MASTER

Design of a monitor for on-the-fly checking of business rules

van Loon, J.H.W.

Award date:
2011

Link to publication
Design of a Monitor for on-the-fly checking of Business Rules

J.H.W. van Loon

March 2011
1.2 Problem Definition

Figures 1.1 and 1.2 illustrate a top-level architecture of an Online Auditing Tool. Later in this research, we will conduct a more detailed investigation of the tools available and compare them with our solution.

SAP possesses the capability to log all events and data that are entered or modified within the system. However, this functionality is typically only activated for debugging purposes. SAP also includes a built-in workflow system. In the system we will use during this research, this workflow system is utilized for the purchasing process. We will combine data retrieved from the workflow system with data stored in the "normal" SAP-tables to generate our event log.

The event log format will be based on the datamodel that was introduced by [vdAvHvdW 2009]. We wish to develop a tool that supports the following functionalities:

- Define and store business rules
- Generate an event log from data extracted from an SAP-system
- Analyze this event log and check for compliance to business rules and regulations

The functionality of the tool we are developing corresponds to the following components of the top-level architecture of an Online Auditing Tool as introduced by [vdAvHvdW 2009], as shown in Figure 1.2:

- Define and store business rules, where definition should not require any programming or other IT-related skills. Dejure Models - Business Rules
- Generate event log from the information system to the runtime data
- Analyze this event log and check for compliance to business rules and regulations Conformance checker

1.2.2 Scoping

This research will focus on the conformance checker part of the conceptual model introduced by [vdAvHvdW+09] and on the transformation of actual real-life...
Abstract

This research has been performed as the final Master Project for the Business Information Systems master at the Eindhoven University of Technology. The project has been executed at Deloitte, which also provided the case study at a large Dutch public company.
The research involves a possible solution for checking the data in an enterprise resource planning system for compliance to a predefined set of business rules. Typical for this research is the focus on substantive testing, i.e. checking whether all cases in a process do not violate a certain set of business rules, and the capability to check compliance in real-time.
## Contents

1 Introduction 7  
   1.1 Context  
   1.2 Business Processes  
   1.3 Business Rules  
     1.3.1 Checking business rules  
   1.4 Proof of concept  
   1.5 Scoping  
   1.6 Report structure  

2 Business Rule Checking using Petri Net Simulation 11  
   2.1 Preliminaries  
     2.1.1 Classical Petri nets  
     2.1.2 Coloured Petri nets  
   2.2 Formalizing business rules  
   2.3 Business rules and Petri Nets  
     2.3.1 Classical versus coloured Petri nets  
     2.3.2 Starting Net  
     2.3.3 Event Generator  
     2.3.4 Starting and closing cases  
     2.3.5 Resources  
   2.4 Petri Net Composition  
     2.4.1 Adding business rules  
     2.4.2 Composition rules  
     2.4.3 Verification of correctness  
   2.5 Verification by Simulation  
     2.5.1 Complexity  

3 Business Rule Templates 23  
   3.1 Task always precedes a task  
   3.2 Restrict update operation  
   3.3 Limit number of repetitions  
   3.4 4-eyes principle  
   3.5 Mutually exclusive agents  
   3.6 Task limit on an agent  
   3.7 Forbidden to write  
   3.8 Limit on entity attribute value for an agent  
   3.9 Limit on entity attribute value for a case  
   3.10 Approval limit
Later in this research we will do a more detailed investigation of the tools available and we will compare them with our solution.

SAP has the ability to log all events and data which is entered or changed in system. However this functionality is in most implementations only turned on for debugging purposes. SAP also has a built-in workflow system. In the system we will be using during this research, this workflow system is also used for the purchasing process. We will combine data retrieved from the workflow system with data which is stored in the 'normal' SAP-tables to generate our event log.

The event log format will be based on the datamodel which was introduced by [vdAvHvdW +09].

We would like to develop a tool which supports the following functionality:

- Define and store business rules
- Generate an event log from data extracted from an SAP-system
- Analyze this event log and check for compliance to business rules and regulations

The functionality of the tool we are going to develop translates to the following components of the top level architecture of an Online Auditing Tool as it was introduced by [vdAvHvdW +09] which is shown in figure 1.2:

- Define and store business rules, where definition should not require any programming or other IT-related skills. ¡=¿ Dejure Models - Business Rules
- Generate an event log from data extracted from an SAP-system ¡=¿ Arows from the information system to the runtime data
- Analyze this event log and check for compliance to business rules and regulations ¡=¿ Conformance checker

1.2.2 Scoping

This research will focus on the conformance checker part of the conceptual model introduced by [vdAvHvdW +09] and on the transformation of actual real-life
Chapter 1

Introduction

1.1 Context

In recent years organizations are enforced to comply to a growing set of rules and regulations. Part of these are enforced by laws like the Sarbanes-Oxley Act [6], others are introduced by the organization itself to improve performance or reduce risk.

Because checking this evergrowing set of business rules requires an increasing amount of time of both internal and external auditors, research on automating the task of business rule checking has been performed the last years. This has also led to some software implementations, but, as discussed in the next section, this software focuses on control testing. For substantive testing solutions have been proposed by [17], which covers an ongoing research to develop the so-called "Iphone for the audit", and by [12], which presents a more statistical approach to continuous auditing. Another part of this research has been performed at the Eindhoven University of Technology and is presented in, among other papers, [15] and [16]. These last 2 papers form the main base for this research.

In this report we will introduce a method and a tool which will check in real-time if data entered into a SAP-system is compliant with a predefined set of business rules.

1.2 Business Processes

Business rules are defined on business processes, so before we can discuss what business rules are we first need to define the following terms often used when talking about business processes:

- Task: An atomic activity executed by an agent, possibly including the consumption or production of resources. In our research we assume that tasks are instantaneous.
- Process: A set of tasks for a higher order activity, with ordering relations between the tasks.
- Case: An instance of a process, i.e. an execution path of a process.
1.2 Problem Definition

Figure 1.1: A top level architecture of an Online Auditing Tool

Figure 1.2: A top level architecture of an Online Auditing Tool

Later in this research we will do a more detailed investigation of the tools available and we will compare them with our solution. SAP has the ability to log all events and data which is entered or changed in system. However this functionality is in most implementations only turned on for debugging purposes. SAP also has a built-in workflow system. In the system we will be using during this research, this workflow system is also used for the purchasing process. We will combine data retrieved from the workflow system with data which is stored in the 'normal' SAP-tables to generate our event log. The event log format will be based on the datamodel which was introduced by [vdAvHvdW +09].

We would like to develop a tool which supports the following functionality:

• Define and store business rules
• Generate an event log from data extracted from an SAP-system
• Analyze this event log and check for compliance to business rules and regulations

The functionality of the tool we are going to develop translates to the following components of the top level architecture of an Online Auditing Tool as it was introduced by [vdAvHvdW +09] which is shown in figure 1.2:

• Define and store business rules, where definition should not require any programming or other IT-related skills. Dejure Models - Business Rules
• Generate an event log from data extracted from an SAP-system Arrows from the information system to the runtime data
• Analyze this event log and check for compliance to business rules and regulations Conformance checker

1.2.2 Scoping

This research will focus on the conformance checker part of the conceptual model introduced by [vdAvHvdW +09] and on the transformation of actual real-life

1.3 Business Rules

According to [2] a business rule is a statement that defines or constrains some aspect of the business. It is intended to assert business structure or to control or influence the behavior of the business. According to [16] business rules can be divided in the following 3 main categories:

• Ordering of events, which limit the order in which events occur
• Agent-based, which limit the agents who execute a certain event
• Resource balancing, which limit the values of resources used or changed in a case

In this research we use a set of 11 example business rules, taken from [16] and covering a broad range of business rule types, to demonstrate our theory. These business rules are:

• Task always precedes another task, e.g. TaskA may not be executed before TaskB has been executed
• Restrict update operation, e.g. All tasks which update ResourceX, may not be executed after TaskA has been executed
• Limit number of repetitions, e.g. TaskA may be executed at most 3 times in a trace
• 4-eyes principle, e.g. TaskA may not be executed by the same agent executes TaskB
• Mutually exclusive agents, e.g. AgentX and AgentY may not both execute tasks in one trace
• Task limit on an agent, e.g. An agent may execute at most three tasks in a trace
• Forbidden to write, e.g. AgentX may not update any resource
• Limit on entity attribute value for an agent, e.g. AgentX is not allowed to create or update ResourceX if the value of ResourceX is greater then 1000
• Limit on entity attribute value for a case, e.g. ResourceX may not have a value greater then 1000 in CaseY
• Approval limit, e.g. AgentX may not be executed TaskY if the value of ResourceZ is greater then 1000
• Three-way Match, e.g. TaskX may not be executed if the value of ResourceA is not less then or equal to the values of ResourceB and less then or equal to the value of ResourceC.

The first three rules in this set are ordering of events rules, the next four are agent-based rules and the last four are resource balancing rules. This set should be a good representation of the business rules which can be checked using our theory. There are also some types of business rules rules which cannot be checked using our theory, for example rules which state that a task should be executed at any time in the future, no matter when. It is not possible to check for this business rule, because this business rule will never evaluate to false.

1.3.1 Checking business rules

There a two ways to prove that a certain business rules holds for a certain process:

1. Prove that the business rule holds by formal verification
2. Check that if the business rule holds for all occurring instances of the process

This research focuses on the second method. Notice that this method is completely different from control testing as it is done by most rule checking software. In fact this control testing is part of the first method as it tests if all controls which are needed to prove that a certain business rule holds are enabled.

As said we will however focus on the second method, which is called substantive testing. Substantive testing has been done manually by auditors for a long time. Automating this task does not only save time of the auditor, but also has the additional advantage that we are able to test all data instead of only a very small sample set of the actual data. Our method will also especially focus on substantive testing in real-time. This has the additional benefit that we will detect violations of a business rule the moment they occur, instead of during some checking run some time after the violation has occurred.

1.4 Proof of concept

To demonstrate the results of the research a prototype has been developed. The current implementation of this prototype provides the following functionality:

• Configure a predefined set of business rules
• Load an event log into the conformance checker and check for violation of business rules
• Support for loading new log entries after checking the initial event log
• Check whether the new log entries violate the business rules, and report which log entries violate one or more business rules
To test the functionality we will also use this prototype in a case study. For this case study we will use the actual data from a large Dutch public company. For the business rules in this case study we will use a small subset of the business rules introduced in the third section of this chapter. We will use only a small set, because we want to validate the results using SQL-queries and otherwise these queries would become too complex and error-prone.

1.5 Scoping

This research will focus on the conformance checker part of the conceptual model introduced by [15]. The formal definition and storage of business rules is also in the scope of this project.

The extraction of data from an SAP-system is already possible with a tool which was developed by Deloitte. In this study we will not cover the real-time extraction of data from the SAP-system, but we will simulate the SAP-system by manually adding events to the event log in the order they occurred in the SAP workflow log.

