Verification of Logs - Revealing Faulty Processes of a Medical Laboratory

Robin Bergenthum and Joachim Schick

Department of Software Engineering and Theory of Programming
FernUniversität in Hagen
{robin.bergenthum,joachim.schick}@fernuni-hagen.de
http://www.fernuni-hagen.de/sttp

Abstract. If there is a suspicion of Lyme disease, a blood sample of a patient is sent to a medical laboratory. The laboratory performs a number of different blood examinations testing for antibodies against the Lyme disease bacteria. The total number of different examinations depends on the intermediate results of the blood count. The costs of each examination is paid by the health insurance company of the patient. To control and restrict the number of performed examinations the health insurance companies provide a charges regulation document. If a health insurance company disagrees with the charges of a laboratory it is the job of the public prosecution service to validate the charges according to the regulation document.
In this paper we present a case study showing a systematic approach to reveal faulty processes of a medical laboratory. First, files produced by the information system of the respective laboratory are analysed and consolidated in a database. An excerpt from this database is translated into an event log describing a sequential language of events performed by the information system. With the help of the regulation document this language can be split in two sets - the set of valid and the set of faulty words. In a next step, we build a coloured Petri net model corresponding to the set of valid words in a sense that only the valid words are executable in the Petri net model. In a last step we translated the coloured Petri net into a PL/SQL-program. This program can automatically reveal all faulty processes stored in the database.

1 Introduction

A lot of information systems are used in the healthcare sector and each system produces some kind of log-data. This is particularly true in the domain of medical laboratories where all samples, materials and examination results have to be stored. This "good laboratory practice" is an important method of quality management and big medical laboratories own records about several millions of processed orders.

Every examination performed by a medical laboratory is paid by a health insurance company. The cost of each examination is rated by a fixed scale of charges given in a so-called charges regulation document. Of course, the correct

application of the regulations have to be proven to the health insurance companies. If a suspicion about irregular application of the regulations arises, it is the job of the public prosecution service to validate the billed charges according to the regulation document. Usually, the prosecution service orders a report investigating the issue from an expert-office.

In this case study we describe an approach using coloured Petri nets which is inspired by the methods of the area of process mining and process discovery to reveal faulty processes given in log-files of a medical laboratory. The files contain data recorded over a period of five years having 1500-2000 orders a day. Each order consisting of 20-30 events, examinations and results. Altogether, we face about 100 million lines of log that need to be analysed and verified. Each line describes an event or a sub-process of the medical laboratory. Each event refers to the occurrence of an action of the information systems and is annotated with a time stamp, order-id, variables etc. Typical actions of the system are *register order*, *register requirements*, *register examination results*, *validate results*, *make invoice*, *archive order* . . . . In addition to these basic actions, a medical laboratory is able to perform a huge number of different examinations. In this case study the prosecution service ordered a report revealing all faulty processes concerned with Lyme disease.

To reveal all faulty processes of a set of log-files we choose a four step approach. We call the first step *consolidation step*. The main goal is to develop a schema to integrate all the recorded files into a relational database. Using the same schema it is also easy to implement a view on top of the database tables. The view abstracts from redundant or superfluous information and reduces the data to events and results corresponding to processes considering Lyme disease. With the help of this view we are able to produce an event log, i.e. a sequence of events bearing only information about order-number, time-stamp and result.

The next step is called the *formalization step*. Each sequence of events corresponds to a sequence of actions. Each sequence of actions is called a word. The set of words is called the language of the event log. The main task in the formalization step is to split this language into two sublanguages, the set of valid and the set of faulty words. This has to be done manually with the help of the charges regulation document. Of course, this is a time consuming task, but we believe that it is very easy and hardly error-prone to classify single words. We could also try to directly build a model of regulations from the charges regulation document to classify the set of words automatically, but often the regulation document is given as plain text. Starting from such a description is error-prone and easily yield a model that does not fit the recorded event log regarding names of actions, values and level of abstraction. Remark, we only need to partition the set of words, we do not classify the complete event log. In the formalization step a set of valid sequences of actions is produced. We call this set the language of regulations.

The third step is called *integration step*. The language of regulations is integrated into a coloured Petri net model. Such an integration can be supported by synthesis or workflow mining algorithms. In our case study the language of

regulations is already highly compressed and settled, such that we construct a corresponding coloured Petri net model by hand using the editor CPN-Tools [1]. The constructed coloured Petri net model is a formalization of the charges regulation document using the language of the recorded files. Only valid process instances of the Lyme disease diagnostic processes are executable in this Petri net model. A big advantage of such a Petri net model is that it can be analysed, simulated and verified.

The fourth step is called *implementation step*. Coloured Petri nets are well readable and have an intuitive formal semantic. We will show how to translate such a coloured Petri net model into a PL/SQL-program. We translate transitions to functions, places to tables and arcs to delete or insert statements. With the help of such a PL/SQL-program all sequences of events can be replayed in the database. If the replay fails, the sequence corresponds to the occurrence of a faulty process of the medical laboratory.

