Enabling Event-data Analysis in R
Demonstration

Gert Janssenswillen[1,2], Marijke Swennen[1], Benoît Depaire[1], Mieke Jans[1], and Koen Vanhoof[1]

[1] Hasselt University, Agoralaan Bldg D, 3590 Diepenbeek, Belgium
[2] Research Foundation Flanders (FWO), Egmontstraat5, 1060 Brussels, Belgium
{gert.janssenswillen,marijke.swennen,
benoit.depaire,mieke.jans,koen.vanhoof}@uhasselt.be

Abstract This demonstration introduces a newly developed R-package, named *edeaR - Exploratory and Descriptive Event-based Data Analysis in R*. The package aims to handle, describe and select event data using a set of predefined methods. Consequently, it enables the vast collection of data manipulation and analysis functionalities within R to be used for event data.

Keywords: Data Analysis, R, Event data

1 Introduction

Due to the enormous growth of event data in the last decades, organisations are dealing with the challenge of extracting useful knowledge from it, and exploiting it to gain competitive advantages. Process mining provides ways to reach this goal, by getting a better understanding of processes and improving them [1].

In order to analyse event data, the process mining framework ProM [6] has been growing rapidly over the last decade, and provides an enormous amount of methods to discover and analyse processes. However, most attention has been given to the discovery and analysis of process models. This approach can be successful when the resulting models are a good representation of the event log, i.e. having a relatively high fitness and precision. However, realistic event logs are often complex and, without any preprocessing of the data, reliable models to describe them are difficult to be found. Exploratory analysis of event data itself is only limited supported by ProM. Therefore, there is a need for an interactive tool which allows direct access to the event log to analyse it. Commercial tools, such as Disco[1] or QPR Process Analyzer[2], partly fill up this gap. However, in general, there does not exist a lot flexibility.

The objective of this demonstration is to introduce a newly developed R-package, named *edeaR - Exploratory and Descriptive Event-based Data Analysis in R*. R is a statistical programming environment which is used for data analysis

[1] http://fluxicon.com/disco/
[2] http://www.qpr.com

and visualisation [4], and has gained a lot of importance over the last few years. R comes with an extensive package eco-system, which results in a limitless range of data analysis applications. The main goal of the package is to provide a grammar to handle event data in R. It includes functionalities to import and export event data from and to XES-files, making it combinable with other process mining tools. It can be used to describe the data of an event log using a set of predefined methods, as well as to perform data selection. But primarily, it enables event data analysts to use the vast range of functionalities in R on event data. Furthermore, it invites data scientists from other disciplines who are familiar with R to look into event data analysis. The R-package edeaR is available on *github*, from where it can be easily installed.[3]

The next section will formally define the terminology used by the R-package. Section 3 will introduce the event log class. Section 4 will discuss methods to describe and explore event-based data. Data selection will be illustrated in Section 5. Section 6 provides a brief application while section 7 will conclude the paper.

2 Preliminaries

In this section, we will define the terminology used by the package to handle event data. An event log is expected to be in the format as illustrated in Table 1. Each record of the event log contains one event. Events are atomic registrations of actions in a process. Each event should contain a timestamp, an activity identifier, a lifecycle transaction and an activity instance identifier. Next to these, events can have multiple other attributes, depending on the context. Attributes at the level of cases should preferable be stored separately for the event log, in a second data.frame, which can be linked to the event log by means of a common case identifier column in both data tables.

A process exists of several activities. Each activity can have one or more activity instances. An activity instance refers to one specific instance of an activity, related to one specific case. Within one case, an activity might occur more than once, leading to the existence of multiple activity instances. Each activity instance consists of one or more events, recording the different stages in the transactional life-cycle of the activity instance. In Table 1, each activity instance has two events: a start event and a complete event. A case refers to one individual process instance. It is a finite sequence of activity instances. Multiple cases might share the same sequence of activities, which is referred to as a trace.

3 `Eventlog` object

Within the context of `R` an event log can be represented as a `data.frame`. However, in order to analyse an event log, it is essential to know which elements

[3] The package is available at `https://github.com/gertjanssenswillen/edeaR`. Using the R-package `devtools`, it can be installed using the following command: `install_github("gertjanssenswillen/edear", build_vignettes = T)`.

