and 7.4. Parts of the material presented in this chapter have been published in [129].

7.1 Evaluation Criteria and Metrics

In this section, we will first review the research questions of the thesis, and define the criteria used to evaluate the contribution of the thesis in addressing the research questions. Then, we will define some evaluation metrics accordingly.

7.1.1 Review of Research Questions and Defining Evaluation Criteria

As pointed out in Section 3.1, the main objective of this research is to investigate an approach that enables data analysts to manage event data analysis on historical data with minimal IT expert intervention. Accordingly, we have defined the following research questions:

- How can we provide a new method that enables complex event processing and facilitates user-driven rule set evolution?
- How can we leverage existing Event Processing Systems to support the new method?
- How can we facilitate the interoperability of event data in any proposed system?

In order to answer these research questions, we proposed EP-RDR architecture (Chapter 4) and an EDMF framework (Chapter 5). In this chapter, we will be evaluating the thesis contribution in addressing these research questions. For this purpose, we now define the following evaluation criteria:

- ***Feasibility and Interoperability***: The EP-RDR architecture should be feasible; the interoperability of event data between the architectural components should be facilitated.

- ***Complex Event Processing Capability***: The system should be capable of conducting complex event processing on event datasets.
- ***User-Driven Rule Set Evolution Capability***: Data analysts with minimal IT experience should have control over rule set evolution in the system.

7.1.2 Evaluation Metrics

We now define some metrics to evaluate our new method according to each of the evaluation criteria above.

- Feasibility and Interoperability

This criterion is satisfied if the proposed EP-RDR architecture in Chapter 4 can be successfully implemented and the interoperability of event data between components (RDR and EPDaaS) can be realised. This will be achieved by:

- – Developing a prototype of the design (which involves developing RDR and EPDaaS components, selecting an underlying EPS, selecting the database, developing a usable GUI).
- – Selecting at least one appropriate event data set, running the prototype with the selected data set, and recording the execution times of one particular rule and several rules respectively.
- – Observing if event data can be transferred between RDR and EPDaaS components successfully, check whether there is any incorrect or missing information incurred when the event data transferred from RDR to EPDaaS, and whether the event pattern occurrences detected by the underlying EPS can be recognised and processed by the RDR component.

- Complex Event Processing Capability

This criterion is satisfied if event processing rules can be properly defined and executed. We should perform the following to accomplish this evaluation:

- – Define at least one scenario in an appropriate domain where event data analysis is involved. Select at least one appropriate event data set.
- – In each scenario, a data analyst should be able to add one event processing rule in an empty rule set and execute this rule only.
- – Observe if this event processing rule can be defined and executed properly.

- We can also compare the result of this rule with the result produced by other existing tools.

- User-Driven Rule Set Evolution Capability

This criterion is satisfied if data analysts are able to easily evolve the system by themselves easily, which means making changes to rule sets according to new cases without affecting other existing rules. This is the most important part of the evaluation. We will perform the following tasks to accomplish this evaluation:

- Define at least one scenario in an appropriate domain where incremental event data analysis is involved. Select at least one appropriate event data set. This can be done together with the evaluation of "Complex Event Processing Capability".
- In each scenario, a data analyst should work iteratively by executing the current rule set, inspecting the resulting list of generated actions and decisions, and adding new rules. This can also be done together with the evaluation of "Complex Event Processing Capability".
- Observe if new rules can be added successfully.
- In at least one particular iteration of rule addition, observe if any other existing rule is affected.
- Compare the time of rule set evolution (or rule change) using our new method with some other existing tools.
- Compare the efficiency of rule change in an appropriate iteration of the rule set evolution. The rule change efficiency (RCE) of the system in one particular iteration of rule change can be calculated using the following formula:

$$\boxed{\textit{RCE = (1 - \% of false negatives) / No. of rules changed}}$$

Now we illustrate the formula above. For a particular iteration of rule change due to a new case, consider in the selected data set there are totally N cases to be detected and asserted an action on. False negatives are cases that should be detected but have not been detected. If the number of false negatives is FN, then:

$$\boxed{\textit{RCE = (N - FN) / (N * No. of rules changed)}}$$

Note that in this formula we have applied the concept of "recall" in the machine learning field [130]:

$$\boxed{\textit{Recall = (N - FN) / N}}$$

7.1.3 Summary

To sum up, we need to do the following to accomplish the evaluation on all the criteria listed above:

- Select appropriate event data sets in a particular domain and define case study scenarios where incremental event data analysis is involved.
- Develop a prototype, and compare the execution time of one particular event processing rule with other tools.
- Observe the various aspects of the system according to different criteria.
- Compare our new method with some other existing tools via observation and calculation.

Now we start the illustration on the case study in the following section, including the selection of a proper domain and appropriate event data sets in this domain, and the definition of scenarios where incremental event data analysis is involved.

7.2 Case Study Background

We have selected financial market data pre-processing as the case study used to evaluate the proposed architecture. First of all, financial market data (e.g. stock data, corporate action data, etc.) comes with a timestamp associated with each record, so it can be considered as a typical example of event data. Secondly, data pre-processing is an integral part of financial market data analysis, which involves addressing time-related data quality issues. Many interesting event pattern types can be defined and used in financial market data pre-processing. Therefore, we are able to find appropriate scenarios to conduct the evaluation discussed above.

In this section, an overview of financial market data pre-processing and some examples are provided in Section 7.2.1. Then, we illustrate in detail two types of financial market data analysis that require complex data pre-processing in Sections 7.2.2 and 7.2.3 respectively.

7.2.1 Overview of Financial Market Data Pre-Processing

Financial market data is a source of knowledge about trading behaviour, strategies and patterns, market design, operations and efficiency, products being traded, and financial news. For this reason, there are many financial market data providers that sell either real-time data or historical data. Examples of real-time financial market data providers include Thomson Reuters [131] and Bloomberg [132]. Examples of historical financial market data providers include Sirca [25] that specialises in Australian data and WRDS [133] that provides mostly US data. In this case study, we will be using data downloaded from Thomson Reuters Tick History (TRTH) provided by Sirca.

Prior to conducting any meaningful financial study, data analysts expend a lot of effort on data collecting and pre-processing (also known as data preparation). Since there are many data quality issues such as missing data, duplicates and inconsistencies, data pre-processing is an indispensible phase where data analysts detect and manage these data quality issues, and standardise the data to facilitate further processing and analysis. Data pre-processing generally involves [21]:

- Data quality control (Cleansing): attempts to fill in missing values, smooth out noise while identifying outliers, and correct inconsistencies in the data.
- Data integration: involves combining data residing in different sources and providing data analysts with more meaningful and valuable information.
- Data transformation (Standardisation): converts a set of data values from the data format of a source data system into the data format of a destination data system.
- Data reduction: is the process of minimising the amount of data that needs to be stored in a data storage environment. Data reduction can increase storage efficiency and reduce costs.

In particular, event data pre-processing handles a number of issues (see Table 7.1), which can be categorised into "time-related" and "non-time-related" issues.

There are many data quality issues associated with Sirca TRTH data that require pre-processing. For example, prior to carrying out an event study [134] using a software tool (e.g. Eventus [135]), the data analyst needs to first download the raw company stock data, and pre-process the dataset, i.e. cleansing the data and calculating returns (see Figure 7.1). If the data pre-processing is not done properly, the data analysis will end up with errors or unreasonable and unreliable results.

Table 7.1 Summary of major issues to be addressed by event data pre-processing

Category	Dimension	Problem Name	Explanation
Non-time -related issues	Accuracy	Illegal values	A value does not comply with the type or does not exist in the attribute dictionary
		Invalid formats	The format of value does not agree with the type
	Completeness	Missing values	Missing attribute values (null or default) within a data-compulsory field
	Consistency	Inconsistent values	Two attributes do not agree
		Inconsistent formats in the same attribute	Values within the same attribute have different formats resulting in difficulty in understanding
Time - related issues	Event Uniqueness	Duplicate events	Similar repeated records (with the same timestamps and ID values) due to event update or verification
	Timeliness	Out-of-date	The data is out-of-date and has to be updated or verified
	Completeness	Missing events	Missing event record which is compulsory
	Event Consistency	Inconsistent event order sequence	Time stream sequence is not in a consistent order
		Time-related key value disagreement	Key value of an event is changed over time, resulting in key value disagreement
		Time overlapping	Time overlapping due to key attribute value change
		Time-related meaning inconsistency	Meaning of an attribute depends on time it was generated

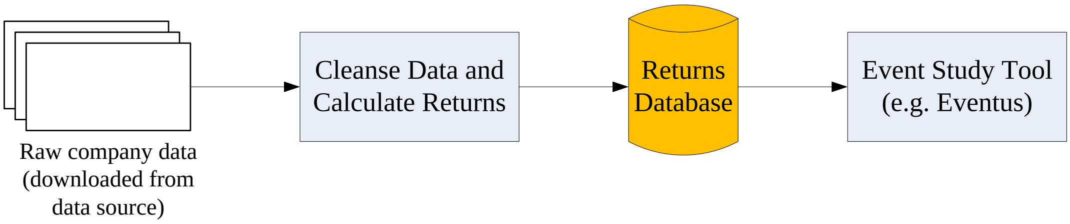

Raw company data (downloaded from data source)
Cleanse Data and Calculate Returns
Returns Database
Event Study Tool (e.g. Eventus)

Figure 7.1 Example of data pre-processing (for event studies).

The rest of this section will introduce two scenarios of TRTH financial market data pre-processing, namely eliminating duplicate dividends and calculating earnings. These two scenarios are both representative of financial market data pre-processing, and involve complex event processing rules and require incremental rule set evolution (i.e. modification on the business logic according to new cases). The purpose of the data analysis involved in these two scenarios is to detect occurrences of particular event pattern types related to the data pre-processing and to generate a CSV file with a list of actions to be taken on the original dataset. The actions can be deleting a particular event, reporting potential problems in data, filling in a value in a field, etc.

7.2.2 Scenario 1: Eliminating Duplicate Dividends

We introduced the duplicate dividends problem in Section 2.1.4 as an example of rule set evolution in event data analysis. This example will constitute the basis of our first scenario in the case study. In the following, we extend this example and illustrate the change of logic when data analysts are dealing with this problem.

There are totally six cases that result in an incremental modification of the rules (business logic). The cases are displayed in Table 7.2. The expected action relating to these cases is listed in Table 7.3.

Specifically, in the most common cases, like Case 1 shown in Table 7.2(a), duplicate events are considered simply as two dividend events with the same timestamp, the same "Div Amt." and the same "Div Ex Date". After a while in the new case (Case 2) shown in Table 7.2(b), the Div Ex Date in this Dividend event refers to a day that has no trading, which means this Dividend event may not be valid. This needs to be reported in the output so the data analyst can avoid the potential errors caused. Thus, a new rule should be defined to rectify the issue. Similarly, other new cases all result in changes of the existing rules (see Table 7.3 for details).

Table 7.2 Different cases in eliminating duplicate dividends.

(a) Case 1 – Simple duplicate dividend records: two events with Type "Dividend" has the same timestamp, the same "Div Amt." and the same "Div Ex Date".

#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 – No "End Of Day" event exists with "Div Ex Date" of a "Dividend" event as the timestamp.

#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

(c) Case 3 – An event with the type "Dividend" has null or empty value in the field "Div Amt." or "Div Ex Date".

#RIC	Date	Type	Div Ex Date	Div Amt	Div ID	Div Delete Marker	Payment Status
ABC	15/10/2013	Dividend			9344378	0	APPD

(d) Case 4 – Although these two dividends are issued at the same time (Date), the Div IDs are different which indicates they 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

(e) Case 5 – A "Dividend" event has a value other than "APPD" in the field "Payment Status". This dividend event is not approved (a value other than APPD) so it should be considered as an out-dated record.

