Extracting the Warehouse Management event logs

Alessandro Berti (a.berti@pads.rwth-aachen.de)

Date: *04 September 2020*

Premise

This document exemplifies the extraction of process mining logs from SAP related to the Warehouse Management. It is assumed to work with a SAP R/3 installation. Moreover, the actual extraction job is done on a SAP IDES instance that contains simulated data.

The data preprocessing is done in Python using the Pandas library and the PM4Py library `http://www.pm4py.org`.

The target is the extraction of three different event logs:

- The lifecycle of a material.
- The lifecycle of a transfer order.
- The lifecycle of a quant: how materials move in/move out the quant.

Preliminaries

The considered SAP tables to extract the event log are:

- **LTBP**: Transfer requirement item data.
- **LTAP**: Transfer order item data. In particular, it is assumed to have the following fields:
  - **TANUM**: Transfer Order Number.
  - **MATNR**: Material Number.
  - **MEINS**: Base Unit of Measure.
  - **UMREZ**: Numberator for Conversion to Base Units of Measure.
  - **UMREN**: Denominator for conversion to base units of measure.
  - **PQUIT**: Indicator: confirmation complete.
  - **QDATU**: Date of confirmation.
  - **QZEIT**: Time of confirmation.
  - **QNAME**: User name.
  - **VLPLA**: Source Storage Bin.
  - **NLPLA**: Destination Storage Bin.

- **LQUA**: Quants data.

Other tables that are interesting for the warehouse management are:

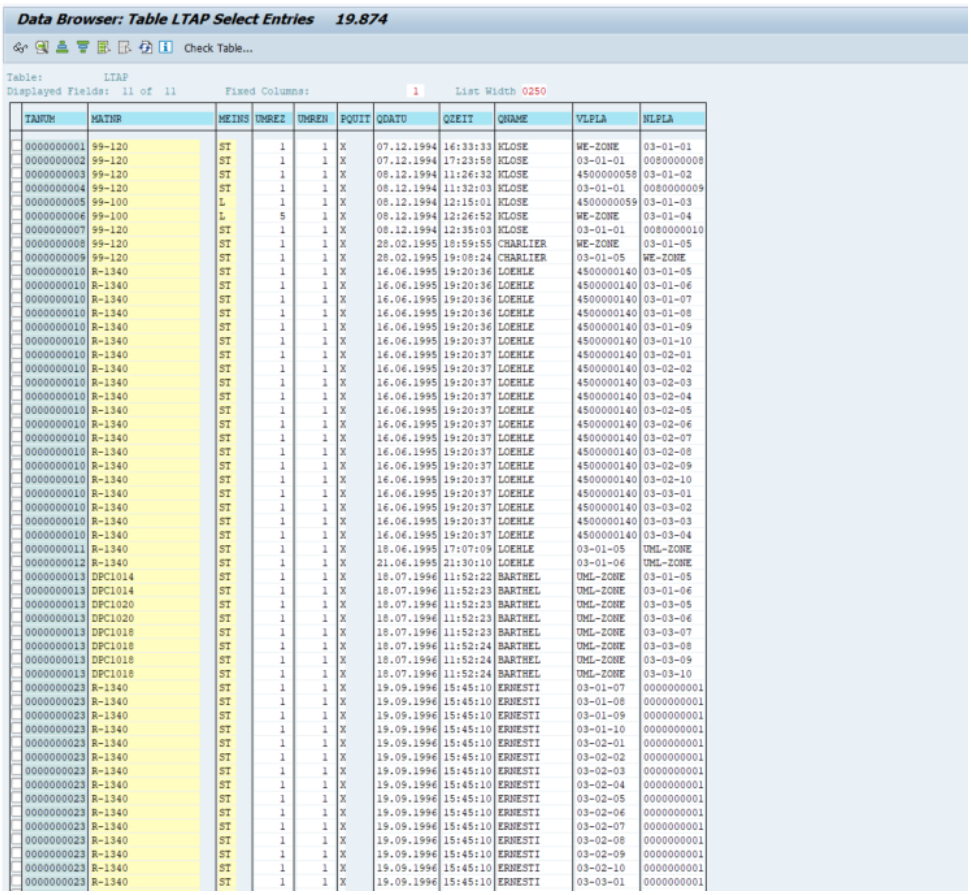
- **LINK**: Inventory document header.
- **LINP**: Inventory document item.

