Analyzing a TCP/IP-Protocol with Process Mining Techniques*

Christian Wakup[1] and Jörg Desel[2]

[1] rubecon information technologies GmbH, Germany
[2] Fakultät für Mathematik und Informatik, FernUniversität in Hagen, Germany

Abstract. In many legacy software systems the communication between client and server is based on proprietary Ethernet protocols. We consider the case that the implementation and specification of such a protocol is unknown and try to reconstruct the rules of the protocol by observation of the network communication. To this end, we translate TCP/IP-logs to appropriate event logs and apply Petri net based process mining techniques. The results of this contribution are a systematic approach to mine client/server protocols, an according tool chain involving existing and new tools, and an evaluation of this approach, using a concrete example from practice.

1 Introduction

Since the 1980s, many software solutions for business software in various application domains have been based on proprietary hardware for a client/server-architecture, and often also on proprietary operating systems. Since this software is highly mature and dependable, emulation software was used in the 1990s for migration to the Windows- or to the UNIX world. This is still today's situation at many places. Communication between client and server is often based on proprietary protocols which respect the requirements of the original terminals. These protocols are determined by fixed rules for the interaction behavior between their respective partners or system components [7]. However, often neither these rules nor a precise definition of the resulting behavior is known today. Substitution of an interface involved in a protocol may cause rule violations, resulting in severe consequences. Therefore, conformance to the protocol definition is highly desirable when interface software is newly implemented. To this end, identification of this definition is an obvious first prerequisite for protocol implementation substitution. So we aim at reconstructing specification models of a protocol from its behavior.

According to the *Process Mining Manifesto* [3], process mining aims at discovering, monitoring and improving real processes by extracting knowledge from event logs. The considered processes are mainly business processes, and the event logs are sequences of observed and recorded occurrences of business process activities. These activities typically involve the considered information system and

* based on the thesis [14] of the first author, supervised by the second author

human interaction, but can also be automatized activities. There exists a large variety of mining techniques and tools as well as suggestions how to systematically choose and apply these techniques, based on the given application.

In our setting, we apply process mining techniques to event logs generated from automatized activities only. The underlying process is more a technical protocol than a business process. So we aim at *discovering protocols* instead of processes from logs. Our core research question is whether process mining techniques are useful in this context as well, and if so, how to adapt existing mining techniques to this application area. The result is quite positive, as the case study at the end of the paper will show.

This case study is based on a real industrial challenge: recover an existing legacy interface protocol without known definition. A new version of this protocol had been implemented in an ad-hoc way, and this implementation has subtle differences to the original one. So the task was not only to recover the original protocol definition, but also to compare it to the new one and find out precisely the differences between both. In this sense, this work can also be seen as a study in *conformance checking*, another main objective of process mining (up to which delta is the new protocol conform to the original protocol definition?).

The protocols to be discovered in our approach describe the interface behavior of clients and servers that communicate via TCP/IP. So this setting does not offer event logs that can immediately be used by process miners. Instead, the only way to acquire information about behavior is to record traffic data from the involved network. Actually, this is the first step of our approach. We apply the *Wireshark* network protocol analyzer [11] which is able to observe and record transmitted packets in the network. Moreover, this program offers useful filtering capabilities that are used to create logs for single communications between specific clients and specific servers.

These logs are not immediately suitable for process mining algorithms. Based on published requirements for event logs for process mining, we develop an approach and introduce a tool to transform them into event logs. To this end, information is gathered from different layers of the network protocol, and from information added by the *Wireshark* tool. Since the information about the performed activity type is not achievable this way, we add a semi-automatic technique to derive this information. This technique is supported by our tool, too.

Finally, the event logs are transformed in the XES standard format [6] so that mining algorithms such as provided by ProM [12] can be applied. We select a collection of algorithms that are promising for this application area and compare their respective results. We apply the techniques described before to a practical example and thus evaluate these techniques and the developed tool. In this case study, process miners are not only applied to generate protocol models but also to check whether models can be viewed as sub-models of others which are generated from additional use cases. Finally the protocol derived from mining is compared to a protocol generated in an ad-hoc way.

The complete sequence of steps of our procedure, together with supporting tools, is summarized in Figure 1.