Table 1. Example event log

Event	Case	Timestamp	Activity instance	Activity	Lifecycle transition
1	0	28/09/2015 16:51:34	1	A	start
2	0	28/09/2015 16:53:01	1	A	complete
3	0	28/09/2015 16:53:51	3	D	start
4	0	28/09/2015 16:54:03	2	C	start
5	0	28/09/2015 16:56:33	2	C	complete
6	0	28/09/2015 16:56:29	3	D	complete
9	1	28/09/2015 16:51:53	5	A	start
10	1	28/09/2015 16:53:18	5	A	complete
11	1	28/09/2015 16:53:53	7	D	start
12	1	28/09/2015 16:55:06	7	D	complete

of the data.frame identify the different cases, activities, activity instances, etc. The `eventlog` class, implemented by the R-package, will do just this. As one can observe, there are five mandatory fields needed for a data.frame to act as an event log. The names of these field will be stored as attributes of an `eventlog` object, which are described in Table 2.

Table 2. Attributes of an `eventlog`-object

Attribute	Description
case_id	The case to which the event belongs.
activity_id	The activity the event refers to.
activity_instance_id	The activity instance the event belongs to.
lifecycle_id	The stage in the transactional life cylce.
timestamp	The timestamp of the event.

For an object of the class `eventlog`, the values of the different arguments can be easily obtained and changed on the go, allowing the user to easily switch between different views on the same event log. The values for the lifecycle transaction can be any of the transitions in the transactional life cycle specified in the XES-standard[3]. Next to these mandatory attributes, each event can have a number of other attributes. In order to avoid data redundancy, case attributes should preferably be stored in a second data.frame, which can be linked to the event log by the case identifier column.

There are two ways to create `eventlog` objects. Firstly, the package comes with import functions to read both event logs and case attributes from XES-files. Secondly, event logs stored as csv-files can be imported using existing R-functionalities, and if needed be preprocessed *manually* to the desired format. For an example on how to preprocess event data accordingly, we refer to the vignettes available in the package.

Objects of the class `eventlog` will be handled differently compared to normal data.frame objects by the generic R-functions `print` and `summary`. When these functions are called upon an event log, the nature of the object will be reported, as well as some relevant information to describe the event log.

4 Descriptive statistics

Several functions have been developed to compute descriptive summary statistics about an event log. These functions are based on the work in [5]. The metrics describe an event log concerning two dimensions - time and structuredness - and are available at different levels of granularity: event log, trace, case and activity. Table 3 gives an overview of the different metrics and the levels of analysis at which they can be used. For the precise definition of all the metrics, we refer to [5] or the package documentation.

Table 3. Descriptive metrics

		Log	Trace	Case	Activity
Time	Processing time	X	X	X	X
	Throughput time	X	X	X	
Structuredness	**Variance**				
	Activity presence in traces				X
	Activity type frequency		X	X	X
	Start activities	X		X	X
	End activities	X		X	X
	Trace length	X	X	X	
	Trace coverage	X	X		
	Number of traces	X			
	Repetitions				
	Number of repetitions	X	X	X	X
	Self-loops				
	Size of self-loops	X	X	X	X
	Number of self-loops	X	X	X	X
	Number of traces with self-loop	X			

Each metric can be calculated by providing the event log and the desired analysis level. Depending on the metric and the level of analysis, the output can be of two types: either a table containing absolute and relative values concerning a certain phenomenon, or a table with measures of locality and spread. For instance, the trace length at trace level will contain for each trace, the length of the trace in absolute terms, i.e. the number of activity instances, and in relative terms, i.e. compared to the overall average trace length of the event log. However,

at a log level, the metric will return descriptive measures of the trace length, e.g. average, median, min, max, quantiles, etc.

5 Data selection

Data selection can be performed in several ways. Firstly, the event log can be subsetted as any regular data.frame in R, e.g. selecting on certain attribute values. Additionally, the developed R-package supports several predefined functions which are able to select data based on specific features of cases or activity instances. Table 4 lists all the available filters, together with the different options and the corresponding output. Note that the workings of some filters differ depending on the arguments provided by the user. Furthermore, each filter contains the argument *reverse*, allowing the user to invert the selection. For instance, selecting the 10% most rare activities instead of the 90% most common.

