Proactive Technology Online(Proton) – Suspicious Account example

The objective of this example is to walk through a simple application in the Proton CEP Engine, and run it with a sample input.

Initialization: After setting up Proton Web Server on Tomcat server, navigate to http://localhost:8080/AuthoringTool/.

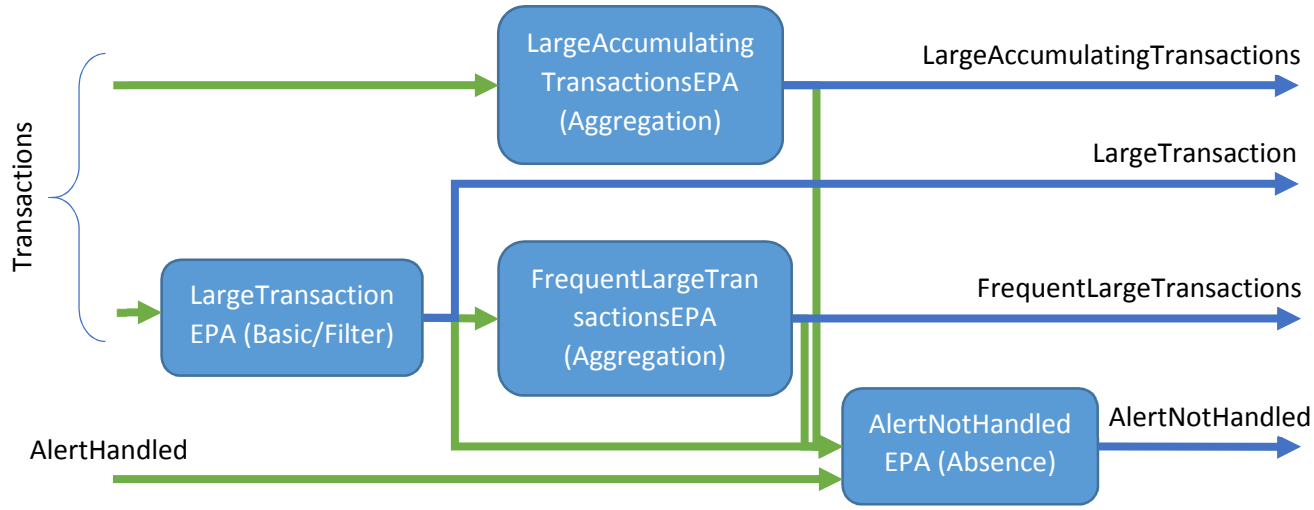
Background story

In this example we are going to create a CEP engine that monitors bank account transactions of customers, and alerts when a suspicious activity is detected. In order to keep this example simple, we will track after the following 3 suspicious activities:

- A single transaction of 1000$ or more (LargeTransaction).
- 3 transactions of 1000$ or more, per customer, in a time window of 24 hours FrequentLargeTransactions).
- Sum of transactions in a time window of 24 hours exceeding 5000$, per customer (LargeAccumulatingTransactions).

Whenever any of these activities is detected, we expect a human operator to handle the alerts created. If any such activity occurs, and a human operator does not handle the alert within 2 hours, we'd like to know about it (AlertNotHandled). In this case, the alert will notify about the CustomerId, and not a specific alert. Furthermore, handling an alert for a customer means handling all alerts that have been produced for a customer.

Event Processing Network (EPN) Diagram



Legend:

Input event

Output event

Events

After creating the project, we will create the input events which the system monitors, and the events which the system derives and outputs.

We will create the following events, each with its own custom attributes:

- **Transaction** – this will be the basic raw event. Each transaction monitored by our system will be of this event type. Custom attributes:

Name	Type	Dimension	Default Value
CustomerId	String	0	
TransactionId	String	0	
Amount	Double	0	0

- **AlertHandled** – this will be our input event indicating that all alerts for a customer with CustomerId have been handled. Custom attributes:

Name	Type	Dimension	Default Value
CustomerId	String	0	

- **LargeTransaction** – this will be our derived event indicating transactions over 1000$ have been made. Custom attributes:

Name	Type	Dimension	Default Value
CustomerId	String	0	
TransactionId	String	0	
Amount	Double	0	0

- **FrequentLargeTransactions** – this will be our derived event indicating that in the past 24 hours, 3 transactions over 1000$ have been made. Custom attributes:

Name	Type	Dimension	Default Value
CustomerId	String	0	
Amount	Double	0	0
TimeWindow	Long	0	0

TimeWindow will indicate the time duration between the first and last transaction in the FrequentLargeTransactions.