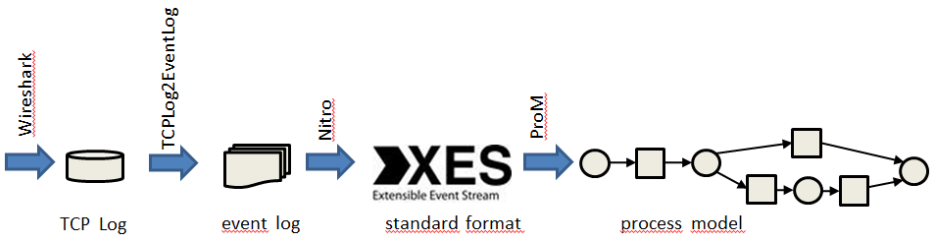

Fig. 1. The complete tool chain of our approach

The topic of this paper has strong relations to *service mining* [1], which is also based on more technical events, but aims at the identification of web services. Service mining can be viewed as a sub-discipline of the more general *protocol mining*, which is, however, not an established field. In particular, mining of protocols is usually not covered by *protocol engineering* [7]. Thus our contribution also suggests a new research area combining mining techniques and engineering techniques for more general protocols.

The paper is structured as follows. In section 2 we report about the generation of logs from the network and the pre-processing necessary to use these logs for process mining. In particular, we introduce our tool *TCPLog2EventLog*. Section 3 is devoted to process mining approaches applied to our logs. We recall characteristics of different miners and select miners that appear to be capable of delivering useful results in our setting. In section 4 we describe our case study in more detail and show results for particular use cases. Section 5 recapitulates the findings of this study.

2 From Network Traffic to Logs

Network protocols are defined on messages between the protocol partners. For mining purposes, these messages have to be observed and logged in an appropriate way. For service mining, [5] suggests an HTTP-listener for the generation of event logs, i.e., runs of web services. This listener precisely filters out and records the relevant data traffic of the web server. The situation in our client/server setting is quite similar to the data exchange between services. Hence we follow the above suggestions for service mining, but adapt the logs and the generation mechanism where appropriate or necessary. To this end, we developed a new software tool, *TCPLog2EventLog*, that supports the construction of event logs from packet based network protocol data.

In this section, we show how to come from network traffic to logs, where each log contains all relevant data for later process mining. For the latter, we refer to [2], which characterizes the requirements for data entries in event logs. These requirements distinguish different cases of processes. Usually, entries of event logs can refer to different cases which have interleaved actions. Since each run of the process to be mined belongs to a single case, each entry has to refer

to its respective case. This, and other requirements of [2], are summarized in the following list:

- Each entry refers to an event at one point in time, not a period.
- Each entry refers to one single event, which is uniquely identifiable.
- Each entry should include a description of the respective event.
- Each entry refers to a single process case.
- Each process case refers to a specific process definition.

Since our approach is not based on HTTP, as in the case of web services, we apply the software *Wireshark*, a tool for analysis of network traffic [11]. This tool allows a complete recording of data exchange between two devices and offers several filtering features. The recorded data is stored on the server, which ensures that traffic between the server and *all* clients is complete, and is available at one place.

This recorded data contains communication information relevant for the protocol of interest, but also the complete further data exchange that happened in the same time interval. Moreover, data items might refer to the establishment of TCP connections and even to duplicated packets in case of retransmission. This additional information complicates the further construction of correct event logs, or it can result in noise. Using the filtering functions provided by Wireshark, we therefore filter this data as follows:

1. Each TCP connection uses a single port of a network. Each network-based communication which is relevant for our purposes uses a particular port. By filtering out all data which does not refer to this port, we get rid of additional traffic from other applications (*Wireshark* filter: `tcp.port==<port number>`).
2. The establishment and deletion of a connection uses particular flags. In an existing connection, usually these flags are `SYN` or `ACK` (*Wireshark* filter: `tcp.flags==0x18`).
3. Repeatedly sent packets should only be recorded once. *Wireshark* also offers a filter support for that: `!tcp.analysis.retransmission`.

This recorded and filtered packet data is logged in a text file by *Wireshark*. Each entry contains the original data of the network packets plus a time stamp representing the time the packet was observed. We will call these files *TCP logs* in the sequel.

TCP logs are not yet suitable for process mining, because important characteristics of the entries, namely case identifier, resource and activity, are still missing [4]. Moreover, TCP logs store data in hexadecimal form which requires translation to suitable ASCII character strings.

For completion of the data, we (i.e., our tool *TCPLog2EventLog*) refer to the different layers of the TCP/IP protocol: Ethernet-Frame, IP-packet, TCP segment and application, see Figure 2. All this information, and additionally a time stamp representing the recording time, is available in TCP logs and will be used for generating suitable log entries for process mining.