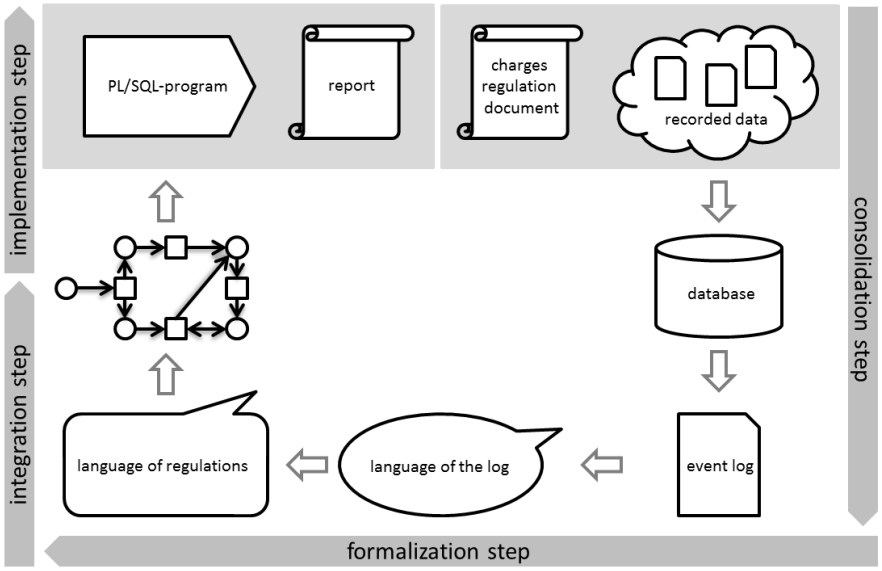

Fig. 1. Approach to reveal faulty processes

Figure 1 depicts an overview of the presented approach. A key feature is that it is built on a chain of formal models. The initial models, i.e. the schema, the view and the event log, consolidate the recorded data. Afterwards, the language of regulations, the coloured Petri net and the PL/SQL-program are build. The constructed models document of the whole inspection procedure, all results can easily be reconstructed, the produced models can be reused when inspecting other laboratories. Of course, stepping from one formal model to another highly supports the validity of the investigation report produced. Each step can be

supported by algorithms and tools. Some steps can even run fully automated using e.g. synthesis algorithms for the construction of the Petri net model or automated generation of the PL/SQL-program.

The chosen approach is inspired by techniques well known in the area of process mining where some recorded behaviour is merged into a formal model of the underlying process [2–4]. Remark, that it is of great importance to choose an appropriate process mining algorithm that does not introduce much additional behaviour to the model. There are language base discovery algorithms [5–7] or even synthesis algorithms [8–11] that meet this requirement. The approach is also inspired by work done in the field of business process modelling and requirements engineering were the starting point of the discovery phase is the construction of a formal and valid specification [12–16]. Nevertheless, there are two major points that are unusual to approaches known in both areas. We model the process of the underlying system by coloured Petri nets since they highly depend on the intermediate results of a chain of different blood examinations. In addition the formal language of the event logs needs to be filtered by hand according to the charges regulation document. This step can not be automated and is crucial for the quality of the report produced.

The paper is organized as follows: Section 2 provides formal definitions. Section 3 presents the approach and our case study. In Section 4, we sum up the results to prove the applicability of the developed approach.

2 Preliminaries

In this section we briefly recall the basic notions of languages, event logs and coloured Petri net.

An alphabet is a finite set A. The set of words over an alphabet A is denoted by A^*. The empty word is denoted by λ. A subset $L \subseteq A^*$ is called language over A.

Business processes describe the flow of work within an organisation [17]. Each process consist of a set of activities that needs to be performed. We denote T the set of all activities and call the execution of an activity an event. Events are labelled with the name of the corresponding activity. Furthermore, events can carry a time stamp showing the time of execution and values denoting results of the execution. We denote V the set of values. A set of events corresponding to the occurrence of a processes is called a case. Recording the behaviour of a system yields a set of interleaved cases we call an event log.

Definition 1. *Given a finite set of activities T, a finite set of values V and a finite set of cases C. An element $\sigma \in (T \times V \times C)^*$ is called an event log. Fix a case $c \in C$ we define the function $p_c : (T \times V \times C) \to (T \times V)$ by*

$$p_c(t, v, c') = \begin{cases} (t, v) & ,\text{if } c = c' \\ \lambda & ,\text{else.} \end{cases}$$

Given an event log $\sigma = e_1 \dots e_n \in (T \times V \times C)^*$ we define the language $L(\sigma)$ of σ by $L(\sigma) = \{p_c(e_1)...p_c(e_i) | i \leq n, c \in C\} \subseteq (T \times V)^*$.