We introduce a small dictionary of terms:

- A *transfer order* is a document used to execute goods movements. It could require approval.
- The *storage bin* is a position in the warehouse where the goods are stored.
- A *quant* is a stock of a material, with similar features, that is stored in one storage bin.

Extraction of the Data

It is assumed that the data is extracted in a CSV-like format from SAP. This can be done directly at the database level. For example purposes, we do it executing the **SE16** transaction in SAP, specifying the table name (**LTBP**, **LTAP** and **LQUA**), and the

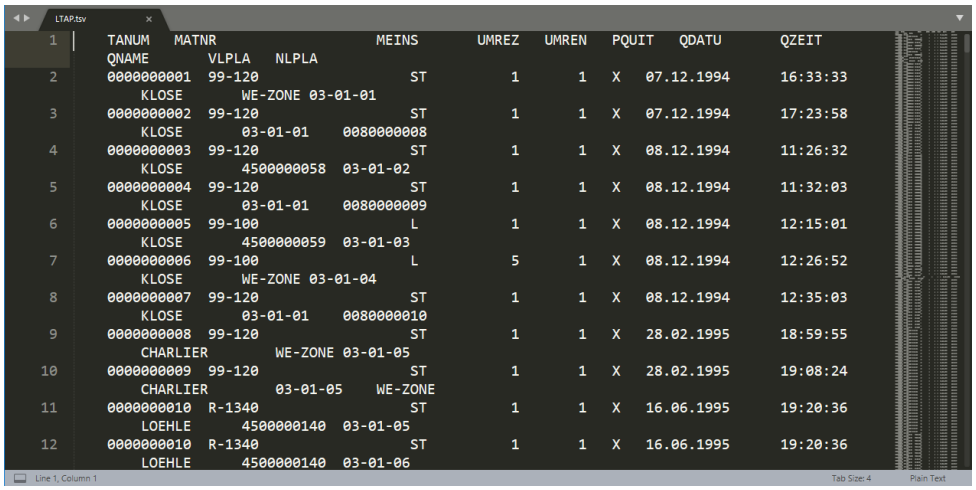

Data Browser: Table LTAP Select Entries 19.874

Check Table...