#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	0	PROP

(f) Case 6 – A "Dividend" event has '1' in the field "Div Delete Marker". This dividend event is virtually deleted by the data provider, so this is not a valid entry.

#RIC	Date	Type	Div Ex Date	Div Amt	Div ID	Div Delete Marker	Payment Status
ABC	26/08/2014	Dividend	11/09/2012	0.15	9654412	1	APPD

Table 7.3 Cases and their expected actions of Scenario 1.

Case	Brief Description	Expected Action
1	Simple duplicate dividend records: Two events with Type "Dividend" have the same timestamp, the same "Div Amt." and the same "Div Ex Date".	Delete the former dividend event
2	No "End Of Day" event exists with "Div Ex Date" of a Dividend event as the timestamp.	Report it: Missing data
3	An event with the type "Dividend" has null or empty value in the field "Div Amt." or "Div Ex Date".	Discard this Div event and report missing value
4	A pair of duplicate dividends (pattern type No. 4) have different "Div Mkt Lvl ID". Although these two dividends are issued at the same time (Date), the Div IDs are different which indicates they are two different dividends rather than a duplicate.	This cannot be counted as a "duplicate dividends" case.
5	A "Dividend" event has a value other than 'APPD' in the field "Payment Status". This dividend event is not approved (a value other than APPD) so it should be considered as an out-dated record.	This dividend event cannot be counted in any "duplicate dividends" case.
6	A "Dividend" event has '1' in the field "Div Delete Marker". This dividend event is virtually deleted by the data provider, so this is not a valid entry.	This dividend event cannot be counted in any "duplicate dividends" case.

7.2.3 Scenario 2: Calculating Earnings

This is a more complicated scenario, which can better show the capability of the new system in handling rules involving complex event processing. In some financial studies, researchers need to calculate and compare the Price-Earnings ratios (P/E) of different companies in the same industry to indicate whether investors of a particular company are expecting higher earnings growth in the future than other companies. This is a very important measure for investors to assess a company. P/E equals market value per share (i.e. price) divided by earnings per share.

P/E = Price/Earnings per share

Thus, to calculate P/E ratio, we need both price data that provides price information and corporate action data that provides earnings information.

The task here is to find the correct value of "earnings per share" from "Earning" data for each trading day ("End Of Day" event) in TRTH data. There are totally six cases we have incrementally found that result in an incremental modification on the rules (business logic). The cases are displayed in Table 7.4. The expected action relating to these cases is listed in Table 7.5.

At the first stage, the data analyst defined a rule based on the most common cases like Case 1 shown in Table 7.4(a). The content of the rule is:

If *an event with type "Earning" (E) happens before an event with type "End Of Day" (EOD)*
(Note that for each EOD find only one closest occurrence if it exists),
then *the "earnings per share" for this EOD event should be calculated by*
*E.epsAmount * $10^{EPS\_scaling\_factor}$*[3]*.*

After a while, the data analyst found some new cases like the one (Case 2) shown in Table 7.4(b) that caused incorrect calculation using the previous rule. In this case, there are "Earning" events with "6" (months) as the value of the "EPS Period Length" field rather than "12" (month) as in the most common cases. The data analyst decided to annualise the "Earning" events, i.e. find two "Earning" events with "6" month "EPS Period Length" and add up "EPS Amount" values in these two records so their EPS periods together constitute one year. Thus, the calculation of the earnings of the End Of Day event should be changed. Similarly, other new cases all result in changes of the existing rules (see Table 7.5 for details).

[3] "epsAmount" denotes the value of the field "EPS Amount" in the "Earning" event, and "EPS_scaling_factor" denotes the value of the field "EPS Scaling Factor"

Table 7.4 Different cases considered in calculating earnings.

(a) Case 1 – Normal earning record.

RIC	Date	Type	EPS Period End Date	EPS Period Length	EPS Amount	EPS Scaling Factor
ABC	17/08/2011	Earning	30/06/2011	12	2306.9	-4
ABC	18/08/2011	EndOfDay				

(b) Case 2 – 6 months' earnings + 6 months' earnings.

RIC	Date	Type	EPS Period End Date	EPS Period Length	EPS Amount	EPS Scaling Factor
ABC	28/02/2012	Earning	31/12/2011	6	2071.7	-4
ABC	22/08/2012	Earning	30/06/2012	6	7974	-5
ABC	23/08/2012	EndOfDay				

(c) Case 3 – 3 months' earnings + 3 months' earnings + 6 months' earnings.

RIC	Date	Type	EPS Period End Date	EPS Period Length	EPS Amount	EPS Scaling Factor
ABC	28/02/2011	Earning	31/12/2011	3	952.5	-4
ABC	25/05/2012	Earning	31/03/2012	3	1350.6	-4
ABC	22/08/2012	Earning	30/06/2012	6	2011.3	-4
ABC	23/08/2012	EndOfDay				

(d) Case 4 – 3 months' earnings + 9 months' earnings.

RIC	Date	Type	EPS Period End Date	EPS Period Length	EPS Amount	EPS Scaling Factor
ABC	28/02/2012	Earning	31/12/2011	3	874	-4
ABC	25/05/2012	Earning	31/03/2012	9	1985	-4
ABC	26/05/2012	EndOfDay				

(e) Case 5 – 3 months' earnings + 3 months' earnings
+ 3 months' earnings + 3 months' earnings.

RIC	Date	Type	EPS Period End Date	EPS Period Length	EPS Amount	EPS Scaling Factor
ABC	28/02/2011	Earning	30/9/2011	3	522	-4
ABC	25/05/2012	Earning	31/12/2011	3	887	-4
ABC	22/08/2012	Earning	31/03/2012	3	1397.3	-4
ABC	22/08/2012	Earning	30/06/2012	3	4352	-5
ABC	26/05/2012	EndOfDay				

(f) Case 6 – 9 months' earnings + 3 months' earnings.

RIC	Date	Type	EPS Period End Date	EPS Period Length	EPS Amount	EPS Scaling Factor
ABC	28/02/2012	Earning	31/12/2011	9	2004	-4
ABC	28/02/2013	Earning	30/09/2012	3	1085.4	-4
ABC	01/03/2013	EndOfDay				

Table 7.5 Cases and their expected actions of Scenario 2.

Case	Brief Description	Expected Action
1	An event with type "Earning" (E) happens before an event with type "End Of Day" (EOD).	Calculate the earnings of the EOD event using the following formula: EOD.earnings = E.epsAmount * $10^{\text{EPS\_scaling\_factor}}$
2	Two events $E_{6(1)}$ and $E_{6(2)}$ with type "Earning" ($E_{6(2)}$ before $E_{6(1)}$) happen before an event with type "End Of Day" (EOD)	Calculate the earnings of the EOD event using the following formula: EOD.earnings = ($E_{6(1)}$.epsAmount + $E_{6(2)}$.epsAmount) * $10^{\text{EPS\_scaling\_factor}}$
3	Three events with type "Earning" ($E_{3(2)}$ before $E_{3(1)}$ before E_6) happen before an event with type "End Of Day" (EOD)	Calculate the earnings of the EOD event using the following formula: EOD.earnings = (E_6.epsAmount + $E_{3(1)}$.epsAmount + $E_{3(2)}$.epsAmount) * $10^{\text{EPS\_scaling\_factor}}$
4	One 3-month earning and one 9-month earning before End Of Day	Calculate the earnings of the EOD event using the following formula: EOD.earnings = E_9.epsAmount + E_3.epsAmount
5	Four 3-month earnings before End Of Day	Calculate the earnings of the EOD event using the following formula: EOD.earnings = ($E_{3(1)}$.epsAmount + $E_{3(2)}$.epsAmount + $E_{3(3)}$.epsAmount + $E_{3(4)}$.epsAmount) * $10^{\text{EPS\_scaling\_factor}}$
6	One 9-month earning and one 3-month earning before End Of Day	Calculate the earnings of the EOD event using the following formula: EOD.earnings = (E_3.epsAmount + E_9.epsAmount) * $10^{\text{EPS\_scaling\_factor}}$

7.2.4 Conventional Tools Used in Financial Market Data Pre-Processing

In real life scenarios, data analysts either employ a programmer to develop a bespoke program, or learn and apply an existing data pre-processing tool to perform pre-processing tasks on datasets. Therefore, we have selected two tools to be compared with our new system (EP-RDR) in the experiments (see Table 7.6). One of them is a bespoke program (BP) we developed which implements the data pre-processing logic in Java for both the "eliminating duplicate dividends" scenario and the "calculating earnings" scenario. The other one is OpenRefine (OP) [9], which is a popular and representative data pre-processing tool that is broadly utilised in various domains.

Table 7.6 Other tools used in the experiments to be compared with EP-RDR.

Abbr.	Name	Description
BP	Bespoke Program	A dedicated program that implements the data pre-processing logic in Java for both the "eliminating duplicate dividends" scenario and the "calculating earnings" scenario.
OP	OpenRefine	One of the most popular and free desktop applications for data pre-processing, previously funded by Google. This tool is easy to use but data pre-processing business logic is performed manually by data analysts.

7.3 Evaluating Feasibility, Interoperability and Event Processing Capability

7.3.1 Feasibility and Interoperability

As discussed in Chapter 6, we have successfully developed a prototype for the proposed artefacts, namely EP-RDR architecture and the event data modelling framework (EDMF). This prototype functioned properly and gave expected results. From the perspective of a software developer, the design is not hard to implement because most complex logic resides with an existing EPS (EventSwarm). Developers are able to implement it using different technologies flexibly. Therefore, the proposed method is feasible.

In both of the two scenarios (eliminating duplicate dividends and calculating earnings) described in Sections 7.2.2 and 7.2.3, the system could execute the corresponding rule set without any problem. Event data was transferred between RDR and EPDaaS components successfully and no incorrect or missing information was incurred. The event pattern occurrences detected by the underlying EPS (EventSwarm) are represented in JSON format, compliant with the data model we built via EDMF. Thus, these occurrences could also be recognised and processed by the RDR component, since they share the same meta-model.

In terms of execution time, we compared the EP-RDR prototype with BP (see Table 7.6). We selected one particular rule in the eliminating duplicate dividends scenario and 2 different data sets to record the execution time interval from starting the program to getting the final result (the list of actions). The rule is based on Case 1 in Table 7.2:

***If** two events with Type "Dividend" has the same timestamp, the same "Div Amt." and the same "Div Ex Date"*

***then** delete the former dividend event.*

Note that we have pre-uploaded three data sets in different sizes onto the EPS (EventSwarm) server prior to execution, so that the remote file upload time are not considered as part of the execution time.

For each of these three datasets, we first executed this rule for ten times and calculated the average execution time. The results are shown in Table 7.7 (a). Then, we executed seven rules defined according to the "eliminating duplicate dividends" scenario (see Section 7.2.2) ten times, using each of these three datasets on EP-RDR and BP respectively, and calculated the average execution time. The results are shown in Table 7.7 (b). Note that these seven rules are displayed in Section 7.4.1. Finally, we have calculated "the number of events per second" under execution time (per rule) to demonstrate the efficiency of both EP-RDR and BP (see Table 7.7(c)). The records of each execution time are listed in Appendix B (see Tables B.1 and B.2).

Table 7.7 Execution time comparison.

(a) Average execution time of 1 rule.