The language of an event log is finite and prefix closed. It reflects the control flow between activities given by the events of the log. Each case adds a word to the set of words called language.

In this paper we use coloured Petri nets to model valid behaviour of a medical laboratory. The underlying Petri net models the control flow between actions while variables control the examination results. The following definition of coloured Petri nets was given in [1].

Definition 2. *A coloured Petri net is a tuple* $CPN = (P, T, F, \Sigma, V, D, G, E, I)$, *where:*

- P *is a finite set of places.*
- T *is a finite set of transitions, such that* $P \cap T = \emptyset$ *holds.*
- $F \subseteq (P \times T) \cup (T \times P)$ *is a set of directed arcs.*
- Σ *is a finite set of non-empty colour sets.*
- V *is a finite set of typed variables such that* $Type[v] \in \Sigma$ *for all variables* $v \in V$.
- $D : P \to \Sigma$ *assigns a colour set to each place.*
- $G : T \to EXP_V$ *assigns a guard to each transition* t *such that* $Type[G(t)] = Bool$.
- $E : F \to EXP_V$ *assigns an arc expression to each arc* f *such that* $Type[E(f)] = D(p)_{MS}$, *where* p *is the place connected to the arc* f.
- $I : P \to EXP_0$ *is an initialisation expression to each place* p *such that* $Type[I(p)] = D(p)_{MS}$.

In contrast to low-level Petri net a place of a coloured Petri net belongs to a given type called colour. According to this colour each place carries values called tokens. Arcs carry variables and if an arc is connected to a place, the tokens of the place can bind to variables of the arc. A binding b of a transition maps variables of related arcs into values of related places. A transition t is executable if there is a binding b such that the transition guard evaluates to $true$. When the transition occurs, as for low-level Petri net, it removes the specified tokens from the input places and produces tokens in the output places (see [1] for a formal definition).

The initialisation function I assigns tokens to places yielding an initial marking. Given a coloured Petri net CPN a sequence of sequential enabled transitions is called an occurrence sequence of CPN. In this paper we add the values of the respective bindings to each transition of an occurrence sequence. The language $L(CPN)$ of CPN is defined as the set of all occurrence sequences. Given an event log $log \in (T \times V \times C)^*$, log is executable in CPN if $L(log) \subseteq L(CPN)$ holds.

3 Verification of Logs

In this section we present an approach to validate a set of given recorded files with the help of a regulation document. In the following case study, on behalf of the public prosecution service, recorded data of an information system of a

medical laboratory has to be reviewed. During a period of five years 1800 files were produced and recorded. Each file contains about 1500 processed orders. The regulation document is given by a charges regulation document provided by health insurance companies. The goal is to identify faulty processes performed by the medical laboratory considering all processes corresponding to Lyme disease diagnostic. An overview of our approach is sketched in Figure 1 given in the introduction. The subsections of this section reflect the four steps of our approach.

3.1 Consolidation Step

In a first step the recorded files need to be consolidated and formalized. The aim of this step is to load the recorded files into a database to extract an event log from it afterwards. For the storage and processing of data, the commercial Oracle Database is used. This database system provides a procedural programming language named PL/SQL for the implementation of the stored procedures. To set up the database an entity-relationship diagram is produced. Of course, to produce this diagram first the recorded files need to be reviewed. Afterwards, we use the Oracle SQL Developer Data Modeler to construct the model.

An excerpt of a file recorded by the medical laboratory is depicted in Figure 2. All files of the laboratory's information system have a hierarchical structure with a flexible record length up to 1024 characters. Each file is a sequence of different types of blocks. Each block corresponds to a set of different actions of the system. The first line of each block is the header of the block and all following lines are indented.

The file depicted in Figure 2 starts with a block corresponding to the registration of a new order for a blood count. The header of this block reads as follows: The first number corresponds to the registration-id 727980834 generated for this new order. This id perfectly fits the need to identify cases in the given file. In our case study each registration-id corresponds to a case of the system. The next two numbers refer to the time the registration occurred, i.e. January 25th 2011, 11:49:54 in our example. The next two strings indicate that this action was manually triggered. The last number of the header encodes the name of the action occurred. In this particular information system the number 10 refers to the action *order blood count*. The inner lines of this first block carry the values of this registration action. Possible values are the name, birthday and address of the patient registered.