- **LargeAccumulatingTransactions** – this will be our derived event indicating that in the past 24 hours, the total sum of transactions' values has reached over 5000$. Custom attributes:

Name	Type	Dimension	Default Value
CustomerId	String	0	
Amount	Double	0	0

- **AlertNotHandled** – this will be our derived event indicating that an alert regarding a certain CustomerId has not been handled by an operator. Custom attributes:

Name	Type	Dimension	Default Value
CustomerId	String	0	

Contexts

Next are contexts. In our example, we will need to use both temporal and segmentation contexts, and combinations of both.

Segmentation Contexts

In our example, we will use the following segmentation context:

- CustomerId – a context used in order to partition events based on CustomerId attribute of events.
  Events included in this context:

Event	Expression
Transaction	Transaction.CustomerId
LargeTransaction	LargeTransaction.CustomerId
FrequentLargeTransactions	FrequentLargeTransactions.CustomerId
LargeAccumulatingTransactions	LargeAccumulatingTransactions.CustomerId
AlertNotHandled	AlertNotHandled.CustomerId
AlertHandled	AlertHandled.CustomerId

Temporal Contexts

In our example, we will use the following temporal contexts:

- Always – a context which is open from startup until shutdown. Will be used for transactions filtering purposes. To set this, check the "At Startup" and "Never Ends" checkboxes.
- LargeTransaction24HoursInterval – a context used in order to start a 24-hours' time interval, starting with the first transaction of over 1000$. This context will be used to detect 3 transactions of 1000$ or more per customer in a 24 hours period.

- Type – temporal interval. The interval starts when a transaction of over 1000$ is received, and ends 24 hours later.
- Event initiators:

Event	Condition	Correlation Policy
LargeTransaction		Add

Note: In this scenario, we detect any 3 consecutive large transactions within 24 hours, meaning that if we have 4 large transactions within a 24 hours window, we will get an alert for the 1st, 2nd, 3rd transactions, and also an alert for the 2nd, 3rd, 4th transactions.

- Relative Time Terminator – Set time to 86400000 ms (which are 24 hours), and type can be either Terminate or Discard.

- Transaction24HoursInterval – a context used in order to start a 24-hours' time interval, starting at the time of a transaction. This context will be used to detect sum of over 5000$ of transactions per customer in a 24 hours period.
  - Type – temporal interval. The interval starts when a transaction is received, and ends 24 hours later.
  - Event initiators:

Event	Condition	Correlation Policy
Transaction		Add

Note: In this scenario, we detect any consecutive transactions that sum up to more than 5000$ within 24 hours, similar to the LargeTransaction24HoursInterval context.

  - Relative Time Terminator: Set time to 86400000 ms (which are 24 hours), and type can be either Terminate or Discard.
- Alert2HoursInterval – a context used in order to start a 2-hours' time interval, starting at the time of an alert. This context will be used to detect an alert that has not been handled 2 hours after it has been produced.
  - Type – temporal interval. The interval starts when an alert is produced, and ends 2 hours later.
  - Event initiators:

Event	Condition	Correlation
LargeTransaction		Add
FrequentLargeTransactions		Add
LargeAccumulatingTransactions		Add

Note: choose correlation policy as Add for all, since there might be a case that 2 alerts are active, but 1 is not handled and the other one is handled at a later time.

  - Relative Time Terminator: Set time to 7200000 ms (which are 2 hours), and type can be either Terminate or Discard.