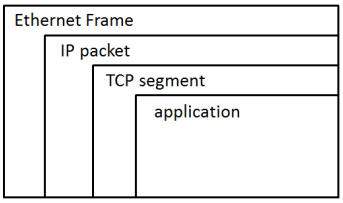

Fig. 2. Schematic structure of an internet packet

- As *time stamps* of event logs, we take the time stamp (recording time) of the TCP log.
- The *resource* is taken from the Source Address of the IPv4 header (IP protocol according to RFC 791, see Figure 3).
- The TCP header (TCP protocol according to RFC 793, see Figure 4) contains, among others, a Sequence Number, an Acknowledgment Number and a Data Field. According to the TCP protocol, Sequence and Acknowledgment Numbers ensure safe transmission of packets between sender and receiver as follows: The first Sequence Number is chosen arbitrarily by the respective host. Each subsequent packet of the same communication uses as Sequence Number the Sequence Number of the previous packet, increased by the length of the Data Field of the previous packet (in bytes). When the receiver answers to a packet, the Acknowledgment Number equals the Sequence Number of the subsequent message of the sender, calculated as above. Using these numbers, we uniquely assign each entry of the TCP log to a conversation between a particular client and a particular server. These conversations receive subsequent, and hence distinct, numbers in our approach. Each conversation number constitutes the *case id* of the respective log entries.
- To obtain *activities* in event logs we use the Data Field of the IP protocol, as discussed below.

As a result of the decision to use the Data Field entries as activity names, very similar tasks might be represented by different activity names. For example, registration activities of different users are not recognized as the same task. This is not desirable. Moreover, this data, taken directly from the TCP header definitions, is unreadable (for humans). Therefore, in a further step we group and classify similar activities (such as all registration activities), and give these groups readable names that eventually should appear in the process model.

To this end, *TCPLog2EventLog* uses activity patterns, each with a semantics and an appropriate name. These patterns are obtained by a previous procedure: We perform atomic user activities within the application and thus within the protocol to be mined (pressing single keys, moving the mouse etc.). The resulting traffic is recorded as described above. Since the obtained logs only refer to the atomic activity, we can deduce which bit strings in the Data Field describe certain single activities, and we assign an appropriate name to this pattern. This

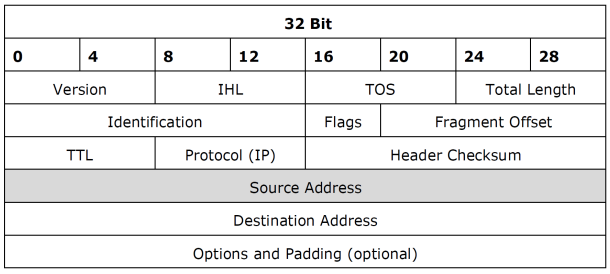

Fig. 3. IPv4 header

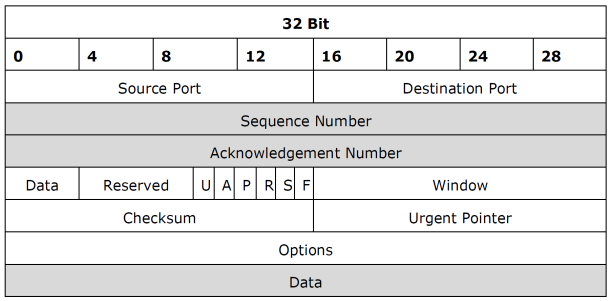

Fig. 4. TCP header

pre-processing is supported by *TCPLog2EventLog*, too. The derived patterns are collected in a template file for further processing.

More technically, the output data of *TCPLog2EventLog*, i.e., the event log without activity classification, is stored and will later be used to identify activity classes. Therefore, each record receives a class name so that this pattern file has the following format: <data of the packet>; <name of the activity class>

Whereas the classification of the activities might contain errors, there is one important attribute of each activity that is easily derived from the data, namely the respective sender of a message (server or client). To capture this information, each class name receives the prefix `Server:` or `Client:`.

The following table summarizes the preliminary transformation of TCP logs to event logs.