1.6 Report structure

In the next chapter we will discuss the theoretical part of the research. We will introduce a way to describe business rules and we will explain why we use Petri nets for rule checking. In the following chapter we describe all the business rule templates from our example set. In Chapter 4 we discuss the design of the software implementation. This chapter is followed by a chapter which covers some technical implementation details of the software. In Chapter 6 the results of a case study at a larger Dutch public sector client are discussed.
Chapter 2

Business Rule Checking using Petri Net Simulation

2.1 Preliminaries

2.1.1 Classical Petri nets

Petri nets, introduced by C. A. Petri in 1962 and of which good tutorials can be found in [13], [7] and [8], provide an elegant and useful mathematical formalism for modeling concurrent systems and their behaviors. A Petri net is a directed bipartite graph, consisting of places and transitions as nodes. The edges of a Petri net, which are called arcs connect places and transitions. Arcs from a place to a transition are called input arcs, arcs from a transition to a place are called output arcs. Places in Petri nets can hold any positive number of tokens. The state of a Petri net can be described by giving the current number of tokens in each place, this is known as the marking of a Petri net.

A Petri net is a 3-tuple \((P,T,F)\) where:

- \(P\) is a finite set of places
- \(T\) is a finite set of transitions \((P \cap T = \emptyset)\)
- \(F \subseteq (P \times T) \cup (T \times P)\) is a set of arcs (flow relation)

A transition \(t\) is enabled at a marking \(m\) if and only if for every input arc \(i\) of \(t\) there is a token in place \(p\), where \(p\) is the place which is connected to the other end of \(i\). If a transition \(t\) is enabled, it may fire by removing a token from each input place \(p1\) for each input arc between \(p1\) and \(t\) and putting a token in each output place \(p2\) for every output arc between \(t\) and \(p2\).
2.1.2 Coloured Petri nets

Coloured Petri Nets (CPNs) were introduced by [10]. Like other high-level Petri nets, coloured Petri nets are an extension to classical Petri nets. In coloured Petri nets every token is equipped with an extra data attribute called the token colour. This colour allows us to store additional information in the token, e.g. its case id or a time stamp. The colour of a token can change when a transition is fired.

In CPNs arcs can have inscriptions. These inscriptions are expressions which determine whether a transition can fire or not and they also describe which colour the token will get. If we want to combine arc inscriptions of 2 or more arcs to determine whether 1 transition can fire, CPNs can also have a guard on a transition. The following more formal definition is based on the definition as it was given in [11].

Structure of CP-nets

In order to define the structure of a coloured Petri net, we first need to define following semantics:

- The elements of a type, T. The set of all elements in T is denoted by the type name T itself.
- The type of a variable, v - denoted by Type(v)
- The type of an expression, expr - denoted by Type(expr)
- The set of variables in an expression, expr - denoted by Var(expr)
- A binding of a set of variables, b : V → D, with b(v) ∈ Type(v)
- The value obtained by evaluating an expression, expr, in a binding, b - denoted by expr < b >. Var(expr) is required to be a subset of the variables of b, and the evaluation is performed by substituting for each variable v ∈ Var(expr) the value b(v) ∈ Type(v) determined by the binding.

Furthermore we define a multi-set m, over a non-empty set S, as a function m ∈ [S → N]. The non-negative integer m(s) ∈ N is the number of appearances of the element s in the multi-set m. By S_MS we denote the set of all multi-sets over S.

Using these semantics we can define a coloured Petri net as a tuple CPN = (Σ, P, T, A, N, C, G, E, I) satisfying the following requirements:

- Σ is a finite set of non-empty types, called colour sets.
- P is a finite set of places.
- T is a finite set of transitions.
- A is a finite set of arcs such that: P ∩ T = P ∩ A = T ∩ A = ∅.
• $N$ is a node function. $N : A \rightarrow P \times T \cup T \times P$.

• $C$ is a colour function. $C : P \rightarrow \Sigma$.

• $G$ is a guard function. It is defined from $T$ into expressions such that: $\forall t \in T : [\text{Type}(G(t)) = \text{Bool} \land \text{Type}(\text{Var}(G(t))) \subseteq \Sigma]$.

• $E$ is an arc expression function. It is defined from $A$ into expressions such that: $\forall a \in A : [\text{Type}(E(a)) = C(p(a))_{MS} \land \text{Type}(\text{Var}(E(a))) \subseteq \Sigma]$ where $p(a)$ is the place of $N(a)$.

• $I$ is an initialization function. It is defined from $P$ into expressions such that: $\forall p \in P : [\text{Type}(I(p)) = C(p)_{MS}]$.

**Behaviour of CP-nets**

Using the definition of the structure of a coloured Petri net we can now define the behaviour of a CP-net.

Var(t) is called the set of variables of $t$ while $E(x_1, x_2)$ is called the expression of $(x_1, x_2)$. The summation indicates addition of expressions (and it is well-defined because all the participating expressions have a common multi-set type). From the argument(s) it will always be clear whether we deal with the function $E \in [A \rightarrow Expr]$ or the function $E \in [(P \times T \cup T \times P) \rightarrow Expr]$. Notice that $A(x_1, x_2) = \emptyset$ implies that $E(x_1, x_2) = \emptyset$ (where the latter $\emptyset$ denotes the closed expression which evaluates to the empty multi-set).

Next we define what we mean by a transition binding. Intuitively, a binding of a transition $t$ is a substitution that replaces each variable of $t$ with a colour. It is required that each colour is of the correct type and that the guard evaluates to true. As defined in the previous section $G(t) < b >$ denotes the evaluation of the guard expression $G(t)$ in the binding $b$. $B(t)$ is used to denote the set of all bindings for $t$.

The order of variables has no importance.

• A token element is a pair $(p, c)$ where $p \in P$ and $c \in C(p)$

• A binding element is a pair $(t, b)$ where $t \in T$ and $b \in B(t)$.

• The set of all token elements is denoted by $TE$

• The set of all binding elements is denoted by $BE$

• A marking is a multi-set over $TE$

• A step is a non-empty and finite multi-set over $BE$

To continue with the real behaviour of a CP-net we define that a step $Y$ is enabled in a marking $M$ if and only if all guards on $t$ evaluate to true and the following property is satisfied:

$$\forall p \in P : \sum_{(t,b) \in Y} E(p,t) < b > \leq M(p)$$
When a step is enabled it may occur, meaning that tokens are removed from the input places and added to the output places of the occurring transitions. The number and colours of the tokens are determined by the arc expressions, evaluated for the occurring bindings:

When a step $Y$ is enabled in a Marking $M_1$ it may occur, changing the marking $M_1$ in marking $M_2$, defined by:

$$\forall p \in P : M_2(p) = (M_1(p) - \sum_{(t,b) \in Y} E(p,t) < b) + \sum_{(t,b) \in Y} E(t,p) < b$$

The first sum describes the tokens removed from the input places and the second describes the tokens added to the output places.

### 2.2 Formalizing business rules

Before we can check an event log against business rules, we will first need a way to formally describe these business rules. We could do this by introducing a formal language. [16] introduces such a formal language called BRL (Business Rule Language). However, in order to focus on the main topic of this research, which is checking the rules by simulation, we choose to describe the business rules using standard predicate calculus. After defining our business rules in predicate calculus we will translate them into Petri net-templates. These templates describe which nodes need to be added to an existing net in order to force compliance to the specified business rule.

### 2.3 Business rules and Petri Nets

#### 2.3.1 Classical versus coloured Petri nets

Using Petri nets for compliance checking, introduced by [16], should provide the following advantages:

- **Good performance**, because we only need to check the newly arrived event instead of the complete event log. This gives us the advantage that we will be able to check business rules in real-time.
- **Detect events** that either may or must lead to violations later on, because in a Petri net we can check whether a certain marking can lead to a deadlock. This early detection is not part of this research.

In [16] an example of modeling business rules in Petri nets is given. This example illustrates the use of Petri nets for an authorization process. It illustrates how permissions to execute a certain task can be delegated to someone else and how to connect this authorization net to the business net. Studying this translation to a Petri net lead us to the conclusion that in fact agent-based rules are a special kind of event-ordering rules. An agent-based rule simply states that the event of granting person $A$ the right to execute task $X$ should happen before the execution of task $X$ and that the event of retracting this same right should happen after the execution of task $X$. 

---

14
The fact that agent-based rules are in fact a special kind of event-order rules is also the reason that this kind of rules can be translated to Petri nets. Classical Petri nets are indeed most suitable for specifying the order of firing of transactions. Modeling resources in a classical Petri net is also possible, however it can be very difficult and may require using inhibitor arcs. A logical step is to use coloured Petri nets, in which the colour of the tokens represents the value of the resource. This will simplify the modeling, but it has the disadvantage that proving properties of a coloured Petri net can be more difficult then proving properties of a classical Petri net. However, because we will only need to prove rather simple CP-nets, this disadvantage does not outweigh the advantage of the simpler modeling.

2.3.2 Starting Net

We want to construct an automaton which accepts only the language which describes all possible sequences of events in an event log which do not violate any of the predefined business rules. This automaton will be defined by a Petri net constructed of a general starting net which accepts every possible firing sequence for the given set of tasks extended by predefined set of nodes and inscriptions for every business rule which has to checked.

This means we have to start with a Petri net which represents our business process and which does not restrict the execution of tasks. To accomplish this we have to map all elements of our business process to elements in a coloured Petri net. We do that in the following way:

- Task: Represented by a transition with the same name as the task.
- Process: A complete coloured Petri net.
- Case: A set of tokens in the coloured Petri net. All tokens in this set have the same case identifier.
- Agent: A token in the coloured Petri net. The colour of the token represents the name of agent.
- Resource: A token, whose colour represents the current value of the resource.
- Event: The firing of the transition corresponding to the executed task.

The most simple net which accepts all firing sequences for a given set of tasks is the so called flower net. In this net there is one central place. Every transition, which relates to a certain task, has one input-arc from that central place and one output-arc to that central place. The inscriptions of those arcs should be identical and should not disable the transition in any situation. There also should not be any guards defined on the transitions. An example of such a starting net is showed in figure 2.1.
2.3.3 Event Generator

In order to simulate our business process we introduce an event generator which will generate random events. This event generator will generate three tokens, one for the case identifier, one for task and one for the agent. After the event has been generated only the transition corresponding to the task of the event is enabled. This transition will consume the three tokens which were generated. Furthermore the firing of the transition will produce a token which re-enables the event-generator. The event generator connected to the previous flower net is shown in Figure 2.2.
2.3.4 Starting and closing cases

In order to keep our statespace limited in size we only want to keep track of the status of cases for which are open. Open cases are all cases for which we are going to execute the first task in the next step or for which we have already executed tasks and expect to execute more tasks in the future. All open cases should have a token with their identifier in the central place of the flower model. Transitions to start and complete cases are introduced which produce and consume this token respectively. These transitions will later on be used to initialize and finalize the status of certain business rules. The additions for starting and closing cases are shown in Figure 2.3.

![Petri net with start and complete transitions](image)

Figure 2.3: Petri net with start and complete transitions

2.3.5 Resources

All resource types we will be using in our business rules should be modeled in our starting net as a place. These places can contain tokens which represent global resources, i.e. resources which are used by all cases, and local resources, i.e. resources which are specific for a certain case. We also already define how and by which tasks these resources are updated. We recognize two types of
updates: updates which overwrite the previous value of the resource with a new value generated by the event generator and updates which add the value from the event generator to the resource. An example of a resource with the second type of update is shown in Figure 2.4.

![Petri net with resources](image)

**Figure 2.4: Petri net with resources**

### 2.4 Petri Net Composition

#### 2.4.1 Adding business rules

Because we want our conformance checker to check multiple business rules at the same time, we need to combine the Petri nets for the different rules into a composite Petri Net which will enforce all these business rules. In this section we will describe what we need to add to the starting net for the different types of business rules. These additions will be further specified in the next chapter. The additions mentioned here are all the changes we have discovered in our set of example business rules, but it is very well possible that new changes are introduced with new (types of) business rules.

