Technische Universiteit
Eindhoven
University of Technology

EDI event logs from the EDImine Correlation plugin

An analysis of existing ProM plugins

Dion Jansen (d.jansen@student.tue.nl)

12/14/2011

0 CONTENTS

1 INTRODUCTION

In this report event logs will be used that are exported from the EDImine Correlation plugin to analyze already existing ProM[1] plugins. The different plugins will be analyzed on how well they perform with the exported data and on usefulness for industry.

The EDImine project is jointly conducted by the Vienna University of Technology and the Eindhoven University of Technology. EDImine is funded by the Vienna Science and Technology Fund (Wiener Wissenschafts-, Forschungs- und Technologiefonds, WWTF - http://www.wwtf.at). The project is about inter-organizational business process mining in the context of Electronic Data Interchange (EDI). [2]

The EDImine Correlation plugin is one of the results from this project and uses Electronic Data Interchange (EDI) messages as input. The plugin lets a user analyze these messages in a clear and understandable way. The plugins main feature is that it allows the user to create traces based on different attributes and finally export these traces to a ProM event log which allows further analysis.

ProM is a process mining framework developed at the Eindhoven University of Technology in the Process Mining Group and is written in JAVA. It already supports a wide range of process mining techniques and supporting functions, one of which is the EDImine Correlation plugin.

The structure of this report will stay close to the structure of the book *Process Mining: Discovery, Conformance and Enhancement of Business Processes* written by prof.dr.ir. W.M.P. van der Aalst, a full professor at the Department of Mathematics & Computer Science at the Eindhoven University of Technology. First the topic of process discovery will be discussed, after that the topic of conformance checking and finally the operational support these techniques provide will be discussed.

This reports structure is as follows. First the EDImine Correlation plugin will be discussed in section 2. After which a brief explanation will follow on EDI messages in section 2.2. Data analysis on the sample data will be performed in section 3. In this section also will be explained what traces will be used throughout this report and will serve as input for the different plugins. Then in section 4 several process discovery plugins will be discussed that are currently present in ProM. The topic of conformance checking will be discussed in section 5, a technique which takes a process and an event log and compares the two. Additional perspectives and operational support will be discussed thereafter in section 6 and section 7 respectively. To conclude this report a description of possible future work and some concluding remarks are given in section 8.

For information and discussion on the plugin and EDI Robert Engel was the person who could be contacted, he was very helpful answering questions and remarks. Prof.dr.ir. W.M.P. van der Aalst helped a lot in shaping this report in what it is now; his book (Aalst, 2011) gave structure and direction.

[1] http://www.promtools.org
[2] Source: http://edimine.ec.tuwien.ac.at/

2 EDIMINE CORRELATION PLUGIN

This section is split into two distinct parts. In the first section (2.1) the main window of the EDImine Correlation plugin will be discussed after which in section 2.2 a small part is reserved to describe what EDI messages are exactly.

2.1 THE PLUGIN

The EDImine Correlation plugin only needs as input a directory with files containing EDI messages, furthermore a custom ontology, a custom Smooks configuration and pre-existing correlation rules can be loaded; as output the plugin produces an event log containing messages as events within a particular trace. The main window of the plugin will be discussed below.

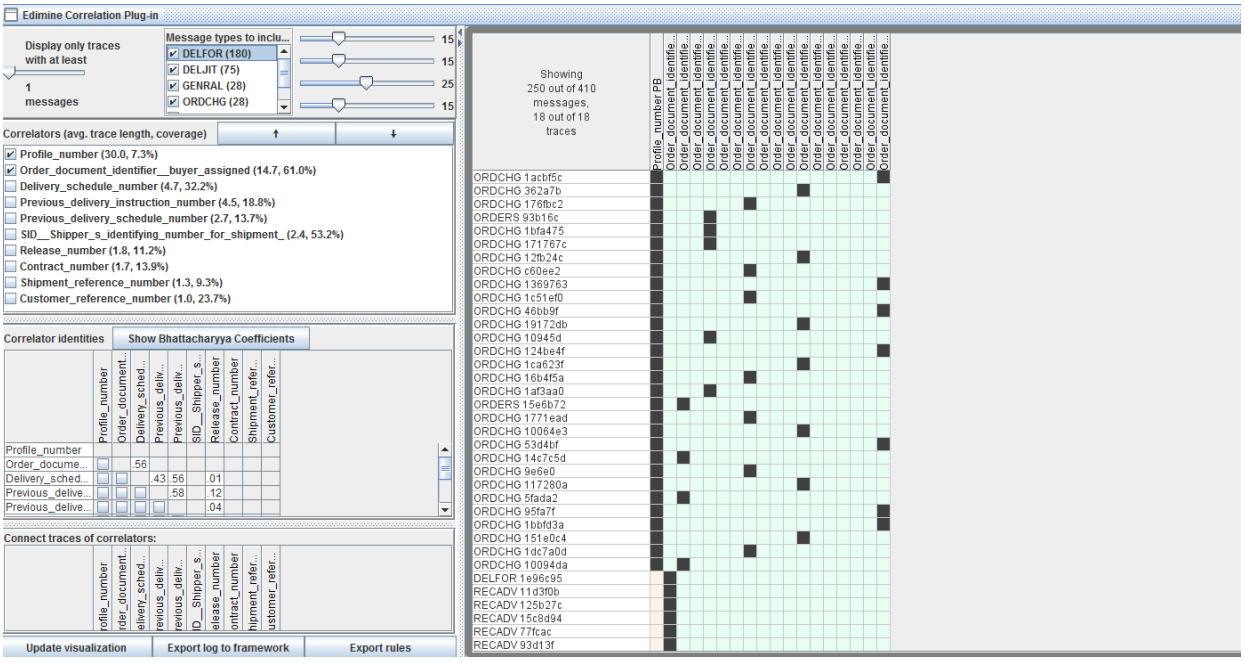

Figure 1 - The main window of the EDImine Correlation plugin. The matrix with the Bhattacharyya distances is used in the sample data analysis in section 3 and is also shown in Table 1.

In the top left there is the possibility to show only traces with a certain amount of messages or to show only messages of a certain type (which are explained in section 3). Below these filters the correlators list contains all the usable attributes which can be used to identify traces.

Below the correlators list resides the correlator identities matrix, which contains Bhattacharyya distances between combinations of two attributes. A high value indicates values of these attributes overlap, which may indicate that these attributes can be used to link messages together so they end up in the same trace eventually. In section 3 on sample data analysis this matrix will be used to construct the traces which in turn will be used throughout this report.

In the bottom left resides another matrix which can be used to connect the traces of different correlators. When a message is related to two or more different attributes, this matrix can be used to connect the traces together that have this message in common.

Last but not least is the window on the right hand side. It contains a table with trace identifiers in the header and message identifiers in the column on the left. The table indicates which messages are used in what trace.

By the end of October the plugin was updated to fix a bug and add an additional feature: the possibility to add artificial start and end events. Prof.dr.ir. W.M.P. van der Aalst pointed out that this functionality was already present in ProM and it seemed valuable to investigate if there were any differences between the two resulting event logs.

When taking a closer look there was one major difference to start with, the EDImine correlation plugin added start events with timestamp "1970-01-01" and end events with timestamp "292278994-08-17", while the plugin already present in ProM has a more elegant solution. The "Add Artificial Events" plugin developed by J. Claes checks each trace for the first and last event and insert an artificial start and end event just before the first event and just after the last event respectively. The difference between the two approaches is shown in Figure 2 and Figure 3.

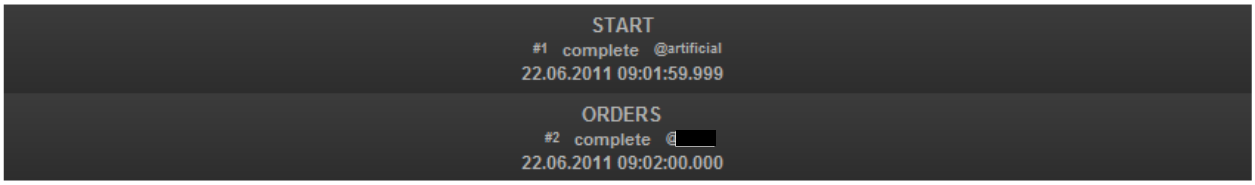

Figure 2 – First two event of some trace after artificial start and end events are added by the "Add Artificial Events" plugin.

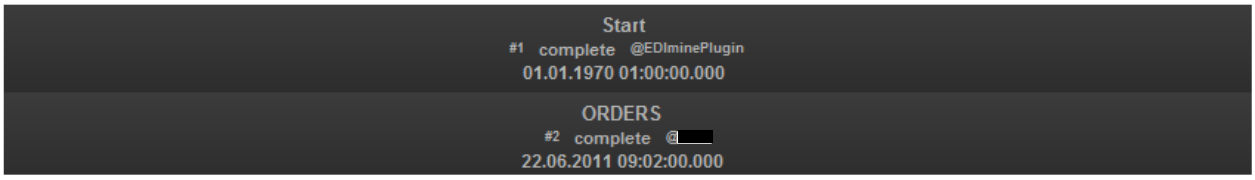

Figure 3 – First two events of some trace where artificial start and end events are added by the EDImine Correlation plugin.

No major difficulties were experience because of this, until the event log and a Fuzzy model were used to animate the process. Apparently the "Animate Event Log in Fuzzy Instance" plugin has some issues with a process that runs for almost 300 million years.

Again, this did not affect other plugins that were analyzed for this report, and therefore no more attention will be paid to it. The existence of the plugin by J. Claes is communicated to Robert Engel[3] as it could be used for the EDImine Correlation plugin; its solution is more elegant and it saves further development time for this particular feature.

[3] This was communicated by e-mail on 15th of November 2011.

2.2 EDI MESSAGES

Electronic Data Interchange (EDI) is a highly structured electronic data communication between parties. Throughout this report an event log will serve as input for the various plugins as will be explained in the next section, but the events contained in this event log are also worth to take a look at.

EDI messages are structured according to guidelines developed internally by companies for their own communication or according to guidelines agreed upon by parties that need to communicate using these messages. Communication in this sense can be between any type of parties, let it be between a service desk and a flight database at the airport or between a buyer and a seller in the steel industry. All messages are in line with a specific standard, for instance EDIFACT[4]. EDIFACT prescribes, for one, that messages should follow a certain structure as shown in Figure 4.