Table: LTAP
Displayed Fields: 11 of 11 Fixed Columns: 1 List Width 0250

TANUM	MATNR	MEINS	UMREZ	UMREN	PQUIT	QDATU	QZEIT	QNAME	VLPLA	NLPLA
0000000001	99-120	ST	1	1	X	07.12.1994	16:33:33	KLOSE	WE-ZONE	03-01-01
0000000002	99-120	ST	1	1	X	07.12.1994	17:23:58	KLOSE	03-01-01	0080000008
0000000003	99-120	ST	1	1	X	08.12.1994	11:26:32	KLOSE	4500000058	03-01-02
0000000004	99-120	ST	1	1	X	08.12.1994	11:32:03	KLOSE	03-01-01	0080000009
0000000005	99-100	L	1	1	X	08.12.1994	12:15:01	KLOSE	4500000059	03-01-03
0000000006	99-100	L	5	1	X	08.12.1994	12:26:52	KLOSE	WE-ZONE	03-01-04
0000000007	99-120	ST	1	1	X	08.12.1994	12:35:03	KLOSE	03-01-01	0080000010
0000000008	99-120	ST	1	1	X	28.02.1995	18:59:55	CHARLIER	WE-ZONE	03-01-05
0000000009	99-120	ST	1	1	X	28.02.1995	19:08:24	CHARLIER	03-01-05	WE-ZONE
0000000010	R-1340	ST	1	1	X	16.06.1995	19:20:36	LOEHLE	4500000140	03-01-05
0000000010	R-1340	ST	1	1	X	16.06.1995	19:20:36	LOEHLE	4500000140	03-01-06
0000000010	R-1340	ST	1	1	X	16.06.1995	19:20:36	LOEHLE	4500000140	03-01-07
0000000010	R-1340	ST	1	1	X	16.06.1995	19:20:36	LOEHLE	4500000140	03-01-08
0000000010	R-1340	ST	1	1	X	16.06.1995	19:20:36	LOEHLE	4500000140	03-01-09
0000000010	R-1340	ST	1	1	X	16.06.1995	19:20:37	LOEHLE	4500000140	03-01-10
0000000010	R-1340	ST	1	1	X	16.06.1995	19:20:37	LOEHLE	4500000140	03-02-01
0000000010	R-1340	ST	1	1	X	16.06.1995	19:20:37	LOEHLE	4500000140	03-02-02
0000000010	R-1340	ST	1	1	X	16.06.1995	19:20:37	LOEHLE	4500000140	03-02-03
0000000010	R-1340	ST	1	1	X	16.06.1995	19:20:37	LOEHLE	4500000140	03-02-04
0000000010	R-1340	ST	1	1	X	16.06.1995	19:20:37	LOEHLE	4500000140	03-02-05
0000000010	R-1340	ST	1	1	X	16.06.1995	19:20:37	LOEHLE	4500000140	03-02-06
0000000010	R-1340	ST	1	1	X	16.06.1995	19:20:37	LOEHLE	4500000140	03-02-07
0000000010	R-1340	ST	1	1	X	16.06.1995	19:20:37	LOEHLE	4500000140	03-02-08
0000000010	R-1340	ST	1	1	X	16.06.1995	19:20:37	LOEHLE	4500000140	03-02-09
0000000010	R-1340	ST	1	1	X	16.06.1995	19:20:37	LOEHLE	4500000140	03-02-10
0000000010	R-1340	ST	1	1	X	16.06.1995	19:20:37	LOEHLE	4500000140	03-03-01
0000000010	R-1340	ST	1	1	X	16.06.1995	19:20:37	LOEHLE	4500000140	03-03-02
0000000010	R-1340	ST	1	1	X	16.06.1995	19:20:37	LOEHLE	4500000140	03-03-03
0000000010	R-1340	ST	1	1	X	16.06.1995	19:20:37	LOEHLE	4500000140	03-03-04
0000000011	R-1340	ST	1	1	X	18.06.1995	17:07:09	LOEHLE	03-01-05	UML-ZONE
0000000012	R-1340	ST	1	1	X	21.06.1995	21:30:10	LOEHLE	03-01-06	UML-ZONE
0000000013	DPC1014	ST	1	1	X	18.07.1996	11:52:22	BARTHEL	UML-ZONE	03-01-05
0000000013	DPC1014	ST	1	1	X	18.07.1996	11:52:23	BARTHEL	UML-ZONE	03-01-06
0000000013	DPC1020	ST	1	1	X	18.07.1996	11:52:23	BARTHEL	UML-ZONE	03-03-05
0000000013	DPC1020	ST	1	1	X	18.07.1996	11:52:23	BARTHEL	UML-ZONE	03-03-06
0000000013	DPC1018	ST	1	1	X	18.07.1996	11:52:23	BARTHEL	UML-ZONE	03-03-07
0000000013	DPC1018	ST	1	1	X	18.07.1996	11:52:24	BARTHEL	UML-ZONE	03-03-08
0000000013	DPC1018	ST	1	1	X	18.07.1996	11:52:24	BARTHEL	UML-ZONE	03-03-09
0000000013	DPC1018	ST	1	1	X	18.07.1996	11:52:24	BARTHEL	UML-ZONE	03-03-10
0000000023	R-1340	ST	1	1	X	19.09.1996	15:45:10	ERNESTI	03-01-07	0000000001
0000000023	R-1340	ST	1	1	X	19.09.1996	15:45:10	ERNESTI	03-01-08	0000000001
0000000023	R-1340	ST	1	1	X	19.09.1996	15:45:10	ERNESTI	03-01-09	0000000001
0000000023	R-1340	ST	1	1	X	19.09.1996	15:45:10	ERNESTI	03-01-10	0000000001
0000000023	R-1340	ST	1	1	X	19.09.1996	15:45:10	ERNESTI	03-02-01	0000000001
0000000023	R-1340	ST	1	1	X	19.09.1996	15:45:10	ERNESTI	03-02-02	0000000001
0000000023	R-1340	ST	1	1	X	19.09.1996	15:45:10	ERNESTI	03-02-03	0000000001
0000000023	R-1340	ST	1	1	X	19.09.1996	15:45:10	ERNESTI	03-02-04	0000000001
0000000023	R-1340	ST	1	1	X	19.09.1996	15:45:10	ERNESTI	03-02-05	0000000001
0000000023	R-1340	ST	1	1	X	19.09.1996	15:45:10	ERNESTI	03-02-06	0000000001
0000000023	R-1340	ST	1	1	X	19.09.1996	15:45:10	ERNESTI	03-02-07	0000000001
0000000023	R-1340	ST	1	1	X	19.09.1996	15:45:10	ERNESTI	03-02-08	0000000001
0000000023	R-1340	ST	1	1	X	19.09.1996	15:45:10	ERNESTI	03-02-09	0000000001
0000000023	R-1340	ST	1	1	X	19.09.1996	15:45:10	ERNESTI	03-02-10	0000000001
0000000023	R-1340	ST	1	1	X	19.09.1996	15:45:10	ERNESTI	03-03-01	0000000001