The next block corresponds to the scheduling of examinations. The header refers to the same case as the first registration block since both ids match. Remark, both recorded actions even occurred within the same second. The difference between both headers is only given by the number at the end of the line. In this block 20 refers to the action *schedule examination*. This block consists of two sub-blocks, both sub-blocks marked by the keyword BORR. BORR stands for Lyme disease and indicates that the scheduled examinations are part of Lyme disease diagnosis. Again, the inner lines carry values of the scheduling where BORG and BORM are abbreviations of two different blood examinations.

```
727980834       250111  114954  SF      erfass  10
        NAME    [redacted]
        GEBDAT  06.01.1979
        ...
727980834       250111  114954  SF      erfass  20
        BORR    BORG            0000
                RESTYPE W
                ST_BA[1]        X
                ST_BA[4]        X
                MVALBER LA
                UNTVERS 0000
        BORR    BORM            0000
                RESTYPE W
                ST_BA[1]        X
                ST_BA[4]        X
                MVALBER LA
                UNTVERS 0000
702673748       250111  115004  SF      erfass  10
        NAME    ...
        ...
702673748       250111  164235  MB      onlval  21
        JB      FT3
                WERT    2.8
                ...
702984083       250111  174847  MB      onlval  21
        TSH1    TSH1
                WERT    0.63
        ...
727980834       270111  123344  US      onlval  21
        BORR    BORG            0000
                WERT    < 10.0
                ST_RES  J
                ST_BA[1]        R
                ST_BA[4]        R
                ST_MVAL J
        BORR    BORM            0000
                WERT    < 18.0
                ST_RES  J
                ST_BA[1]        R
                ST_BA[4]        R
                ST_MVAL J
        ...
727980834       270111  195346  AB      abschl  13
        ...
727980834       280111  071406  HW      rechdr  11
        RNR     KV110128
        ROK     OK
```

Fig. 2. An excerpt of a recorded file of the medical laboratory.

In this example the block corresponds to the occurrence of two different actions. A BORG-examination and a BORM-examination is scheduled.

The sixth block shown in Figure 2 corresponds to the recording of results of the scheduled examinations. The number 21 refers to the action *receive result*. This block matches the schedule examination block besides two important differences. First, the keyword ONLVAL indicates that this event was automatically triggered by the information system when the results of examinations are received. Second, the inner lines of the block carry the results of these examinations. In this example the results of the BORG- and the BORM-examinations are received. The value of the BORG-examination is smaller than 10.00 and the value of the BORM-examination is smaller than 18.00. Both values show the absence of the corresponding antibodies, i.e. both examinations are negative and no further examinations need to be scheduled.

After knowing the structure of the files an entity-relationship diagram is built. With the help of this schema a PL/SQL-program is written to load all files into the Oracle Database. If all the data is stored, the next step is to extract a consolidated and formal event log from this database. The event log only contains events and values corresponding to processes that need to fulfil regulations given in the charges regulation document concerning Lyme disease diagnostic. We omit a detailed description of the produced entity-relationship diagram, but give a short impression in Figure 3.

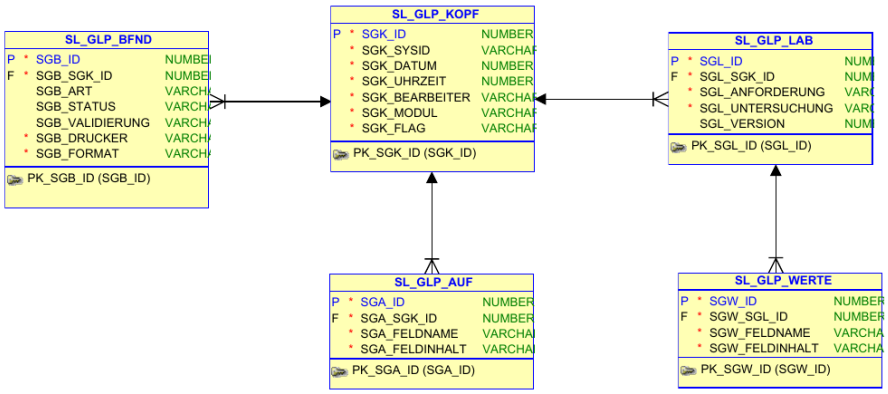

Fig. 3. Entity-relationship diagram of our Oracle Database.

Given the entity-relationship diagram it is easy to implement a view on top of the tables of the database to receive an appropriate event log. In our example, the excerpt depicted in Figure 2 only contains four blocks corresponding to Lyme disease diagnostic. In the first and in the third block two new orders arrived and both patients are registered. In the second block a BORG- and a BORM-examination for the first order is scheduled. The sixth block shows the results

of both examinations. We are able to discard all other blocks shown in Figure 2. If we apply the constructed view to this excerpt we get the event log shown in Table 1. This event log abstracts from additional events and values. It shows the six events corresponding to the four blocks concerned with Lyme disease of Figure 2.

id	action	value	stamp
727980834	10		25.01.11, 11:49:54
727980834	20 BORG		25.01.11, 11:49:54
727980834	20 BORM		25.01.11, 11:49:54
702673748	10		25.01.11, 11:50:04
727980834	21 BORG	< 10	27.01.11, 12:33:44
727980834	21 BORM	< 18	27.01.11, 12:33:44

Table 1. Event log of the file depicted in Figure 2.

With the help of the Oracle Database and the implemented view arbitrary extracts of the recorded files can be shown as event logs. These logs are the results of the consolidation step of our approach. In the next steps these logs are filtered with the help of the regulation document and integrated to an executable model.