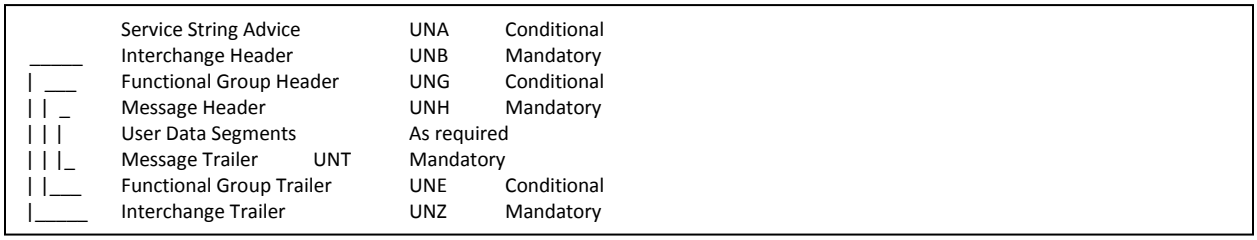

Figure 4 - This figure shows the different parts in an EDI message following the EDIFACT standard (taken from ISO 9735 document). Figure 5 contains a sample message with the different segments.

In Figure 5 an example EDI message is shown; it is not human readable but contains all the information needed by a system to serve its purpose. These messages can contain orders, invoices or even payment information in a compact and highly structured format.

```
UNA:+.? 'UNB+IATB:1+6XPPC+LHPPC+940101:0950+1'UNH+1+PAORES:93:1:IA'MSG+1:45'IFT+3+XYZCOMPANY
AVAILABILITY'ERC+A7V:1:AMD'IFT+3+NO MORE FLIGHTS'ODI'TVL+240493:1000::1220+FRA+JFK+DL+400+C'
PDI++C:3+Y::3+F::1'APD+74C:0:::6++++++6X'TV+240493:1740::2030+JFK+MIA+DL+081+C'PDI++C:4'APD+EM
2:0:1630::6+++++++DA'UNT+13+1'UNZ+1+1'
```

Figure 5 - An example of an EDI message (taken from http://en.wikipedia.org/wiki/EDIFACT). As the EDIFACT standard prescribes (Figure 4) this message contains all mandatory segments (UNB, UNH, UNT and UNZ) and one optional segements (UNA). The segements after the UNH and before the UNT segments is the actual information send by a party, the rest is container data.

Since these messages are used very often within a small time window you can imagine that a lot of data comes with it that can be used within the domain of process mining. The EDImine Correlation plugin is able to transform these unreadable messages into XML and from there based on input from the user is able to construct traces in an event log. How the event log that is used throughout this report came to be is described in the next section.

[4] http://www.unece.org/trade/untdid/welcome.htm

3 DATA ANALYSIS & TRACE SELECTION

In this section the available sample data will be briefly discussed and finally it is describe how the traces that are going to be used were build.

There appear to be 410 messages in the sample EDI data each one of six types. The types are:

- DELFOR – Delivery Schedule Message, which contains delivery schedule instructions for the future.
- DELJIT – Delivery Just in Time Message, which contains delivery schedule instructions for the near future.
- GENRAL – General Purpose Message, which contains textual information.
- ORDERS – Purchase Order Message, which contains initial order data.
- ORDCHG – Purchase Order Change Message, which contains information changing the initial order.
- RECADV – Receiving advice message, which contains information regarding (un)successful delivery.

The EDImine Correlation Plugin determines if attributes are similar using Bhattacharyya distance (Table 1). It shows high values between the following attributes:

- Delivery schedule number
- Previous delivery schedule number
- Order document identifier buyer assigned
- Previous delivery instruction number

The relation between delivery schedule number and previous delivery schedule number seems obvious, how the latter two relate to the first two will require a closer look at the actual data.

The matching values between (previous) delivery schedule number and previous delivery instruction number are assumed to be a coincidence. This overlap in values probably occurred because the sample dataset is relatively small and by accident some identical numbers have been used. In studying traces of messages belonging together the possible relationship between these attributes will not be used.

There is however a relation between delivery schedule number and order document identifier buyer assigned. Both attributes contain non-trivial identifiers 550062644, 550238240, 550238241, 550238446, 550265766 and 550267417, which is probably not a coincidence. Therefore delivery schedule number and order document number will be used as one and the same attribute; this can be done in the plugin by selecting the cell in the correlator identities matrix linking the two attributes as explained in section 2.1.

If we link these correlators together we get 23 traces containing in total 382 messages. The messages which are not used are GENRAL messages, containing only human readable warning texts which do not contain any information and thus cannot be used. If we add up the number of GENRAL messages, 18, to the 283 messages in the traces we again have 410 messages.

Throughout this report these 23 traces will be used in combination with existing ProM plugins to see if ProM can be used for useful analysis for the industry. The events in these traces will be referred to as EDI messages.

Manual analysis showed the event log describes four parties shown in Figure 1. In section 6.2 on social network analysis these parties and how to use them in the future will be discussed.

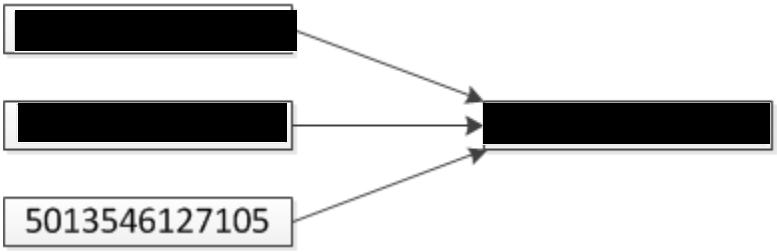

Figure 6 - A graph depicting different communication channels between parties. Good to note is that in the data both [redacted] and [redacted] were mentioned as receiving parties. In this report will be assumed these are one and the same company. The mining of these communication relations will primarily be discussed in section 6.2 on social network analysis.

Since we have to deal with sample data which is incomplete it is important to note how the process is expected to look like. Figure 7 shows this process, and throughout this report it will be referred to as the intuitive process model.

	Profile_number	Order_document_identifier__buyer_assigned	Delivery_schedule_number	Previous_delivery_instruction_number	Previous_delivery_schedule_number	SID__Shipper_s_identifying_number_for_shipment_	Release_number	Contract_number	Shipment_reference_number	Customer_reference_number
Profile_number										
Order_document_identifier__buyer_assigned			0.56							
Delivery_schedule_number				0.43	0.56		0.01			
Previous_delivery_instruction_number					0.58		0.12			
Previous_delivery_schedule_number							0.04			
SID__Shipper_s_identifying_number_for_shipment_										
Release_number										
Contract_number										
Shipment_reference_number										
Customer_reference_number										

Table 1 – This table shows the Bhattacharyya distances between attributes. It indicates possible related attributes which can be used to link orders (and later traces) together. The underlined attributes are assumed to be related as described in this section.

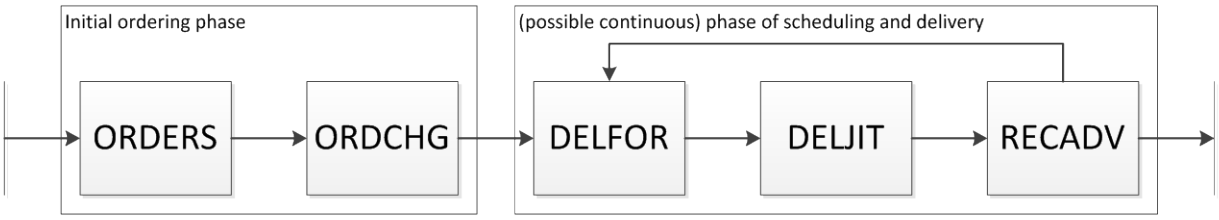

Figure 7 - This figure shows the expected process; it is made up out of two phases. In the first phase is where the actual ordering takes place and in the second phase a possible continuous process takes place where deliveries are scheduled and actually confirmed.

4 PROCESS DISCOVERY

Process mining algorithms try to construct a process model from event logs. In case the event log contains only complete traces and the actual process contains little to no hidden behavior the mined process model will not differ (too much) from the actual process. Since we are dealing here with incomplete sample event log, and cannot assume in the future we would actually get complete event log the process discovery techniques discussed here immediately tested on their tolerance of noise and incompleteness.

In this section the α-algorithm, the ILP mining algorithm, the heuristic mining algorithm, the genetic mining algorithm and the Declare mining algorithm will be discussed. This section on process discovery is finalized with some concluding remarks.

During the writing of this report, and more specifically, during analysis of the heuristic mining results a possible issue was formulated that was noticed while analyzing the 23 traces formulated before. This problem will be discussed in a short subsection (section 4.6) before the concluding remarks.

4.1 A-ALGORITHM

The α-algorithm was one of the first process discovery algorithms that could adequately deal with concurrency (Aalst, et al., 2004) (Aalst, 2011). Aalst (2011) mentions that it is not a very practical technique though, it has problems with noise, incomplete behavior and complex routing constructs.

Unfortunately as mentioned in the previous section, we are dealing with an incomplete event log. The result of using this technique will nevertheless be discussed so this report contains analysis on most of the techniques that can be used within ProM.

Figure 8 shows the result of the alpha miner part of ProM, and you might notice the petri net has an unexpected structure. The algorithm is as said before very sensitive to noise and incompleteness it is probably not the best process discovery technique to use for analyzing EDI communication messages. In Figure 9 the petri net is shown that was mined from the event log containing artificial start and end events.

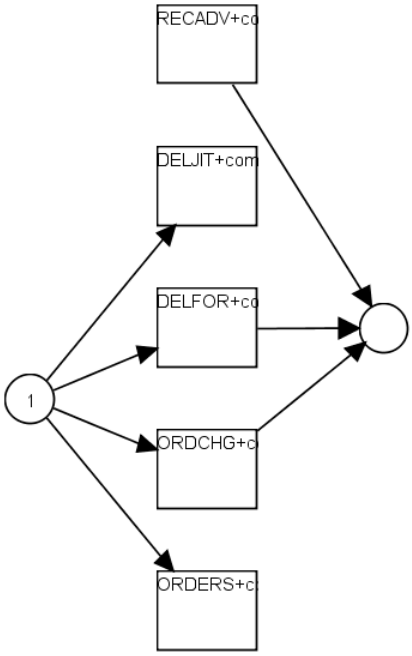

Figure 8 - The petri net that was mined by the α-algorithm without artificial start or end events. Figure 9 shows the petri with the artificial events.