**Ordering of events rules**

- Adding new places and arcs to and from these places only, where the places store the information about the already executed events and the limitations on these executions
1.2 Problem Definition

Figure 1.1: A top level architecture of an Online Auditing Tool
Figure 1.2: A top level architecture of an Online Auditing Tool

Later in this research we will do a more detailed investigation of the tools available and we will compare them with our solution.

SAP has the ability to log all events and data which is entered or changed in system. However this functionality is in most implementations only turned on for debugging purposes. SAP also has a built-in workflow system. In the system we will be using during this research, this workflow system is also used for the purchasing process. We will combine data retrieved from the workflow system with data which is stored in the ‘normal’ SAP-tables to generate our event log.

The event log format will be based on the datamodel which was introduced by [vdAvHvdW +09].

We would like to develop a tool which supports the following functionality:

• Define and store business rules
• Generate an event log from data extracted from an SAP-system
• Analyze this event log and check for compliance to business rules and regulations

The functionality of the tool we are going to develop translates to the following components of the top level architecture of an Online Auditing Tool as it was introduced by [vdAvHvdW +09] which is shown in figure 1.2:

• Define and store business rules, where definition should not require any programming or other IT-related skills.
• Adding arcs and transitions which cleanup tokens (finalizing cases). Again these arcs only connect to places added in the same addition.

Agent-based rules

• Adding places which store information about agents who have executed and who are allowed to execute certain tasks.
• Adding arcs and guards which validate the business rule. Arcs are again only connected to newly added places.

Resource balancing rules

• Adding configuration places, which define the limits on resources for the business rule.
• Adding arcs and guards which validate the business rule. Arcs are only connected to the configuration places.

Besides these additions it will sometimes be necessary to declare new functions in the CP-net, which will perform some more advanced comparison functions, e.g. comparing the items of two lists.

2.4.2 Composition rules

To be sure that our Petri net accepts a firing sequences if and only if it satisfies all business rules defined in that Petri net we have to assure two properties:

• Additions for one business rule assure that if these additions are added to the starting net, the resulting Petri net accepts a firing sequences if and only if it satisfies that business rule.
• Additions made to the Petri net for different rules do not interfere with each other. This means the additions do not update the value of tokens which are not exclusively used to validate the corresponding business rule.

The first point will be discussed on a per rule basis in the next chapter. The second requirement can be fulfilled if the following rules are obeyed:

• Additions do not add arcs to or from places which were already in the Petri net
• If a guard is added to a transition which already has a guard, both the existing guard and the new guard have to be fulfilled, so they are connected by conjunction.
• Inscriptions of existing arcs are not changed.
• The initial marking of the places which were already in the Petri net is not changed.
• The value of variables already in the Petri net is not changed.
In the next section we will proof that if we obey these rules when adding business rules to any Petri net, we will only limit the behaviour of the net in the same way as when we add it to any other net using induction on the firing sequence.

2.4.3 Verification of correctness

In this section we will prove that the additions made for an extra business rule, only limit the behaviour of our Petri net, if we obey the rules stated in the previous section. We will prove this using induction on the accepted firing sequence. We introduce the coloured Petri net $N$. To this net $N$ we will add the additions for one extra business rule, obeying the rules stated above. The result of this addition is a new coloured Petri net $N'$. Now we have to prove that $N'$ only limits the behaviour of $N$, so we have to prove that if $N'$ accepts a firing sequence $\sigma$ then $N$ should also accept that firing sequence.

**Basis** The basis of our induction proof is the empty firing sequence $\sigma_0$. Because every Petri net will accept the empty firing sequence it is trivial that $N'$ does not limit $N$ in accepting the empty firing sequence.

**Inductionhypothesis**

\[
\begin{align*}
M'_0 & \xrightarrow{\sigma} M'_1 \implies M_0 \xrightarrow{\sigma[T_N]} M_1 \\
\forall p \in P_N : M(p) = M'(p)
\end{align*}
\]

Where $M_0$ and $M_1$ are the markings of $N$ and $M'_0$ and $M'_1$ are the markings of $N'$ and $\sigma[T_N]$ is the projection of $\sigma$ on $T_N$, i.e. the transitions of $N$.

Now we have to prove for our induction step $\sigma' = \sigma \bullet t$ that if $N'$ accepts $\sigma'$ then $N$ should also accept $\sigma'[T_N]$. We will split this proof in two cases.

**Case 1:** $t \in T'_N \setminus T_N$

\[
\begin{align*}
N' : M'_0 & \xrightarrow{\sigma'} M'_1 \xrightarrow{\sigma'[T_N]} M'_2 \\
\sigma'[T_N] & = \sigma'[T_N] \text{ and so } N : M_0 \xrightarrow{\sigma'[T_N]} M_1
\end{align*}
\]

We have to prove that $\forall p \in P_N : M_1(p) = M'_2(p)$.

Because by the rules stated in previous section $t$ is not connected to any $p \in P_N$, $t$ cannot change the marking of any $p \in P_N$. Therefore $\forall p \in P_N : M'_2(p) = M'_1(p)$ and because $\forall p \in P_N : M_1(p) = M'_1(p)$ we can conclude that $\forall p \in P_N : M_1(p) = M'_2(p)$.
Case 2: \( t \in T_N \)

\[
N' : M'_0 \xrightarrow{a} M'_1 \xrightarrow{t} M'_2 \\
N : M_0 \xrightarrow{\sigma(T_N)} M_1 \xrightarrow{t} M_2
\]

In order to prove that \( N' \) only limits the behaviour of \( N \) we have to prove that:

i: \( t \) is enabled in \( M_1 \)

\[\forall p \in P_N : M'_2(p) = M_2(p)\]

ii: \( t \) is enabled in \( M_1 \)

The input of \( t \) may consist of places from \( P_N \) and from \( P_N \setminus P_N \). For all places in \( P_N \) the input is the same, because \( M'_1(p) = M_1(p) \). The other places \( (p \in P_N \setminus P_N) \) can only limit the enabledness of \( t \). Also any extra guards can only limit the enabledness of \( t \). But because \( t \) is enabled in \( M'_1 \) it is also enabled in \( M_1 \).

\[\exists t \in T \text{ is firing}\]

The tokens in \( p \in t \bullet \) are determined by the arc inscriptions of \( t \rightarrow p \). As stated by the rules in the previous section, these arc inscriptions are not changed, so \( M'_2(p) = M_2(p) \)

This results to the conclusion that if \( N' \) accepts \( \sigma' \) then \( N \) will also accept \( \sigma' \). And because both \( N' \) and \( N \) accept the empty firing sequence we can conclude that \( N' \) only limits the behaviour of \( N \).

### 2.5 Verification by Simulation

To verify whether a certain event log satisfies all business rules which are defined by a certain Petri net, we first translate the event log into a firing sequence. We do this by first sorting the event log based on the time at which the events occurred. Events do not have to be grouped by cases and are solely sorted on their timestamp. After sorting we translate every event in the log to the firing of the transition which corresponds to the task of the event.

The simulation model will consist of the Petri net composed of the starting net and the additions defined by all applicable business rules. If and only if our event log does not violate any of the business rules our simulation model will accept the firing sequence.

Please note that our approach does not include a check of which business rule has been violated. It is however possible to determine this by evaluating which node(s) and/or guard(s) prohibit a transition from firing. Solving this issue should be part of future research.
2.5.1 Complexity

Memory usage

One of the reasons we choose using Petri nets for conformance checking is the fact that memory usage of the algorithm is bounded by the number of open cases instead of the length of the complete event log, as with most log checking algorithms. Because we keep the state of every business rule for every open trace in memory we do not need to keep any events in memory. When a new event arrives, we just check the last event against the business rules and update the state of every business rule for the corresponding trace. After that the event may be disposed from memory. As the number of open cases is limited in the real-world, the memory usage of our monitor will also be limited.

Calculation time

Our algorithm will keep the status of all open cases in memory in the form of the marking of a Petri net. This results in only having to calculate the effect of a newly arrived event on the status of the open case. This calculation involves the following steps:

- Check enabledness of transition
- Fire transition

The checking of whether a transition is enabled or not does not require a lot of processing. It checks whether all input places have the correct tokens, i.e. tokens of the case, and it evaluates at most one simple expression per business rule.

Firing the transition will also not require heavy processing, because firing also only includes simple expressions, which do not require the processing of large datasets or complex algorithms.
Chapter 3

Business Rule Templates

In the following paragraphs we will demonstrate the Petri nets for a given set of different business rules. This set of business rules is taken from [16] and should cover a wide range of business rules which will be supported by our tool. This will exclude business rules which cannot be checked by using simulation of Petri nets at all, which include (as discussed in the previous chapter) the following types of business rules:

- Rules which require some task to happen at any time, no matter when. For example "Every invoice should be paid once."

The following types of business rules are also not included, not because they are impossible to check using our method, but because it would make all templates, so not just the template for that particular rule, significantly more complex:

- Business rules concerning more than one case
- Agent-based business rules. Basically these are business rules concerning two cases, where one case belongs to the process of delegation of rights for a certain agent and one case belongs to the actual business process
- Rules for which the simulation engine needs to be aware of a current time. For example the rule "Every invoice should be paid within 20 days." would require the simulation engine to know when these 20 days are passed.

For every rule template we will give a short description in natural language and the formal description taken from [16]. We will also give an example of the use of the particular template and a Petri net which implements this example. In this Petri net the additions to the starting net, as described in 2.4.1, will be shown in red. If we need information on the agent which executed the task or need information about a resource, which are both already part of the Petri net without the business rule, we will draw this parts in green. Finally we will proof that the given Petri net implements the business rule.
3.1 Task always precedes a task

Category
Ordering of tasks

Description
A task t2 should always be performed before task t1 in any case of process u.

Formal Description

\[
\text{TaskAlwaysPrecedesTask}(u : \text{Process}, t_1, t_2 : \text{Task}) := \\
\forall x_1 \in \text{Event} : p(c(x_1)) = u \land t(x_1) = t_1 \Rightarrow \\
\exists x_2 \in \text{Event} : t(x_2) = t_2 \land c(x_1) = c(x_2) \land (x_2, x_1) \in \text{prev}^*
\]

Example
There should always be a "Goods Received" event before an invoice is paid.

Net additions
The required additions to the Petri net are shown in figure 3.1.
1.2 Problem Definition

1.2.1 Problem Definition

Later in this research we will do a more detailed investigation of the tools available and we will compare them with our solution. SAP has the ability to log all events and data which is entered or changed in system. However this functionality is in most implementations only turned on for debugging purposes. SAP also has a built-in workflow system. In the system we will be using during this research, this workflow system is also used for the purchasing process. We will combine data retrieved from the workflow system with data which is stored in the ‘normal’ SAP-tables to generate our event log. The event log format will be based on the datamodel which was introduced by "vdAvHvdW+09".