	Companies	Date Range	No. of events	Dataset Size (KB)	Average Execution Time (s)	
					EP-RDR	BP
Dataset 1	SGB.AX	01/06/2007 - 31/05/2009	764	143	0.374	0.247
Dataset 2	BHP.AX and ZYL.AX	01/01/2005 - 29/04/2011	5300	1001	0.870	0.753
Dataset 3	36 Dubai companies[4]	01/12/2006 - 04/12/2013	90262	32605	7.829	5.500

[4] The RICs of these 36 Dubai companies are listed in Appendix B (see Table B.3)

(b) Average execution time of 7 rules.

	Companies	Date Range	No. of events	Dataset Size (KB)	Average Execution Time (s)	
					EP-RDR	BP
Dataset 1	SGB.AX	01/06/2007 - 31/05/2009	764	143	2.585	0.591
Dataset 2	BHP.AX and ZYL.AX	01/01/2005 - 29/04/2011	5300	1001	4.817	1.845
Dataset 3	36 Dubai companies	01/12/2006 - 04/12/2013	90262	32605	35.140	10.149

(c) Number of events per second under execution time.

		EP-RDR	BP
1 Rule	Dataset 1	2043 events/sec	3093 events/sec
	Dataset 2	6092 events/sec	7039 events/sec
	Dataset 3	11529 events/sec	16411 events/sec
7 Rules	Dataset 1	2069 events/sec	9049 events/sec
	Dataset 2	7702 events/sec	20108 events/sec
	Dataset 3	17981 events/sec	62256 events/sec

As we can see from the results, EP-RDR is generally a little slower than BP. This is mainly due to the fact that the underlying EPS (EventSwarm) is only available remotely rather than locally. Therefore, the EP-RDR prototype requires remote invocation of the underlying EPS, which largely relies on the Internet connection status. While executing multiple rules, multiple remote invocations are required (one invocation for each event pattern type), so the execution time difference becomes larger. If the underlying EPS could be invoked locally, the performance would be more comparable. The comparison in this section is mainly used to evaluate the feasibility of the proposed approach and the interoperability of the components.

In regard to the rule modification time, we compared the EP-RDR prototype with the BP and OP. We tried to build a rule set for all the cases shown in Table 7.2, BP required an IT expert to implement the rules and it took several days for each rule to be implemented. In OP, not all the rules can be implemented. In EP-RDR, the rule building process was much easier, and took only five minutes for each rule to be added. We also asked a data analyst with very limited IT knowledge to try build the rule base by himself and it was successfully done within an hour, without assistance by the IT expert.

7.3.2 Complex Event Processing Capability

To assess the ability of the system in detecting complex event pattern types, we tried to define two particular rules. The first rule is based on Case 1 in Table 7.2:

If *two events with Type "Dividend" has the same timestamp, the same "Div Amt." and the same "Div Ex Date"*
then *delete the former dividend event.*

The second rule is based on Case 2 in Table 7.4:

If *the following pattern occurs:*

$E_{6(2)}$ -> $E_{6(1)}$ -> *EOD*

Two events $E_{6(1)}$ *and* $E_{6(2)}$ *with type "Earning" (*$E_{6(2)}$ *before* $E_{6(1)}$*) happen before an event with type "End Of Day" (EOD) with:*

- The "EPS Period Length" of both $E_{6(1)}$ and $E_{6(2)}$ is 6;
- $E_{6(2)}$.epsEndDate + $E_{6(2)}$.epsLength = $E_{6(1)}$.epsEndDate

(Find only one closest occurrence for each EOD if it exists)
then *Calculate the earnings of the EOD event using the following formula:*
EOD.earnings = ($E_{6(1)}$*.epsAmount +* $E_{6(2)}$*.epsAmount) ** $10^{EPS\_scaling\_factor}$.

OP was not capable of handling the above rules as they required complex time handling (i.e. event pattern detection). BP and EP-RDR could handle these time-related rules and automate the process but BP had to be implemented by an IT expert, while EP-RDR was customised by data analysts. The result of this comparison is shown in Table 7.8.

Table 7.8 Complex Event Processing capability comparison.

√ : can be implemented and customised by data analysts
√ *: can be implemented but cannot be customised by data analysts
×: cannot be implemented

	EP-RDR	BP	OP
The 1st Rule	√	√*	×
The 2nd Rule	√	√*	×

7.4 Evaluating User-Driven Rule Set Evolution Capability

7.4.1 Scenario 1 - Eliminating Duplicate Dividends

In the first scenario - eliminating duplicate dividends, using EP-RDR, the event data analyst initially defined three rules (Table 7.9(a)). After four iterations, there are finally seven rules (Table 7.9(b)). The actions and inference actions in ***italic*** are those that were added or changed. In terms of using BP, the data analyst initially asked the IT expert to define the identical three rules as shown in Table 7.9(a). For each iteration of rule change, the analyst asked the IT expert to modify the program. As for using OP, the data analyst first tried to define the identical three rules as shown in Table 7.9(a). For each iteration of rule change, the analyst tried to modify the rules by himself. Table 7.10 displays all the event pattern types involved in these event processing rules.

Table 7.9 Initial rule set for eliminating duplicate dividends

(a) Initial rule set for eliminating duplicate dividends

Rule No.	Event pattern type	Action	Inf. action (true)	Inf. action (false)
1	Dividend event	N/A	2	2
2	Duplicate Dividends	Delete the former one	3	3
3	Missing an EOD event on DED	Report it as an error	-1	-1

(b) Final rule set for eliminating duplicate dividends

Rule No.	Event pattern type	Action	Inf. action (true)	Inf. action (false)
1	Dividend event	N/A	***4***	2
2	Duplicate Dividends	Delete the former one	***5***	3
3	Missing an EOD event on DED	Report it as an error	-1	-1
4	Missing value in a Div event	***Discard this Div event and report missing value***	***2***	***2***
5	Different Div ID	***Retrieve the last action***	***3***	***6***
6	Status is not 'APPD'	***Retrieve the last action***	***3***	***7***
7	Delete Marker is not 0	***Retrieve the last action***	***3***	***3***

Table 7.10 Event pattern types used in the rule set for eliminating duplicate dividends.

ID	Name	Event Pattern Description
1	Dividend event	An event is a "Dividend" event.
2	Duplicate Dividends	Two events with Type "Dividend" have the same timestamp, the same "Div Amt." and the same "Div Ex Date".
3	Missing an EOD event on DED	No "End Of Day" event exists with "Div Ex Date" of a "Dividend" event as the timestamp.
4	Missing value in a Div event	An event with the type "Dividend" has null or empty value in the field "Div Amt." or "Div Ex Date".
5	Different Div ID	A pair of duplicate dividends (pattern type No. 4) have different "Div Mkt Lvl ID".
6	Status is not 'APPD'	A "Dividend" event has a value other than 'APPD' in the field "Payment Status".
7	Delete Marker is not '0'	A "Dividend" event has '1' in the field "Div Delete Marker".

We did an experiment with the "eliminating duplicate dividends" scenario to compare the rule change efficiency of both BP and EP-RDR. Specifically, a data analyst ran the initial rule set on twenty-two Australian companies to detect duplicate dividends. The RIC codes of the twenty-two companies are listed in Appendix B (see Table B.4). Then the data analyst tried to make changes to the rule base according to Case 3 in Table 7.2(c). Finally we calculated the rule change efficiency (RCE) of each system:

$$\boxed{\textit{RCE = (1 - \% of false negatives) / No. of rules changed}}$$

In this case, we know that there should be totally 300 dividends removed due to duplicate dividends or invalid entries (missing values). False negatives will be Dividend events not removed but should have been removed.

Specifically, after running the initial rule set in Table 7.9(a), a programmer was asked to modify the code of BP due to Case 3 in Table 7.2(c) for the data analyst; meanwhile, the data analyst added a new rule in the rule set used by EP-RDR (Rule 4 in Table 7.9(b)). We found that with respect to BP, the change of the first rule in the initial code affected the two other following rules in the code, which resulted in three iterations of changes. In fact, it took quite a long time (approximately one day) to have it working in the right way. In OP, it is the same case as with BP, but Rule 3 (a time-related rule) could not be properly defined. By contrast, in EP-RDR, we simply added one rule (Rule 4 in Table 7.9(b)) and the existing rules were not affected, so the result came out correctly without any side effect. The time of this rule addition is approximately five minutes, considerably shorter than using BP or OP. Table 7.11 shows the comparison of rule change efficiency of the three systems.

Compared with BP and OP, EP-RDR enables the event data analyst to update the rule base simply and neatly without assistance from the IT expert, consuming a significantly less amount of time. Therefore, we can draw a conclusion that the efficiency of rule changes in our new system out-performs BP and OP.

Table 7.11 Comparison in rule change efficiency.

Tool	Rules changed	Total duplicates not removed	Execution repetitions	Rule Change Efficiency (RCE)
BP & OP	1	300/300	3	33.3%
	2	12/100		
	3	0/100		
EP-RDR	1	0/300	1	100%

7.4.2 Scenario 2 - Calculating Earnings

In the second scenario - calculating earnings, initially, only one rule (Table 7.12(a)) was defined by the data analyst. When using EP-RDR, after five iterations of rule addition by the data analyst, there were finally six rules (Table 7.12(b)). The actions and inference actions in ***italic*** are those that were added or changed. This more complicated scenario is to demonstrate the more complex event processing capabilities of EP-RDR. Table 7.13 displays all the event pattern types involved in these event processing rules.

Table 7.12 Initial and final rules for calculating earnings

(a) Initial rule set for calculating earnings

Rule No.	Event pattern type	Action	Inf. action (true)	Inf. action (false)
1	Earning before End Of Day	EOD.earnings = E.epsAmount * $10^{\text{EPS\_scaling\_factor}}$	-1	-1

(b) Final rule set for calculating earnings

Rule No.	Event pattern type	Action	Inf. action (true)	Inf. action (false)
1	Earning before End Of Day	EOD.earnings = E.epsAmount * $10^{EPS\_scaling\_factor}$	***2***	***2***
2	Two 6-month earnings before End Of Day	$\boldsymbol{EOD.earnings = (E_{6(1)}.epsAmount + E_{6(2)}.epsAmount) * 10^{EPS\_scaling\_factor}}$	***3***	***3***
3	Two 3-month earnings and one 6-month earning before End Of Day	$\boldsymbol{EOD.earnings = (E_{6}.epsAmount + E_{3(1)}.epsAmount + E_{3(2)}.epsAmount) * 10^{EPS\_scaling\_factor}}$	***4***	***4***
4	One 3-month earning and one 9-month earning before End Of Day	$\boldsymbol{EOD.earnings = E_{9}.epsAmount + E_{3}.epsAmount}$	***5***	***5***
5	Four 3-month earnings before End Of Day	$\boldsymbol{EOD.earnings = (E_{3(1)}.epsAmount + E_{3(2)}.epsAmount + E_{3(3)}.epsAmount + E_{3(4)}.epsAmount) * 10^{EPS\_scaling\_factor}}$	***6***	***6***
6	One 9-month earning and one 3-month earning before End Of Day	$\boldsymbol{EOD.earnings = (E_{3}.epsAmount + E_{9}.epsAmount) * 10^{EPS\_scaling\_factor}}$	***-1***	***-1***

Table 7.13 Event pattern types used in the rule set for calculating earnings.