Composite Contexts

We want to track transactions for each customer. We have segmentation contexts to partition these transactions in the EPAs by CustomerId, and we have temporal contexts in order to track transactions over periods of 24 hours in order to detect suspicious activity. We want to combine

these 2 together, in order to track transactions over 24 hours' time periods, and for each customer. For this, we use Composite Contexts, which combine between several other contexts. We will use the following composite contexts:

- LargeTransaction24HoursComposite – this composite context will be used to detect a transaction of over 1000$, per CustomerId, to initiate the EPA used to detect 3 transactions of over 1000$ per customer in a 24 hours window.
    - Temporal contexts – LargeTransaction24HoursInterval.
    - Segmentation contexts – CustomerId.
- Transaction24HoursComposite – this composite context will be used to detect any transaction, per CustomerId, to initiate the EPA used to detect a sum of transactions of over 5000$ per customer in a 24 hours window.
    - Temporal contexts – Transaction24HoursInterval.
    - Segmentation contexts – CustomerId.
- Alert2HoursComposite – this composite context will be used to detect alerts to initiate the EPA used to detect an alert that has not been handled in a 2 hours interval.
    - Temporal contexts – Alert2HoursInterval.
    - Segmentation contexts – CustomerId.

Event Processing Agents (EPAs)

Now, we are going to define the EPAs which will do the actual monitoring of events to detect suspicious activity. We will use the following EPAs:

- LargeTransactionEPA – an EPA used to detect transactions of over 1000$.
    - Type: Basic (Filter). We want to only filter transactions of over 1000$.
    - Context: Always.
    - Participant events:

Event	Alias	Condition
Transaction	T1	T1.Amount>1000

    - Derivation:
        - LargeTransaction. Attributes' values:

Attribute	Type	Expression
CustomerId	String	T1.CustomerId
TransactionId	String	T1.TransactionId
Amount	Double	T1.Amount

- FrequentLargeTransactionsEPA – an EPA used to detect 3 transactions of over 1000$ per customer, in a 24 hours period.
    - Type: Aggregate.
    - Context: LargeTransaction24HoursComposite.
    - Participant events:

Event	Alias	Condition	Consumption
LargeTransaction	T1		Consume

    - Computed variables:

Name	Aggregation Type	T1 Expr

NumOfTransactions	Count	1
SumOfTransactions	Sum	T1.Amount

We choose T1 Expr to be '1' because we want to count event occurrences, and T1 Expr is the number that is added to Count for each event instance taken into consideration in the EPA.

- Condition: NumOfTransactions==3.
- Evaluation policy: Immediate, so we get an immediate alert when 3 transactions over 1000$ occur.
- Cardinality policy: Single. This is because in the Temporal context we already allowed multiple windows (and thus also EPAs), which will detect further matching sets of the EPA.
- Derivation:
  - FrequentLargeTransactions. Attributes' values:

Attribute	**Type**	**Expression**
CustomerId	String	context.CustomerId
Amount	Double	SumOfTransactions
TimeWindow	Long	context.windowSize

Note: the context object enables access to the segmentation and temporal parameters associated with the context, e.g CustomerId in order to get the CustomerId of the context instance, or windowSize in order to get the current duration of the context.

- LargeAccumulatingTransactionsEPA – an EPA used to detect transactions' sum of over 5000$ per customer, in a 24 hours period.
  - Type: Aggregate.
  - Context: Transaction24HoursComposite.
  - Participant events:

Event	**Alias**	**Condition**	**Consumption**
Transaction	T1		Consume

  - Computed variables:

Name	**Aggregation Type**	**T1 Expr**
SumOfTransactions	Sum	T1.Amount

  - Condition: SumOfTransactions>5000.
  - Evaluation policy: Immediate, so we get an immediate alert when transactions' sum goes over 5000$.
  - Cardinality policy: Single. This is because in the Temporal context we already allowed multiple windows (and thus also EPAs), which will detect further occurrences of the EPA.
  - Derivation:
    - LargeAccumulatingTransactions. Attributes' values:

Attribute	**Type**	**Expression**
CustomerId	String	context.CustomerId
Amount	Double	SumOfTransactions

- AlertNotHandledEPA – an EPA used to detect a situation where alerts for a customer are not handled within 2 hours after they are produced.