3.2 Formalisation Step

In the second step of our approach first the event log is used to define the formal language of the recorded behaviour. Then, in a next step, this behaviour is filtered with the help of the charges regulation document yielding a language of valid words. The aim of this formalisation step is to bring together the recorded behaviour and the regulation document given as plain text. Remark, that it is much easier to only evaluate the recorded language with the help of the regulations and not to build an independent model of all regulations hoping it will fit the language of the recorded behaviour.

To deduct a formal language from the event log, first the actions of the system need to be identified. In our case study the list of significant actions reads as follows:

$$\begin{aligned} T = \{&10, 20BORG, 20BORM, 20BVLSEG, 20BP39G, 20BP83, 20BIV1, \\ &20BIV2, 20BIV3, 20BIV4, 20BOSPC, 20BVLSEM, 20BP39M, \\ &21BORG, 21BORM, 21BVLSEG, 21BP39G, 21BP83, 21BIV1, \\ &21BIV2, 21BIV3, 21BIV4, 21BOSPC, 21BVLSEM, 21BP39M\} \end{aligned}$$

The numbers 10, 20 and 21 indicate if a blood count for a patient is registered, an examination is scheduled or if a result is received. The attached letters are

the abbreviations of the corresponding examinations. There are 12 different tests corresponding to Lyme disease diagnostic leading to 25 different actions in total. Every action having a name starting with 21 carries a value of the type *boolean* (i.e. either the examination is negative or positive). At this point we are able to abstract from any other value given in the files such that any other action occurs without additional data. As stated above all events having the same registration-id belong to the same case. Events of the same case can be ordered by their time stamp. If we apply this knowledge to our event log we get a set of words. The following table shows three example words given by the event log of our case study:

$L(log) = \{$10 20BORG 20BORM (21BORG,false) (21BORM,false),

10 20BORG 20BORM 20BVLSEG 20BP39G 20BP83 20BIV1 20BIV2 20BIV3 20BIV4 20BOSPC 20BVLSEM 20BP39M (21BORG,true) (21BORM,true) (21BVLSEG,false) (21BP39G,false) (21BP83,false) (21BIV1,false) (21BIV2,false) (21BIV3,false) (21BIV4,false) (21BOSPC,false) (21BVLSEM,false) (21BP39M,false),

10 20BORG 20BORM 20BVLSEG 20BP39G 20BP83 20BIV1 20BIV2 20BIV3 20BIV4 20BOSPC 20BVLSEM 20BP39M (21BORG,false) (21BORM,false) (21BVLSEG,false) (21BP39G,false) (21BP83,false) (21BIV1,false) (21BIV2,false) (21BIV3,false) (21BIV4,false) (21BOSPC,false) (21BVLSEM,false) (21BP39M,false),

...

Table 2. The language of the event log.

The language depicted in Table 2 was automatically processed from the given event log. This language is a complete and formal description of the set of processes occurred in the information system of the medical laboratory. Any new sequence of actions and values given by the events of a case yields a new word in the language of the log. Of course, the language of the log is much smaller than the event log since cases corresponding to the same process are not distinguished.

Given the language of the log the next step is to distinguish valid and faulty words. This is a major task in the presented approach which can not be automated. The charges regulation document is given as text. It is absolutely necessary to understand the given regulations and apply them to the set of words. The main advantage of the presented approach is that the set of words is given in a very compact and formal style. There is no room for interpretations or ambiguities. The rules of the charges regulation document do not need to be modelled explicitly, they just need to be applied to the given language. As stated in [13, 16] a single word is much easier to understand than a whole system. The evaluation of single words can be performed by experts on the regulation document. There is no need that these experts know how to model a system or even can read the files or know how the information system works.

In our example given in Table 2 the first two words are valid. The third word is faulty since the set of examinations {BVLSEG, BP39G, BP83, BIV1, BIV2, BIV3, BIV4, BOSPC, BVLSEM, BP39M } may only be preformed if one of the BORG- and BORM-examination is positive. According to the regulation document the blood count needs to be performed in two steps. First, the BORG- and BORM-examination results need to be evaluated, if one of these is positive, a more detailed set of examinations should be performed.

The result of the formalization step is the set of valid words. This set can be seen as the relevant part of the language of the charges regulation document given in the language of the information system. If this language is found, the most challenging task of the investigation process has been completed. In the next steps this set is integrated into an executable model.

3.3 Integration Step

The third step of our approach is called integration step. The aim is to build an executable model having the language of the charges regulation document. As suggested in [18] it would be possible to skip this integration step and just filter the event log with the help of the set of valid words given by the language constructed in the former step, but there are mainly two important reasons to build an integrated model first. A model provides a more compact representation of the set of words such that the model can more easily be simulated and analysed. For this purpose there exist a lot of well known Petri net algorithms in the literature. Second, an executable model can easily be translated into executable code in the last step of our approach.