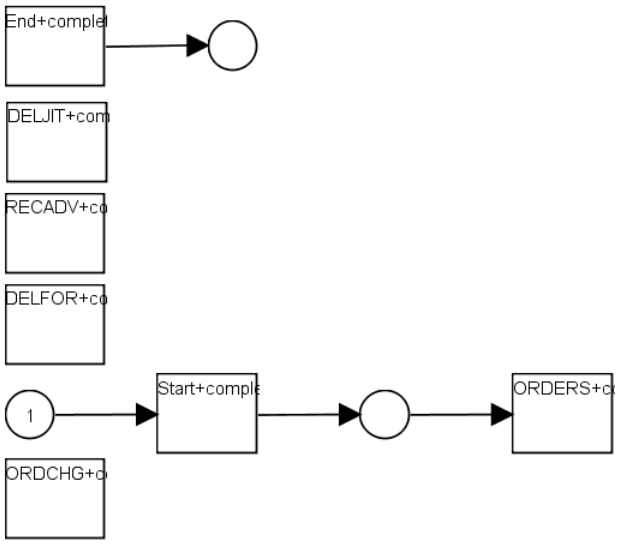

Figure 9 - The petri net mined by the alpha algorithm using an event log which contained artificial start/end events.

Unfortunately the petri net with the artificial start and end events is also not the process model we would expect from this event log. For obvious reasons the α-algorithm is not suitable for mining an intuitive model for the EDI communication.

4.2 ILP MINING

In the search for mining algorithm that outputs a petri net for the process also the ILP mining plugin is discussed which can also mine a petri net from an event log using another algorithm. The plugin has been re-implemented by T. van der Wiel during the writing of his Master's Thesis (Wiel, 2010). Van der Wiel (2010) explains: *the Integer Linear Programming approach … gives a partial ordering over all regions of a language that allows for searching unrelated and minimal regions. This way, the unnecessary addition of places that have little added value to the behavioral restriction of the net is avoided.*

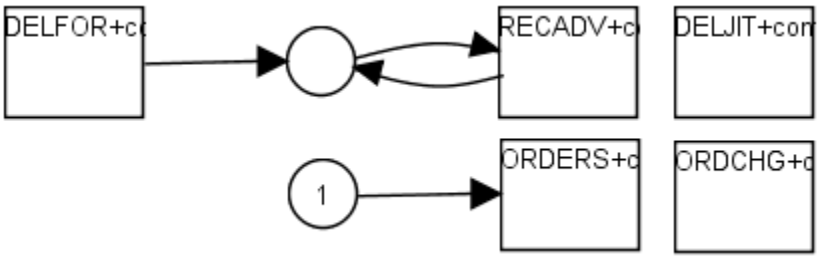

Figure 10 – The petri net mined by the ILP miner from the event log without artificial start and end events. To compare, Figure 11 contains the process model with added artificial start and end events.

As you might notice, Figure 10 still does not show an intuitive process. Adding artificial start and end event, as shown in Figure 11, creates at least a 'good looking' process; however the mined petri net allows for ALL behavior which is also not desirable of course. From this we conclude that the ILP miner cannot be used to capture the EDI behavior that is documented within the event log. As you will see in the next section the heuristic mining algorithm has more luck finding an intuitive model.

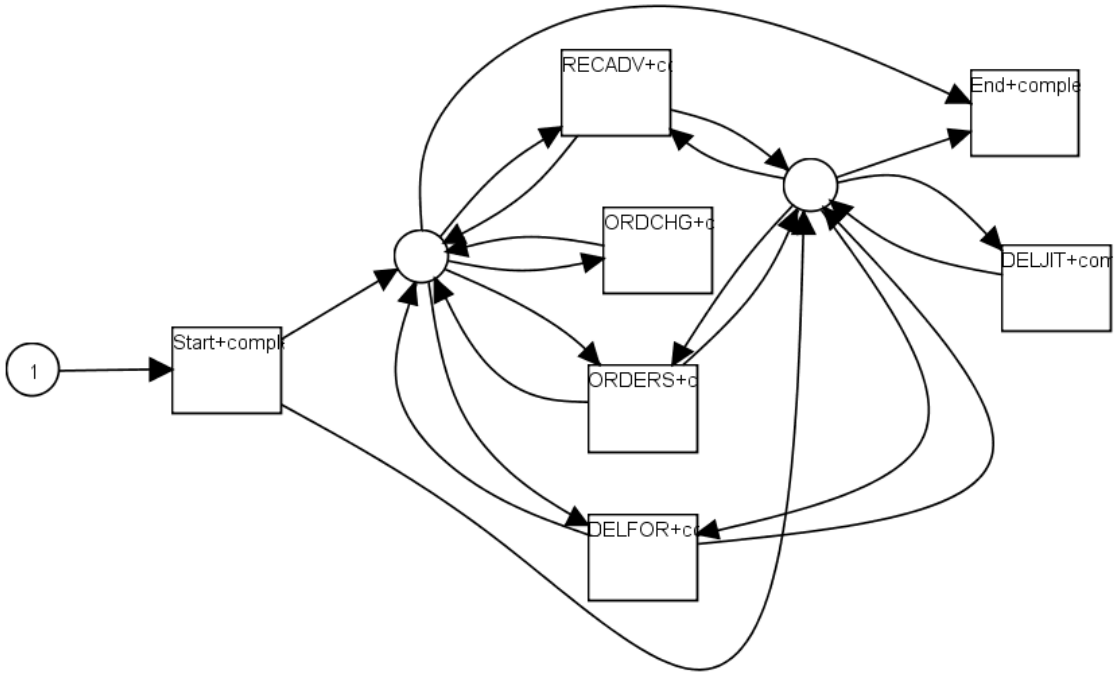

Figure 11 - The petri net mined by the ILP miner from the event log with artificial start and end events. Note that this petri net allows all behavior.

4.3 HEURISTIC MINING

An advantage of heuristic mining over mining a process model with the α-algorithm is that heuristic mining takes frequency of occurrences of traces into account. In this section a heuristic net and a causal net will be mined from the event log. The heuristic net will be converted to a flexible heuristic net which will serve as input for conformance checking which is discussed in section 5.

4.3.1 HEURISTIC NET

Figure 12 shows the heuristic net mined by the heuristic miner in ProM. The advantage over the previously visited process discovery algorithms is that this time it really shows some kind of process occurring.

In real life this should be a fully connected net, but due to the incompleteness of the sample event log this cannot be achieved. It is most likely that the sample data is from a too narrow time window and therefore not a complete process is captured. Intuitively the net should also follow a DELFOR·DELJIT·RECADV order (as shown in the intuitive model in Figure 7) while the mined net shows a DELJIT·DELFOR·RECADV. Robert Engel, the creator of the EDImine correlation plugin, actually found the same heuristic net as shown in Figure 12 and discussed it with the company that supplied them with the data. The company was of opinion that the current communication should follow a DELFOR·DELJIT·RECADV order, while the heuristics net shows something different. In the possible problem definition in section 4.6 will be further explained what might be the problem.

The heuristic net with artificial start and end events is shown in Figure 13; it seems to constrain some behavior. For instance a constraint is that the DELJIT·DELFOR·RECADV loop ends with the DELFOR message. The reason is that there exist traces with only DELFOR messages; this heuristic net takes these traces into account. A plus is that now we have a heuristic net which is fully connected and it has a fitness of 0.884 which is relatively high.

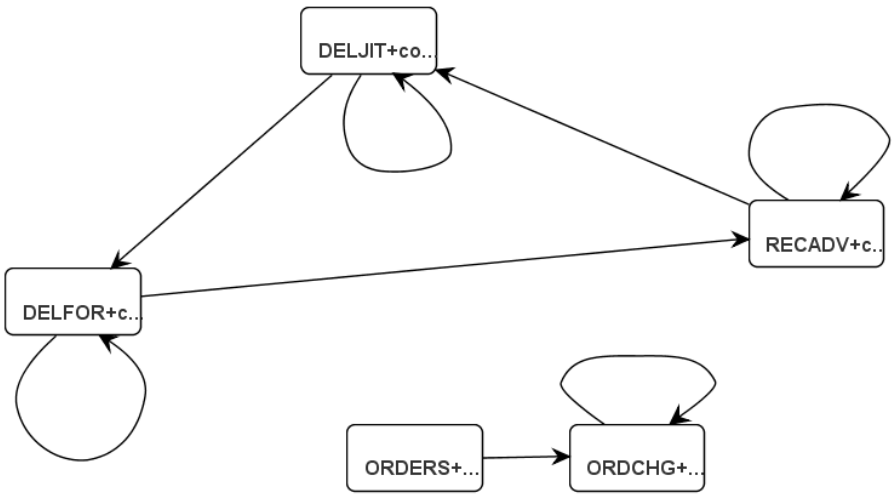

Figure 12 - The heuristic net found by the heuristic miner. Note that in the event log used no artificial start or end events were inserted. It shows the separate phases that were identified in Figure 7, therefor the reason it is not entirely connected can be the limited time window of the event log. Figure 13 shows a complete process after artificial start and end events are added.

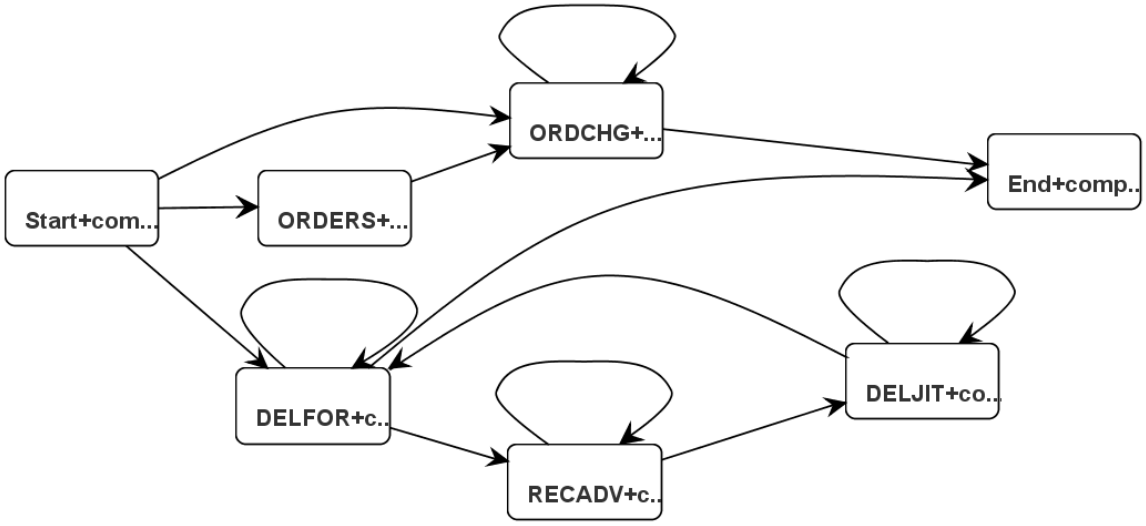

Figure 13 - The heuristic net found by the heuristic miner. This time each trace had artificial start and end events, also shown in the heuristic net. It seems that this net constrains the behavior more than the heuristic net shown in Figure 12, but as in Figure 12 the two different parts of the process are shown.