EOD ≡ End Of Day, E ≡ Earning

An arrow defines the chronological order of two events, i.e. a -> b means a happens "strictly" before b (we do not count it when a and b happen on the same date); E_n means an "Earning" event with *n* as the value of "EPS Period Length"; EOD denotes an End-of-day event (price data); ‘epsEndDate’ denotes the field "EPS Period End Date"; ‘epsLength’ denotes the field "EPS Period Length"

ID	Name	Event Pattern Description
1	Earning before End Of Day	*E -> EOD* An event with type "Earning" (E) happens before an event with type "End Of Day" (EOD). (Find only one closest occurrence for each EOD if it exists.)
2	Two 6-month earnings before End Of Day	$E_{6(2)}$ -> $E_{6(1)}$ -> *EOD* Two events $E_{6(1)}$ and $E_{6(2)}$ with type "Earning" ($E_{6(2)}$ before $E_{6(1)}$) happen before an event with type "End Of Day" (EOD) with: • The "EPS Period Length" of both $E_{6(1)}$ and $E_{6(2)}$ is 6; • $E_{6(2)}$.epsEndDate + $E_{6(2)}$.epsLength = $E_{6(1)}$.epsEndDate • Find only one closest occurrence for each EOD if it exists.
3	Two 3-month earnings and one 6-month earning before End Of Day	$E_{3(2)}$ -> $E_{3(1)}$ -> E_6 -> *EOD* Three events with type "Earning" ($E_{3(2)}$ before $E_{3(1)}$ before E_6) happen before an event with type "End Of Day" (EOD) with: • The "EPS Period Length" of $E_{3(2)}$ and $E_{3(1)}$ is 3; The "EPS Period Length" of E_6 is 6; • $E_{3(2)}$.epsEndDate + $E_{3(2)}$.epsLength = $E_{3(1)}$.epsEndDate • $E_{3(1)}$.epsEndDate + $E_{3(1)}$.epsLength = E_6.epsEndDate • Find only one closest occurrence for each EOD if it exists.

4	One 3-month earning and one 9-month earning before End Of Day	*E_3 -> E_9 -> EOD* Two events E_3 and E_9 with type "Earning" (E_3 before E_9) happen before an event with type "End Of Day" (EOD) with: • The "EPS Period Length" of E_3 is 3 and The "EPS Period Length" of E_9 is 9; • E_3.epsEndDate + E_3.epsLength = E_9.epsEndDate • Find only one closest occurrence for each EOD if it exists.
5	Four 3-month earnings before End Of Day	*$E_{3(4)}$ -> $E_{3(3)}$ -> $E_{3(2)}$ -> $E_{3(1)}$ -> EOD* Four events $E_{3(1)}$, $E_{3(2)}$, $E_{3(3)}$, and $E_{3(4)}$ with type "Earning" ($E_{3(4)}$ before $E_{3(3)}$ before $E_{3(2)}$ before $E_{3(1)}$) happen before an event with type "End Of Day" (EOD) with: • The "EPS Period Length" of $E_{3(1)}$, $E_{3(2)}$, $E_{3(3)}$, and $E_{3(4)}$ is 3; • $E_{3(i)}$.epsEndDate + $E_{3(i)}$.epsLength = $E_{3(i-1)}$.epsEndDate (i=2,3,4) • Find only one closest occurrence for each EOD if it exists.
6	One 9-month earning and one 3-month earning before End Of Day	*E_9 -> E_3 -> EOD* Two events E_3 and E_9 with type "Earning" (E_9 before E_3) happen before an event with type "End Of Day" (EOD) with: • The "EPS Period Length" of E_3 is 3 and The "EPS Period Length" of E_9 is 9; • E_9.epsEndDate + E_9.epsLength = E_3.epsEndDate • Find only one closest occurrence for each EOD if it exists.

7.5 Discussion

In summary, the proposed system has successfully addressed all the evaluation criteria: feasibility and interoperability, complex event processing capability and user-driven rule set evolution capability.

- ***Feasibility and Interoperability***: The design does not put many restrictions on the technologies to be used and is not difficult for a developer to implement. The prototype works as expected and it is a bit slower than a local bespoke program due to repetitive remote invocations of the underlying EPS (for each event pattern type to be detected). However, considering the time of remote invocation, the speed is not a big issue. If the underlying EPS could be invoked locally, the performance would be more comparable. The data models used in both the RDR component and the EPDaaS component (including the underlying EPS - EventSwarm) are all built via the EDMF, sharing the same meta-model. This ensures the exchange of data between them.
- ***Complex Event Processing Capability***: EPDaaS as a component plays the role of invoking the underlying EPS, which has the capability to detect event pattern occurrences. With EPS technology leveraged in the system, the system can address complex event processing effectively.
- ***User-Driven Rule Set Evolution Capability***: With the involvement of the RDR approach, the system allows for incremental, error-triggered rule management, i.e. it is possible to add a new rule when an incorrect action is found without corrupting the existing rule base.

By contrast, a bespoke program is normally dedicated to automating a specific data pre-processing logic, which can hardly be customised by data analysts. Time-related issues might be specifically handled but normally it is very inefficient. Although existing tools, such as OpenRefine, provide the function to save all used rules so data analysts can apply them to similar datasets automatically, this is still extremely error-prone. Most importantly, data pre-processing tools, like OpenRefine, are not capable of addressing time-related issues.

An additional advantage of our new system over others is that as all inference actions are always stored along with existing rules at any stage of the rule set evolution, data analysts can run historical rule sets so as to repeat whatever has been done before. Specifically, in both scenarios, in BP, once the program is modified, it cannot be converted back so previous rules at earlier stages cannot be repeated. In OpenRefine, we need to save the rule set manually (e.g. in a JSON file) at any stage for future use, otherwise, the rules will not be retained. In EP-RDR, we were able to execute previous rule sets at any stage of the rule set evolution process, which is a plus for the EP-RDR system.

There are some limitations in the evaluation. Firstly, the rule sets were not very large and specific to one particular domain. Future work will need to design experiments with bigger rule sets in other domains. Secondly, while the data analyst could play the role of managing the rules, an IT expert was still involved in defining and coding event pattern types in the underlying EPS and in building the data model via EDMF. However, taking advantage of existing EPLs, the work of the IT expert was much easier than developing the entire tool or maintaining the logic in the code according to data analyst evolving requirements. Last but not least, the underlying EPS was invoked remotely, which resulted in the deterioration of performance when the number of rules becomes larger. In the future, we could invoke an underlying EPS locally, so that the performance is more comparable.

Chapter 8 Conclusion and Future Work

The primary contribution of this thesis lies in a rule-based architecture for event data analysis called EP-RDR and an event data modelling framework called EDMF, which have been subsequently implemented and evaluated. In this final chapter, Section 8.1 provides a summary of the main ideas and findings of the thesis. Next, Section 8.2 presents a critical evaluation on how our proposed method has addressed the research questions. Following that, we describe the main contributions of the thesis in Section 8.3, and identify the main limitations in Section 8.4. Finally, we discuss potential future work in Section 8.5.

8.1 Thesis Summary

This thesis has investigated the role of event data analysis and the importance of rule management when the analysis is being conducted by data analysts who have limited IT knowledge (fundamental usage of popular operating systems, e.g. clicking buttons using the mouse, opening and closing a data file, etc.). It has found that event data has some unique characteristics, which make it more difficult to process than other types of data. To conduct event analysis tasks, data analysts have to rely on IT experts either to implement a bespoke program/service or to customise an event processing system (EPS) according to their needs. Because of constant changes in business needs and the environment, data analysts need to communicate their new requirements to IT experts to update and maintain the event data analysis business logic - event processing rules. When it comes to dealing with large rule sets, it is hard for an event data analyst with limited IT knowledge to manage them, as any change on one particular rule in the rule set may affect other existing rules, thereby corrupting the whole rule set. In this case, a knowledge engineer is needed to manage the event processing rules.

Through a thorough exploration on three major categories of potential approaches for event data analysis: generic approaches (e.g. bespoke programs, data pre-processing tools, database management systems, and statistical data analysis tools), Event Processing Systems or EPSs (e.g. Esper, EventSwarm), and rule-based systems (e.g. Ripple Down Rules or RDR), this thesis compared all these approaches in terms of three capabilities: automation, event processing and user-driven rule set evolution. The research found that no single approach possesses all these capabilities at the same time. To be more specific, the automation capability can be easily met as long as the event data analysis logic can be defined as a runnable process. The capacity to conduct complex event processing can only be provided by an EPS. This, however, does not facilitate user-driven rule set evolution of the system for addressing changing requirements. The user-driven rule set evolution is satisfied by a rule-based system such as RDR, but this technology does not support event processing and thus has not yet been applied to event data analysis. The research gap is "*There is no existing approach for event data analysis that enables event processing and facilitates user-driven rule set evolution.*" Consequently, this thesis has sought to address the research gap by investigating a new approach to enabling data analysts in managing event data analysis on historical data with minimal IT expert intervention. The three research questions are:

- How can we provide a new method that enables complex event processing and facilitates user-driven rule set evolution?
- How can we leverage existing Event Processing Systems to support the new method?
- How can we facilitate the interoperability of event data in any proposed system?

In answer to these research questions, this thesis proposes the "Event-Processing RDR" architecture (EP-RDR), which leverages EPS technologies as well as the power of RDR systems. This architecture has two main components in the business layer, namely RDR and EPDaaS (Event Pattern Detection as a Service). The RDR component is re-designed for event processing purposes and plays the role of managing the event processing logic, while still supporting incremental rule insertion that enables data analysts to define and add rules by themselves. Any EPS can be integrated and invoked by EPDaaS so that the system allows data analysts to conduct event processing without any concern about which event processing language/engine to use. To enable the interoperability between components in the architecture, this thesis also proposes an "Event Data Modelling Framework" (EDMF), which allows data model builders to construct an event data model more easily and in a consistent manner. EDMF consists of an Event Data Meta-Model and its associated Operational Guidelines. The Event Data Meta-Model is a high-level model that provides sufficient abstractions from various co-existing event data representation formats so that the semantics of different approaches can be captured in a consistent manner, and the meta-model can be easily extended to different types of datasets and domains. The Operational Guidelines are a number of steps to be followed by the data model builder. A data model built based on EDMF will allow event pattern types and occurrences to be defined, and will abstract existing events and event pattern occurrences data formats in a consistent manner.

In order to evaluate the effectiveness of the proposed method, a prototype was implemented. This thesis applied three criteria to conduct the evaluation, namely, feasibility and interoperability, event processing capability, and user-driven rule set evolution capability. A case study on financial market data pre-processing was elaborated and the prototype was used in the two real-life scenarios (eliminating duplicate dividends and calculating earnings) to check the ability of the proposed method to satisfy the evaluation criteria.

8.2 Addressing the Research Questions

This section discusses how the proposed method (EP-RDR and EDMF) has answered the research questions. The discussion is based on characteristics of the method reported in this thesis as well as the results from the case study evaluations. Specifically,

- The first research question "*How can we provide a new method that enables complex event processing and facilitates user-driven rule set evolution?*" has been addressed by the idea of leveraging the EPS technology with an RDR approach. According to the validation results in Section 7.3.1, integrating EPS technology with an RDR approach within a single system is feasible. In particular, the RDR component maintains the rule base and allows for incremental, error-triggered rule management, i.e. adding a new rule when an incorrect action is inspected without corrupting the existing rule base. The experiments described in Section 7.4 prove that the proposed method can facilitate user-driven rule set evolution.
- The second research question "*How can we leverage existing Event Processing Systems to support the new method?*" has been addressed by the proposed EP-RDR architecture in which the EPDaaS component plays the role in invoking any underlying EPS for conducting event pattern detection. The experiments described in Section 7.3.2 prove that the proposed method can support complex event processing via a commercial EPS (EventSwarm).
- The third research question "*How can we facilitate the interoperability of event data in any proposed system?*" has been addressed by the proposed EDMF, which allows event data models to be built in a consistent manner. The data models used in both the RDR component and the EPDaaS component should all be built using EDMF so that they share the same meta-model, and thus enable the interoperability between components of the EP-RDR architecture. The interoperability of the proposed method has been evaluated in Section 7.3.1, and the results prove that the components in the architecture are interoperable.