We would like to develop a tool which supports the following functionality:

- Define and store business rules
- Generate an event log from data extracted from an SAP-system
- Analyze this event log and check for compliance to business rules and regulations

The functionality of the tool we are going to develop translates to the following components of the top level architecture of an Online Auditing Tool as it was introduced by "vdAvHvdW+09" which is shown in figure 1.2:

- Define and store business rules, where definition should not require any programming or other IT-related skills. "Dejure Models - Business Rules"
- Generate an event log from data extracted from an SAP-system "Arrows from the information system to the runtime data"
- Analyze this event log and check for compliance to business rules and regulations "Conformance checker"

1.2.2 Scoping

This research will focus on the conformance checker part of the conceptual model introduced by "vdAvHvdW+09" and on the transformation of actual real-life

Proof

Transition "Approve Payment" can only be enabled for case 'q' if place "GR Complete" contains a token (q, true). Place "GR Complete" is connected to four transitions and each transition has a different effect on the tokens in this place:

- Start Case: Places (q, false) into the place
- Complete Case: Removes the token for case 'q' from the place
- Approve Payment: Places the same token back as it removes
- Goods Received: Place (q, true) into the place, after removing the token for case 'q'

This results in the fact that there will be always one token for case 'q' in place "GR Complete" between the firing of "Start Case" and the firing of "Complete Case" for that particular case. Because "Approve Payment" will always happen between "Start Case" and "Complete Case", the only way "Approve Payment" can be enabled when it occurs in a firing sequence is when there is a token (q, true) in place "GR Complete". And as "Goods Received" is the only transition which can place such a token in that place, "Approve Payment" can only be enabled after "Goods Received" has fired.

This results in the desired behavior that "Approve Payment" is enabled for case 'q' if and only if "Goods Received" has been fired for case 'q'.

25
3.2 Restrict update operation

Category
Ordering of tasks

Description
After task u is performed in a case, no entity of type x can be updated anymore in that case.

Formal Description

\[
RestrictUpdate(u : Task; x : EntityType) := \\
\forall e_1, e_2 \in Event : c(e_1) = c(e_2) \land t(e_1) = u \land (e_1, e_2) \in prev^* \\
\land \neg(\exists y \in UpdateEntity : p(y) = e_2 \land t(e(y)) = x)
\]

Example
After payment of an invoice has been approved, no updates to that invoice are allowed.

Net additions
Our tool will determine which task(s) update entity ‘x’. In the following net it is assumed that “Update Invoice” is the only task which can update an invoice. The required additions to the Petri net are shown in figure 3.2.

Proof
Transition ”Update Invoice” can only be enabled for case ’q’ if place ”Payment Approved” contains a token (q, false). Place ”Payment Approved” is connected to 4 transitions and each transition has a different effect on the tokens in this place:

- Start Case: Places (q, false) into the place
- Complete Case: Removes the token for case ‘q’ from the place
- Update Invoice: Places the same token back as it removes
- Approve Payment: Place (q, true) into the place, after removing the token for case ‘q’

This results in the fact that there will be always 1 token for case ‘q’ in place ”Payment Approved” between the firing of ”Start Case” and the firing of ”Complete Case” for that particular case. Because ”Update Invoice” will always
happen between "Start Case" and "Complete Case", the only way "Update Invoice" can be disabled when it occurs in a firing sequence is when there is a token \((q, \text{true})\) in place "Payment Approved". And as "Approve Payment" is the only transition which can place such a token in that place, "Update Invoice" can only be disabled after "Approve Payment" has fired.

The only way to re-enable "Update Invoice" after "Approve Payment" has fired is to put a token \((q, \text{false})\) in "Payment Approved". The only transition capable of doing this is "Start Case", but because by definition "Start Case" cannot occur after "Approve Payment" in a firing sequence it is not possible to re-enable "Update Invoice".

This results in the desired behavior that "Update Invoice" is enabled for case 'q' if and only if "Approve Payment" has not been fired for case 'q'.

Figure 3.2: Restrict update operation
3.3 Limit number of repetitions

Category
Ordering of tasks

Description
In any case of process P task u cannot be executed more than n times.

Formal Description

\[
\text{LimitNrOfRepititions}(u : \text{Process}; z : \text{Task}) := \\
\forall w \in \text{Case} : p(w) = u \Rightarrow \\
|\{x \in \text{Event}|c(x) = w \land t(x) = z\}| \leq n
\]

Example
The task "resolve P-block" should not occur more than 5 times in any case.

Net additions
The required additions to the Petri net are shown in figure 3.3.

![Figure 3.3: Limit number of repetitions](image-url)
Proof

Define places x, y and rl (Repetition limit)
Define transitions sc (Start case), cc, (Complete case), rp (Resolve P-block) and cu (Cleanup).

#x defines the number of times transition x has been fired.
We want to limit #rp to 5 per case, so we want an invariant 5#sc >= #rp.
The number of tokens in place rl is defined as #rl = 5#sc - #rp - #cu.
Because the number of tokens in a place can never be negative we have
5#sc - #rp - #cu >= 0
5#sc >= #rp + #cu
Because the number of firings of a transitions can also not be negative, this results in
5#sc >= #rp
This is our desired invariant.
3.4  4-eyes principle

Category
Agent-Based

Description
Two tasks t1 and t2 in the same case should always be executed by different agents.

Formal Description

\[
4EyesPrinciple(t_1, t_2 : Task) :=
\forall x, y \in Event : (e(x) = e(y) \land t(x) = t_1 \land t(y) = t_2) \Rightarrow
execBy(x) \neq execBy(y)
\]

Example
The approval of an order should be done by a different person than the one who approves the payment.

Net additions
The required additions to the Petri net are shown in figure 3.4.

Proof
2 places are added to net, 1 which holds 1 token for each case with a list of actors which executed task A and 1 which holds 1 token for each case with a list of actors which executed task B. These lists are updated on firing of the transition corresponding to the task. The guards on the transitions for both tasks is \([\text{mem } y \text{ Actor} = \text{false}]\). Which means these transitions can only fire if the actor which executed the corresponding task is not in list y. And as list y holds all actors which executed the other task, this means task A and task B can not be executed by the same actor in 1 case.
Figure 3.4: 4-eyes principle
3.5 Mutually exclusive agents

Category
Agent-Based

Description
Two agents a1 and a2 should never appear together in a case.

Formal Description

\[ MutualExclusiveAgents(a_1, a_2 : Agent) := \neg \exists u_1, u_2 \in Event : u_1 \neq u_2 \land c(u_1) = c(u_2) \land \text{execBy}(u_1) = a_1 \land \text{execBy}(u_2) = a_2 \]

Example
Two persons who are directly related are not allowed to work together on the same case.

Net additions
The required additions to the Petri net are shown in figure 3.5. The following function declaration is also added to the Petri net:

\[
\text{fun CheckMutualActors(act : STRING, c::ca : Actors, ma : MutualActors) =}
\]

\[
\text{if mem ma (act, c) then false else CheckMutualActors(act,ca,ma) | CheckMutualActors(_,nil,_) = true;}
\]

Proof
The function 'CheckMutualActors' is a recursive function. It recursively runs through the list of actors which have previously executed a task in the case. If that list is empty 'true' is returned, meaning the transition may fire because the actor cannot conflict with an actor on an empty list. If the list is not empty the function checks whether the product of the top element of the list and the actor of the transition are in the list with mutually exclusive agents. If this is the case 'false' is returned and the transition may not fire, otherwise the function is called with this top element removed from the list of actors previously having executed a task in the case.

As the result of this function fulfills the requirements of our business rule, the guard consists only of a call of this function with the correct parameters.
Figure 3.5: Mutually exclusive agents
3.6 Task limit on an agent

Category
Agent-Based

Description
An agent a cannot do more than n tasks in any case of process u.

Formal Description

\[
\text{TaskLimitOnAgent}(U : \text{Process}, a : \text{Agent}, n : \text{Nat}) := \\
\forall w \in \text{Case} : (p(w) = u) \Rightarrow \\
|\{x \in \text{Event} | c(x) = w \land \text{execBy}(x) = a\}| \leq n
\]

Example
A manager should not execute more than a total of 2 tasks in a case.

Net additions
The required additions to the Petri net are shown in figure 3.6.

![Figure 3.6: Task limit on an agent](image-url)
Proof

The place 'actor limits' contains a predefined set of tokens with the limit per actor. The place 'task count' contains per case a list with the number of tasks each agent has executed. The function 'AddExec' updates this taskcount on every execution of the specified task. The function 'CheckTaskLimit' validates whether this taskcount is lower than the limit. If this is true the guard evaluates to true and the transition will be enabled.
3.7 Forbidden to write

Category
Agent-Based

Description
An agent a1 is not allowed to update any local entity in a process u.

Formal Description

\[
\text{ForbiddenToWrite}(a : \text{Agent}, u : \text{Process}) := \\
\forall x \in \text{Event} : (\text{execBy}(x) = a \land p(c(x)) = u) \Rightarrow \\
\neg(\exists y \in \text{UpdateEntity} : p(y) = x)
\]

Example
An agent without purchasing rights is not allowed to insert or update any information in a purchase order.

Net additions
The required additions to the Petri net are shown in figure 3.7.

Figure 3.7: Forbidden to write
Proof

The place ‘authorized actors’ contains a predefined set of tokens, one for each actor authorized to execute the update task. The guard on the update transition states [ z = y ]. This ensures that the transition can only fire if the aforementioned place contains a token which is equal to actor executing the task.
3.8 Limit on entity attribute value for an agent

Category
Value-Based

Description
An agent $a$ is not allowed to write an entity of type $b$ with value of attribute $x$ larger than $n$.

Formal Description

$$LimitEntAgent(a : Agent; b : EntityType; x : A; n : V) :=$$
$$\forall z \in Event; y \in UpdateEntity : (p(y) = z \land t(e(y)) = b \land$$
$$execBy(z) = a) \Rightarrow e(y).x \leq n$$

Example
An agent is not allowed to create or update a purchase order with a total value above 15000 euros.

Net additions
The required additions to the Petri net are shown in figure 3.8.

![Figure 3.8: Limit on entity attribute value for an agent](image-url)
Proof

The place 'actor limits' contains a predefined set tokens with the limit per actor. The place 'total invoice value' contains the value of each invoice. This value is updated by the transition 'change invoice', this is done by the arc inscription 'id=(id x), value=(value x) + y'. Where x is the record of the invoice before the firing of task and y is the value which is added by the task.

The guard on the 'change invoice' transition is [((value x) + y) ≤ al], where al is the limit for the actor executing the task corresponding to the transition. This ensures that the transition can only fire if the previous total value of the invoice plus the value added by transition is less than or equal to the limit for the actor.
3.9 Limit on entity attribute value for a case

Category
Value-Based

Description
For each entity of type b written in case w, the value of attribute x is lower than n.

Formal Description
In this description Λ issetofattributesofanentity.

Example
An invoice can only be approved if the total value is below 15000 euros.

Net additions
The required additions to the Petri net are shown in figure 3.9.
Proof

The place ‘case limits’ contains a predefined set tokens with the limit per case. The place ‘total invoice value’ contains the value of each invoice. This value is updated by the transition ‘change invoice’, this is done by the arc inscription ‘id=(id x), value=(value x) + y’. Where x is the record of the invoice before the firing of task and y is the value which is added by the task. The guard on the ‘change invoice’ transition is [((value x) + y) \leq cl], where cl is the limit for the case. This ensures that the transition can only fire if the previous total value of the invoice plus the value added by transition is less than or equal to the limit for the case.
3.10 Approval limit

Category
Value-Based

Description
An agent \( a \) can only perform task \( u \) in a case if for each entity of type \( b \) written in that case, attribute \( x \) is lower than value \( n \).