Thirdly, one can define custom filtering methods which may be based on the provided metrics. For instance, one might be interested to select only the cases in which a self-loop never occurs. Furthermore, custom features of cases can be developed, computed and used for data filtering. This clearly shows the flexibility and extendability of the approach. Finally, also case attributes, stored in a second data table, can be used for filtering. Any subsetting on the case attributes can be done, and subsequently used for event log filtering based on the case identifier.

6 Application

In this section, a brief illustration on the use of the package is given. The analyses are conducted using data from the BPI Challenge 2014 [2]. In particular, the *Incident details* are used as case attributes, while the *Activity log for incidents* is used as the event log itself. A sample of both datasets describing a total of 4000 cases is included in the package. In order to know the nature of the event log, the *mapping* function can be called, as is shown in Code 1.1. This function prints out the different identifiers connected to the event log.

```
1 BPIC14_incident_log %>% mapping
```

The result is printed out as follows

```
1 Case identifier:              incident_id
2 Activity identifier:          incident_activity_type
3 Activity instance identifier: incident_activity_number
4 Timestamp:                    data_stamp
5 Lifecycle transition:         lifecycle
```

Code 1.1. Mapping

Table 4. Filtering methods

Filter	Options	Description
Activity frequency	Percentile x	Select the x% most frequent activities
Endpoints	{Start activities, end activities}	Select cases with the provided start and/or end activities
	Percentile x	Select cases with the x% most common start and end activities
Throughput time	Interval $[a, b]$	Select cases with a throughput time in the interval $[a, b]$
	Percentile x	Select the x% shortest cases
Trace frequency	Interval $[a, b]$	Select cases of which the corresponding trace has a frequency in the interval $[a, b]$
	Percentile x	Select the x% of cases which correspond to the most freqeunt trace
Trace length	Interval $[a, b]$	Select cases with a trace length in the interval $[a, b]$
	Percentile x	Select the x% most shortest cases
Trim	Start activities, end activities}	Restrict all cases to the first activity instance of a start activity until the last activity instance of an end activity
Precedence	{Antecedents, consequents, directly follows, each}	Select cases where each of the antecedents is at least once directly followed by each of the consequents
	{Antecedents, consequents, eventually follows, each}	Select cases where each of the antecedents is eventually followed by each of the consequents
	{Antecedents, consequents, directly follows, one of}	Select cases where at least one of the antecedents is directly followed by one of the consequents
	{Antecedents, consequents, eventually follows, one of}	Select cases where at least one of the antecedents is eventually followed by one of the consequents
Time period	{Intersecting, $[a, b]$}	Select cases with at least one activity instance in the time interval $[a, b]$
	{Contained, $[a, b]$}	Select cases where all activity instances are in the time interval $[a, b]$
	{Start, $[a, b]$}	Select cases which started in the time interval $[a, b]$
	{Complete, $[a, b]$}	Select cases which completed in the time interval $[a, b]$
	{Trim, $[a, b]$}	Select activity instances in the time interval $[a, b]$

Note that, for readability, the piping symbol %>% as defined by the R-package *dplyr* is used. This symbol inserts the output of its *predecessor* as the first argument of its successor. Several basic characteristics of the event log, such as the number of cases or the number of activities, can be computed with simple auxiliary functions, like *n_cases* and *n_activities*, respectively. An brief overview of characteristics can be printed using the generic R-function *summary*, which will recognize the data as an event log, as depicted in Code 1.2.

```
BPIC14_incident_log %>% summary
```

The result is printed out as follows

```
Number of events:      39911
Number of cases:       4000
Number of traces:      2430
Number of activities:  36
Average trace length:  9.97775

Start eventlog:        2013-02-06 13:07:45
End eventlog:          2014-04-02 14:04:51
```

Code 1.2. Event log summary

The following command will compute the number of self-loops in the log, and output the result as log-level summary statistics. The output in Code 1.3 shows that while at least 50% of the cases do not include self-loops (i.e. the median is equal to zero), there exist at least one case in which the number of self-loops peaks at 34. To drill down in this result, the number of self-loops can be calculated at the case level and visualised, e.g. using ggplot2, as in Code 1.4. The resulting graph is displayed in Figure 1[4].

```
BPIC14_incident_log %>% number_of_selfloops("log")
```

The result is printed out as follows

```
   min   q1 median  mean   q3   max  st_dev  iqr
     0    0      0 0.8444    1    34 1.459654    1
```

Code 1.3. Number of self-loops

```
BPIC14_incident_log %>% number_of_selfloops("case") %>% ggplot() +
    geom_histogram(aes(absolute)))
```

Code 1.4. Visualizing the occurrence of self-loops

This output can then be easily compared with other descriptives. For example, Figure 2 shows the relationship between the number of self-loops in a

[4] Note that some formatting commands have been left out in Code 1.4, as they are out of the scope of this demonstration.