Figure 1: Executing the SE16 transaction, it is possible to retrieve the contents of a table (if it is not too big).



LTAP.tsv

	TANUM	MATNR	MEINS	UMREZ	UMREN	PQUIT	QDATU	QZEIT
	QNAME	VLPLA	NLPLA					
2	0000000001	99-120	ST	1	1	X	07.12.1994	16:33:33
	KLOSE	WE-ZONE 03-01-01						
3	0000000002	99-120	ST	1	1	X	07.12.1994	17:23:58
	KLOSE	03-01-01	0080000008					
4	0000000003	99-120	ST	1	1	X	08.12.1994	11:26:32
	KLOSE	4500000058	03-01-02					
5	0000000004	99-120	ST	1	1	X	08.12.1994	11:32:03
	KLOSE	03-01-01	0080000009					
6	0000000005	99-100	L	1	1	X	08.12.1994	12:15:01
	KLOSE	4500000059	03-01-03					
7	0000000006	99-100	L	5	1	X	08.12.1994	12:26:52
	KLOSE	WE-ZONE 03-01-04						
8	0000000007	99-120	ST	1	1	X	08.12.1994	12:35:03
	KLOSE	03-01-01	0080000010					
9	0000000008	99-120	ST	1	1	X	28.02.1995	18:59:55
	CHARLIER	WE-ZONE 03-01-05						
10	0000000009	99-120	ST	1	1	X	28.02.1995	19:08:24
	CHARLIER	03-01-05	WE-ZONE					
11	0000000010	R-1340	ST	1	1	X	16.06.1995	19:20:36
	LOEHLE	4500000140	03-01-05					
12	0000000010	R-1340	ST	1	1	X	16.06.1995	19:20:36
	LOEHLE	4500000140	03-01-06					

Line 1, Column 1 Tab Size: 4 Plain Text

Figure 2: With the Sublime text editor, it is possible to polish the CSV file obtained with the **SE16** transaction, or with a direct database extraction, and prepare it for the following processing phases.

number of records to be retrieved (set it to a big number such as 9999999). A view on a table is contained in Fig. 1.

Then, with a right click on the table contents, select the *Download* voice and save the table as a CSV file (choosing the *Text with Tabs*) option.

The extracted CSV (that is tab-separated) needs to be polished. For doing that, we use the Sublime text editor, opening the CSV file, removing the lines that are not the content or the header, and then we save the file using the encoding UTF-8.