The problem of integrating a set of words into a Petri net is a well known problem. There exists a lot of work tackling the problem in the area of process mining [2, 19, 20, 7] and in the area of language based synthesis [8, 21, 11, 9, 6]. Algorithms from both areas can by applied to support the integration step. In the presented case study we built the corresponding model by hand. The constructed language of the charges regulation document was already compressed in such a way that there was no need for automated integration. At first, a transition is constructed for every action of the given language. According to the ordering of actions given in the language places are added to this set of transitions such that only words of the language are executable in the resulting net. In a second step the values carried by actions yield coloursets added to the constructed Petri net. Variables are added to arcs connected with the respective transitions corresponding to actions carrying a value. The coloured Petri net is adjusted in such a way that each pair of an action and value given in the language corresponds to a transition and a binding. In a last step, like it is common for coloured Petri nets, it is possible to merge some transitions. Similar parts of the Petri net are folded yielding additional coloured tokens representing each part. For modelling we use CPN-Tools [22, 23]. CPN-Tools is developed at the AIS group of the Technische Universiteit Eindhoven and supports all editing and simulation features for coloured Petri net.

In our case study our initial low-level Petri net contains 25 transitions corresponding to the 25 actions of our process identified in the formalization step. The control flow is rather simple and we just add the corresponding places. First, a blood count have to be registered, then an arbitrary number of the 12 examinations concerning Lyme disease can be scheduled. The execution of these 12 examinations must follow the simple rule, that first the BORG- and BORM-examination need to be performed before the other examinations occur. Remark, the control flow of the initial low-level net is independent form the values given in the language. Rules and regulation concerning values are added in the next step. All actions that corresponds to an examination result carry a value. For this reason we introduced a *boolean* colourset called RES and allow each such transition to be executed while binding to *true* or *false*. At this point we are able to require that a BORG- or BORM-examination must be positive before any other examination can be executed. In a last step we folded transitions if possible. The resulting net is depicted in Figure 4.

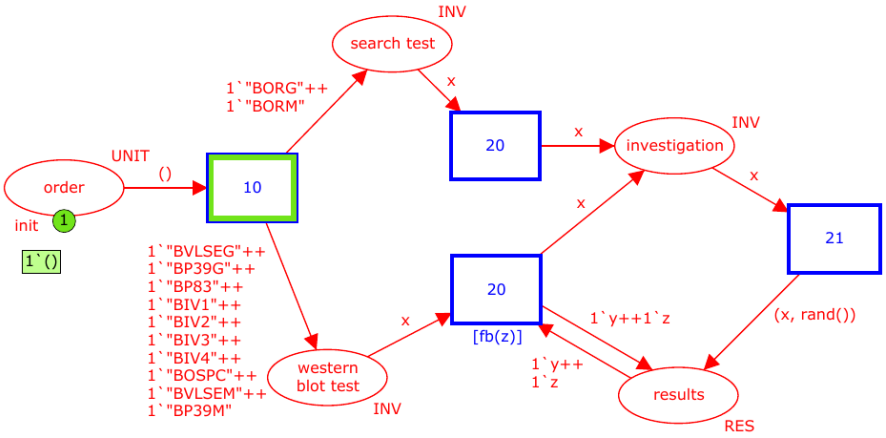

Fig. 4. Coloured Petri net representing the charges regulation document.

In Figure 4 the transition named 10 is enabled in the initial marking. If transition 10 fires, a BORG- and a BORM-token is produced in the place *search test* and tokens corresponding to all other examinations are produced in the place *western blot test*. In such a marking only the upper transition 20 is enabled. If transition 20 fires a BORG- or a BORM- examination is scheduled. As soon as an examination is scheduled transition 21 is enabled. If transition 21 fires, it consumes a token from the place *investigation* and moves this token to the place *results*. While the token is moved a random boolean value is attached. The lower transition named 20 is enable if the western blot tests are scheduled and if there are at least two tokens in the places *results*. The arc inscription $1'y + +1'z$ denotes a pair of tokens. One token is assigned to the variable y

and another token is assigned to the variable z. The guard [fb(z)] ensures that the token called z carries the value *true*. It follows that in the model shown in Figure 4 the western blot tests can only be preformed if the results of the BORG- and BORM-examination are present and at least one of these examinations was evaluated with *true*.

The model shown in Figure 4 is only able to reproduce one single run of the information system. In some sense it is a model of valid words, not a model of the running information system. Our goal is to replay each case of the event log in this model, there is no need to construct a model which is able to handle multiple cases at once.

Besides the possibility to validate the produced model by simulation, CPN-Tools provides some model checking algorithms (see [1] for details). Table 3 depicts a small part of the CPN-Tools state space report of the model shown in Figure 4.

Liveness Properties ————————
Dead Transition Instances: None
Live Transition Instances: None
Fairness Properties ————————
No infinite occurrence sequences.