Formal Description

\[
ApprLim(a : Agent; u : Task; b : EntityType; x : \Lambda; n : V) := \\
\forall y \in Event : \text{execBy}(y) = a \land t(y) = u \Rightarrow \\
\text{LimitEntinCase}(c(y); b; x; n)
\]

Example
An agent can not approve a payment if the total value of the invoice is above the agent’s approval limit.

Net additions
The required additions to the Petri net are shown in figure 3.10.

![Net additions](image-url)
Proof

The place 'approval limits' contains a predefined set tokens with the approval limit per actor. The place 'invoices' contains the value of each invoice. The guard on the 'approve payment' transition is \([x \leq y]\), where \(y\) is the limit for the actor executing the task corresponding to the transition and \(x\) is the value of the invoice being approved. This guard ensures that the transition can only fire if the total value of the invoice is less than or equal to the approval limit for the actor.
3.11 Three-way Match

Category
Value-Based

Description
In each case of process n, if task u is executed, then the sum of the values of entities of type a and the sum of the values of entities of type b should be equal to or greater then the sum of the values of entities of type c.

Formal Description

ThreeWayMatch(u : Task; a;b;c : EntityType) :=
\forall w \in Case; v \in Event; x;y;z \in UpdateEntity :
(t(v) = u \land c(p(x)) = c(p(y)) = c(p(z)) = c(v) = w) \Rightarrow
(e(x) : value \geq e(z) : value \land e(y) : value \geq e(z) : value \land
 t(e(x)) = a \land t(e(y)) = b \land t(e(z)) = c)

Example
Payments can only be approved when goods received and purchase order total values are equal or greater then the total value of approved invoices plus the value of the invoice which is going to be approved

Net additions
The required additions to the Petri net are shown in figure 3.11.

Proof
The places 'Purchase Order', 'Goods Received' and 'Invoices' all held a token for each case with the total value for that document in a case. The guard for the 'approve payment' transition is:

[TotalValue(#Value x) >= p+v, 
 TotalValue(#Value y) >= p+v, 
 #ID x=q, 
 #ID y=q]

The lines of the guards ensure the following:

• Total value of the Purchase Orders is greater then or equal to total paid amount plus amount of the invoice which is going to be approved.

• Total value of the Goods Received is greater then or equal to total paid amount plus amount of the invoice which is going to be approved.
Later in this research we will do a more detailed investigation of the tools available and we will compare them with our solution. SAP has the ability to log all events and data which is entered or changed in system. However this functionality is in most implementations only turned on for debugging purposes. SAP also has a built-in workflow system. In the system we will be using during this research, this workflow system is also used for the purchasing process. We will combine data retrieved from the workflow system with data which is stored in the 'normal' SAP-tables to generate our event log. The event log format will be based on the datamodel which was introduced by [vdAvHvdW+09].

We would like to develop a tool which supports the following functionality:

• Define and store business rules
• Generate an event log from data extracted from an SAP-system
• Analyze this event log and check for compliance to business rules and regulations

The functionality of the tool we are going to develop translates to the following components of the top level architecture of an Online Auditing Tool as it was introduced by [vdAvHvdW+09] which is shown in figure 1.2:

• Define and store business rules, where definition should not require any programming or other IT-related skills. Dejure Models - Business Rules
• Generate an event log from data extracted from an SAP-system Arrows from the information system to the runtime data
• Analyze this event log and check for compliance to business rules and regulations Conformance checker

1.2.2 Scoping

This research will focus on the conformance checker part of the conceptual model introduced by [vdAvHvdW+09] and on the transformation of actual real-life

Figure 3.11: Three-way Match

- The Purchase Orders belong to the same case as the invoice which is going to be approved.
- The Goods Received belong to the same case as the invoice which is going to be approved.

These are exactly the requirements stated by the description of the business rule.
Chapter 4

Design

4.1 Conceptual Design

The conceptual design is based on the conceptual design which was introduced by [15]. All components which are not relevant for the conformance checker were removed from this design. Furthermore extra components for ETL (Extract, Transform, Load) are introduced. This leads to the conceptual design shown in 4.1, where the rectangle indicates the scope of our project.

Besides the challenge of checking an event log in real-time the other big problem is the creation of a suitable event log. To create such a log we have to extract data from the actual SAP-system, then we have to transform it into an event log and finally we have to load that event log into our compliance checker. The problems and process of this preprocessing step have been discussed in [9].

This paper, in combination with the tools already available and the research in [14], will be the base for our preprocessing tools. Tools which perform these preprocessing tasks are often referred to as ETL-tools, where ETL stands for extract, transform, load. In the following subsections we will describe each of these steps and we will introduce the design for the tool or component which will perform that step.

4.2 Event Definition

As shown in figure 4.1 the definition of the events is the base of our software. In this component we define the events which occur in the process as well as how to group these events into cases.

The definition of events depends a lot on the information system, as well on the software which is used as on the way it is implemented. We can however split the software into 2 main categories, based on how data is stored:

- Data is stored based on documents, as in traditional ERP-systems
- Data is stored based on events, as in workflow systems (and SAP transactions)
1.2 Problem Definition

Figure 1.1: A top level architecture of an Online Auditing Tool

Figure 1.2: A top level architecture of an Online Auditing Tool

Later in this research we will do a more detailed investigation of the tools available and we will compare them with our solution. SAP has the ability to log all events and data which is entered or changed in system. However this functionality is in most implementations only turned on for debugging purposes. SAP also has a built-in workflow system. In the system we will be using during this research, this workflow system is also used for the purchasing process. We will combine data retrieved from the workflow system with data which is stored in the ‘normal’ SAP-tables to generate our event log.

The event log format will be based on the datamodel which was introduced by [vdAvHvdW + 09].

We would like to develop a tool which supports the following functionality:

• Define and store business rules
• Generate an event log from data extracted from an SAP-system
• Analyze this event log and check for compliance to business rules and regulations

The functionality of the tool we are going to develop translates to the following components of the top level architecture of an Online Auditing Tool as it was introduced by [vdAvHvdW + 09] which is shown in figure 1.2:

• Define and store business rules, where definition should not require any programming or other IT-related skills ¡=¿ Dejure Models - Business Rules
• Generate an event log from data extracted from an SAP-system ¡=¿ Arrows from the information system to the runtime data
• Analyze this event log and check for compliance to business rules and regulations ¡=¿ Conformance checker

1.2.2 Scoping

This research will focus on the conformance checker part of the conceptual model introduced by [vdAvHvdW + 09] and on the transformation of actual real-life extraction.

Templates

Events

Required

Business

Rules

SAP

System

Data Extractor

SAP

Raw Data

Event Log

Reports

Triggers

Testing Tool

Data Transformer

Compliance Checker

Business Rules

Definition Tool

Figure 4.1: System Overview

Both of these systems require a slightly different approach for event and case definition. We will however create a datamodel which is able to store event definitions for both types of systems.

• Timestamp
• Case identifier
• Agent
• Task (Workflow Element)

4.2.1 Workflow systems

A workflow system already stores events. However we still need to define the case identifier. This might be the same identifier as used by the workflow system, but it can also be another one.

Besides the case identifier we might also need to define some additional data. If for example we have an event "change purchase order" we might also need the new total value of the purchase order. This data will be stored as additional attributes of the event entry.

4.2.2 Document-centric systems

In a document-centric system we would have to develop a method to generate events ourselves. This should be possible but it is outside the scope of this project.
1.2 Problem Definition

Later in this research we will do a more detailed investigation of the tools available and we will compare them with our solution. SAP has the ability to log all events and data which is entered or changed in system. However this functionality is in most implementations only turned on for debugging purposes. SAP also has a built-in workflow system. In the system we will be using during this research, this workflow system is also used for the purchasing process. We will combine data retrieved from the workflow system with data which is stored in the 'normal' SAP-tables to generate our event log. The event log format will be based on the datamodel which was introduced by [vdAvHvdW+09].

We would like to develop a tool which supports the following functionality:

- Define and store business rules
- Generate an event log from data extracted from an SAP-system
- Analyze this event log and check for compliance to business rules and regulations

The functionality of the tool we are going to develop translates to the following components of the top level architecture of an Online Auditing Tool as it was introduced by [vdAvHvdW+09] which is shown in figure 1.2:

- Define and store business rules, where definition should not require any programming or other IT-related skills. = Dejure Models - Business Rules
- Generate an event log from data extracted from an SAP-system = Arrows from the information system to the runtime data
- Analyze this event log and check for compliance to business rules and regulations = Conformance checker

1.2.2 Scoping

This research will focus on the conformance checker part of the conceptual model introduced by [vdAvHvdW+09] and on the transformation of actual real-life

4.3 Rule Definition

Business rules related data can be stored in different formats for different parts of the system:

- XML-templates which are built by people with knowledge of the system implementation
- Database with used rule-templates and parameters, created by the auditor
- Internal storage of the complete business rule Petri Net, which can be read in CPN Tools

In the current version of the tool the templates are hard-coded into the tool and the parameters for the templates have to be set each time the tool is executed. This is something that has to be developed further later on.

How these data are related is shown in figure 4.2.

4.3.1 Ruletemplates

In the next sections we will discuss the format of the ruletemplates. As discussed before a ruletemplate is a collection of additions to a Petri net. First we will discuss adding single Petri net nodes (low-level additions) and thereafter we will explain some high-level additions, which combine a set of low-level additions into 1 addition. This way we have an easier and more readable way to describe our business rule templates.

4.3.2 Low-level Petri Net additions

addPlace

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pType</td>
<td>ColSet</td>
<td>Type of tokens stored in this place</td>
</tr>
<tr>
<td>pReturn</td>
<td>ID</td>
<td>ID of the newly added place</td>
</tr>
</tbody>
</table>

Description

Add a new place of type pType to the top-level business net.
A different perspective

Deloitte & Touche LLP

Annual Report 2009

1.2 Problem Definition

Figure 1.1: A top level architecture of an Online Auditing Tool

Figure 1.2: A top level architecture of an Online Auditing Tool

Later in this research we will do a more detailed investigation of the tools available and we will compare them with our solution.

SAP has the ability to log all events and data which is entered or changed in system. However this functionality is in most implementations only turned on for debugging purposes. SAP also has a built-in workflow system. In the system we will be using during this research, this workflow system is also used for the purchasing process. We will combine data retrieved from the workflow system with data which is stored in the ‘normal’ SAP-tables to generate our event log.

The event log format will be based on the datamodel which was introduced by [vdAvHvdW+09].