TCP log	**event log**
time stamp	time stamp
sequence number/ acknowledgement number	process ID
source IP-address	resource
abstracted user data via template identified class	activity

After creation of activity class names, it remains to assign single activities to according classes. This is done using the above mentioned template file. Since the Data Field entry of a packet is not identical to the previously found patterns for known activities in general, we implemented the Levenshtein Algorithm [8], which studies similarities between strings of characters, in *TCPLog2Eventlog*. If

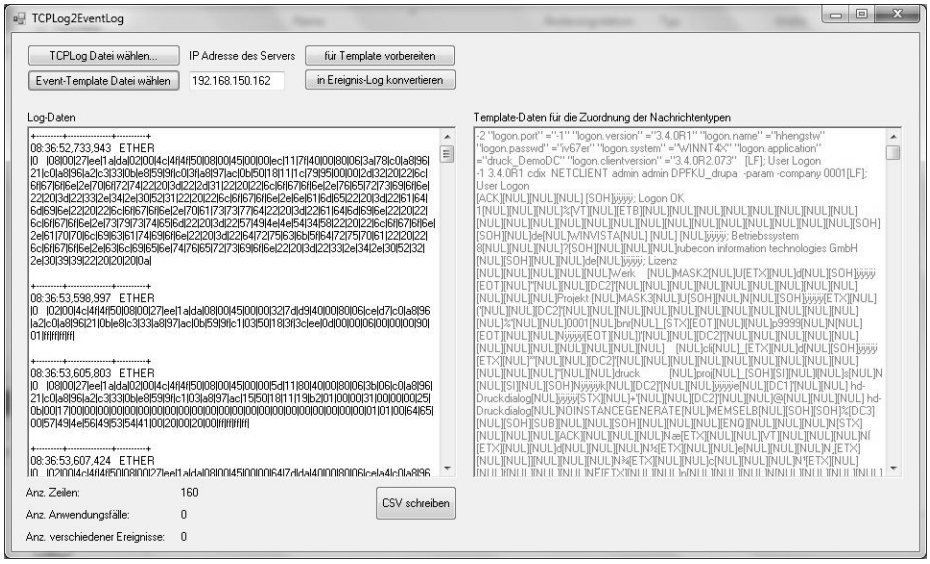

Fig. 5. User interface of *TCPLog2Eventlog*
see also https://wiki.fernuni-hagen.de/sttp/images-sttp/d/d6/Figure5.jpg

the Data Field entry of a TCP log entry is identified similar to a known Data Field then the respective activity is assigned to the respective class, and the existing array is extended by the new name. If no assignment is possible, then the result will be a new activity class `Answer xx`, where `xx` is a consecutive number. It is desirable to have not too much of these exceptional activities.

Finally the program *TCPLog2Eventlog* outputs the data for process ID, time stamp, resource and activity as a CSV file. Figure 5 shows the user interface of *TCPLog2Eventlog*. The right side shows a template file, the left side a generated event log of an imported TCP log.

The above mentioned prerequisites for event logs (for process mining) have been standardized in form of the *Meta Models Mining eXtensible Markup Language (MXML)* in [4]. The *eXtensible Event stream (XES)* [6] standard, developed by the IEEE task Force on Process Mining, can be viewed as a successor approach that better supports extensibility. Both approaches are supported by today's process mining tools, and both are based on XML.

So our pre-processing ends with a conversion of the event log to the XES format, which will serve as input format of mining tools such as *ProM* [12]. This conversion is done by the commercial tool *Nitro* developed by *Fluxicon*.

3 Mining of Event Logs

There is no *best* process miner, each miner and each mining algorithm has particular advantages and drawbacks. The selection of a process miner primarily

depends on the goal of mining. For our purposes, the following mining algorithms (all available via ProM[3]) seem to be suitable. In this list, we refer to the *ProM Tips* provided in [13].

- **Alpha Miner**. A simple algorithm based on dependency relations between events. Needs ideal event logs without noise. Improved by the α^+-miner, that also handles "short loops", i.e., mutual dependencies between events. The output is a Petri net. Not recommended in [13] for real logs, because highly sensitive w.r.t. quality of logs.
- **Heuristics Miner**. Based on the mechanisms of the Alpha Miner, but additionally takes the number of identified relations into account and is therefore more robust against noise. The Heuristic Miner in ProM produces a so-called Heuristic Net, which can easily be transformed in a normal Petri net. Its use is recommended for real data with a limited amount of different activities, in particular if a Petri net is desired [13].
- **Fuzzy Miner**. Good if behavior shows little structure and has many different activities. Can simplify very complex data or a very complex model. The output is a so-called Fuzzy Model, which is not easily transformed in another language.
- **Multi Phase Miner**. Aims at constructing event-driven process chains with logical connectors, as used by the ARIS tools. Needs noise-free input.
- **Genetic Miner**. Based on genetic algorithms, this miner can cope with incomplete logs and noise, resulting in a Petri net or in a Heuristic Net.