Preprocessing

The following steps of the preprocessing are done in Python 3.x. We suggest a version of Python that is greater or equal to 3.7 for better compatibility with the Python ecosystem. Moreover, we require the installation of the Pandas library (*pip install pandas*) and of the PM4Py library (*pip install pm4py*).

Reading the CSV files

We use the Pandas library, that offers an efficient data structure (a Pandas dataframe) to ingest data from CSV files. We proceed to import the CSV files into Pandas dataframes. To the command *read_csv*, we need to provide:

- The name of the file.
- The separator of the CSV file.
- The default type of the columns: in this case, it is specified that the columns must be imported as strings. This avoids some problem in numerical conversions of columns.

Listing 1: Loading the CSV files.

```
lqua = pd.read_csv("LQUA.tsv", sep="\t", dtype=str)
ltap = pd.read_csv("LTAP.tsv", sep="\t", dtype=str)
ltbp = pd.read_csv("LTBP.tsv", sep="\t", dtype=str)
```

Removing dummy columns

Sometimes, two consecutive columns are separated by more than an occurrence of the separator character. This leads to the insertion of unnamed columns. For the analysis, we want to remove these columns.

Listing 2: Removing anomalous columns.

```
lqua = lqua[[x for x in lqua.columns if not x.startswith("Unnamed")]]
ltap = ltap[[x for x in ltap.columns if not x.startswith("Unnamed")]]
ltbp = ltbp[[x for x in ltbp.columns if not x.startswith("Unnamed")]]
```

Timestamp and Activity Processing

To apply process mining, three basic columns need to be there:

- The case identifier: groups events belonging to the same execution.
- The activity: describe what happens in the event.
- The timestamp: describe when the event happens.

In the following, we want to create some columns in the Pandas dataframes that are the case identifier, the activity and the timestamp. To do this, we will use the PM4Py conventions:

- The case identifier is associated to the **case:concept:name** column.
- The activity is associated to the **concept:name** column.
- The timestamp is associated to the **time:timestamp** column, and has *datetime* format. The timestamp contains both the date and the time.

For the three tables **LQUA**, **LTAP** and **LTBP**, we assume that all the rows of the tables have equal activity (*Create Quant for Material*, *Execute Transfer of Material*, *Request Transfer of Material* respectively). Once the timestamp is composed as union of the date and of the time, a *parsing* operation is needed to convert the string to a datetime object. The format of the conversion is *%d.%m.%Y %H:%M:%S* for all the three tables. If some timestamps are not specified, the *errors='coerce'* option replaces them with null objects, and with the *dropna* command we remove the rows that have an empty timestamp.

Listing 3: Inserting the timestamp and the activity in the LQUA table.

```
lqua["time:timestamp"] = lqua["BDATU"] + "␣" + lqua["BZEIT"]
lqua["time:timestamp"] = pd.to_datetime(lqua["time:timestamp"],
format="%d.%m.%Y␣%H:%M:%S", errors="coerce")
lqua = lqua.dropna(subset=["time:timestamp"])
lqua["concept:name"] = "Create␣Quant␣for␣Material"
```

For the LTAP table, we have a *QNAME* column that is the originator performing the event (that is the **org:resource** in PM4Py).

Listing 4: Inserting the timestamp and the activity in the LTAP table.

```
ltap["time:timestamp"] = ltap["QDATU"] + "␣" + ltap["QZEIT"]
ltap["time:timestamp"] = pd.to_datetime(ltap["time:timestamp"],
format="%d.%m.%Y␣%H:%M:%S", errors="coerce")
ltap = ltap.dropna(subset=["time:timestamp"])
ltap["concept:name"] = "Execute␣Transfer␣of␣Material"
ltap = ltap.rename(columns={"QNAME": "org:resource"})
```

Listing 5: Inserting the timestamp and the activity in the LTBP table.

```
ltbp["time:timestamp"] = ltbp["TDATU"] + "␣" + ltbp["TZEIT"]
ltbp["time:timestamp"] = pd.to_datetime(ltbp["time:timestamp"],
format="%d.%m.%Y␣%H:%M:%S", errors="coerce")
ltbp = ltbp.dropna(subset=["time:timestamp"])
ltbp["concept:name"] = "Request␣Transfer␣of␣Material"
```

Log Extraction - Lifecycle of a Material

In this case, we choose the **MATNR** (material number) column as case identifier. We do the assignation for all the three dataframes. Then, we concatenate the three dataframes into a single dataframe, and we sort the values by timestamp.