We would like to develop a tool which supports the following functionality:

• Define and store business rules
• Generate an event log from data extracted from an SAP-system
• Analyze this event log and check for compliance to business rules and regulations

The functionality of the tool we are going to develop translates to the following components of the top level architecture of an Online Auditing Tool as it was introduced by [vdAvHvdW+09] which is shown in figure 1.2:

• Define and store business rules, where definition should not require any programming or other IT-related skills. Dejure Models - Business Rules
• Generate an event log from data extracted from an SAP-system Arrows from the information system to the runtime data
• Analyze this event log and check for compliance to business rules and regulations Conformance checker

1.2.2 Scoping

This research will focus on the conformance checker part of the conceptual model introduced by [vdAvHvdW+09] and on the transformation of actual real-life

addArc

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pType</td>
<td>Enum</td>
<td>Indicates the direction of the arc</td>
</tr>
<tr>
<td>pTransition</td>
<td>ID</td>
<td>Transition the arc is connected to</td>
</tr>
<tr>
<td>pPlace</td>
<td>ID</td>
<td>Place the arc is connected to</td>
</tr>
<tr>
<td>pVariableType</td>
<td>ColSet</td>
<td>Type of arc inscription</td>
</tr>
<tr>
<td>pWeight</td>
<td>INT</td>
<td>Weight of the arc (Default = 1)</td>
</tr>
<tr>
<td>pReturn</td>
<td>ID</td>
<td>ID of the newly added arc</td>
</tr>
</tbody>
</table>

Description

Add a new arc to the net. Depending on pType the arc will be from pPlace to pTransition or from pTransition to pPlace.

addGuard

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pTransition</td>
<td>ID</td>
<td>Transition to add this guard to</td>
</tr>
<tr>
<td>pType</td>
<td>ColSet</td>
<td>Type of equation (equals / greater then / smaller then)</td>
</tr>
<tr>
<td>pVariable1</td>
<td>ID</td>
<td>Left variable of equation</td>
</tr>
<tr>
<td>pVariable2</td>
<td>ID</td>
<td>Right variable of equation</td>
</tr>
<tr>
<td>pReturn</td>
<td>Bool</td>
<td>Boolean to indicate if the guard has been added successfully</td>
</tr>
</tbody>
</table>

Description

Extends the guard of pTransition. The added formula is of the form: pVariable1 pType pVariable2. The existing guard and the newly added formula are connected with an AND-operator.

addMarking

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pPlace</td>
<td>ID</td>
<td>Place to add the new tokens to</td>
</tr>
<tr>
<td>pList</td>
<td>List</td>
<td>List of tokens to be added</td>
</tr>
<tr>
<td>pValues</td>
<td>List</td>
<td>List with pairs of record attributes</td>
</tr>
<tr>
<td>pReturn</td>
<td>Boolean</td>
<td>Boolean to indicate if the guard has been added successfully</td>
</tr>
</tbody>
</table>

Description

For every item of pList add an item to the list in pPlace. pValues contains the mapping between the attributes of the records in pList and the elements of the records of the list in pPlace.
4.3.3 High-level Petri Net additions

addInitArc

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pPlace</td>
<td>ID</td>
<td>Place which needs to be initialized for every case</td>
</tr>
<tr>
<td>pWeight</td>
<td>INT</td>
<td>Weight of the initialization arc</td>
</tr>
</tbody>
</table>

Description

This arc will add pWeight number of tokens to pPlace just before the first event of every case. The tokens have type ID, which represents the case-id.

Low-level additions

addInitArc

<table>
<thead>
<tr>
<th>AddCommand</th>
<th>Parameters</th>
</tr>
</thead>
</table>
| addArc     | Type = TtoP  
Transition = InitCase  
Place = pPlace  
VariableType = ID  
Weight = pWeight  
Return = InitArcID, InitArcVarName |

addFinArc

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pFrom</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pWeight</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Description

This arc will remove pWeight number of tokens from pPlace just before every case is closed. This is to guarantee there will be no tokens left behind when a case is closed.

Low-level additions

addFinArc

<table>
<thead>
<tr>
<th>AddCommand</th>
<th>Parameters</th>
</tr>
</thead>
</table>
| addArc     | Type = PtoT  
Transition = CloseCase  
Place = pPlace  
VariableType = ID  
Weight = pWeight  
Return = FinArcID, FinArcVarName |
4.3.4 Example

An example of how this template design is used in an actual business rule template will be presented in this section. We choose the "Limit number of repetitions" template as this is one of the less complex and easy understandable templates. In the next paragraphs the formal definition of this template as it has been implemented in the tool is given.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pTask</td>
<td>ID</td>
<td>Task which should be limited in number of repetitions</td>
</tr>
<tr>
<td>pLimit</td>
<td>INT</td>
<td>Number of allowed repetitions</td>
</tr>
</tbody>
</table>

Formal additions

<table>
<thead>
<tr>
<th>AddCommand</th>
<th>Parameters</th>
</tr>
</thead>
</table>
| addPlace   | Name = Taskcount  
             Type = ID  
             return = TaskCountID |
| addPlace   | Name = AntiTaskCount  
             Type = ID  
             return = AntiTaskCountID |
| addInitArc | Place = TaskCountID  
             Weight = pLimit |
| addArc     | From = TaskCountID  
             To = pTask  
             VariableType = ID  
             Return = LimitArcID, LimitArcVarName |
| addArc     | From = AntiTaskCountID  
             To = TaskCountID  
             VariableType = ID  
             Return = CountArcID, CountArcVarName |
| addGuard   | Transition = pTask  
             Type = Equals  
             Variable1 = LimitArcVarName  
             Variable2 = InputArcFromXVarName  
             Return = BoolSucces |
| addCleanup | SourcePlace = TaskCountID  
             TargetPlace = AntiTaskCountID  
             |
| addFinArc  | From = AntiTaskCountID  
             Weight = pLimit |

4.4 ETL-Tool

4.4.1 Extract, Transform, Load

Extract

The extraction of data from an SAP-system is already possible with a tool which was developed by Deloitte. This tool connects to the middleware of the
SAP-system, so not directly to its database, and extracts the tables which where defined by the user. This tool stores the extracted data in an encrypted SQLite-database.

The data which should be extracted from the SAP-system is stored in a template. This template is configured with a tool called GRMT, which is also being developed at Deloitte. In our project these templates will be generated by the business rule definition tool, because the data which is needed depends on the rules which need to be checked for compliance.

The same tool which extracts the data from the SAP-system is also capable to decrypt this data and export the data to the following fileformats:

- Excel-files (.xls)
- Comma-separated values (.csv)
- Access-databases (.mdb)
- Sqlite-databases (.db)
- Eqsmart-files (Deloitte proprietary format)
- Grdata-files (Deloitte proprietary format)

Currently we are manually importing the data which is exported to csv- or Sqlite-files into an SQL-database. We are investigating the possibility of implementing the export to an SQL-database into the extraction tool.

**Transform**

As described in the previous subsection our raw SAP-data can be imported into an SQL-database and also our compliance checker will be using an SQL-database for the event log. Therefor the transformation from the raw SAP-data to an usable event log will be performed by executing SQL-queries. The format in which we will transform our data is based on the datamodel which is introduced in [16] and which is shown in figure 4.3. Because we left the maintenance of roles and permissions for agents out of the scope of this project, we can will not be using the organizational definition.

A few big steps on identifying events in SAP procurement data have been made in [14]. However because we will be using the data from the SAP workflow system we will not be using this method. Instead we only have to join events from the workflow events tables into traces. This work executed by some rather simple SQL-queries.

**Load**

The event log is loaded into the conformance checker directly from the SQL-database. The conformance checker will load 1 event log entry at a time and then check that entry. The conformance checker will keep track of which entries have been checked and which entry needs to be read from the database next.
Later in this research we will do a more detailed investigation of the tools available and we will compare them with our solution.

SAP has the ability to log all events and data which is entered or changed in system. However this functionality is in most implementations only turned on for debugging purposes. SAP also has a built-in workflow system. In the system we will be using during this research, this workflow system is also used for the purchasing process. We will combine data retrieved from the workflow system with data which is stored in the ‘normal’ SAP-tables to generate our event log. The event log format will be based on the datamodel which was introduced by [vdAvHvdW+09].

We would like to develop a tool which supports the following functionality:

- Define and store business rules
- Generate an event log from data extracted from an SAP-system
- Analyze this event log and check for compliance to business rules and regulations

The functionality of the tool we are going to develop translates to the following components of the top level architecture of an Online Auditing Tool as it was introduced by [vdAvHvdW+09] which is shown in figure 1.2:

- Define and store business rules, where definition should not require any programming or other IT-related skills.
- Generate an event log from data extracted from an SAP-system.
- Analyze this event log and check for compliance to business rules and regulations.

1.2.2 Scoping

This research will focus on the conformance checker part of the conceptual model introduced by [vdAvHvdW+09] and on the transformation of actual real-life definitions in constraints (i.e. queries), thus allowing us to implement the conformance checker with a SQL engine.

5.1.1. Business Data Definition

Processes involve business data, e.g., entities like invoices, products and customers. To describe the type of business data and the relationships between these data elements, we introduce the Business Data Definition. It stores the Entity Types of business data and the binary relationship between them. In fact, this component stores general data models as introduced in Section 2. However, we link them via form links to tasks.

4.4.2 Real-time ETL

If we want to build an online auditing system, we need to do ETL in real-time. However, currently our data extraction tool does not support real-time extraction and we can only filter data which needs to be extracted on predefined values. We could of course keep track of the records which are extracted in our own tool and let the extraction tool only extract records with an higher ID from the SAP-system, but then we would miss changes to records which are not recorded in the CDHDR and CDPOS tables.

Because solving this problem requires a significant amount of work, it will not be part of this research assignment.

4.5 Conformance Checker

4.5.1 Connection to CPN Tools

Our compliance checker will communicate with CPN Tools using TCP-sockets. The CPN-simulator can be setup and controlled by sending messages in APN-ML over these TCP-sockets. We will use the commands of APN-ML to send the complete Petri net to the simulator and to control the simulation.

4.5.2 Execution loop

Once our conformance checker is setup and the Petri Net with all applicable business rules is built and loaded into the Petri Net simulator, it will enter a loop. Each cycle of this loop checks 1 entry of event log and goes through the following steps:

- Load event entry (from logfile / database / external application)
1.2 Problem Definition

Later in this research we will do a more detailed investigation of the tools available and we will compare them with our solution.

SAP has the ability to log all events and data which is entered or changed in system. However this functionality is in most implementations only turned on for debugging purposes. SAP also has a built-in workflow system. In the system we will be using during this research, this workflow system is also used for the purchasing process. We will combine data retrieved from the workflow system with data which is stored in the 'normal' SAP-tables to generate our event log. The event log format will be based on the datamodel which was introduced by [vdAvHvdW +09].

We would like to develop a tool which supports the following functionality:

- Define and store business rules
- Generate an event log from data extracted from an SAP-system
- Analyze this event log and check for compliance to business rules and regulations

The functionality of the tool we are going to develop translates to the following components of the top level architecture of an Online Auditing Tool as it was introduced by [vdAvHvdW +09] which is shown in figure 1.2:

- Define and store business rules, where definition should not require any programming or other IT-related skills.
- Generate an event log from data extracted from an SAP-system.
- Analyze this event log and check for compliance to business rules and regulations.

1.2.2 Scoping

This research will focus on the conformance checker part of the conceptual model introduced by [vdAvHvdW +09] and on the transformation of actual real-life event log data.

Figure 4.4: Execution Loop

- Set marking of 'local'-places using data from the event-log
- Check enabledness of transition corresponding to the current task
- Fire the transition
- Mark event entry 'completed'

This loop and the communication between the different components involved is illustrated in figure 4.4.

4.5.3 Handling of violations

When a violation of the business rules occurs, the transition which should be fired next is not enabled. Because we always check the whether or not the transition is enabled before firing it, we can detect violations of the business rules before forcing the Petri Net simulator into exceptions. This way we can check the markings of all input-places and find out why a transition is not enabled. This in turn gives us to possibility to give a detailed report of which business rule was violated and in some cases to which extend it was violated (i.e. report whether the approval limit is exceeded by 1 euro or by 1 million euro).