The Process Mining Manifesto [3] defines five maturity levels of event logs, from the highest level 5 of excellent quality (correct and complete) to the lowest level 1 (missing events, wrong events). Before selecting process miners to be considered further, we classify the event logs of our setting. The maturity of our event logs can be assumed to be on level 3 (automatic recording, missing completeness, little noise).

The Multi Phase Miner seems inappropriate for our setting because it results in an event driven process chain whereas other miners construct Petri nets and are thus better comparable. Moreover, the Multi Phase Miner assumes a particularly high quality of the input event logs which we cannot guarantee.

The Fuzzy Miner has the advantage to provide abstraction of activities dynamically, i.e., distinct activities that always appear in the same context, are embraced. Since in our procedure this step was done just before mining, it won't turn out to be helpful again.

So we apply the mining algorithms Alpha Miner, Heuristic Miner and Genetic Miner in the sequel and will compare the respective results in the next section.

4 Case Study

This work is motivated by a real application. The software company *rubecon information technologies GmbH* [9] is specialized on software solutions for the

[3] Our study is based on ProM 6.2

printing industry. One of its products is the software *hd-druckdialog* which is based on the CDIX Business Application Framework. This product was migrated to Windows. The precise implementation of the windows client is unknown, and so is the protocol. Recently, a new software client for the CDIX protocol was developed, called *SmartClient*. Its behavior is *not* based on the mining approach of this paper. Tests showed that it slightly differs from the original protocol.

The current aim is to reconstruct the original protocol from observed behavior and to formalize it by means of a model, compare this model with a model of the *SmartClient* implementation and to identify and explain differences.

For the sake of this case study, we derive a model for a particular use case. This has the advantages that the model is not too complex and that we have a better chance to assign activities to classes correctly. A disadvantage is that the usable input data for mining is comparably small. If the mining procedure is successful, the use case (and thus the model) can be extended.

The simple use case in our experiment is given as follows: The software *hd-druckdialog* is started; then the user authenticates herself, and then the software is terminated. Its extension opens after successful athentification the customer management windows, which requires navigation in the menu structure of the software. Also in this use case, the software is subsequently terminated.

These use cases were performed a number of times, using the classical protocol implementation. In doing so, possible variants of the user interface were chosen, such as a wrong password or different navigation paths, so that this part of the communication protocol is captured as complete as possible. The traffic was recorded and the steps described in the previous sections were performed, leading to a minable event log.

The software *TCPLog2EventLog* determines the number of use cases that appear in the present event log. The further considerations only make sense, if this number coincides with the real number (two in our case). If this number deviates, the log is too faulty and can not be used. A further criterion for the quality of the log is the appearance of xx-activities, i.e., of activities that were not mapped to an activity class. If this is the case, either the generation of the template file was insufficient or the event log contains activities not known before, which can also be due to non-conformity reasons.

In our case study, we have a comparable small amount of use cases and hence can assume that the event logs are incomplete. On the other hand, since we constructed the event logs carefully manually, we assume lack of noise. So we tried the Alpha Miner first, yielding the model shown in Figure 6. All server activities are located in the upper part of the figure, all client activities in the lower one. The transitions of this (Petri net) model are inscribed by the sender of a message, followed by the content type. We checked completeness of this model , i.e., found out that all use case activities are possible. Subsequently, we did the same for the extended use case and checked whether the new model is in fact an extension of the first one. This model is shown in Figure 7. It turned out to be complete as well. A conformance check by ProM additionally confirmed that in fact the first model is included in the second, because the log of the first