8.3 Thesis Contributions

Part of the materials presented in this thesis have been published in [116-118, 129]. The main contributions of the research carried out in this thesis can be summarised into:

- ***EP-RDR Architecture***: EP-RDR is designed for building and conducting user-driven event data analysis. Its two main components are a Ripple-Down Rule (RDR) system and an event pattern detection as a service (EPDaaS) which can interface with any event processing system (EPS). It enables incremental rule management by data analysts from the perspective of managing the whole rule base, and to leverage the functionality of existing Event Processing Systems. Taking advantage of the existing EPS, the IT expert's intervention in the system is simplified, i.e. the work of IT expert was much easier than developing the entire tool or managing the logic in the programming code. Additionally, this architecture can have applications outside event data analysis. For instance, applications based on this architecture can be developed for analysing other types of datasets, for which there are existing tools but managing the rules for the analysis can be difficult especially when the rule set is huge. Under this circumstance, the underlying EPS in the architectural design can be replaced by tools for this particular type of data analysis.
- ***Event Data Modelling Framework***: We have proposed a new Event Data Modelling Framework (EDMF) that facilitates event data modelling regardless of the domain, data source, or the EPS to be used in the EP-RDR architecture. EDMF is beneficial to both EP-RDR developers and data analysts. For developers, applying EDMF to the EP-RDR architecture, it will be easier to implement generic components that do not depend on how event pattern occurrences are represented. The data model provides the essential link between the two components of EP-RDR, i.e. RDR and EPDaaS, enabling the exchange of data in between. This framework can also be applied to enable interoperability of event datasets between different types of event processing systems. Such systems may have different data models, as long as the data models can be linked with each other via EDMF's meta-model. EDMF provides the basis for a new standard format for representing event datasets.

8.4 Thesis Limitations

This section discusses the main limitations of the proposed method and this thesis.

In terms of the scope of the thesis, the focus is on data analysis of *historical* event data only. The thesis assumes that the targeted users of the proposed system are data analysts in research institutions who do not require "real-time" processing. Thus, no consideration of real-time analysis was given. However, in the contemporary information age, real-time analysis with minimal lag is gradually becoming crucial for data analysts, especially those in enterprises, to ensure the timeliness of analysis results.

In terms of the design of the proposed system, one limitation is that an IT expert is still involved in the data analysis process. That is, while a data analyst is empowered to manage the event processing rules, an IT expert is still needed to define event pattern types associated with the rules. Therefore, essentially, data analysts do not have full control of the system in the Rule Addition use case (see Section 4.7.1).

The Event Data Modelling Framework (EDMF) consists of a high-level Event Data Meta-Model and its associated Operational Guidelines. The Event Data Meta-Model provides sufficient abstractions from various co-existing event data representation formats to capture the semantics of different approaches in a consistent manner. However, since only one type of event dataset (financial market data) has been used, there is no certainty that all event and event pattern types can be represented using the Event Data Meta-Model.

Another significant limitation lies in the evaluation part of the thesis, which has been done merely via a case study in the financial domain. The thesis proposed EP-RDR architecture and the EDMF, which claim the advantage of being able to be applied in various domains, but this has not been demonstrated in the thesis. As for the "Complex Event Processing Capability", the thesis evaluated the system using the "Eliminating Duplicate Dividends" and "Calculating Earnings" scenarios. However, the work must be demonstrated on more complex event pattern types.

Additionally, as mentioned earlier, a typical data analysis process has the following four phases: data collection, data cleansing (data pre-processing), data transformation and statistical analysis (see Section 2.1.1). Usually, data cleansing, data transformation and statistical analysis can be conducted separately or together. In the evaluation, we merely selected data pre-processing in the case study so the feasibility of performing data transformation and statistical analysis has not been demonstrated.

8.5 Future Work

This section describes some potential research issues and future work that has arisen as a result of the research presented in this thesis.

8.5.1 Short-Term Future Work

In the short term, another case study can be performed so as to evaluate the proposed method to a better extent. We may do the following to make it happen:

- Selecting another domain (e.g. health, biology, etc.) for which there are large amounts of event data that need to be analysed.
- Identifying a particular scenario that requires a large rule set in the selected domain. The rules should vary in the complexity of the underlying event pattern types. In other words, the rule set should combine very simple as well as very complex event pattern types. It would be better to include at least one event pattern type that EventSwarm (the Event Processing System we used in this thesis) cannot handle. This scenario could cover more phases of data analysis (e.g. data transformation, statistical analysis).
- Integrating another powerful EPS (e.g. Esper) that can bridge the gap that EventSwarm cannot fill. With two integrated EPSs, the claimed benefit of the proposed system (making use of multiple EPSs to enhance the performance of the system) can be evaluated.
- Develop a stable prototype of the proposed system in the context of a large-scale scientific software development project, and make it publicly available to academic data analysts, so that they are able to make use of it and provide valuable feedback.

8.5.2 Long-Term Future Directions

In the long term, it would be helpful to:

- Extend the proposed approach to real-time event data analysis. This will enable new commercial event data analysis applications in areas such as disaster management and business intelligence. Such effort requires a revised version of the proposed EP-RDR architecture to support real-time features. Most importantly, for real-time analysis, timeliness needs to be fulfilled. Thus, efficiency of the system will be the priority on top of the functionality, so further optimisation of the system’s architecture is necessary.

- Include additional components into the EP-RDR architecture to enhance the functionality of the system. For instance, external statistical computing modules could be integrated to make the system more suitable for statistical analysis.
- Extend the Event Data Meta-Model in the Event Data Modelling Framework (EDMF) described in Section 5.3, so that it can facilitate the representation of as many event pattern types as possible. This will make it considerably easier for the IT expert to build a data model for data in a particular domain and to implement more complex event pattern types in the underlying EPS.

References

[1] P. McFedries. (2011) The Coming Data Deluge. *IEEE Spectrum*. 19. Available: http://spectrum.ieee.org/at-work/innovation/the-coming-data-deluge

[2] P. Meier, "Data Flooding and Platform Scarcity," in *iRevolutions*, ed, 2009.

[3] I. G. Israel, "Information explosion and university libraries: Current trends and strategies for intervention," *Chinese Librarianship: an International Electronic Journal,* 2010.

[4] T. Hey, S. Tansley, and K. e. Tolle, *The Fourth Paradigm: Data-Intensive Scientific Discovery*. Redmond, Washington, 2009.

[5] IBM. (2013). *The Four V's of Big Data*. Available: http://www.ibmbigdatahub.com/infographic/four-vs-big-data

[6] B. Appelbe and D. Bannon, "eResearch - Paradigm Shift Or Propaganda?," *Journal of Research and Practice in Information Technology*, vol. 39:2, pp. pp. 83-90, 2007.

[7] Microsoft. (2015). *Excel*. Available: http://products.office.com/en-us/excel

[8] D. M. Harcar, "Justification and Expected Benefits of Data Analysis Automation Projects," 2015.

[9] OpenRefine. (2015). Available: http://openrefine.org/

[10] Oracle. (2015). *MySQL*. Available: http://www.mysql.com/

[11] SAS. (2014). *SAS*. Available: http://www.sas.com/en_us/home.html

[12] Esper. (2013). Available: http://esper.codehaus.org/

[13] C. M. Judd and G. H. McCleland, *Data Analysis: A Model-Comparison Approach*: Harcourt College Pub, 1989.

[14] R. Sapsford and V. Jupp, *Data Collection and Analysis*, 2nd ed.: SAGE Publications Ltd, 2006.

[15] T. Friedman and A. Bitterer, "Magic Quadrant for Data Quality Tools," Gartner. 28 July 2011.

[16] A. Bitterer, "Who's Who in Open-Source Data Quality," Gartner. 18 January 2012.

[17] A. K. Jain and R. C. Dubes, *Algorithms for clustering data*: Prentice-Hall, Inc., 1988.

[18] T. Milo and S. Zohar, "Using Schema Matching to Simplify Heterogeneous Data Translation," presented at the 24th VLDB, 1998.

[19] S. Gibilisco. (2014). *Statistical Analysis*. Available: http://whatis.techtarget.com/definition/statistical-analysis

[20] R. Schutt and C. O'Neil, *Doing Data Science: Straight Talk from the Frontline*: O'Reilly Media, Inc., 2013.

[21] J. Han and M. Kamber, *Data Mining: Concepts and Techniques*, 2nd ed. ed. San Francisco, CA, USA: Morgan Kaufmann Publishers, 2006.

[22] D. Luckham, *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. MA, USA: Addison Wesley Professional, 2002.

[23] O. Etzion and P. Niblett, *Event Processing in Action*: Manning Publications Co., 2011.

[24] F. A. Rabhi, L. Yao, and A. Guabtni, "ADAGE: A Framework for Supporting User-Driven Ad-hoc Data Analysis Processes," *Computing,* vol. 94, pp. 489-519, 2012.

[25] Sirca. (2015). Available: http://www.sirca.org.au/

[26] datos.gob.es, "Data Processing and Visualisation Tools," European PSI Platform2013.

[27] GitHub. (2014). *Documentation For Users*. Available: https://github.com/OpenRefine/OpenRefine/wiki/Documentation-For-Users

[28] DataWrangler. (2013). Available: http://vis.stanford.edu/wrangler/

[29] E. F. Codd, "A relational model of data for large shared data banks," *Commun. ACM,* vol. 13, pp. 377-387, 1970.

[30] Microsoft. (2015). *Access*. Available: https://products.office.com/en-au/access?legRedir=true&CorrelationId=78383e98-8ec5-4f52-b2aa-1fe0e3e73d86

[31] Apache. (2010). *OpenOffice Base*. Available: https://www.openoffice.org/product/base.html

[32] PostgreSQL. (2015). Available: http://www.postgresql.org/

[33] D. Rosenberg. (2010). *Are databases in the cloud really all that different?* Available: http://www.cnet.com/au/news/are-databases-in-the-cloud-really-all-that-different/

[34] K. Grolinger, W. Higashino, A. Tiwari, and M. Capretz, "Data management in cloud environments: NoSQL and NewSQL data stores," *Journal of Cloud Computing: Advances, Systems and Applications,* vol. 2, p. 22, // 2013.

[35] Apache. (2014). *CouchDB*. Available: http://couchdb.apache.org/

[36] Apache. (2014). *Hadoop*. Available: http://hadoop.apache.org/

[37] Apache. (2009). *Cassandra*. Available: http://cassandra.apache.org/

[38] Neo4J. (2015). Available: http://neo4j.com/

[39] MangoDB. (2015). Available: http://www.mongodb.org/

[40] K. Shailender and K. Shailender, "A Study of Temporal Database Research," *International Journal of Artificial Intelligence & Knowledge Discovery,* vol. 3, 2013.

[41] C. J. Date, H. Darwen, and N. A. Lorentzos, *Temporal Data and the Relational Model*: Morgan Kauffman Publishers, 2002.

[42] C. M. Saracco, M. Nicola, and L. Gandhi, "A matter of time: Temporal data management in DB2 10," IBM2012.

[43] C. J. Date, *An Introduction to Database Systems,* 8th ed.: Addison Wesley, 2003.

[44] R. (2015). *The R Project for Statistical Computing*. Available: http://www.r-project.org/

[45] G. Williams, *Data Mining with Rattle and R*: Springer, 2011.

[46] S. Zaidi and M. Nasir, *Teaching and Learning Methods in Medicine*: Springer, 2015.