The main problem about a violation of a business rule is however not the question of how to report, but the question of what to do next. There a few possibilities and all seem to have some advantages and disadvantages. In table 4.1 you will find the possibilities we discussed with their advantages and disadvantages. A good solution would be to give an auditor the chance to choose from this possibilities once a violation has been found.
Figure 1.1: A top level architecture of an Online Auditing Tool

Figure 1.2: A top level architecture of an Online Auditing Tool

Later in this research we will do a more detailed investigation of the tools available and we will compare them with our solution. SAP has the ability to log all events and data which is entered or changed in system. However this functionality is in most implementations only turned on for debugging purposes. SAP also has a built-in workflow system. In the system we will be using during this research, this workflow system is also used for the purchasing process. We will combine data retrieved from the workflow system with data which is stored in the ‘normal’ SAP-tables to generate our event log. The event log format will be based on the datamodel which was introduced by [vdAvHvdW+09].

We would like to develop a tool which supports the following functionality:

- Define and store business rules
- Generate an event log from data extracted from an SAP-system
- Analyze this event log and check for compliance to business rules and regulations

The functionality of the tool we are going to develop translates to the following components of the top level architecture of an Online Auditing Tool as it was introduced by [vdAvHvdW+09] which is shown in figure 1.2:

- Define and store business rules, where definition should not require any programming or other IT-related skills. — Dejure Models - Business Rules
- Generate an event log from data extracted from an SAP-system — Arrows from the information system to the runtime data
- Analyze this event log and check for compliance to business rules and regulations — Conformance checker

### 1.2.2 Scoping

This research will focus on the conformance checker part of the conceptual model introduced by [vdAvHvdW+09] and on the transformation of actual real-life

<table>
<thead>
<tr>
<th>Next step</th>
<th>(Dis)advantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mark complete case as violation and do not check any other events of the</td>
<td>Advantages:</td>
</tr>
<tr>
<td>case</td>
<td>• Easy to implement</td>
</tr>
<tr>
<td></td>
<td>• No chance of putting the simulator into exceptions</td>
</tr>
<tr>
<td></td>
<td>Disadvantages:</td>
</tr>
<tr>
<td></td>
<td>• Further events in the same case are not checked for violations</td>
</tr>
<tr>
<td>Mark event as violation, revert back to situation before event and skip</td>
<td>Advantages:</td>
</tr>
<tr>
<td>event</td>
<td>• No chance of putting the simulator into exceptions</td>
</tr>
<tr>
<td></td>
<td>Disadvantages:</td>
</tr>
<tr>
<td></td>
<td>• Event-order business rules whose status depend on this event are not checked</td>
</tr>
<tr>
<td></td>
<td>correctly for future events</td>
</tr>
<tr>
<td>Mark event as violation, fire transition which simulates the event has</td>
<td>Advantages:</td>
</tr>
<tr>
<td>completed</td>
<td>• Further events in the same case are checked for violations</td>
</tr>
<tr>
<td></td>
<td>• Event-order rules are all checked</td>
</tr>
<tr>
<td></td>
<td>Disadvantages:</td>
</tr>
<tr>
<td></td>
<td>• High risk of putting the simulator into exceptions</td>
</tr>
<tr>
<td></td>
<td>• Complex implementation</td>
</tr>
</tbody>
</table>

Table 4.1: Handling of violations
Chapter 5

Implementation

5.1 Technology

5.1.1 Microsoft .NET

The conformance checker will be build using Microsoft .NET technology.

- Uses Microsoft technologies which are also used by other tools developed at Deloitte (for example the data extraction tool, which we will be using to extract data from the SAP-system).
- Lots of resources and documentation are available for these technologies
- Proven technology

5.1.2 CPN Tools

The simulation of the Petri net we will build will be simulated using CPN Tools. CPN Tools (CPN stands for Colored Petri Nets) is a Petri Net modeling and simulating tool developed by the CPN Group, University of Aarhus, Denmark [1] and currently maintained at the Eindhoven University of Technology. We have compared the tool against some other tools as shown in Table 5.1. The main reasons for choosing CPN over other simulation tools are:

- The simulation engine from the tool can also be used without the included GUI. It is possible to communicate directly with the simulation engine over TCP.
- It supports Petri nets, and as outlined in Chapter 2 Petri nets are a well suited modeling technique for our rule checking monitor.
- It is a mature Petri net simulation engine without any graphical user interface, whose main purpose is to simulate Petri nets efficiently, we may assume that these core Petri net simulation tasks are processed very efficiently. Tests using the included user interface also showed that the performance of the simulation engine was good.
To enable communication with the CPN simulation engine from our own tool we need to run the simulation engine on a computer running Linux and connect to the simulation engine using TCP-communication. Implementing the communication with the simulation engine is a cumbersome project. It requires sending messages using the APN ML protocol, which is not very well documented. Although not all functionality of the protocol has been implemented in the current version of the tool, we managed to implement all functionality necessary to load a model into the simulator and to control the simulation. For now this will be enough to run our case study.

### 5.2 Performance

The main goal of this research is to develop a system which is capable of doing real-time conformance checking. Considering our case study which has roughly 1 million events in a period 6 months, this means our system should be able to handle 15 events per minute at the very least. After running some tests on an artificial log this seems no problem at all. Our tool processed 1000 events in a matter of seconds.

However when the size of the log grows the simulation engine stops responding at a certain point. Investigating the memory and cpu usage of the simulation process did not indicate any problems. Later on when simulating the log from the case study, which has a lot more tasks and therefore a lot more places in the Petri net, we discovered that the problem arises earlier if the simulator is simulating a bigger Petri net. This indicates that there is a problem with storing large state spaces. However because this problem did not prohibit us to prove our technology, we decided not to invest too much time into solving this problem and to just work around it during the case study.

<table>
<thead>
<tr>
<th>Tool</th>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPN Tools [1]</td>
<td>Fast, tcp interface allows control from an external tool</td>
<td>cons</td>
</tr>
<tr>
<td>Yasper [3]</td>
<td>Easy to use, source available</td>
<td>No interface to control from an external tool</td>
</tr>
<tr>
<td>Renew [5]</td>
<td>Source available so should be possible to integrate in a tool</td>
<td>Java-library, requires Java knowledge</td>
</tr>
<tr>
<td>ProM [4]</td>
<td>Source available so should be possible to integrate in a tool</td>
<td>Not meant as Petri net simulator. Complex product.</td>
</tr>
</tbody>
</table>

Table 5.1: Petri net simulation options
Chapter 6

Case Study

6.1 Process Context

A case study has been performed at a large public sector company. In this case study we will investigate the purchase-to-pay process. The client’s main information system is SAP with a working workflow system. We extracted data from their system which included all invoices from the past 6 months. Some metrics of the resulting log are shown in table 6.1 and a complete list of events occurring in this log with their total frequency, the number of times they occur as the first event of case and the number of times they occur as the last event is shown in table 6.2.

<table>
<thead>
<tr>
<th>Cases</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Events</td>
<td>Total</td>
</tr>
<tr>
<td></td>
<td>Tasks</td>
</tr>
<tr>
<td>Actors</td>
<td>Total</td>
</tr>
<tr>
<td>Amount</td>
<td>Total</td>
</tr>
<tr>
<td></td>
<td>Average</td>
</tr>
<tr>
<td>Throughput time</td>
<td>Min (days)</td>
</tr>
<tr>
<td></td>
<td>Max (days)</td>
</tr>
<tr>
<td></td>
<td>Avg (days)</td>
</tr>
</tbody>
</table>

Table 6.1: Metrics of log

6.2 Business Rules

Because our log was rather limited, it only included data for the invoicing phase of the purchase-to-pay process and did not have the proper usernames recorded in all events, our choice of business rules was also limited. And because we also wanted to verify the results of our newly developed tool we choose to test some business rules which we can also test using rather straight-forward SQL-queries on our event log. This verification requirement, combined with the fact that we cannot yet check on which of the business rules a case fails, made us decide to check no more then 2 business rules. As checking more business rules results
Later in this research we will do a more detailed investigation of the tools available and we will compare them with our solution.

SAP has the ability to log all events and data which is entered or changed in system. However this functionality is in most implementations only turned on for debugging purposes. SAP also has a built-in workflow system. In the system we will be using during this research, this workflow system is also used for the purchasing process. We will combine data retrieved from the workflow system with data which is stored in the ‘normal’ SAP-tables to generate our event log.

The event log format will be based on the datamodel which was introduced by [vdAvHvdW +09].

We would like to develop a tool which supports the following functionality:

• Define and store business rules
• Generate an event log from data extracted from an SAP-system
• Analyze this event log and check for compliance to business rules and regulations

The functionality of the tool we are going to develop translates to the following components of the top level architecture of an Online Auditing Tool as it was introduced by [vdAvHvdW +09] which is shown in figure 1.2:

• Define and store business rules, where definition should not require any programming or other IT-related skills. ¡=¿ Dejure Models - Business Rules
• Generate an event log from data extracted from an SAP-system ¡=¿ Arrows from the information system to the runtime data
• Analyze this event log and check for compliance to business rules and regulations ¡=¿ Conformance checker

1.2.2 Scoping

This research will focus on the conformance checker part of the conceptual model introduced by [vdAvHvdW+09] and on the transformation of actual real-life

<table>
<thead>
<tr>
<th>Tasks</th>
<th>Start of # cases</th>
<th>End of # cases</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start: Receive logistic invoice</td>
<td>2813</td>
<td>0</td>
<td>2847</td>
</tr>
<tr>
<td>Start: Receive light invoice</td>
<td>2599</td>
<td>0</td>
<td>2735</td>
</tr>
<tr>
<td>04: Add remark</td>
<td>0</td>
<td>0</td>
<td>70</td>
</tr>
<tr>
<td>04: Change registration</td>
<td>0</td>
<td>0</td>
<td>3074</td>
</tr>
<tr>
<td>04: Check registration</td>
<td>0</td>
<td>0</td>
<td>6318</td>
</tr>
<tr>
<td>06: Change registration</td>
<td>0</td>
<td>0</td>
<td>200</td>
</tr>
<tr>
<td>06: Choose purchasing group</td>
<td>0</td>
<td>0</td>
<td>165</td>
</tr>
<tr>
<td>06: Process registration</td>
<td>0</td>
<td>0</td>
<td>379</td>
</tr>
<tr>
<td>11: Add remark</td>
<td>0</td>
<td>0</td>
<td>118</td>
</tr>
<tr>
<td>11: Approve scanned document</td>
<td>0</td>
<td>0</td>
<td>1113</td>
</tr>
<tr>
<td>11: Choose purchasing group</td>
<td>0</td>
<td>0</td>
<td>803</td>
</tr>
<tr>
<td>11: Enter text</td>
<td>0</td>
<td>0</td>
<td>25</td>
</tr>
<tr>
<td>11: Show registration</td>
<td>0</td>
<td>0</td>
<td>60</td>
</tr>
<tr>
<td>141: Add attachment</td>
<td>0</td>
<td>0</td>
<td>46</td>
</tr>
<tr>
<td>141: Change purchase order</td>
<td>0</td>
<td>0</td>
<td>870</td>
</tr>
<tr>
<td>141: Enter text</td>
<td>0</td>
<td>0</td>
<td>412</td>
</tr>
<tr>
<td>141: Process invoice</td>
<td>0</td>
<td>0</td>
<td>1618</td>
</tr>
<tr>
<td>18: Add attachment</td>
<td>0</td>
<td>0</td>
<td>505</td>
</tr>
<tr>
<td>18: Create logistic invoice with MIR4</td>
<td>0</td>
<td>0</td>
<td>611</td>
</tr>
<tr>
<td>18: Create logistic invoice with OCR</td>
<td>0</td>
<td>0</td>
<td>3603</td>
</tr>
<tr>
<td>18: Enter text</td>
<td>0</td>
<td>0</td>
<td>216</td>
</tr>
<tr>
<td>18: Process scanned document</td>
<td>0</td>
<td>0</td>
<td>5156</td>
</tr>
<tr>
<td>28: Change registration</td>
<td>0</td>
<td>0</td>
<td>32</td>
</tr>
<tr>
<td>28: Choose purchasing group</td>
<td>0</td>
<td>0</td>
<td>2744</td>
</tr>
<tr>
<td>29: Add remark</td>
<td>0</td>
<td>0</td>
<td>44</td>
</tr>
<tr>
<td>29: Cancel logistic invoice</td>
<td>0</td>
<td>0</td>
<td>382</td>
</tr>
<tr>
<td>29: Handle logistic invoice</td>
<td>0</td>
<td>0</td>
<td>440</td>
</tr>
<tr>
<td>32: Add individual remark</td>
<td>0</td>
<td>0</td>
<td>146</td>
</tr>
<tr>
<td>32: Approve scanned document</td>
<td>0</td>
<td>0</td>
<td>1723</td>
</tr>
<tr>
<td>32: Enter text</td>
<td>0</td>
<td>0</td>
<td>98</td>
</tr>
<tr>
<td>32: Show registration</td>
<td>0</td>
<td>0</td>
<td>735</td>
</tr>
</tbody>
</table>
Later in this research we will do a more detailed investigation of the tools available and we will compare them with our solution.