Every alert will open a new temporal context, and if no AlertHandled event with the same CustomerId arrives within the 2 hours window, the EPA will generate a derived event for each of the temporal contexts.

- Type: Absence.
- Context: Alert2HoursComposite.
- Participant events:

Event	Alias	Condition
AlertHandled	A1	

- DerivationsList:
  - AlertNotHandled. Attributes' values:

Attribute	Type	Expression
CustomerId	String	context.CustomerId

Producers

In order to input events into Proton, we will use an input file which will contain all of our transactions. We will use a producer of type Timed, so the system will be able to simulate long periods of time (e.g 48 hours) in seconds, and process the results.

- TransactionsInput – our producer, streaming transactions into Proton.
  Type – Timed.
  Attributes:

Name	Value
filename	transactions.txt
formatter	tag
delimiter	;
tagDataSeparator	=

Consumers

In order to see Proton's output, we will direct it to output all of the output events to a text file.

- Output – our consumer.
  Type – file.
  Attributes:

Name	Value
filename	output.txt
formatter	tag
delimiter	;
tagDataSeparator	=
sendingDelay	1000

Received Events:

Name	Condition
LargeTransaction	
FrequentLargeTransactions	
LargeAccumulatingTransactions	
AlertNotHandled	

Scenario

The complete definitions file is attached to this example under the name 'AccountFraud.json'.

In order to simulate the system in a reasonable time, we will not use 2 hours and 24 hours contexts, but instead we will change them to 20 seconds and 40 seconds respectively.

Changes to be made:

- Temporal context Alert2HoursInterval – Relative Time Terminator – 20000.
- Temporal context Transaction24HoursInterval – Relative Time Terminator – 40000.
- Temporal context LargeTransaction24HoursInterval – Relative Time Terminator – 40000.

These changes are applied in a file called 'AccountFraud_simulation.json' in the files attached with this example, which is the file we will want to run for the simulation with the example input and output files.

Both the input 'transactions.txt' and output 'output_expected.txt' files are also attached to this example's files.

The scenario we are going to run is as follows:

Note: Superscripted number above event names describes an Id for the event, only for the use of this table, for reference in other places.

CustomerId	**Raw input event**	**Derived events**
1111	Transaction[1]: • Amount=100 • OccurrenceTime=0	
	Transaction[2]: • Amount=1010 • OccurrenceTime=3000	LargeTransaction[6]
	Transaction[3]: • Amount=1010 • OccurrenceTime=13000	LargeTransaction[7]
	AlertHandled[4]: • OccurrenceTime=14000	
	Transaction[5]: • Amount=2900 • OccurrenceTime=15000	LargeTransaction[8] FrequentLargeTranscations[9] – 6, 7, 8 LargeAccumulatingTransactions[10] – 1, 2, 3, 5
	AlertHandled[4] will cancel the alerts for 6, 7, but we will get multiple alerts for 8, 9, 10	

2222	Transaction[1]: • Amount=1100 OccurrenceTime=2000	LargeTransaction[8]
	AlertHandled[2]: OccurrenceTime=4000	
	Transaction[3]: • Amount=900 OccurrenceTime=5000	
	Transaction[4]: • Amount=900 OccurrenceTime=6000	
	Transaction[5]: • Amount=900 OccurrenceTime=17000	
	Transaction[6]: • Amount=900 OccurrenceTime=18000	
	Transaction[7]: • Amount=900 OccurrenceTime=25000	LargeAccumulatingTransactions[9] – 1, 3, 4, 5, 6
	AlertHandled[2] will cancel alert for 8, but we will get alert for 9	
3333	Transaction[1]: • Amount=900 OccurrenceTime=7000	
	Transaction[2]: • Amount=900 OccurrenceTime=8000	
	Transaction[3]: • Amount=900 OccurrenceTime=9000	
	Transaction[4]: • Amount=900 OccurrenceTime=10000	
	Transaction[5]: • Amount=900 OccurrenceTime=11000	
	Transaction[6]: • Amount=900 OccurrenceTime=50000	*
	* – This will not produce a derived event, since it is more than 40 seconds after the first transaction, so the accumulating sum of over 5000 is in a window bigger than 40 seconds.	