[47] StataCorp. (2015). *Stata*. Available: http://www.stata.com/

[48] IBM. (2015). *SPSS*. Available: http://www-01.ibm.com/software/au/analytics/spss/

[49] IBM. (2015). *SPSS Modeler*. Available: http://www-01.ibm.com/software/au/analytics/spss/products/modeler/

[50] S. Chakravarthy and D. Mishra, "Snoop: an expressive event specification language for active databases," *Data Knowl. Eng.,* vol. 14, pp. 1-26, 1994.

[51] M. Gero, F. Ludger, and P. Peter, *Distributed event-based systems*, 2006.

[52] D. Luckham, *Event Processing for Business: Organizing the Real-Time Enterprise*. Hoboken, New Jersey: John Wiley & Sons, Inc., 2012.

[53] J. Bacon, K. Moody, J. Bates, R. Hayton, C. Ma, A. McNeil*, et al.*, "Generic Support for Distributed Applications," *IEEE Computer,* pp. 68-76, 2000.

[54] R. Barga and H. Caituiro-Monge, "Event Correlation and Pattern Detection in CEDR," in *Current Trends in Database Technology – EDBT 2006*. vol. 4254, T. Grust, H. Höpfner, A. Illarramendi, S. Jablonski, M. Mesiti, S. Müller*, et al.*, Eds., ed: Springer Berlin Heidelberg, 2006, pp. 919-930.

[55] M. Eckert, "Complex Event Processing with XChangeEQ," Dissertation, LMU Munich, 2008.

[56] E. Wu, Y. Diao, and S. Rizvi, "High-performance complex event processing over streams," presented at the Proceedings of the 2006 ACM SIGMOD international conference on Management of data, Chicago, IL, USA, 2006.

[57] G. Cugola and A. Margara, "TESLA: a formally defined event specification language," presented at the Proceedings of the Fourth ACM International Conference on Distributed Event-Based Systems, Cambridge, United Kingdom, 2010.

[58] D. Gyllstrom, E. Wu, H.-J. Chae, Y. Diao, P. Stahlberg, and G. Anderson, "SASE: Complex event processing over streams," 2006.

[59] K. Patroumpas and T. Sellis, "Window Specification over Data Streams," in *Current Trends in Database Technology – EDBT 2006*. vol. 4254, T. Grust, H. Höpfner, A. Illarramendi, S. Jablonski, M. Mesiti, S. Müller*, et al.*, Eds., ed: Springer Berlin Heidelberg, 2006, pp. 445-464.

[60] A. Arasu, S. Babu, and J. Widom, "The CQL continuous query language: semantic foundations and query execution," *The VLDB Journal,* vol. 15, pp. 121-142, 2006.

[61] S. Boll and G. U. Westermann, "MediAEther -- an Event Space for Context-Aware Multimedia Experiences," presented at the ACM SIGMM Workshop on Experiential Telepresence (ETP'03), Berkeley, California, 2003.

[62] SAP. (2015). *Continuous Computation Language*. Available: http://help.sap.com/saphelp_esp51sp08gsg/helpdata/en/e7/931cee6f0f101486068ade550250ad/content.htm?frameset=/en/e7/74cec06f0f1014bd1e98b8d524e830/frameset.htm&current_toc=/en/e7/74cec06f0f1014bd1e98b8d524e830/plain.htm&node_id=45&show_children=false

[63] L. Chung-Sheng, "Real-time event driven architecture for activity monitoring and early warning," in *Emerging Information Technology Conference, 2005.*, 2005, p. 4 pp.

[64] P. Mariño, C. Siguenza, J. Nogueira, F. Poza, and M. Dominguez, "An event driven software architecture for enterprise-wide data source integration," in *Information Technology: Coding and Computing, 2000. Proceedings. International Conference on*, 2000, pp. 140-145.

[65] Y. Ping-Peng, C. Gang, D. Jin-Xiang, and H. Wei-Li, "An event and service interacting model and event detection based on the broker/service model," in *Computer Supported Cooperative Work in Design, The Sixth International Conference on, 2001*, 2001, pp. 20-24.

[66] A. Berry and Z. Milosevic, "Real-Time Analytics for Legacy Data Streams in Health: Monitoring Health Data Quality," presented at the 17th IEEE International Enterprise Distributed Object Computing Conference (EDOC), 2013, Vancouver, BC, 2013.

[67] Sybase. (2014). *Sybase Aleri Event Stream Processor*. Available: http://infocenter.sybase.com/help/topic/com.sybase.infocenter.dc01286.0311/pdf/ProductOverview.pdf?noframes=true

[68] Oracle. (2014). *CQL*. Available: http://docs.oracle.com/cd/E17904_01/apirefs.1111/e12048/intro.htm

[69] TIBCO. (2014). *StreamSQL Guide*. Available: http://www.streambase.com/developers/docs/latest/streamsql/

[70] IBM. (2015). *Netcool/Impact Policy*. Available: http://publib.boulder.ibm.com/infocenter/tivihelp/v8r1/topic/com.ibm.netcoolimpact.doc6.1/PolicyReferenceGuide.pdf

[71] SoftwareAG. (2015). *Apama*. Available: http://www.softwareag.com/corporate/products/apama_webmethods/analytics/overview/default.asp

[72] Deontik. (2013). *EventSwarm*. Available: http://deontik.com/Products/EventSwarm.html

[73] RedHat. (2015). *Drools Fusion*. Available: http://drools.jboss.org/drools-fusion.html

[74] TIBCO. (2015). *TIBCO BusinessEvents*. Available: http://www.tibco.com/products/event-processing/complex-event-processing/businessevents/default.jsp

[75] A. Adi, D. Botzer, and O. Etzion, "The Situation Manager Component of Amit — Active Middleware Technology," in *Next Generation Information Technologies and Systems*. vol. 2382, A. Halevy and A. Gal, Eds., ed: Springer Berlin Heidelberg, 2002, pp. 158-168.

[76] IBM. (2015). *InfoSphere Streams*. Available: http://www-03.ibm.com/software/products/en/infosphere-streams/

[77] Google. (2014). Available: https://code.google.com/p/etalis/

[78] Prova. (2011). Available: https://prova.ws/confluence/display/EP/Event+processing

[79] J. Chen, D. J. DeWitt, F. Tian, and Y. Wang, "NiagaraCQ: a scalable continuous query system for Internet databases," presented at the Proceedings of the 2000 ACM SIGMOD international conference on Management of data, Dallas, Texas, USA, 2000.

[80] L. Liu, C. Pu, and W. Tang, "Continual Queries for Internet Scale Event-Driven Information Delivery," *IEEE Trans. on Knowl. and Data Eng.*, vol. 11, pp. 610-628, 1999.

[81] K. Robert, "Evaluation of the Stream Query Language CQL," ed. Sweden: The European Library, 2010.

[82] A. Paschke and A. Kozlenkov, "Rule-Based Event Processing and Reaction Rules," in *Rule Interchange and Applications*. vol. 5858, G. Governatori, J. Hall, and A. Paschke, Eds., ed: Springer Berlin Heidelberg, 2009, pp. 53-66.

[83] N. Jain, S. Mishra, A. Srinivasan, J. Gehrke, J. Widom, H. Balakrishnan, *et al.*, "Towards a streaming SQL standard," *Proc. VLDB Endow.*, vol. 1, pp. 1379-1390, 2008.

[84] D. Phan Minh and P. Mancarella, "Production systems with negation as failure," *Knowledge and Data Engineering, IEEE Transactions on,* vol. 14, pp. 336-352, 2002.

[85] K. Dittrich, S. Gatziu, and A. Geppert, "The active database management system manifesto: A rulebase of ADBMS features," in *Rules in Database Systems*. vol. 985, T. Sellis, Ed., ed: Springer Berlin Heidelberg, 1995, pp. 1-17.

[86] N. W. Paton and O. D, "Active database systems," *ACM Comput. Surv.*, vol. 31, pp. 63-103, 1999.

[87] J. Erikson, "CEDE: Composite Event Detector in An Active Database," 1993.

[88] R. Meo, G. Psaila, and S. Ceri, "Composite events in Chimera," in *Advances in Database Technology — EDBT '96*. vol. 1057, P. Apers, M. Bouzeghoub, and G. Gardarin, Eds., ed: Springer Berlin Heidelberg, 1996, pp. 56-76.

[89] N. H. Gehani, H. V. Jagadish, and O. Shmueli, "Event specification in an active object-oriented database," *SIGMOD Rec.*, vol. 21, pp. 81-90, 1992.

[90] C. Collet and T. Coupaye, "Composite events in NAOS," in *Database and Expert Systems Applications*. vol. 1134, R. Wagner and H. Thoma, Eds., ed: Springer Berlin Heidelberg, 1996, pp. 244-253.

[91] U. Dayal, B. Blaustein, A. Buchmann, U. Chakravarthy, M. Hsu, R. Ledin, *et al.*, "The HiPAC project: combining active databases and timing constraints," *SIGMOD Rec.*, vol. 17, pp. 51-70, 1988.

[92] IBM. (2015). *Amit*. Available: http://www.research.ibm.com/haifa/projects/software/extreme_blue/papers/eXB_AMIT.pdf

[93] S. Gatziu and K. Dittrich, "Events in an Active Object-Oriented Database System," in *Rules in Database Systems*, N. Paton and M. H. Williams, Eds., ed: Springer London, 1994, pp. 23-39.

[94] I. Schmerken. (2008). *Deciphering the Myths Around Complex Event Processing*. Available: http://www.wallstreetandtech.com/latency/deciphering-the-myths-around-complex-event-processing/d/d-id/1259489?

[95] D. Luckham. (2006). *What's the Difference Between ESP and CEP?* Available: http://www.complexevents.com/?p=103

[96] Eprentise. (2014). *Eprentise Data Quality software*.

[97] K. Duncan and D. Wells, "Rule Based Data Cleansing for Data Warehousing," 1999.

[98] I. Bratko, "Prolog Programming for Artificial Intelligence ", 4th ed, 2011.

[99] A. P. Reynolds, G. Richards, B. de la Iglesia, and V. J. Rayward-Smith, "Clustering Rules: A Comparison of Partitioning and Hierarchical Clustering Algorithms," *Journal of Mathematical Modelling and Algorithms,* vol. 5, pp. 475-504, 2006/12/01 2006.

[100] P. Compton and R. Jansen, "A philosophical basis for knowledge acquisition," *Knowledge Acquisition 2,* pp. 241-257, 1990.

[101] T. M. Cao and P. Compton, "A Simulation Framework for Knowledge Acquisition Evaluation," presented at the 28th Australasian Computer Science Conference, 2005.

[102] P. Compton, L. Peters, T. Lavers, and Y. S. Kim, "Experience with Long-term Knowledge Acquisition," *K-CAP,* 2011.

[103] D. Richards, "Two Decades of Ripple Down Rules Research," *The Knowledge Engineering Review,* vol. 24:2, pp. 159-184, 2009.

[104] P. Compton, L. Peters, G. Edwards, and T. G. Lavers, "Experience with Ripple-Down Rules," *Knowledge Based Systems,* vol. 19, pp. 356-362, 2006.

[105] V. Ho, W. Wobcke, and P. Compton, "EMMA: An E-Mail Management Assistant," in *IEEE/WIC International Conference on Intelligent Agent Technology*, 2003.

[106] V. H. Ho, P. Compton, B. Benatallah, J. Vayssiere, L. Menzel, and H. Vogler, "An Incremental Knowledge Acquisition Method for Improving Duplicate Invoices Detection," in *IEEE 25th International Conference on Data Engineering*, 2009, pp. 1415-1418.