SAP has the ability to log all events and data which is entered or changed in system. However this functionality is in most implementations only turned on for debugging purposes. SAP also has a built-in workflow system. In the system we will be using during this research, this workflow system is also used for the purchasing process. We will combine data retrieved from the workflow system with data which is stored in the ‘normal’ SAP-tables to generate our event log.

The event log format will be based on the datamodel which was introduced by [vdAvHvdW+09].

We would like to develop a tool which supports the following functionality:

- Define and store business rules
- Generate an event log from data extracted from an SAP-system
- Analyze this event log and check for compliance to business rules and regulations

The functionality of the tool we are going to develop translates to the following components of the top level architecture of an Online Auditing Tool as it was introduced by [vdAvHvdW+09] which is shown in figure 1.2:

- Define and store business rules, where definition should not require any programming or other IT-related skills.
- Generate an event log from data extracted from an SAP-system.
- Analyze this event log and check for compliance to business rules and regulations.

1.2.2 Scoping

This research will focus on the conformance checker part of the conceptual model introduced by [vdAvHvdW+09] and on the transformation of actual real-life tasks.

<table>
<thead>
<tr>
<th>Tasks</th>
<th>Start of # cases</th>
<th>End of # cases</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>34: Add remark</td>
<td>0</td>
<td>0</td>
<td>138</td>
</tr>
<tr>
<td>34: Choose group and purchaser</td>
<td>0</td>
<td>0</td>
<td>2068</td>
</tr>
<tr>
<td>34: Create goods receipt</td>
<td>0</td>
<td>0</td>
<td>31</td>
</tr>
<tr>
<td>34: Display order</td>
<td>0</td>
<td>0</td>
<td>1756</td>
</tr>
<tr>
<td>34: Enter text</td>
<td>0</td>
<td>0</td>
<td>174</td>
</tr>
<tr>
<td>34: Handle Q block (group)</td>
<td>0</td>
<td>0</td>
<td>4366</td>
</tr>
<tr>
<td>35: Add remark</td>
<td>0</td>
<td>0</td>
<td>530</td>
</tr>
<tr>
<td>35: Create purchase order</td>
<td>0</td>
<td>0</td>
<td>1184</td>
</tr>
<tr>
<td>35: Enter text</td>
<td>0</td>
<td>0</td>
<td>780</td>
</tr>
<tr>
<td>35: Handle Q block individual</td>
<td>0</td>
<td>0</td>
<td>4821</td>
</tr>
<tr>
<td>35: Show purchase order</td>
<td>0</td>
<td>0</td>
<td>1165</td>
</tr>
<tr>
<td>38: Add attachment</td>
<td>0</td>
<td>0</td>
<td>442</td>
</tr>
<tr>
<td>38: Create registration OCR</td>
<td>0</td>
<td>0</td>
<td>3617</td>
</tr>
<tr>
<td>38: Determine process type</td>
<td>0</td>
<td>0</td>
<td>77</td>
</tr>
<tr>
<td>38: Enter text</td>
<td>0</td>
<td>0</td>
<td>208</td>
</tr>
<tr>
<td>38: Process scanned document</td>
<td>0</td>
<td>0</td>
<td>4478</td>
</tr>
<tr>
<td>53: Add remark</td>
<td>0</td>
<td>0</td>
<td>51</td>
</tr>
<tr>
<td>53: Change purchase order</td>
<td>0</td>
<td>0</td>
<td>200</td>
</tr>
<tr>
<td>53: Enter text</td>
<td>0</td>
<td>0</td>
<td>21</td>
</tr>
<tr>
<td>53: Resolve P block</td>
<td>0</td>
<td>0</td>
<td>429</td>
</tr>
<tr>
<td>53: Retrieve purchase orders for group</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Process credit note</td>
<td>0</td>
<td>0</td>
<td>561</td>
</tr>
<tr>
<td>Request credit note</td>
<td>0</td>
<td>0</td>
<td>108</td>
</tr>
<tr>
<td>End: Approve light invoice for payment</td>
<td>0</td>
<td>2658</td>
<td>2658</td>
</tr>
<tr>
<td>End: Approve logistic invoice for payment</td>
<td>0</td>
<td>2754</td>
<td>2754</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>5412</strong></td>
<td><strong>5412</strong></td>
<td><strong>69906</strong></td>
</tr>
</tbody>
</table>

Table 6.2: Events in the log
in a lot of cases violating more than 1 business rule and therefore in a more
difficult verification. We choose the following 2 business rules:

- Restrict update: It is not allowed to change an invoice after it has been
  approved for payment.

- Limit number of repetitions: The task "Handle Q block (group)" should
  not be executed more than 3 times.

### 6.3 Hypothesis

Because our log was constructed using the assumption that approving payment
is the final step in a case, none of the cases will violate rule 1.

We checked the same business rules by running SQL-queries on the same
database-table in the event log. The SQL-query for the second business rules
looks like this:

```sql
SELECT [caseId]
FROM [EventLog].[dbo].[Events]
WHERE TaskName = '34: Handle Q block (group)'
GROUP BY caseId
HAVING COUNT(DISTINCT EventId) > 3
```

This query resulted in a set of 366 case identifiers. As rule 1 should not be
violated by any case, our hypothesis is that the tool reports the exact same set
of case identifiers.

### 6.4 Execution

The log file is put into chronological order to feed the CPN simulator. The
default net is extended with the specific additions needed to validate the 2
business rules and is loaded into the CPN simulator. A visual representation
of this Petri net is shown in figure 6.1.

In each execution cycle of our tool the following steps are executed:

1. Load 1 event from the log

2. Check if the case is already started, otherwise:
   
   (a) Place token with case id in place 'IDs'
   
   (b) Try to fire transition 'Start case'

3. Check if the transition corresponding to the task of the loaded event is
   enabled. If enabled, fire this transition, otherwise report a violation for
   the corresponding case.
The violations mentioned in step 3 resulted in a list of case identifiers of cases which fail 1 of the aforementioned business rules. Comparing this list with the list mentioned in the hypothesis resulted in no differences. This means our hypothesis is not falsified and our tool worked as expected.

### 6.5 Performance

The simulation was run on a laptop with an Intel Core 2 Duo T75000 @ 2.20GHz with 2GB of internal memory. Main operating system of this laptop is Windows Vista 32-bit. A virtual machine with Ubuntu 9.10 and 256MB RAM was run on this laptop in order to facilitate the CPN simulation engine.

The simulation of the complete log of the case study, consisting of 69906 events and 2 business rules, took 80 minutes to complete. Limiting resource during this simulation was the CPU capacity on the virtual machine. So running the simulation engine on dedicated hardware with a more recent processor should increase the performance significantly. Memory usage was almost constant during the execution of the simulation and total memory usage never came near memory capacity nor on the virtual ubuntu machine nor on the host windows machine.
Chapter 7

Conclusions and Recommendations

In this chapter, we will draw conclusions about the research. The main objective of the research was to find out if it is possible to use simulation of Petri nets in order to check whether a given log of a certain process complies with a predefined set of business rules on this process.

In the first part of this research we were able to develop a model on how to build a coloured Petri net which only accepts log files which comply with the corresponding set of business rules. We have proved that we can use this method to combine any combination of business rules in one Petri net as long the separate business rules are suitable for our method. To build a Petri net from business rules we have introduced a set of 11 templates, which can be used in any combination in order to build the Petri net. We have also discussed that this method is not suitable for all sorts of business rules, but it does suit a very large and common set of business rules.

After this we have developed a tool which shows this theory in practice and in fact proves that it is possible to use simulation of coloured Petri nets to check whether or not a certain log violates a predefined set of business rules. We have used this tool in a case study on a set of data from a large public company. This case study also proved that our method is working. It also showed us that the tool we have developed would be fast enough to check the business rules in real-time, i.e. right after the moment they are executed.

7.1 Further research

Although we have been able to develop a working prototype for our business rule monitor, there are still some crucial features missing in the tool. As discussed in this research it is currently not possible to determine which business rule has been violated once a transition which should fire is not enabled. It should however be possible to determine which rule was blocking the simulation of the Petri net. Therefore a method to determine this should be part of any
further research.

Furthermore in order to have a fully real-time business rule monitor we do not only need a tool which is able to check the occurring events fast enough, but we will also need a way to retrieve the logdata in real-time. In our case study it took a considerable amount of time and research to create the event log used by monitor from the data retrieved from the information system. The generation of such event logs should therefore be further automated and a method to generate this data in real-time should be the subject of further research.
Bibliography


Later in this research we will do a more detailed investigation of the tools available and we will compare them with our solution. SAP has the ability to log all events and data which is entered or changed in system. However this functionality is in most implementations only turned on for debugging purposes. SAP also has a built-in workflow system. In the system we will be using during this research, this workflow system is also used for the purchasing process. We will combine data retrieved from the workflow system with data which is stored in the ‘normal’ SAP-tables to generate our event log. The event log format will be based on the datamodel which was introduced by [vdAvHvdW+09].

We would like to develop a tool which supports the following functionality:

• Define and store business rules
• Generate an event log from data extracted from an SAP-system
• Analyze this event log and check for compliance to business rules and regulations

The functionality of the tool we are going to develop translates to the following components of the top level architecture of an Online Auditing Tool as it was introduced by [vdAvHvdW+09] which is shown in figure 1.2:

• Define and store business rules, where definition should not require any programming or other IT-related skills. Dejure Models - Business Rules
• Generate an event log from data extracted from an SAP-system Arrows from the information system to the runtime data
• Analyze this event log and check for compliance to business rules and regulations Conformance checker

1.2.2 Scoping

This research will focus on the conformance checker part of the conceptual model introduced by [vdAvHvdW+09] and on the transformation of actual real-life