Table 3. CPN-Tools state space report of the model shown in Figure 4.

The integration step yields a sound and integrated model of the valid language produced during the formalization step. Of course, if analysing this model uncovers faults or additional requirements, the language produced in the formalization step needs to be adopted according to the change made in the model. If model and language match and describe the valid behaviour of the underlying charges regulation document, in the last step of our approach, the model is translated into executable PL/SQL-code.

3.4 Implementation Step

The fourth and last step of our approach is called the implementation step. Although, the coloured Petri net model is executable we translate the produced Petri net into PL/SQL-code. PL/SQL is a proprietary programming language which is integrated in the Oracle Database. Since it can execute SQL statements directly it is more suitable than Java or C++ in our approach. The aim is to get an executable program directly running next to the recorded data. With the help of this program faulty processes preformed by the medical laboratory can automatically be revealed.

During the case study the coloured Petri net model depicted in Figure 4 is transformed into PL/SQL mainly using the following ideas:

(i) Each place of the coloured Petri net yields a temporary table in the database. The tables are able to store records representing tokens and their values.

(ii) Each transition of the coloured Petri net yields a parametrized function in the database. A function returns *true* only if the corresponding transition is executable. To check if a transition is executable arcs of the Petri net are translated into SQL-statements. Roughly speaking, these statements check if there exist appropriate values in the tables corresponding to places in the preset of the transition.
(iii) Each arc of the coloured Petri net yields an SQL-statement in the database. Arcs leading from a place to a transition correspond to DELETE-statements consuming tokens from tables. Arcs leading from a transition to a place correspond to INSERT-statements producing tokens in tables.
(iv) Each guard or function of the coloured Petri net yields a function in the database. The SML-functions given in the coloured Petri net can easily be translated.

CPN	PL/SQL-program
COLOUR	a list of attributes
PLACE	a table having a COLOUR
TOKEN	a record in PLACE
PT-Arc	DELETE from PLACE return TOKEN
TP-Arc	INSERT into PLACE values TOKEN
EXPRESSION	a WHERE expression
TRANSITION	a function using ARCS
GUARD	a sub-function of TRANSITION

Table 4. Pattern of transformations form a CPN into a PL/SQL-program.

A table of transformation patterns is given in Table 4. With the help of these transformation rules it is even possible to implement a fully automated transformation procedure.

To actually verify the event log with the help of the PL/SQL-program the set of cases of the event log is replayed. The registration-ids of the set of faulty cases is stored in an additional table. With the help of this procedure faulty processes can be revealed. The set of faulty processes is the basis of the report produced for the prosecution service. The specific results produced in our case study are presented in the next section.

4 Results and Conclusion

In the context of the presented case study, a set of files of a medical laboratory has been verified. The files record all occurred actions of the information system of the laboratory over a period of five years. In that given period, 22432 orders of Lyme disease diagnostic have been performed by the laboratory. The PL/SQL-program produced by our approach calculates the following results:

recorded processes	valid	faulty	runtime
22432	3311	19121	11 minutes

Table 5. Results of the presented case study.

As shown in Table 5 only 15% of the recorded behaviour is valid according to the charges regulation document. It turned out, that the considered laboratory in almost every case performed the complete set of 12 examinations in a first step. The regulations require that the BORG- and BORM-examination precede all other examinations. Only if one of the two examinations is positive, the set of all examinations can be charged.

To get a more detailed view on the recorded data, in a second step, we adjusted our coloured Petri net model. We removed the transition guard requiring a positive result from one of the BORG- or BORM-examination, assuming a more sloppy interpretation of the regulation document. If we repeat the validation procedure we get that 50% of all recorded processes are valid concerning this more liberal model. In other words, even if we allow that all 12 examinations can be performed at once, 50% of all processes contain additional faults like unnecessary actions or manual changing of examination values.

In the paper we presented an approach together with a case study to verify logs revealing faulty processes of a medical laboratory. The produced PL/SQL-program can directly be applied to any medical laboratory using the same information system. The main advantage of the presented approach is that it is based on a chain of formal models. With the help of these models it is easy to keep track of the validity of the produced report. Most of the steps can be supported using algorithms or tools well known in the area of Petri nets. Furthermore, experts on the regulation document can support the formalisation step without any knowledge about modelling techniques. If a model is produced it also can be adopted and reused. This can help to generate different criteria regarding only parts of the regulation document. Of course, all calculated results can be reproduced at any time, if this is required by the public prosecution service.

From the experience we gained in the case study we feel that the approach forces us to tackle the given task in a very structured way. The approach provides good documented, traceable results. In the future, we will test the presented approach on a larger regulation document yielding a larger regulation model and try to automate each step of the approach further.