An event log is eventually obtained by converting the dataframe to an event log and calling the PM4Py extraction.

Listing 6: Inserting MATNR as case identifier, concatenating the dataframes and calling the PM4Py exporter.

```
lqua["case:concept:name"] = lqua["MATNR"]
ltbp["case:concept:name"] = ltbp["MATNR"]
ltap["case:concept:name"] = ltap["MATNR"]

dataframe = pd.concat([lqua, ltbp, ltap])
dataframe = dataframe.sort_values("time:timestamp")

log = xes_converter.apply(dataframe,
parameters={"stream_postprocessing": True})
xes_exporter.apply(log, "MATERIAL_WAREHOUSE.xes")
```

The resulting log contains three activities (*Create Quant for Material*, *Request Transfer of Material* and *Execute Transfer of Material*). A directly-follows graph in the log is represented in Fig. 3. The most frequent arc is the one connecting *Execute Transfer of Material* with itself, since in the lifecycle of a material it is expected that different transfers are executed on the same material.

Interesting analyses on this log:

- For a material, checking the average time between two transfers.
- Understanding which materials need more transfers, and in which situations the request of transfer is mandatory.
- Identifying the time between the request of transfer and the effective transfer (as it is a dead time in the process).

Log Extraction - Lifecycle of a Transfer Order

In this case, we choose the **TANUM** (identifier of the transfer order) column as case identifier. First of all, we make sure that the LTBP dataframe is reduced to all the rows that have non-empty **TANUM** (since they are requests that never came to effective transfers). Then, we merge together the two dataframes related to LTBP and LTAP (in this case, we don't include the table LQUA since the quant information is not relevant for the context).

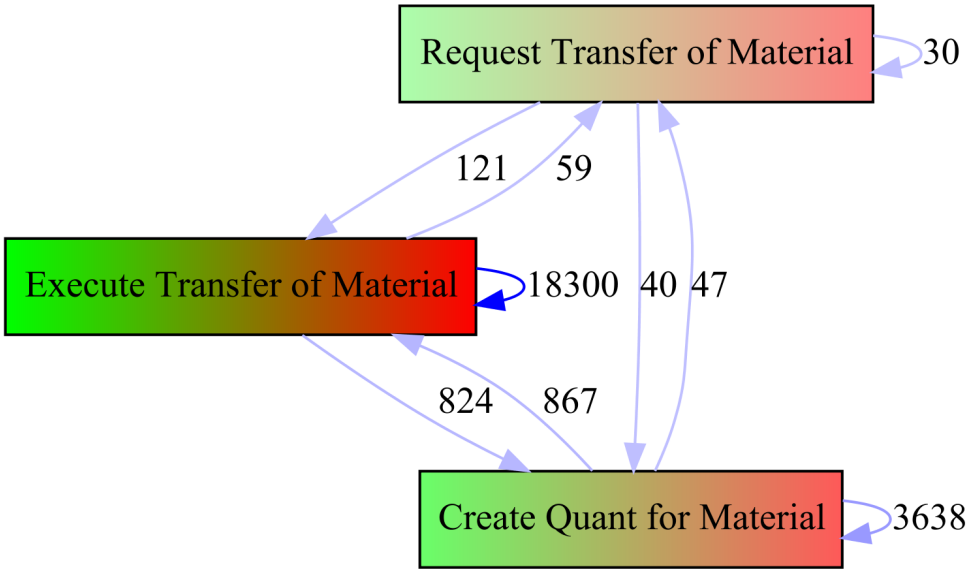


Figure 3: A directly-follows graph that has been extracted on the MATERIAL_WAREHOUSE.xes event log.