4.3.2 CAUSAL NET

Causal nets are different from heuristic nets in semantics. In a causal net, nodes represent an activity, or in our case a message and the arcs represent causal dependencies (Aalst, 2011). Each node has a set of ingoing and outgoing arrows, which thus represent the set of possible preceding messages and the set of messages that can follow. Causal nets are required to have a single start and end event, and therefore only the event log will be used with those, in this case artificial, events in place. Figure 14 shows the mined causal net which still shows the two separate phases also identified in Figure 7 showing the intuitive process model. ProM also shows the possible input and output information for each node, an example of output information of a node is shown in Figure 15.

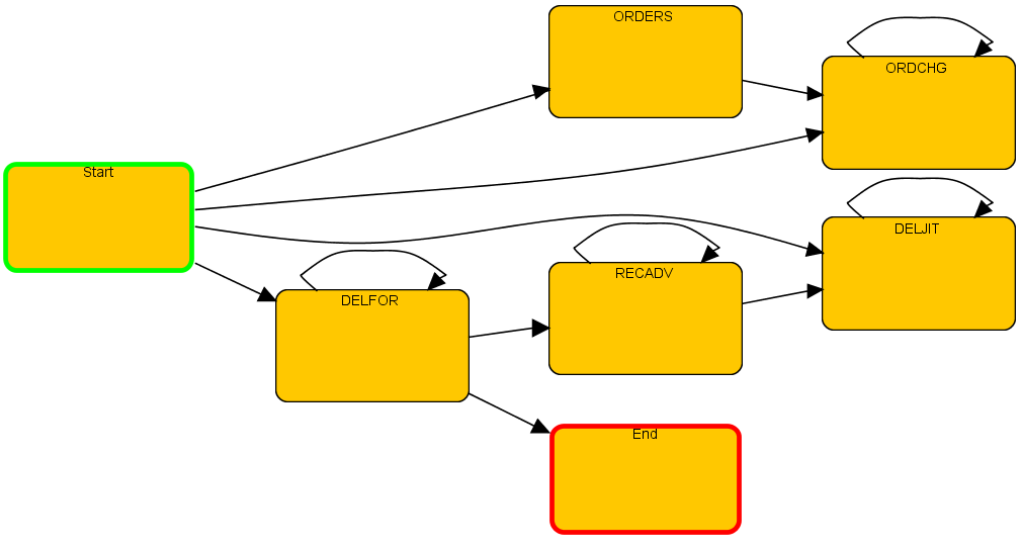

Figure 14 - The causal net found by the heuristic miner, showing a lot of similarities with the heuristic net in Figure 13 w ith respect to separating the ordering and scheduling/delivery phases.

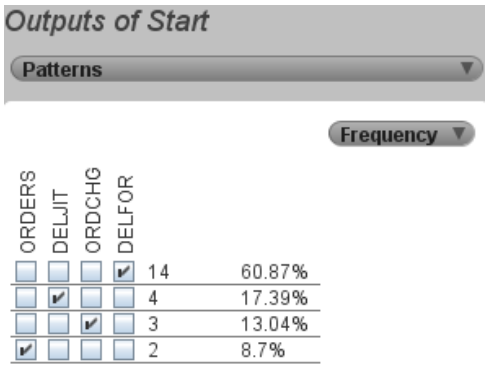

Figure 15 – This figure shows that the possible outputs for the artificial start event are {ORDERS}, {DELFOR}, {DELJIT} and {ORDCHG}. The fact that 91.3% of the traces do not start with an ORDERS message is an indication for the limited time window documented in the event log.

4.3.3 FLEXIBLE HEURISTIC NET

Flexible heuristic net miner is an update of the heuristic net miner it uses more features of the process representation language that is also used for the genetic miner (Weijters & Ribeiro, 2010). Since the flexible heuristic miner outputs causal nets, and causal nets require a single start and end event, only the event log will be used with the artificial start and end events added.

The flexible heuristic net generated by converting the heuristic net in Figure 13 is shown in Figure 16; it shows the same behavior as the heuristic net. The biggest benefit of this conversion though is that this flexible heuristic net can serve as input for conformance checking. This flexible heuristics net is used for this purpose in section 5.1.

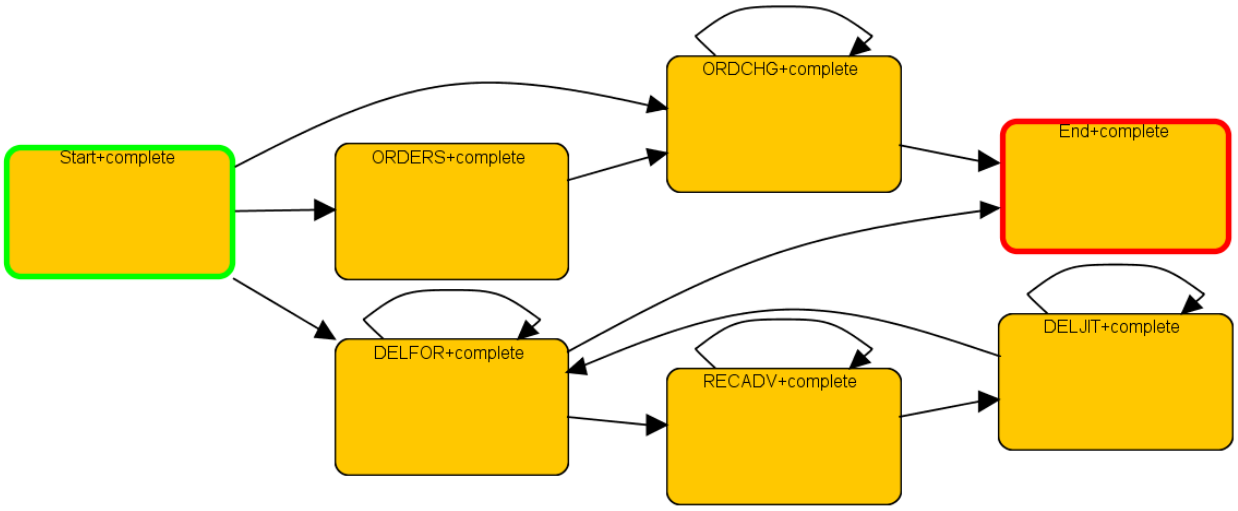

Figure 16 - This figure shows the flexible model as a result of the conversion from the heuristic net in Figure 13.

4.4 GENETIC MINING

ProM contains two ways using the genetic miner on an event log, by mining for a best-fitting heuristic net or sample based mining. The genetic miner uses a different technique of finding process models than the heuristic miner.

Genetic algorithms are adaptive search methods that try to mimic the process of evolution. These algorithms start with an initial population of individuals. Every individual is assigned a fitness measure to indicate its quality. In our case, an individual is a possible process model and the fitness is a function that evaluates how well the individual is able to reproduce the behavior in the log. Populations evolve by selecting the fittest individuals and generating new individuals using genetic operators such as crossover (combining parts of two or more individuals) and mutation (random modification of an individual) (de Medeiros, et al., 2007).

Unfortunately, because the genetic miner is looking for the best-fitting process model, the process model does not look anything like the intuitive process. In case of a more complete event log containing a lot more messages that describe the whole process of ordering, scheduling and delivery it would probably result in a model that is closer to reality.

Figure 17 shows the model mined by the genetic miner from the sample event log; in this case no artificial start or end events were added to the event log. As you see the genetic miner finds the best fitting process, but it does not show a single process, rather it shows several decoupled parts.

Figure 18 shows the model for which artificial start and end events were added to the event log. This model differs from the model mined by the heuristic miner. The genetic miner tries its best to find the best fitting process model, but does not find the intuitive model in Figure 7 or closely related to it the heuristic net in Figure 13.

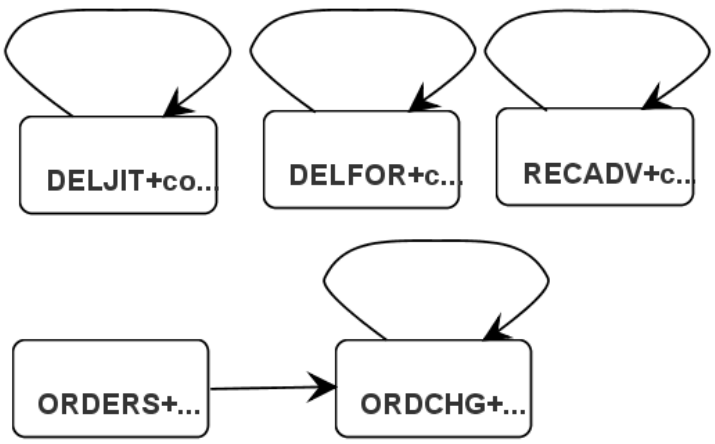

Figure 17 - The best-fitting heuristic net mined by the genetic miner with a fitness of 0.811. For comparison, Figure 18 shows the result when the genetic miner is used on the event log with artificial start and end events.

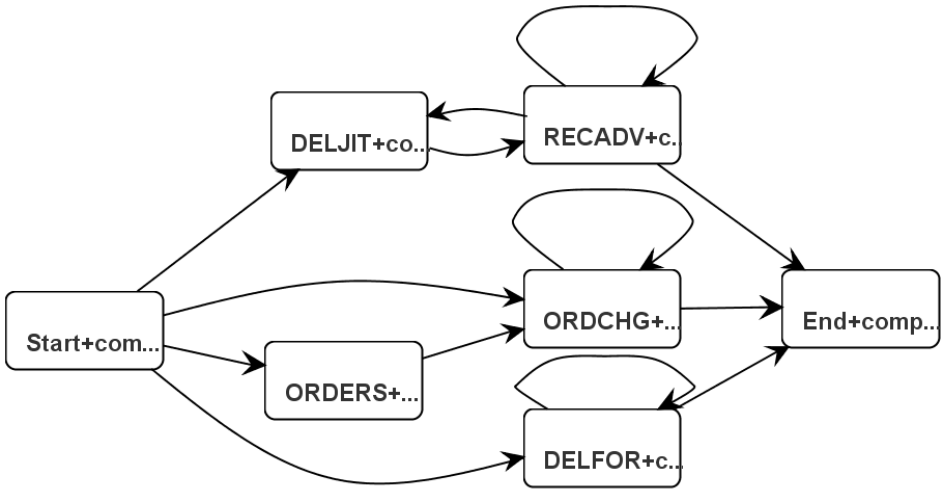

Figure 18 - The best-fitting heuristic net with artificial start/end events. Fitness is 0.91.