[107] M. N. Dani, T. A. Faruquie, R. Garg, G. Kothari, M. K. Mohania, K. H. Prasad, *et al.*, "A Knowledge Acquision Method for Improving Data Quality in Services Engagements," presented at the IEEE International Conference on Services Computing, 2010.

[108] R. Arora and H. Karanam, "Leverage a ripple-down rules framework in InfoSphere QualityStage standardization rule set development," IBM Corporation2011.

[109] K. H. Prasad, T. A. Faruquie, S. Joshi, S. Chaturvedi, L. V. Subramaniam, and M. Mohania, "Data Cleansing Techniques for Large Enterprise Datasets," in *Annual SRII Global Conference (SRII)*, 2011, pp. 135-144.

[110] A. Hinze, K. Sachs, and A. Buchmann, "Event-based applications and enabling technologies," presented at the Proceedings of the Third ACM International Conference on Distributed Event-Based Systems, Nashville, Tennessee, 2009.

[111] K. Chandy and W. Schulte, *Event Processing: Designing IT Systems for Agile Companies*: McGraw-Hill, Inc., 2010.

[112] S. Sen and N. Stojanovic, "GRUVe: A Methodology for Complex Event Pattern Life Cycle Management," in *Advanced Information Systems Engineering*. vol. 6051, B. Pernici, Ed., ed: Springer Berlin Heidelberg, 2010, pp. 209-223.

[113] H. Obweger, J. Schiefer, M. Suntinger, P. Kepplinger, and S. Rozsnyai, "User-Oriented Rule Management for Event-Based Applications," presented at the Proceedings of the 5th ACM international conference on Distributed event-based system, New York, New York, USA, 2011.

[114] R. Murch, *Project Management: Best Practices for IT Professionals*: Prentice Hall PTR, 2000.

[115] G. V. Post and D. L. Anderson, *Management Information Systems: Solving Business Problems with Information Technology*: Irwin/McGraw-Hill, 2000.

[116] W. Chen, "E-Research Event Data Quality," presented at the Workshop in 29th IEEE International Conference on Data Engineering (ICDE'13), Brisbane, Australia, 2013.

[117] W. Chen and F. Rabhi, "An RDR-Based Approach for Event Data Analysis," presented at the Third Australasian Symposium on Service Research and Innovation (ASSRI'13), Sydney, Australia, 2013.

[118] W. Chen and F. Rabhi, "Enabling User-Driven Rule Management in Event Data Analysis," *Information Systems Frontiers*, 2014.

[119] W. Eckerson, "Three Tier Client/Server Architecture: Achieving Scalability, Performance, and Efficiency in Client Server Applications," in *Open Information Systems 10*, 1995.

[120] Microsoft. (2015). *Three-Layered Services Application*. Available: http://msdn.microsoft.com/en-us/library/ff648105.aspx

[121] P. Compton and R. Jansen, "Knowledge in context: a strategy for expert system maintenance," in *2nd Australian Joint Artificial Intelligence Conference*, 1988, pp. 292-306.

[122] B. H. Kang, P. Compton, and P. Preston, "Multiple Classification Ripple Down Rules: Evaluation and Possibilities," presented at the 9th Banff Knowledge Acquisition for Knowledge Based Systems Workshop, 1995.

[123] P. Compton, Y. Kim, and B. Kang, "Linked Production Rules: Controlling Inference with Knowledge," in *Knowledge Management and Acquisition for Smart Systems and Services*. vol. 8863, Y. Kim, B. Kang, and D. Richards, Eds., ed: Springer International Publishing, 2014, pp. 84-98.

[124] WADL. (2009). Available: http://www.w3.org/Submission/wadl/

[125] F. A. Rabhi, A. Guabtni, and L. Yao, "A data model for processing financial market and news data," *International Journal of Electronic Finance*, vol. 3, pp. 387-403, 01/01/ 2009.

[126] IBM. (2015). *IBM Health Plan Data Model*. Available: http://www-03.ibm.com/software/products/en/healthcare/

[127] J. Schiefer, S. Rozsnyai, C. Rauscher, and G. Saurer, "Event-driven rules for sensing and responding to business situations," presented at the Proceedings of the 2007 inaugural international conference on Distributed event-based systems, Toronto, Ontario, Canada, 2007.

[128] OMG, "OMG Unified Modeling Language (OMG UML), Superstructure. Version 2.4.1," 2011.

[129] W. Chen and F. Rabhi, "Validating an Incremental Rule Management Approach for Financial Market Data Pre-Processing," presented at the Workshop on Enterprise Applications, Markets and Services in the Finance Industry (FinanceCom 2014), Sydney, Australia, 2014.

[130] C. E. Metz, "Basic principles of ROC analysis," *Seminars in Nuclear Medicine,* vol. 8, pp. 283-298, 1978.

[131] Thomson-Reuters. (2014). Available: http://www.thomsonreuters.com.au/

[132] Bloomberg. (2015). Available: http://www.bloomberg.com/

[133] WRDS. (2015). *Wharton Research Data Services*. Available: http://wrds-web.wharton.upenn.edu/wrds/

[134] J. Binder, "The Event Study Methodology Since 1969," *Review of Quantitative Finance and Accounting,* vol. 11, pp. 111-137, 1998/09/01 1998.

[135] Eventus. (2012). Available: http://www.eventstudy.com/

Appendix A: Additional Information for EDMF

Table A.1. The XML schema of the proposed meta-model.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="Functor Type" type="Functor Type"/>
  <xs:complexType name="Functor Type">
    <xs:sequence>
      <xs:element name="name" type="xs:string" minOccurs="1" maxOccurs="1"/>
      <xs:element name="datatype" type="xs:string" minOccurs="1" maxOccurs="1"/>
    </xs:sequence>
  </xs:complexType>

  <xs:element name="Source" type="Source"/>
  <xs:complexType name="Source">
    <xs:sequence/>
  </xs:complexType>
  <xs:element name="Source" type="Source"/>
  <xs:complexType name="Source">
    <xs:sequence>
      <xs:element name="Functor Type" type="Functor Type" minOccurs="1" maxOccurs="1"/>
    </xs:sequence>
  </xs:complexType>

  <xs:element name="Timestamp" type="Timestamp"/>
  <xs:complexType name="Timestamp">
    <xs:sequence>
      <xs:element name="Functor Type" type="Functor Type" minOccurs="1" maxOccurs="1"/>
    </xs:sequence>
  </xs:complexType>

  <xs:element name="ComplexEventType" type="ComplexEventType"/>
  <xs:complexType name="ComplexEventType">
    <xs:complexContent>
      <xs:extension base="EventType">

        <xs:sequence>
          <xs:element name="EventPatternType" type="EventPatternType" minOccurs="1"
maxOccurs="1"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>

  <xs:element name="AtomicData" type="AtomicData"/>
  <xs:complexType name="AtomicData">
```

```
    <xs:sequence>
      <xs:element name="AtomicDataType" type="AtomicDataType" minOccurs="1"
maxOccurs="1"/>
    </xs:sequence>
  </xs:complexType>

  <xs:element name="Event" type="Event"/>
  <xs:complexType name="Event">
    <xs:sequence>
      <xs:element name="EventType" type="EventType" minOccurs="0"
maxOccurs="unbounded"/>
      <xs:element name="BaseEventType" type="BaseEventType" minOccurs="1"
maxOccurs="1"/>
    </xs:sequence>
  </xs:complexType>

  <xs:element name="EventPatternOcurrence" type="EventPatternOcurrence"/>
  <xs:complexType name="EventPatternOcurrence">
    <xs:sequence>
      <xs:element name="generated by" type="ComplexEvent" minOccurs="1" maxOccurs="1"/>
      <xs:element name="EventPatternType" type="EventPatternType" minOccurs="1"
maxOccurs="1"/>
      <xs:element name="PDAG Instance" type="PDAG Instance" minOccurs="1"
maxOccurs="1"/>
    </xs:sequence>
  </xs:complexType>

  <xs:element name="EventPatternType" type="EventPatternType"/>
  <xs:complexType name="EventPatternType">
    <xs:sequence>
      <xs:element name="PDAG Type" type="PDAG Type" minOccurs="1" maxOccurs="1"/>
      <xs:element name="Pattern Description" type="Pattern Description" minOccurs="1"
maxOccurs="1"/>
    </xs:sequence>
  </xs:complexType>

  <xs:element name="Pattern Description" type="Pattern Description"/>
  <xs:complexType name="Pattern Description">
    <xs:sequence>
      <xs:element name="Node Type" type="Node Type" minOccurs="1" maxOccurs="1"/>
    </xs:sequence>
  </xs:complexType>

  <xs:element name="SimpleEvent" type="SimpleEvent"/>
  <xs:complexType name="SimpleEvent">
    <xs:complexContent>
      <xs:extension base="Event">
        <xs:sequence/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>

  <xs:element name="ComplexEvent" type="ComplexEvent"/>
  <xs:complexType name="ComplexEvent">
    <xs:complexContent>
      <xs:extension base="Event">
        <xs:sequence>
          <xs:element name="ComplexEventType" type="ComplexEventType" minOccurs="1"
maxOccurs="1"/>
          <xs:element name="AtomicData" type="AtomicData" minOccurs="1"
maxOccurs="unbounded"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>

  <xs:element name="EventType" type="EventType"/>
  <xs:complexType name="EventType">
    <xs:sequence>
      <xs:element name="name" type="xs:string" minOccurs="1" maxOccurs="1"/>
      <xs:element name="function for" type="Functor Type" minOccurs="1"
maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>

  <xs:element name="PDAG Type" type="PDAG Type"/>
  <xs:complexType name="PDAG Type">
    <xs:sequence>
      <xs:element name="defined in" type="Node Type" minOccurs="1"
maxOccurs="unbounded"/>
      <xs:element name="defined by" type="Edge Type" minOccurs="1"
maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>

  <xs:element name="Node Type" type="Node Type"/>
  <xs:complexType name="Node Type">
    <xs:sequence>
      <xs:element name="EventType" type="EventType" minOccurs="1" maxOccurs="1"/>
    </xs:sequence>
  </xs:complexType>

  <xs:element name="Edge Type" type="Edge Type"/>
  <xs:complexType name="Edge Type">
    <xs:sequence>
      <xs:element name="ordering semantics" type="xs:string" minOccurs="1"
maxOccurs="1"/>
      <xs:element name="Node Type" type="Node Type" minOccurs="2" maxOccurs="2"/>
    </xs:sequence>
  </xs:complexType>

  <xs:element name="AtomicDataType" type="AtomicDataType"/>
  <xs:complexType name="AtomicDataType">
    <xs:sequence>
      <xs:element name="Format" type="Format" minOccurs="1" maxOccurs="1"/>
      <xs:element name="Schema" type="Schema" minOccurs="1" maxOccurs="1"/>
    </xs:sequence>
  </xs:complexType>

  <xs:element name="Format" type="Format"/>
  <xs:complexType name="Format">
    <xs:sequence/>
  </xs:complexType>

  <xs:element name="Schema" type="Schema"/>
  <xs:complexType name="Schema">
    <xs:sequence/>
  </xs:complexType>

  <xs:element name="Functor" type="Functor"/>
  <xs:complexType name="Functor">
    <xs:sequence>
      <xs:element name="Functor Type" type="Functor Type" minOccurs="1" maxOccurs="1"/>
      <xs:element name="AtomicDataType" type="AtomicDataType" minOccurs="1"
maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>