Listing 7: Inserting TANUM as case identifier, concatenating the dataframes and calling the PM4Py exporter.

```
ltbp = ltbp.dropna(subset=["TANUM"])

ltap["case:concept:name"] = ltap["TANUM"]
ltbp["case:concept:name"] = ltbp["TANUM"]

dataframe = pd.concat([ltap, ltbp])
dataframe = dataframe.sort_values("time:timestamp")

log = xes_converter.apply(dataframe,
parameters={"stream_postprocessing": True})
xes_exporter.apply(log, "TRANSFER_NUMBER_WAREHOUSE.xes")
```

The resulting log contains two activities, *Request Transfer of Material* and *Execute Transfer of Material*). A directly-follows graph in the log is represented in Fig. 4. The most frequent arc is the one connecting *Execute Transfer of Material* with itself, sience in the same transfer order different materials are transported.

Interesting analyses on this log:

- Checking the throughput times of transfers, from their request to their execution.
- Understanding in which situations the request of transfer is mandatory.

Log Extraction - Lifecycle of a Quant

A *quant* is a stock of a material, with similar features, that is stored in one storage bin. Each row of LQUA describes the features of a quant, and contains the creation/update timestamps.

Moreover, the rows of the LTAP table are describing transfers of materials from a quant to the other.

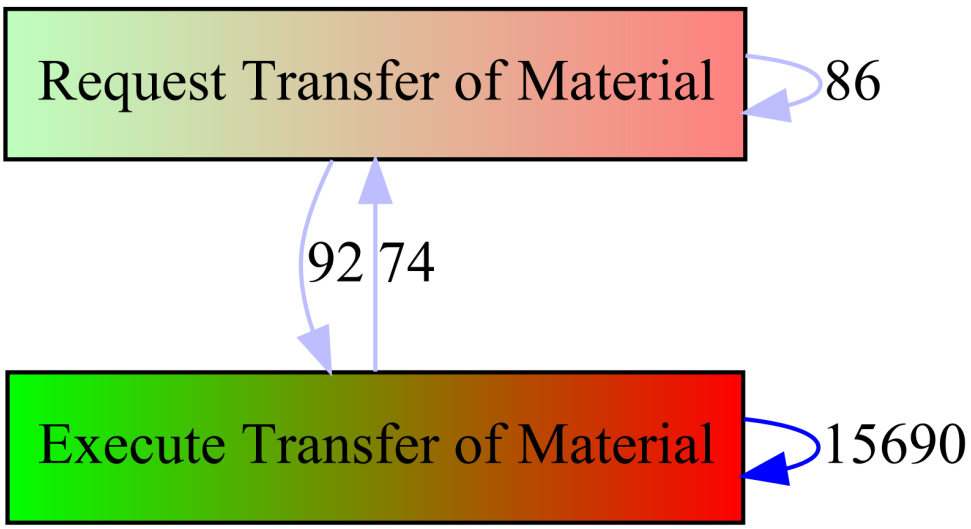


Figure 4: A directly-follows graph that has been extracted on the TRANSFER_NUMBER_WAREHOUSE.xes event log.

In this case, we choose a combination between the identifier of the quant and the material number. The identifier of the quant is:

- For the rows of LQUA, contained in the **LGPLA** column.
- For the rows of LTAP, the source quant is contained in the **VLPLA** column, while the target quant is contained in the **NLPLA** column.

We want that each row of LTAP ends up in two different cases: in the case associated to the source quant, and in the case associated to the target quant. To do so, we replicate twice the dataframe. In the first copy, we replace the activity with *Move from Quant*. In the second copy, we replace the activity with *Move to Quant*.

Then, we use PM4Py to export a XES log.