4.5 DECLARE MINING

A plugin that is quite interesting and differs from previous discussed process discovery techniques is the Declare Miner. It discovers Declare models based on the sample event log. Because the sample log is probably incomplete the rules are possibly not of interest in real situations, but it is a welcome demonstration of what this ProM plugin is capable of.

Declare models can capture behavior in a compact set of rules (Pesic & van der Aalst, 2006) (Maggi, et al., 2011). There are several templates that can be applied, each template in accordance with some set of predefined rules. Each template is explained in the paper written by Maggi et al. (2011) and will not be discussed in detail here. Below you will find several examples of rules found in the example event log below in Figure 19, Figure 20, Figure 21 and Figure 22.

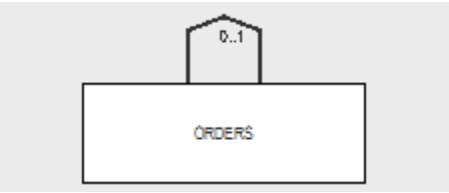

Figure 19 - An example of absence template which defines an upper bound for the amount of appearances of a certain message in a trace, in this case the ORDERS message will appear at most once. The counterpart of the absence template is the existence template which indicates a lower bound.

Figure 20 - An example of the response template. The response relationship means that DELJIT messages are eventually followed by DELFOR messages and ORDERS messages are followed by ORDCHG messages.

Figure 21 - An example of the precedence template. This relation indicates RECADV messages may only occur after a DELFOR message.

Figure 22 - An example of the chain response template. This relation indicates that an ORDCHG message has to come directly after and ORDERS message.

There can be made a difference between relaxed or stricter rules. Relaxed rules specified in for instance the response template (Figure 20) are less strictly defined in the chain response template (Figure 22). It is doubtful the relaxed rules are very useful for something important as exchanging EDI messages between parties.

Defining a lower bound (existence template) or upper bound (absence template) for the number of messages in a trace might be useful, as well as defining what message should be the first in a trace (init template). Defining the exact number of a certain message-type in a trace (exact template) is just a combination of using a lower- and upper bound and therefor is just as useful, although from at this moment it was only applicable for the ORDERS message.

The more relaxed rules like responded existence, co-existence, response, precedence and succession are too weak to base any guidelines for communication on. Alternate response, alternate precedence and alternate succession, based on their weaker brothers, are already stronger since they constrain what needs to happen after some type of message before the same type of message can occur again. Unfortunately a useful or non-trivial example of application of this rule was not found. Stricter rules are chain response, chain precedence and chain succession, indicating a message-type should directly be followed by another message type. At this moment only the chain response rule shown in Figure 22 was found. This is partly because DELJIT, DELFOR and RECADV message types, as shown in the heuristics net by the self-loop of messages in Figure 12, can occur multiple times, so a chain response template cannot be applied to those loops.

4.6 INTERMEZZO: POSSIBLE PROBLEM

The problem of finding a different order of messages than expected is a concern that needs some attention. The possible problem is we do not know how to generate the appropriate traces from the attributes at hand. While now the order numbers are used to connect messages together to form traces, there is need for another way to do this. Without going into more detail on EDI messages again a possible explanation on what the problem could be is given below.

In DELFOR messages a delivery in the distant future is scheduled, as well as a possible delivery schedule for continuous delivery. In DELJIT messages a delivery in the near future is scheduled, containing more detailed delivery information. RECADV messages are used to confirm delivery of goods. Based on these descriptions a DELFOR·DELJIT·RECADV message sequence would make the most sense.

What probably happens at this moment is that, probably due to some message scheduling arrangement between parties, DELFOR messages for goods belonging to the same order occur between DELJIT messages and RECADV messages of another order. These orders are however related to each other by order number or previous order number, but communication about these goods is split up because of logistics or planning. Figure 23 shows a visualization of the problem.

A combination of a goods identifier as well as scheduling data to identify unique traces could solve this problem; at this point, though, there has been no success in finding such combination and it is also out of the scope for this report.

Note: After communication with Robert Engel, the creator of the EDImine Correlation plugin, about this possible problem, it appeared this problem was already thought of[5]. Also the solution as suggested here was tried but unsuccessful so far.

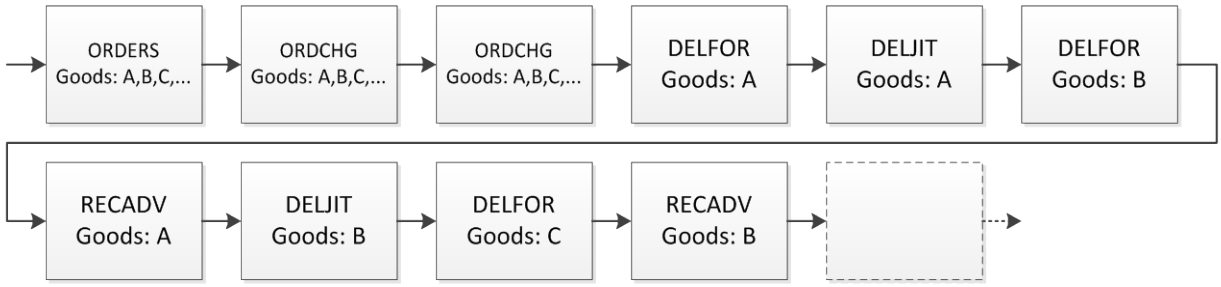

Figure 23 - This model shows a visualization of the possible problem. An order is placed for at least products A, B and C. After some ORDCHG messages the order is getting processed; product A is delivered first, then product B, then product C… When product A is about to be delivered, and after a DELJIT message is send specifying at what time exactly, a DELFOR message is also issued scheduling the delivery of product B in the future before actually sending a RECADV message confirming the receiving of product A. This could be the reason that messages belonging to the same order can cause the mined process model to differ from the actual communication process.

[5] The topic was discussed in an earlier mentioned e-mail conversation between the 6th and 12th of October 2011

4.7 CONCLUSIONS ON PROCESS DISCOVERY

Working with incomplete traces was not beneficial for analyzing the different plugins in ProM, but no one can guarantee a complete event log in the future. Where a user wants to mine for an intuitive and sound process model the different plugins discussed in previous sections all found different ones. By analyzing the event log by hand (using for instance the dotted chart analysis) and some common sense it was found that the heuristic mining algorithms work best with this kind of data.

The mining algorithms based on semantics of petri nets, the α-algorithm and the ILP miner do not possess the capabilities to find an intuitive process model in the big unstructured pile of EDI messages. The main reason is probably the earlier mentioned incomplete event log, but in real communication process several different kinds of traces occur also, which these algorithms probably cannot differentiate.

The heuristic mining algorithms show great promise in combination with the event logs based on EDI communication. The heuristic mining algorithm is the only process discovery approach that successfully found an model related to the intuitive model, and with the addition of the artificial start and end event found a connected process. Of course this 'connected' process only connected the two separate parts from the model without artificial start and end events, but it is a good indication the heuristic miner is able to keep separate process parts indeed separately.

The genetic mining algorithm tries to find the best-fit, which is a major con in case of incomplete and small event logs. The models don't look like the intuitive models that were found earlier with the heuristic mining algorithm.

To conclude this section on process discovery possible applications of process mining based on event logs generated from EDI messages will be discussed. In industry it is important for a company to know how their internal processes work in order to formulate performance standards and improve their business processes. Why should this be any different for the process of communication? EDI messages are not human-readable and sent many times a day; which makes it a tedious job to analyze this type of communication. The EDImine Correlation plugin offers an interface to analyze these messages based on attributes and exports the different traces that are detected.

Based on this event log the heuristic mining plugin can visualize the communication process which immediately shows the outline of what exactly happens. There are additional types of messages than the five that were found in the sample data, and imagine a company sending and receiving a lot of these messages a day. The heuristic miner allows rapid model generation. At first glance researchers can inspect this model to see if it indeed is the process they expect. If the mined process model is what they were looking for, this model can serve as input for conformance checking to find deviations. Conformance checking allows for a company to monitor their current communication and if necessary detect were things go wrong. Conformance checking will be discussed in the next section.

The rules that follow from the Declare miner can be used to manage the communication channel between companies, for example based on these rules it can be determined if a certain order of message-types is used. Based on these rule agreements come to exist between parties and warnings can be issued when a message order is detected which is not allowed. In the extreme case a new order of messages can be negotiated. The LTL checker which will be discussed in section 5.3 can use Declare rules as input for analyzing an event log.

5 CONFORMANCE CHECKING

Conformance checking allows for comparison between an existing process model and an event log. Conformance checking can be used for checking if a generated process model conforms to the reality as it is depicted in the event log. This promises to be valuable activity for industry since it allows for control and checking, relatively simple.

5.1 CONFORMANCE ANALYSIS ON FLEXIBLE HEURISTIC NET

A flexible heuristic net or a causal net can be used to replay event logs with for conformance and performance checking. In this section the results of replaying the flexible heuristic net from Figure 16 are discussed with the event log that was used throughout this report. Table 2 shows the results indicating a minimum fitness of 0.9273.

Noteworthy is that the skipped activities occur because of the incompleteness of the event log, but it can also be the case that something actually went wrong in the communication between parties.

Property	Value
Cost-based fitness/case	0.9922
#Unobservable activities	0
#Skipped activities	12
Minimum fitness	0.9273
#Cases replayed	23
#Synchronous activities (log+model)	434
#Inserted activities	0
Std. deviation fitness value	0.0183
Maximum fitness	1
#Violating synchronous activities	0

Table 2 - This table shows the result of replaying the event log with the mined flexible heuristic net generated from the heuristic net in Figure 13.