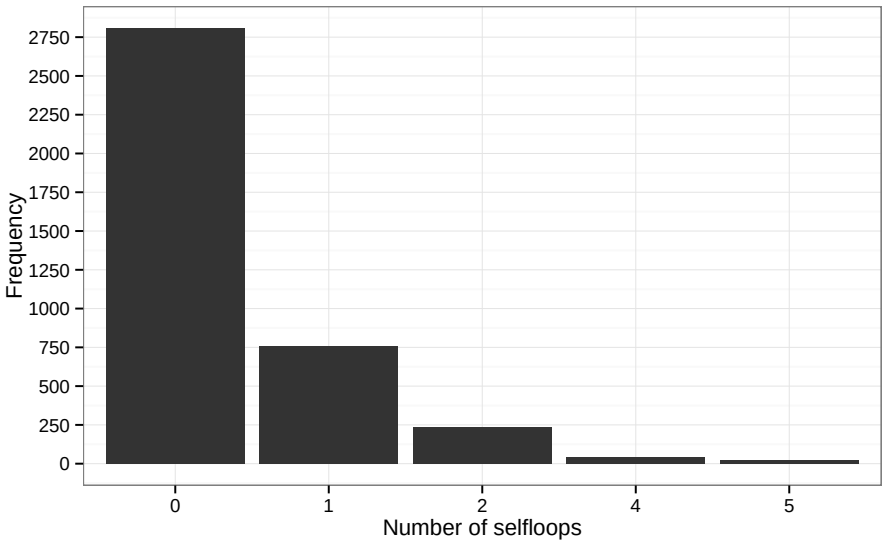

Figure 1. Distribution of number of self-loops per case

case and the throughput time of a case. It can be seen that the occurrence of self-loops has a negative impact on the performance, as could be expected. The analysis of self-loops at the level of activities furthermore indicated that the activities *Assignment* and *Operator Update* are most problematic in this respect[5].

Next to visualising the descriptives, they can be used as input for other analyses, such as clustering. For instance, one might be interested in describing which cases score worse on certain performance characteristics, such as throughput time or the number of repetitions, often signalling rework. As an illustration, a kmeans clustering was conducted on the bases of 5 metrics on case-level: the throughput time, the trace length, the number of repetitions, the number of self-loops, and the average activity frequency. Upon inspection of the SSE for different number of clusters, a cluster design with three different clusters was chosen. The largest cluster contained 92% of the cases which all scored good at the different metrics. For example, throughput time was less that 35 days for all cases and no more than 4 self-loops occurred. A second cluster, containing 6.9% of the cases, included cases which scored somewhat worse, though not exceptionally bad. For these cases, the maximum throughput time was still limited to 86 days. The third cluster contained about 1 % percent of exceptionally bad behaviour. For these cases, throughput time ranged between 90 and 363 days. For simplicity, these clusters were given the names *High*, *Medium* and *Low*, respectively, refering to the performance level of the cases they contain. Subsequently,

[5] Full results were omitted due to limited space.