  <xs:element name="Simple Event" type="Simple Event"/>
  <xs:complexType name="Simple Event">
    <xs:complexContent>
      <xs:extension base="Event">
        <xs:sequence>
          <xs:element name="AtomicData" type="AtomicData" minOccurs="1"
maxOccurs="1"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>

  <xs:element name="BaseEventType" type="BaseEventType"/>
  <xs:complexType name="BaseEventType">
    <xs:sequence>
      <xs:element name="Source" type="Source" minOccurs="1" maxOccurs="1"/>
      <xs:element name="Timestamp" type="Timestamp" minOccurs="1" maxOccurs="1"/>
      <xs:element name="EventType" type="EventType" minOccurs="1" maxOccurs="1"/>
    </xs:sequence>
  </xs:complexType>

  <xs:element name="PDAG Instance" type="PDAG Instance"/>
  <xs:complexType name="PDAG Instance">
    <xs:sequence>
      <xs:element name="PDAG Type" type="PDAG Type" minOccurs="1" maxOccurs="1"/>
    </xs:sequence>
  </xs:complexType>

  <xs:element name="Node" type="Node"/>
  <xs:complexType name="Node">
    <xs:sequence>
      <xs:element name="contained by" type="Event" minOccurs="0"
maxOccurs="unbounded"/>
      <xs:element name="PDAG Instance" type="PDAG Instance" minOccurs="1"
maxOccurs="1"/>
      <xs:element name="Node Type" type="Node Type" minOccurs="1" maxOccurs="1"/>
    </xs:sequence>
  </xs:complexType>

  <xs:element name="Edge" type="Edge"/>
  <xs:complexType name="Edge">
    <xs:sequence>
      <xs:element name="PDAG Instance" type="PDAG Instance" minOccurs="1"
maxOccurs="1"/>
      <xs:element name="Edge Type" type="Edge Type" minOccurs="1" maxOccurs="1"/>
      <xs:element name="source" type="Event" minOccurs="1" maxOccurs="1"/>
      <xs:element name="target" type="Event" minOccurs="1" maxOccurs="1"/>
    </xs:sequence>
  </xs:complexType>

</xs:schema>
```

Table A.2. “Duplicate Dividends” event pattern occurrences in JSON format.

```
{
            "Occurrences": [
              {
                "Pattern Description": "Duplicate dividends",
                "PDAG": {
                  "nodes": [
                    {
                      "id": "e3373e90-30ca-5f81-95f3-981d8ea05102",
                      "source": "TR_Stream",
                      "startTime": "2008-10-29 00:00:00.000, +10",
                      "endTime": null,
                      "dataList": {
                        "Period Length": "6",
                        "Period Units": "12",
                        "Mandatory Voluntary Indicator": "MAN",
                        "Div Ex Date": "18-Nov-08",
                        "Franking %": "100",
                        "Feature": "40",
                        "Frequency": "99",
                        "Currency": "AUD",
                        "Ind. An Amount": "1.82",
                        "Descr": " Dividend Rinvestment plan in operation.",
                        "Div End Date": "30-Sep-08",
                        "Reinvestment Plan Available": "Y",
                        "Source of Fund": "8",
                        "Tax Marker": "2",
                        "Payment Status": "APPD",
                        "Corporate Action ID": "9035725",
                        "Spec. Tax Rules": "7",
                        "Div Reg. Date": "24-Nov-08",
                        "Div Amt.": "0.31",
                        "Payment Type": "CDI",
                        "Div Delete Marker": "0",
                        "Div Ann. Date": "29-Oct-08",
                        "Type": "Dividend",
                        "Date[L]": "29-Oct-08",
                        "Div Mkt Lvl ID": "8018473",
                        "Div Pay Date": "18-Dec-08",
                        "Type Marker": "70",
                        "#RIC": "SGB.AX",
                        "Tax Rate": "30"
                      }
                    },
                    {
                      "id": "913fa16f-7af6-5398-9c92-f6d29da36810",
                      "source": "TR_Stream",
                      "startTime": "2008-10-29 00:00:00.000, +10",
        "endTime": null,
        "dataList": {
          "Period Length": "6",
          "Period Units": "12",
          "Mandatory Voluntary Indicator": "MAN",
          "Div Ex Date": "18-Nov-08",
          "Franking %": "100",
          "Feature": "40",
          "Frequency": "99",
          "Currency": "AUD",
          "Ind. An Amount": "1.82",
          "Descr": "Dividend Rinvestment plan in operation.",
          "Div End Date": "30-Sep-08",
          "Reinvestment Plan Available": "Y",
          "Source of Fund": "8",
          "Tax Marker": "2",
          "Payment Status": "APPD",
          "Corporate Action ID": "9035725",
          "Spec. Tax Rules": "7",
          "Div Reg. Date": "24-Nov-08",
          "Div Amt.": "0.31",
          "Payment Type": "CDI",
          "Div Delete Marker": "0",
          "Div Ann. Date": "29-Oct-08",
          "Type": "Dividend",
          "Date[L]": "29-Oct-08",
          "Div Mkt Lvl ID": "8018473",
          "Div Pay Date": "18-Dec-08",
          "Type Marker": "70",
          "#RIC": "SGB.AX",
          "Tax Rate": "30"
        }
      }
    ],
    "edges": [
      {
        "ordering": "start(source) >= start(target)",
        "target": "nodes[0]",
        "source": "nodes[1]"
      }
    ]
  }
},
{
  "Pattern Description": "Duplicate dividends",
  "PDAG": {
    "nodes": [
      {
        "id": "2a2552f7-2875-5af4-b578-164c9df38b0a",
      "source": "TR_Stream",
      "startTime": "2008-10-29 00:00:00.000, +10",
      "endTime": null,
      "dataList": {
        "Period Length": "6",
        "Period Units": "12",
        "Mandatory Voluntary Indicator": "MAN",
        "Div Ex Date": "18-Nov-08",
        "Franking %": "100",
        "Feature": "40",
        "Frequency": "2",
        "Currency": "AUD",
        "Ind. An Amount": "1.82",
        "Qual. Income Elig.": "10",
        "Descr": " Dividend Rinvestment plan in operation.",
        "Div End Date": "30-Sep-08",
        "Reinvestment Plan Available": "Y",
        "Source of Fund": "2",
        "Tax Marker": "2",
        "Payment Status": "APPD",
        "Corporate Action ID": "9035722",
        "Spec. Tax Rules": "7",
        "Div Reg. Date": "24-Nov-08",
        "Div Amt.": "0.94",
        "Payment Type": "CDI",
        "Div Delete Marker": "0",
        "Div Ann. Date": "29-Oct-08",
        "Type": "Dividend",
        "Date[L]": "29-Oct-08",
        "Div Mkt Lvl ID": "8018469",
        "Div Pay Date": "18-Dec-08",
        "Qual. Income %": "100",
        "Type Marker": "61",
        "#RIC": "SGB.AX",
        "Tax Rate": "30"
      }
    },
    {
      "id": "1f175487-0b36-5152-a2a2-3dbbfbcb2385",
      "source": "TR_Stream",
      "startTime": "2008-10-29 00:00:00.000, +10",
      "endTime": null,
      "dataList": {
        "Period Length": "6",
        "Period Units": "12",
        "Mandatory Voluntary Indicator": "MAN",
        "Div Ex Date": "18-Nov-08",
        "Franking %": "100",
        "Feature": "40",
              "Frequency": "2",
              "Currency": "AUD",
              "Ind. An Amount": "1.82",
              "Qual. Income Elig.": "10",
              "Descr": "Dividend Rinvestment plan in operation.",
              "Div End Date": "30-Sep-08",
              "Reinvestment Plan Available": "Y",
              "Source of Fund": "2",
              "Tax Marker": "2",
              "Payment Status": "APPD",
              "Corporate Action ID": "9035722",
              "Spec. Tax Rules": "7",
              "Div Reg. Date": "24-Nov-08",
              "Div Amt.": "0.94",
              "Payment Type": "CDI",
              "Div Delete Marker": "0",
              "Div Ann. Date": "29-Oct-08",
              "Type": "Dividend",
              "Date[L]": "29-Oct-08",
              "Div Mkt Lvl ID": "8018469",
              "Div Pay Date": "18-Dec-08",
              "Qual. Income %": "100",
              "Type Marker": "61",
              "#RIC": "SGB.AX",
              "Tax Rate": "30"
            }
          }
        ],
        "edges": [
          {
            "ordering": "start(source) >= start(target)",
            "target": "nodes[0]",
            "source": "nodes[1]"
          }
        ]
      }
    }
  ]
}
```

Appendix B: Additional Results for Case Study

This appendix presents supplementary materials to the case study discussed in Chapter 7.

Table B.1. Recorded execution times of one rule used in Section 7.3.1.

All values in the table are time in seconds.

	Dataset 1		Dataset 2		Dataset 3	
	EP-RDR	BP	EP-RDR	BP	EP-RDR	BP
1	0.435	0.314	0.845	0.739	8.027	5.748
2	0.414	0.235	0.895	0.743	7.798	5.334
3	0.380	0.240	0.909	0.703	7.804	5.323
4	0.421	0.228	0.859	0.736	7.560	5.440
5	0.324	0.248	0.933	0.915	8.102	5.712
6	0.335	0.226	0.821	0.753	7.827	5.557
7	0.317	0.245	0.818	0.718	7.667	5.313
8	0.316	0.248	0.889	0.740	7.317	5.555
9	0.366	0.244	0.892	0.751	8.179	5.593
10	0.432	0.246	0.834	0.735	8.007	5.423
avg	0.374	0.247	0.870	0.753	7.829	5.500

Table B.2. Recorded execution times of seven rules used in Section 7.3.1.

All values in the table are time in seconds.

	Dataset 1		Dataset 2		Dataset 3	
	EP-RDR	BP	EP-RDR	BP	EP-RDR	BP
1	2.553	0.681	5.001	1.914	33.656	10.324
2	2.629	0.589	4.805	1.849	34.593	9.943
3	2.563	0.581	4.719	1.852	34.560	10.080
4	2.621	0.572	4.888	1.850	35.724	10.582
5	2.560	0.572	4.782	1.802	36.289	10.079
6	2.700	0.576	4.706	1.778	39.754	10.130
7	2.578	0.576	4.725	1.972	37.592	10.012
8	2.575	0.559	4.955	1.784	32.158	10.075
9	2.559	0.606	4.791	1.766	33.021	10.123
10	2.508	0.599	4.799	1.878	34.052	10.143
avg	2.585	0.591	4.817	1.845	35.140	10.149

Table B.3. RICs of the Dubai companies in Dataset 3 in Section 7.3.1.

AIRA.DU	ASCI.DU	DISB.DU	EMAR.DU	JEEM.DU	SALAMA.DU
AJBNK.DU	CBD.DU	DNIN.DU	ENBD.DU	MASB.DU	SHUA.DU
AMAN.DU	DEYR.DU	DSI.DU	ERC.DU	NBDD.DU	TABR.DU
AMLK.DU	DFM.DU	DTKF.DU	GGIC.DU	NCC.DU	TAML.DU
ARMX.DU	DINC.DU	DU.DU	GNAV.DU	NGIN.DU	TKFE.DU
ARTC.DU	DINV.DU	EBIL.DU	IAIC.DU	OIC.DU	UPRO.DU

Table B.4. RICs of the companies used in Section 7.4.1.

ANZ.AX	AIX.AX	AMP.AX	BEN.AX	BOQ.AX	BXB.AX
BHP.AX	AJL.AX	ASX.AX	BKN.AX	BPT.AX	BIL.AX
AGO.AX	AMC.AX	AWE.AX	BLD.AX	BSL.AX	
AIO.AX	APN.AX	AUN.AX	BLY.AX	BWP.AX	