As concluded in section 4.7 on process discovery conformance checking allows a company to inspect their communication processes and analyze possible deviations from an earlier specified process models. Figure 24 shows the interface of the conformance analysis plugin. At first glance it looks very intuitive with on the left hand side the traces and on the right hand side various statistics on how well the process model performed with the event log that served as input. But this can of course also be read the other way around, the statistics show how well the messages in the event log correspond to the process model, which is an assumed model of how the process works in real life.

The plugin shows messages occurring as predicted by the model in green and uses different colors to depict different inconsistencies between event log and process model. In Figure 24 you can see several pink messages among a lot of green message; these messages are supposed to happen according to the model but did not happen. The plugin inserts the necessary messages to make the trace conform to the process model; this insertion can be seen as notification something unforeseen has happened.

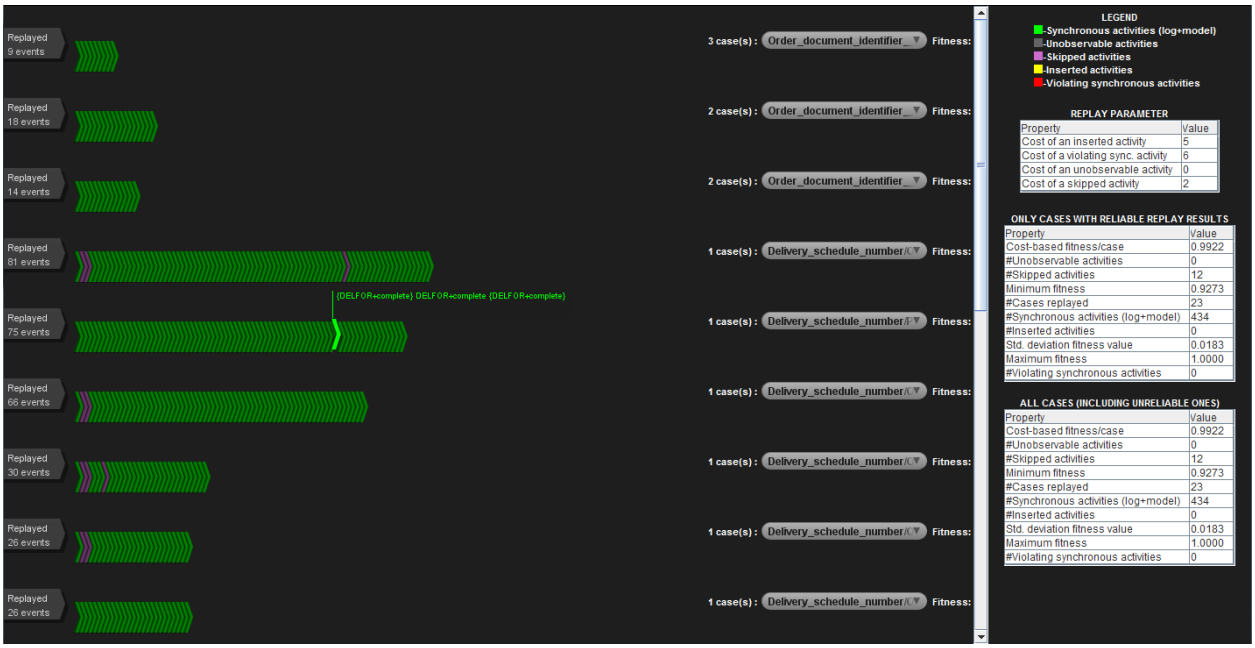

Figure 24 - The main screen of the conformance checker plugin. With on the left hand side the different traces, with possible 'missing' actions and on the right hand side various statistics which also are shown in the table above.

Note that in this section conformance analysis on a flexible heuristic net is shown, but this is also possible for petri nets and causal nets for instance. Conformance analysis on a petri net is discussed in the next section.

5.2 CONFORMANCE ANALYSIS ON PETRI NET

Fitness can also be computed on a petri net and an event log. For the purpose of testing a Heuristic net is converted to a petri net, since the petri net mined by the α-algorithm did not seem useful and the petri net mined by the ILP miner supported all behavior which of course leads to a fitness of 1. Unfortunately the plugin to calculate fitness did not run properly when using it in combination with this model. Fortunately there is also a conformance checker for petri nets as concluded in the previous section and as you can see in Figure 25 which will be discussed instead.

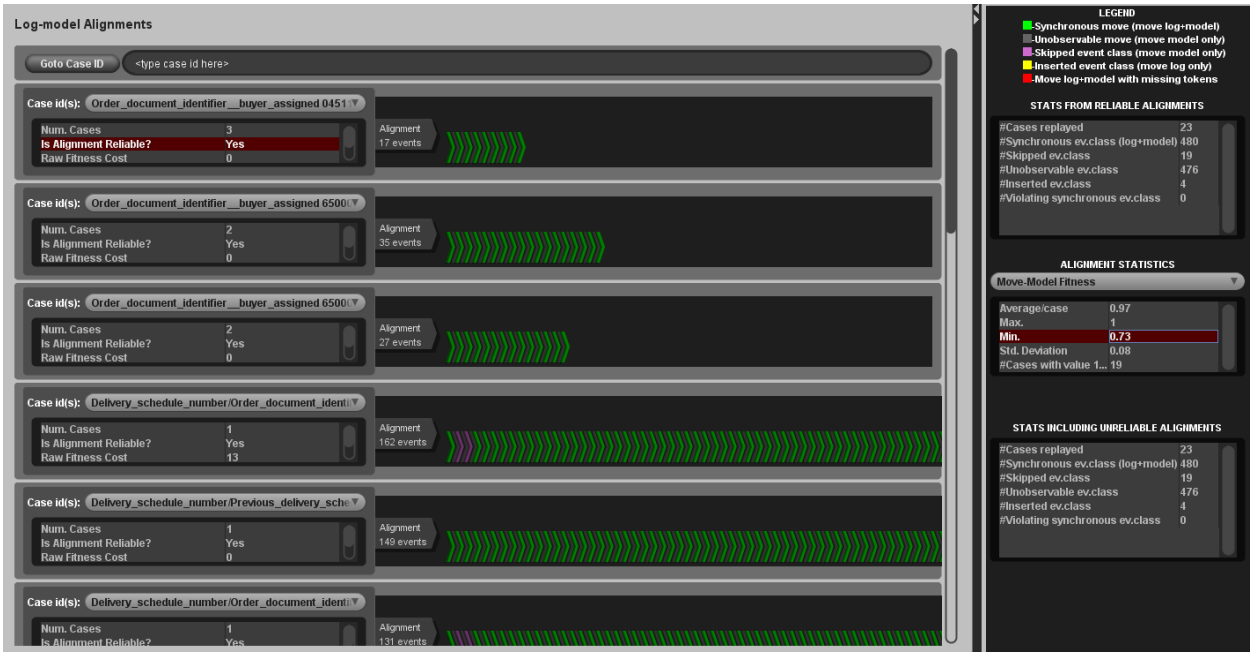

Figure 25 - The main window for petri net conformance checking, minimum (0.71), maximum (1) and average (0.97) fitness are calculated as are much more statistics.

The conformance checking plugin has a different look as the plugin used for conformance checking on the flexible heuristic net in the previous section, but the functionality is the same. Is allows visualization of the different traces and shows where it differs from the expected behavior. It also gives valuable information about the fitness of different traces and possible costs if applicable.

5.3 LTL CHECKER

As mentioned in section 4.5 on the Declare miner, declarative languages can be used to describe behavior. Mined rules from the Declare Miner can be tested with the LTL checker on the sample event log. ProM allows converting the mined Declare model to LTL model that can be used as input for this plugin.

The plugin allows for two different views, see per rule what traces followed the rules or see per trace what rules were violated or not. Figure 26 shows one of the views; selecting a rule the plugin shows what traces did not comply with the selected rule. In this case five of the 23 traces violated this rule, so the rule has coverage of 0.78.

This plugin can be of great use for industry to identify rule violation in communication between parties.

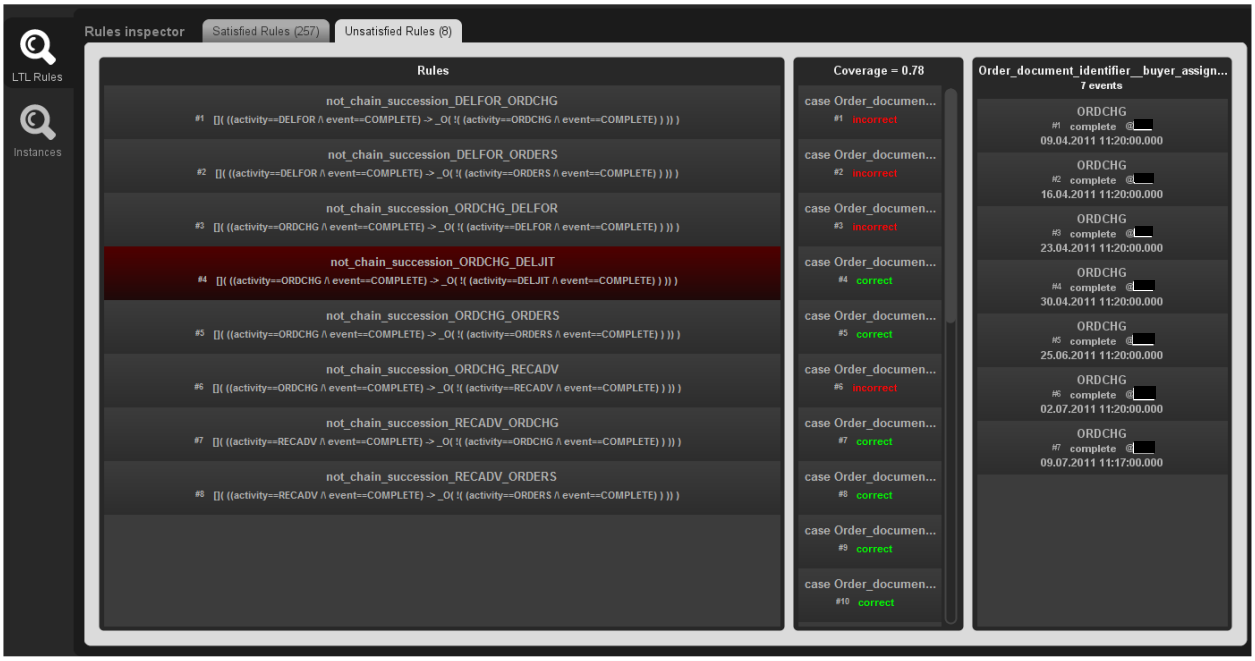

Figure 26 – The main window of the LTL checker plugin; rules can be tested per trace, but it also shows the trace tested per rule in a different tab.