References

1. Jensen, K., Kristensen, L.M.: Coloured Petri Nets - Modelling and Validation of Concurrent Systems. Springer (2009)
2. van der Aalst, W.M.P.: Process Mining - Discovery, Conformance and Enhancement of Business Processes. Springer (2011)
3. van der Aalst, W.M.P., Adriansyah, A., van Dongen, B.F.: Replaying History on Process Models for Conformance Checking and Performance Analysis. Wiley Interdisc. Rew.: Data Mining and Knowledge Discovery **2**(2) (2012) 182–192
4. Rozinat, A.: Process Mining: Conformance and Extension. PhD thesis, TU Eindhoven (2010)
5. van Dongen, B.F., van der Aalst, W.M.P.: Multi-phase Process Mining: Building Instance Graphs. In Atzeni, P., Chu, W.W., Lu, H., Zhou, S., Ling, T.W., eds.: ER. Volume 3288 of Lecture Notes in Computer Science., Springer (2004) 362–376
6. Bergenthum, R., Mauser, S.: Mining with User Interaction. In Desel, J., Yakovlev, A., eds.: Proceedings of the Workshop Applications of Region Theory, Petri Nets 2011. Volume 725 of CEUR Workshop Proceedings. (2011) 79–84
7. Bergenthum, R., Mauser, S.: Folding Partially Ordered Runs. In Desel, J., Yakovlev, A., eds.: Proceedings of the Workshop Applications of Region Theory, Petri Nets 2011. Volume 725 of CEUR Workshop Proceedings. (2011) 52–62
8. Badouel, E., Darondeau, P.: Theory of Regions. In Reisig, W., Rozenberg, G., eds.: Petri Nets. Volume 1491 of Lecture Notes in Computer Science., Springer (1996) 529–586
9. Bergenthum, R., Desel, J., Mauser, S., Lorenz, R.: Construction of Process Models from Example Runs. Petri Nets and Other Models of Concurrency **2** (2009) 243–259
10. Darondeau, P.: Synthesis and Control of Asynchronous and Distributed Systems. In Basten, T., Juhás, G., Shukla, S.K., eds.: ACSD, IEEE Computer Society (2007) 13–22
11. Bergenthum, R., Desel, J., Kölbl, C., Mauser, S.: Experimental Results on Process Mining Based on Regions of Languages. In: Proceedings of the Workshop CHINA, Petri Nets 2008, China (2008) 73–87
12. Glinz, M.: Improving the Quality of Requirements with Scenarios. In: Second World Congress on Software Quality, Yokohama (2000) 55–60
13. Desel, J.: From Human Knowledge to Process Models. In Kaschek, R., Kop, C., Steinberger, C., Fliedl, G., eds.: UNISCON. Volume 5 of Lecture Notes in Business Information Processing., Springer (2008) 84–95
14. Weske, M.: Business Process Management - Concepts, Languages, Architectures, 2nd Edition. Springer (2012)
15. Mayr, H.C., Kop, C., Esberger, D.: Business Process Modeling and Requirements Modeling. In: ICDS, IEEE Computer Society (2007) 8
16. Mauser, S., Bergenthum, R., Desel, J., Klett, A.: An Approach to Business Process Modeling Emphasizing the Early Design Phases. In: Proceedings of the Workshop Algorithmen und Werkzeuge für Petrinetze. Volume 501 of CEUR Workshop Proceedings. (2009) 41–56
17. van der Aalst, W.M.P., Stahl, C.: Modeling Business Processes - A Petri Net-Oriented Approach. Cooperative Information Systems series. MIT Press (2011)
18. Harel, D.: Come, Let's Play - Scenario-based Programming using LSCs and the play-engine. Springer (2003)

19. van der Werf, J.M.E.M., van Dongen, B.F., Hurkens, C.A.J., Serebrenik, A.: Process Discovery using Integer Linear Programming. Fundam. Inform. **94**(3-4) (2009) 387–412
20. IEEE Task Force on Process Mining: Process Mining Manifest. In Daniel, F., Barkaoui, K., Dustdar, S., eds.: Business Process Management Workshop. Volume 99 of Lecture Notes in Business Information., Springer (2012) 169–194
21. Bergenthum, R., Desel, J., Lorenz, R., Mauser, S.: Process Mining Based on Regions of Languages. In Alonso, G., Dadam, P., Rosemann, M., eds.: BPM. Volume 4714 of Lecture Notes in Computer Science., Springer (2007) 375–383
22. Ratzer, A.V., Wells, L., Lassen, H.M., Laursen, M., Qvortrup, J.F., Stissing, M.S., Westergaard, M., Christensen, S., Jensen, K.: CPN Tools for Editing, Simulating, and Analysing Coloured Petri Nets. In van der Aalst, W.M.P., Best, E., eds.: ICATPN. Volume 2679 of Lecture Notes in Computer Science., Springer (2003) 450–462
23. Westergaard, M.: CPN Tools 4: Multi-formalism and Extensibility. In Colom, J.M., Desel, J., eds.: Petri Nets. Volume 7927 of Lecture Notes in Computer Science., Springer (2013) 400–409