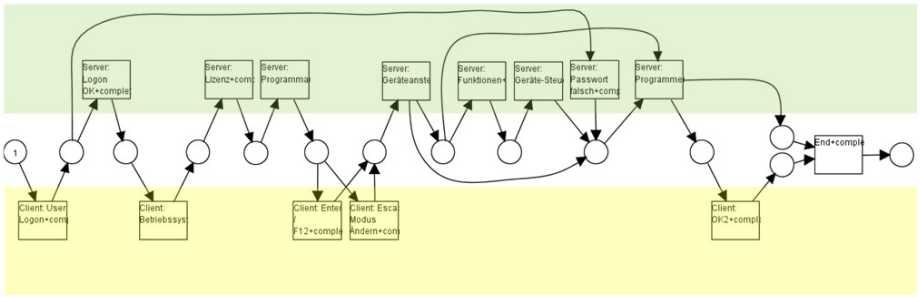

Fig. 6. Communication of registration process within the original windows client see also https://wiki.fernuni-hagen.de/sttp/images-sttp/0/05/Figure6.jpg

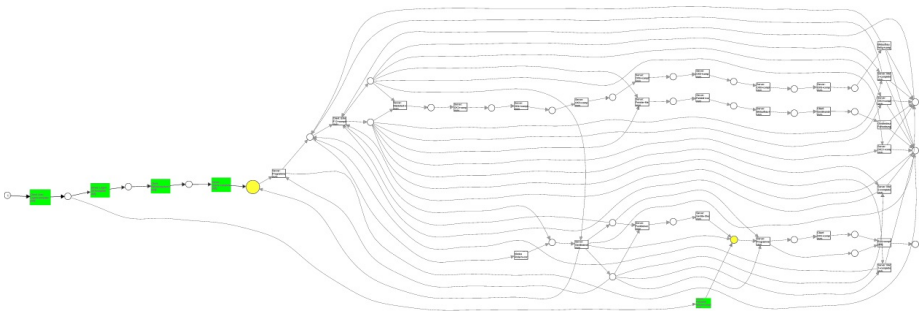

Fig. 7. New model: the extended use case (in color the replay of the event log of the simple use case), see also https://wiki.fernuni-hagen.de/sttp/images-sttp/9/90/Figure7.jpg

example can be played on the second model, too (the inclusion of the first model is depicted in Figure 7).

This example already shows that in principle process mining is applicable to recover protocols from network traffic. We also tried other mining algorithms, with the following results: The Heuristic Miner yields a structure shown in Figure 8, which is semantically equivalent to the result of the Alpha Miner. The Genetic Miner derives a considerably different model, shown in Figure 9 (the differences are depicted). This model is not conform to the event log.

A final step of our case study is the comparison of the obtained model derived from the classical protocol with the new protocol *SmartClient*. To this end, we constructed a model of this new protocol (shown in Figure 10) for the single use case in the same way as for the classical protocol. Both protocols have obvious differences, depicted in the figure, and they also behave differently:

- After the program start, SmartClient maximizes the window size.
- The classical client distinguishes three possibilities to terminate the program, SmartClient has only one such possibility.
- Only SmartClient sends no confirmation after program termination.

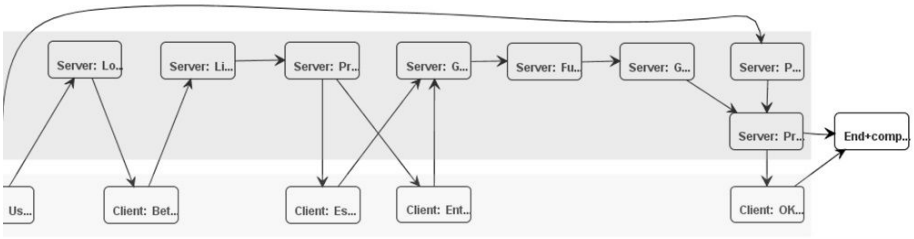

Fig. 8. Result of the Heuristic Miner
see also https://wiki.fernuni-hagen.de/sttp/images-sttp/a/a3/Figure8.jpg

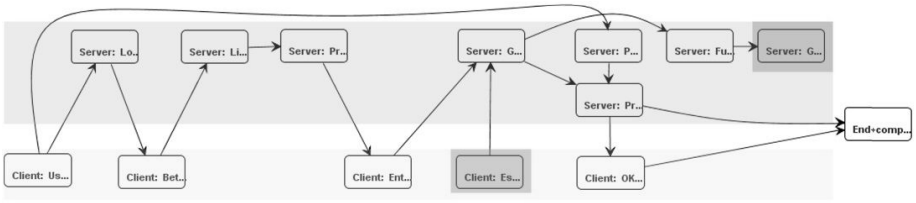

Fig. 9. Result of the Genetic Miner
see also https://wiki.fernuni-hagen.de/sttp/images-sttp/1/17/Figure9.jpg

We finally checked whether SmartClient is still conform to the classic client. Since also the classical protocol allowed to change window size, this behavioral derivation does not spoil correctness. The other differences, however, are more serious although they do not lead to any misbehavior observable by the user.