5.4 CONCLUSIONS ON CONFORMANCE CHECKING

Process discovery lets companies and researchers discover processes from event logs, which can be very helpful in understanding what exactly happens as a process. By using conformance checking these generated models can be made useful by comparing it to reality, or comparing reality to the assumed process.

The conformance analyses that were discussed in the first two subsections were perfect examples of plugins that based on some process model and an event log from a real process, show a clear and compact summary on what exactly happens. Replaying these event logs on the process model allows the user to see what traces do conform to the process model, but more importantly see what traces did something that was not predicted? What happened? Was this just a hiccup or does the process model miss some behavior and is there apparently something that happens which is not expected? So, apart from just understanding the process, conformance checking will help detect a mismatch between the assumed process model and the reality. Of course it can also be the case that there did go something wrong in reality and because it was detected by replaying the trace with the process model action can be taken to do something about it.

The LTL checker is a different kind of conformance checking than described above. It still checks certain behavior in the event log, but does this based on declarative rules instead of whole process models. These rules are predetermined and allow companies to test these rules on-the-fly, as will be discussed in section 7 on operational support.

Understanding the process model, whether this is for the internal organization of a company or in this case the communication process, is very important because not knowing what happens can cost money and time. Also, keeping a close eye on the process allows for correcting mistakes and avoiding possible difficulties in the future.

6 ADDITIONAL PERSPECTIVES

In this section two additional views on the event log generated from the plugin are discussed; the dotted chart analysis and the social network analysis.

6.1 DOTTED CHART ANALYSIS

The dotted chart analysis provides a 'helicopter view' on the data provided in the even log. It shows the different traces with their messages on a timeline. This timeline can be actual time (Figure 27), relative time since the start of the trace (Figure 28), relative with respect to the completion of the trace (Figure 29) and logically ordered (Figure 30).

The dotted chart analysis gave the first hints that the sample event log was incomplete. In the first two figures mentioned here it is clear that there is a big gap in the event log which cannot be explained. The last figure on the dotted chart analysis, Figure 30, just puts the messages together consecutively and immediately shows that we have to deal with a continuous process (because of the continuous loop mentioned earlier).

The dotted chart analysis lends itself perfectly for a broad view on what happens, but provides little value to actually discovering a process model.

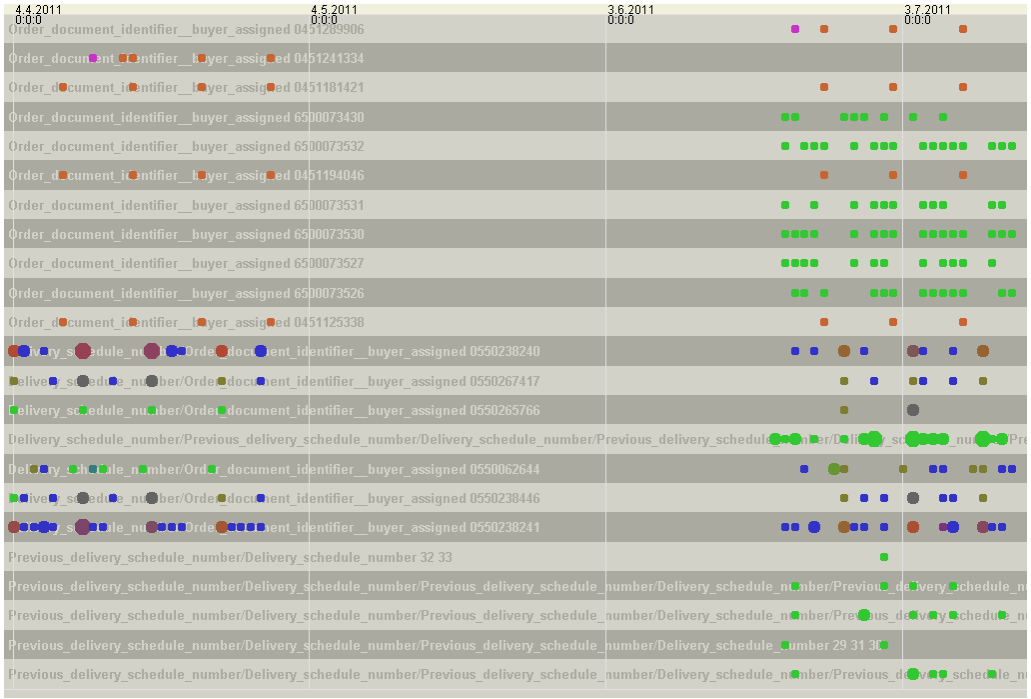

Figure 27 – The dotted chart with 23 timelines; the timelines are in actual time.

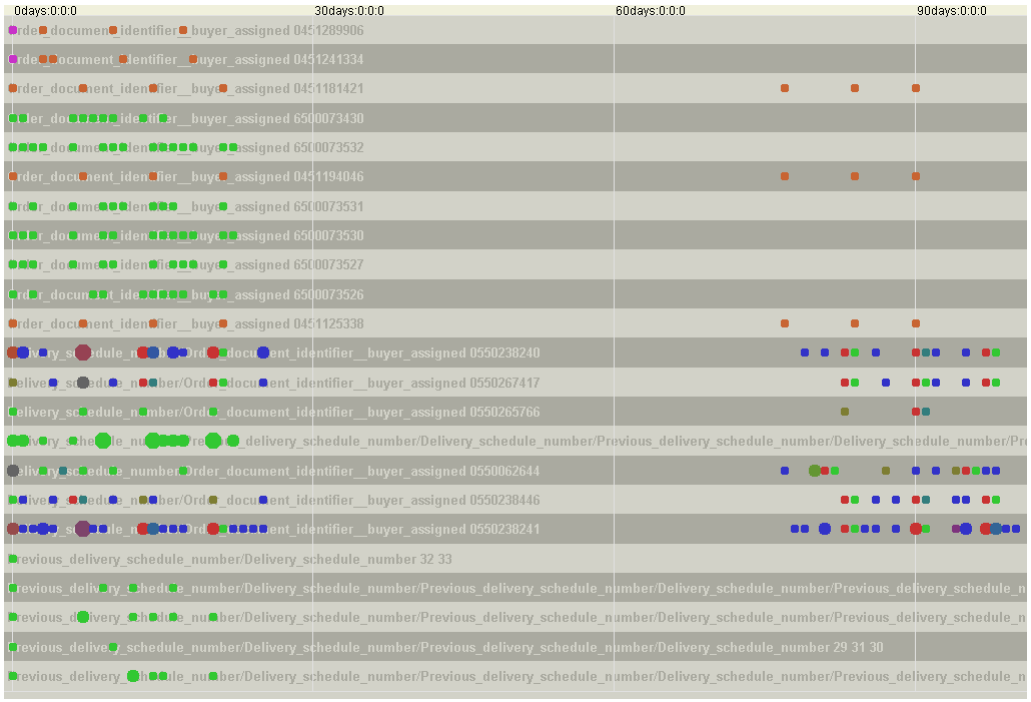

Figure 28 – The dotted chart with the 23 timelines. This time the timeline is ordered to relative time with respect to the start of the trace.

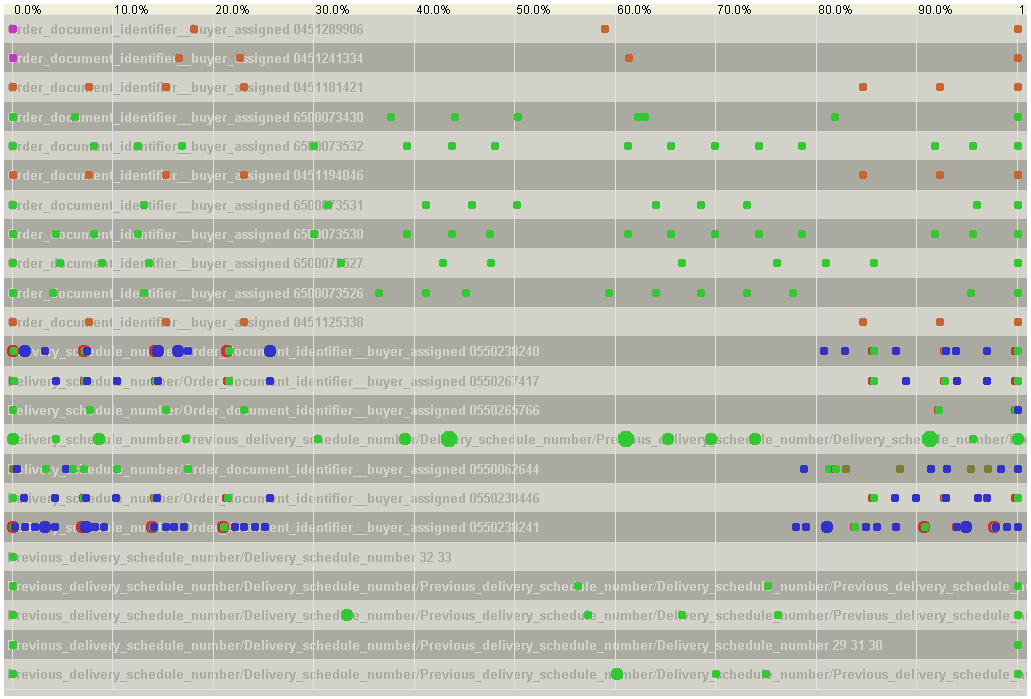

Figure 29 – The dotted chart with the 23 traces showing timelines relative to the complete trace. Note that the milestones in the timelines are now percentages.