Listing 8: Inserting the quant as case identifier, concatenating the dataframes and calling the PM4Py exporter.

```
ltap_source = ltap.copy()
ltap_source["concept:name"] = "Move␣from␣Quant"
ltap_source["case:concept:name"] = ltap_source["VLPLA"] + "␣"
+ ltap_source["MATNR"]

ltap_target = ltap.copy()
ltap_target["concept:name"] = "Move␣to␣Quant"
ltap_target["case:concept:name"] = ltap_target["NLPLA"] + "␣"
+ ltap_source["MATNR"]

lqua["case:concept:name"] = lqua["LGPLA"] + "␣" + lqua["MATNR"]
dataframe = pd.concat([lqua, ltap_source, ltap_target])

dataframe = dataframe.dropna(subset=["case:concept:name"])

log = xes_converter.apply(dataframe,
parameters={"stream_postprocessing": True})
xes_exporter.apply(log, "QUANT_MOVEMENT.xes")
```

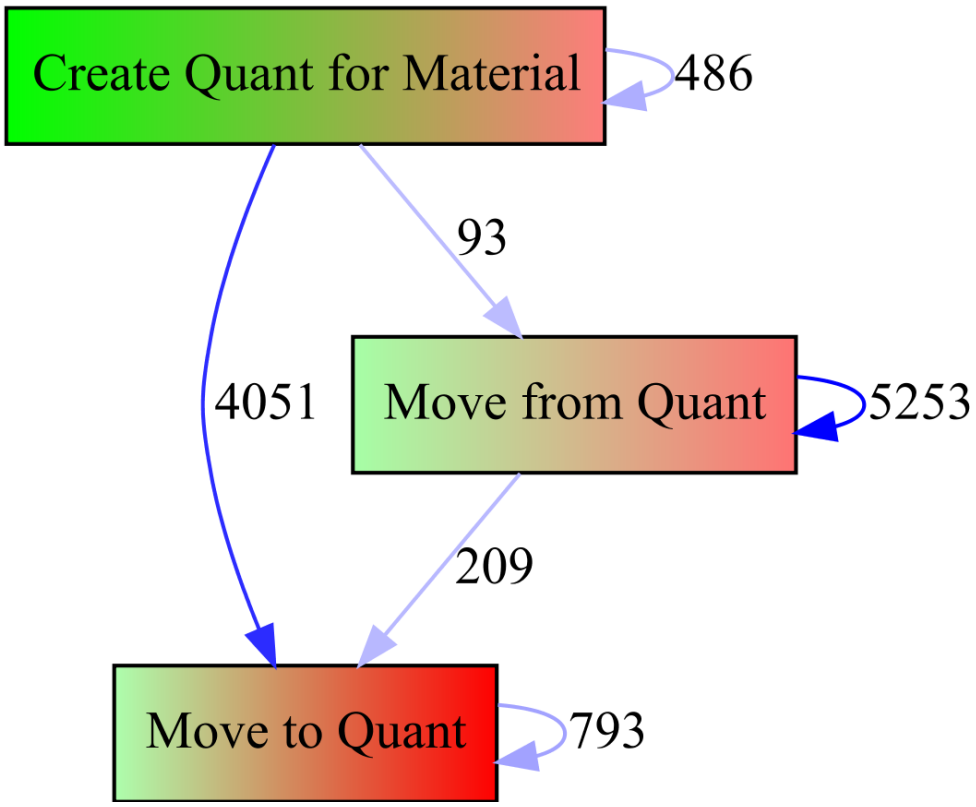


Figure 5: A directly-follows graph that has been extracted on the QUANT_MOVEMENT.xes event log.

The resulting log contains three activities (*Create Quant for Material*, *Request Transfer of Material* and *Execute Transfer of Material*). A directly-follows graph in the log is represented in Fig. 5.

Interesting analyses on this log:

- Checking which quants are more subject to pick up / deposit in the warehouse.
- Checking the average times between moves from quant.
- Checking the average times between moves to a quant.
- Observing the creation of new quants for the material.

Conclusion

In this document, we examined the extraction of event logs for the SAP Warehouse Management.

As future work, it will be nice to consider also the management of the inventary as part of these logs.