5 Conclusion

This paper developed a concept to derive event logs from network traffic, which makes it possible to apply process mining techniques and tools for protocol mining. The concept has several steps, many of them supported by a novel software program. A use case from practice proved the concept to work in general.

A different approach would have been to view clients and servers of our setting as respective loosely coupled services. Then many already existing tools for services could have been applied, such as *FIONA*, *WENDY*, *LOLA* or *BPEL2OWFN* [10]. The underlying formalism, *Open Petri Nets*, would have been an ideal language for protocol definitions. However, all these tools, as well as the underlying techniques, start with services modeled in the *Business Process Execution Language (BPEL)*. Since we could neither derive service definitions formulated in BPEL nor open workflow nets with defined message exchange activities by application of the mining tools, this approach was not (yet) viable.

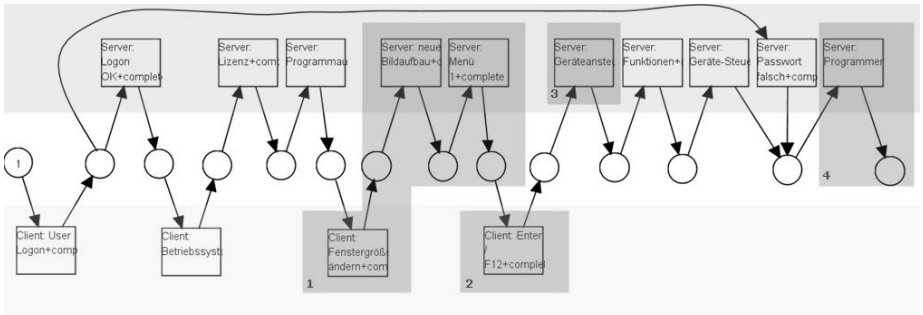

Fig. 10. Model of the communication process of the simple use case of *SmartClient* (differences of the models depicted)
see also https://wiki.fernuni-hagen.de/sttp/images-sttp/b/bc/Figure10.jpg

References

1. Wil M. P. van der Aalst. Service mining: Using process mining to discover, check, and improve service behavior. *IEEE T. Services Computing*, 6(4):525–535, 2013.
2. Wil M. P. van der Aalst and Boudewijn F. van Dongen. Discovering Petri nets from event logs. http://wwwis.win.tue.nl/ wvdaalst/publications/z2.pdf, 2013. [Online; accessed 10-May-2013].
3. Florian Daniel et al. Process Mining Manifesto. In *BPM Workshops, Part I*, volume 99 of *Lecture Notes in Bunsiness Information Processing*, pages 169–194. Springer, 2012.
4. Boudewijn F. van Dongen and Wil M. P. van der Aalst. A meta model for process mining data. In *EMOI-INTEROP*, 2005.
5. Schahram Dustdar and Robert Gombotz. Discovering web service workflows using web services interaction mining. *Int. J. Business Process Integration and Management*, 1(4):256–266, 2006.
6. Christian W. Günther and Erik Verbeek. Xes standard definition. http://www.xes-standard.org/meida/xes/xesstandarddefinition-1.4.pdf, 2012. [Online; accessed 26-April-2013].
7. Hartmut König. *Protocol Engineering*. Springer, 2012.
8. Vladimir I. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. Doklady Akademii Nauk SSSR, 163(4), 1965.
9. rubecon information technologies GmbH. hd-druckdialog. http://www.rubecon.de, 2013.
10. service technology.org. http://service-technology.org/tools/start, 2013. [Online; accessed 10-May-2013].
11. Wireshark Foundation. Wireshark version 1.6.3. http://www.wireshark.org.
12. ProM. process mining workbench. http://www.promtools.org.
13. Anne Rozinat. ProM tips – which Mining Algorithm sould you use? http://fluxicon.com/blog/2010/10/prom-tips-mining-algorithm, 2012. [Online; accessed 28-April-2013].
14. Christian Wakup. Konzeption und Evaluation eines Verfahrens zur Ermittlung der internen Funktionsweise eines TCP/IP-basierten Kommunikationsprotokolls mit Hilfe von Process-Mining. Bachelorarbeit, FernUniversität in Hagen, 2013.