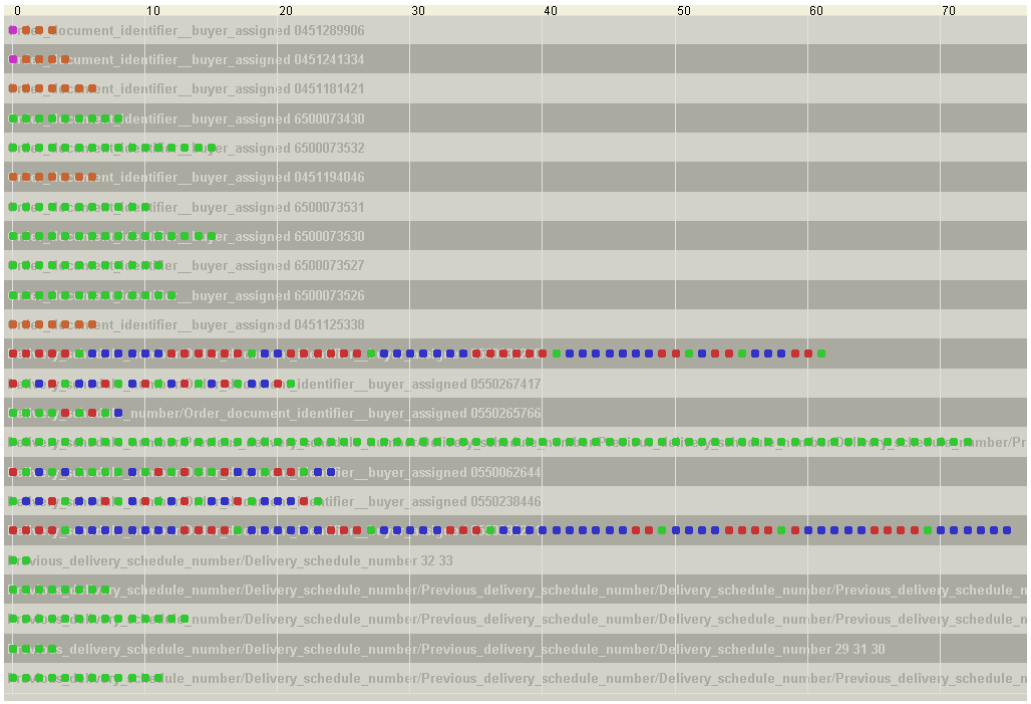

Figure 30 – The dotted chart with the 23 timelines which shows the messages in a logical order. As you can see there exist traces with ORDERS·ORDCHG, only DELFOR and traces containing the DELJIT·DELFOR·RECADV loops.

6.2 SOCIAL NETWORK ANALYSIS

At this moment the EDImine Correlation plugin exports event logs containing both the sender and the receiver information of an EDI message in the event[6]; however messages are exported as completion events with only the sender as the resource used. When mining for a social network, for instance for a handover-of-work social network, the results are disappointing as shown in Figure 31.

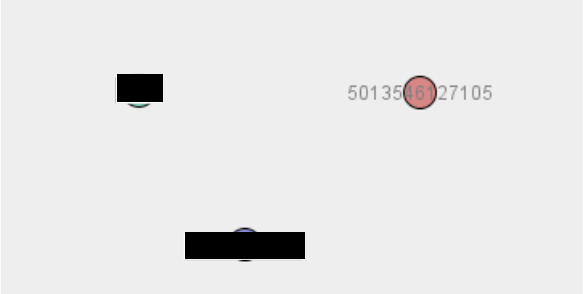

Figure 31 - This figure shows the result of the handover-of-work social network miner on the sample event log.

In none of the different type of social network miners the three parties are connected in any way, while as mentioned in section 3 there are actually four parties and these parties are communicating, the three parties shown in Figure 31 and one receiving party.

With a small addition to the EDImine Correlation plugin communication between parties could actually be analyzed too. The solution is simple and can be constructed as follows.

Instead of adding completion events with the sending party as a resource, it should be able to create two events per message, one starting event for the message containing the sending party as a resource and one completion event for the message containing the receiving party as a resource. This way a social network miner can be used to analyze communication between parties. The result of an event log generated like this is shown in Figure 32.

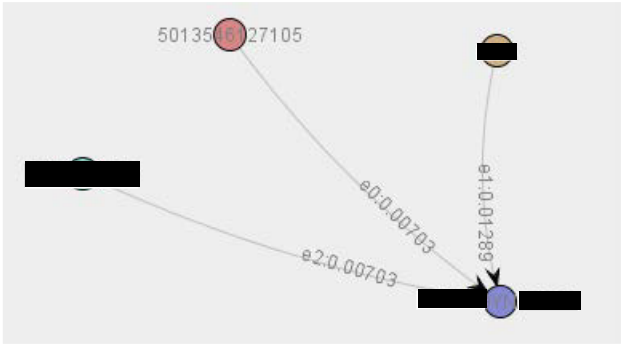

Figure 32 - This figure shows the result of the handover-of-work social network miner on the sample event log when start and completion events are used to indicate the difference between a sender and a receiver. [redacted] is introduced to merge [redacted] and [redacted] receiving parties.

[6] Throughout this report the events in the event log were called messages to emphasize we have with communication; in this section we call events by their name since we are discussing the concept of events and event logs. Messages in this section are the message contained in the imported data.

7 OPERATIONAL SUPPORT

Most process mining techniques analyze events that belong to cases that have already been completed (Aalst, 2011). For operational support also cases are considered that not yet have finished and are still active in the process. Partial traces that have been generated so far by still running cases can be used for three distinct categories of actions, namely detecting, predicting and recommending:

- *Detect:* An operational support system is able to detect from a partial trace belonging to a running case if deviations occur from the path that should have been followed according to the process model.
- *Predict:* An operational support system is able to, based on a partial trace from a running case, predict properties of a case on beforehand. For example the expected costs or total running time.
- *Recommend:* An operational support system is able to, based on a partial trace from a running case, recommend users of the system on what activity should be performed next with some certainty.

From these descriptions is it clear that the act of recommending is not relevant; the sending of EDI messages is fully automated so there is no need for a system to recommend a certain course of action to a human being. The other two activities can be useful for EDI communication and can be supported by ProM, already or in the future.

Conformance checking is a great way of detecting deviations in traces of running cases. As shown in section 5.1 conformance checking of traces using a heuristic net and an event log offers an intuitive dashboard from which the EDI communication can be supervised. In case messages are send following some process model trace are generally colored green, deviations from the process model are colored differently so they are noticeable by users. These deviations can have multiple reasons, of which lost messages can be one. In case the same problem occurs regularly it may be a problem with the assume process model. Thus keeping a close eye on the process allows managing the process both ways; the communication is checked against what is assumed to happen, but also the assumed process is checked against what happens in reality.

Another way of detecting deviations is the use of a combination of Declare rules and the LTL checker, described in section 4.5 and section 5.3 respectively. Declare rules lend themselves perfectly for automatic detecting of deviations, because the simple rules describe a small part of the overall process. These rules can confirm communication agreements between parties made earlier, or can confirm expected behavior. In case something in the communication process differs from the expected behavior the corresponding case can be flagged and necessary actions can be taken. The LTL checker plugin also allows for checking rules against the event log, and vice versa. In case some rule gets below a certain fitness alarm bells could ring because the quality of the communication process may be in danger.

Prediction can only be semi-supported by plugins. EDI messages generally contain a lot of information; information on when the next delivery should be and what amount of what product should be delivered is already known. This information anchors down a lot more, for example costs, expected delivery date and what party is responsible for shipment. Though, from the dotted chart analysis discussed in section 6.1 can de deduced that messages are send at certain intervals. A plugin that is able to mine for time patterns should be able to predict how long it takes until the next message should arrive. Time patterns can also be discovered and a corresponding conformance checker can in turn determine if messages are indeed received within the time frame they are expected.

8 CONCLUSION & FUTURE WORK

To conclude this report it must be stated that familiarization with the EDImine Correlation plugin was very interesting and analyzing the communication process that was hidden in the unreadable EDI messages was a great challenge.

The EDImine Correlation plugin allows researchers and industry to convert the unreadable EDI messages to better readable XML and export those messages in the form of traces to ProM event logs. These event logs can be used in combination with several plugins as shown in this report and allows users to further analyze documented behavior. This offers great opportunities which offer better understandability of the communication process. This concept of better understanding can lead to optimizations in or around the process which can save money or time by helping to detect difficulties and errors in the process for example.

As mentioned in the description of the EDImine Correlation plugin (section 2) and in the discussion on the social network miner (section 6.2) some upgrades and additions are possible for the plugin; especially the possibility to split the messages into two events, one for the sender and one for the receiver, to allow the use of social network mining algorithms seems to be a worthy addition. Also further analysis with a complete event log would be welcome; apart from the two-month gap in the initial sample data, it only contained six types of messages while there are many more types out there. Analysis on sample data originating from different parties would probably show different kind of processes and maybe shed a new light on how ProM plugins can be used to further analyze this kind of data.

Apart from additions to the plugin, developing a plugin for ProM that mines for timing patterns and allows predicting messages based on these time patterns is a good suggestion for future work. In case such a plugin exists it can be used to detect anomalies when certain messages are sent or received.

Hopefully a proper initial picture on what is possible with the event logs exported from the EDImine Correlation plugin is painted in this report. It promises to be a welcome plugin in the world of process mining.

9 REFERENCES

Aalst, W. v. d., 2011. *Process Mining : Discovery, Conformance and Enhancement of Business Processes.* s.l.:Springer Verlag.

Aalst, W. v. d., Weijters, A. & Maruster, L., 2004. Workflow Mining: Discovering process models from event logs. *IEEE Transactions on Knowledge and Data Engineering,* 16(9), pp. 1128 - 1142.

de Medeiros, A., Weijters, A. & van der Aalst, W., 2007. Genetic process mining: an experimental evaluation, Vol. 4, Iss: 2. *Data Mining and Knowledge Discovery,* pp. 245-304.

Engel, R. et al., 2011. Process Mining for Electronic Data Interchange. In: C. a. S. T. Huemer, ed. *E-Commerce and Web Technologies.* s.l.:Springer Berlin Heidelberg, pp. 77 - 88.

Maggi, F., Mooij, A. & van der Aalst, W., 2011. User-guided discovery of declarative process models. *2011 IEEE Symposium on Computational Intelligence and Data Mining,* pp. 192-199.

Pesic, M. & van der Aalst, W., 2006. A Declarative Approach for Flexible Business Processes Management. In: *Business Process Management Workshops.* s.l.:Springer Berlin / Heidelberg, pp. 169-180.

Weijters, A., Aalst, W. v. d. & Medeiros, a. A. A. d., 2006. *Process Mining with the Heuristics Miner-algorithm.* s.l.:BETA Working Paper Series, WP 166, Eindhoven University of Technology, Eindhoven.

Weijters, A. & Ribeiro, J., 2010. Flexible Heuristics Miner (FHM). *Beta Working Paper series, WP 334.*

Wiel, T. v. d., 2010. *Process Mining using Integer Linear Programming,* Eindhoven: Eindhoven University of Technology.