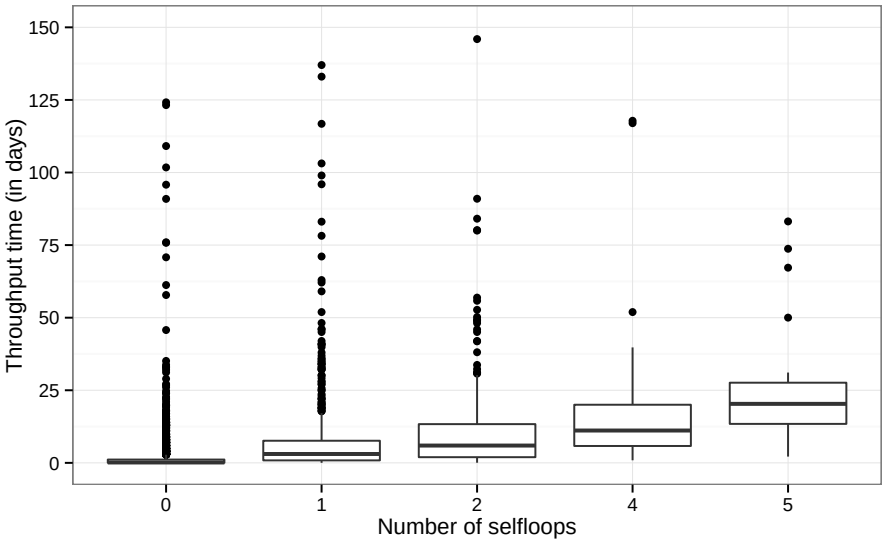

Figure 2. Throughput related to the number of self-loops

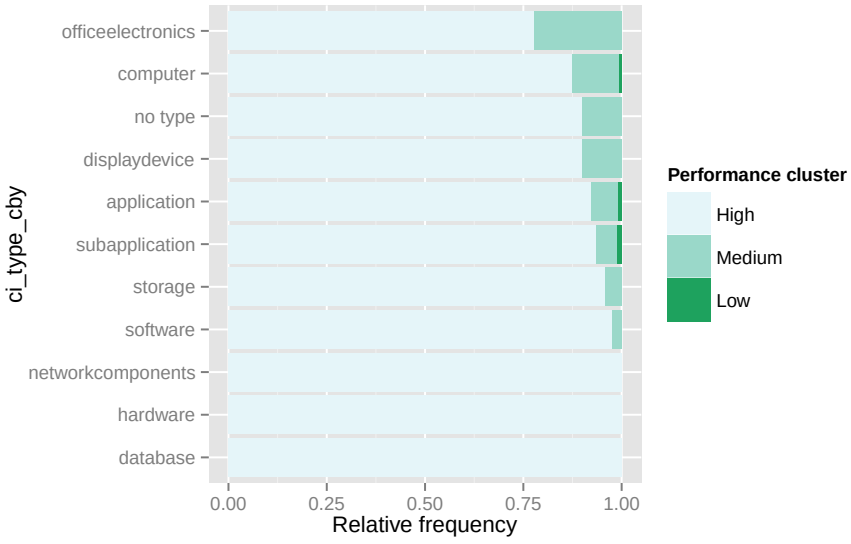

Figure 3. Performance clusters related to ci_type_cby

it is very straightforward to connect these clusters to different case attributes, in order to explain which case characteristics are related to cases which score less good on the selected criteria. For example, Figure 3 shows how the clusters are represented differently among different subtypes of configuration items (CI). Although the example is rather illustrative, it highlights how the vast amount of statistical and data analysis functionality available in R and its package ecosystem can be used in the analysis of event data.

7 Conclusions

This paper briefly introduced a newly developed R-package which allows users to handle event data in R. In order to do so, the `eventlog` object class was presented, which is compatible with the XES-standard. Furthermore, a collection of functions was defined to describe event data and to perform event data selection.

Introducing event data in R enables us to use a vast amount of functionalities available in R, ranging from statistical analysis over data visualization to data mining. For instance, output of several descriptives can be easily visualized. The current package is available on github. A manual describing the different methods is available, as well as several vignettes to illustrate their workings.

References

1. van der Aalst, W.: Process mining: discovery, conformance and enhancement of business processes. Springer, Heidelberg (2011)
2. van Dongen, B.F.: BPI Challenge. Rabobank Nederland. Dataset (2014), `http://dx.doi.org/10.4121/uuid:c3e5d162-0cfd-4bb0-bd82-af5268819c35`
3. Günther, C.W., Verbeek, H.: Xes-standard definition. Tech. rep., Technische Universiteit Eindhoven (2012)
4. Ihaka, R., Gentleman, R.: R: a language for data analysis and graphics. Journal of computational and graphical statistics 5(3), 299–314 (1996)
5. Swennen, M., Janssenswillen, G., Jans, M., Depaire, B., Vanhoof, K.: Capturing Process Behavior with Log-Based Process Metrics. In: Submitted to Simpda 2015 (2015)
6. Verbeek, H.M.W., Buijs, J.C.A.M., Van Dongen, B.F., Van Der Aalst, W.M.P.: Xes, xesame, and prom 6. In: Soffer, P., Proper, E. (eds.) Information Systems Evolution. pp. 60–75. Springer (2011)