

# Abstracting Common Business Rules to Petri Nets

Kees van Hee<sup>1</sup>, Jan Hidders<sup>2</sup>, Geert-Jan Houben<sup>2</sup>, Jan Paredaens<sup>3</sup>, and  
Philippe Thiran<sup>4</sup>

<sup>1</sup> Department of Mathematics and Computer Science  
Technische Universiteit Eindhoven  
`k.m.v.hee@tue.nl`

<sup>2</sup> Software Technology Department  
Technische Universiteit Delft

`{a.j.h.hidders, g.j.p.m.houben}@tudelft.nl`

<sup>3</sup> Department of Mathematics and Computer Science  
Universiteit Antwerp  
`jan.paredaens@ua.ac.be`

<sup>4</sup> PRcISE Research Centre, Faculty of Computer Science  
Facultés Universitaires Notre-Dame de la Paix, Namur  
`philippe.thiran@fundp.ac.be`

**Abstract.** Business information systems are mostly very complex and difficult to modify. As a consequence, if we would like to ensure that certain business rules are enforced in a business, it is often easier to design a separate information system, called a monitor, that collects the business events from the business information system in a log and verifies whether the rules are satisfied. If not, the monitor can report the problem or it can intervene in the business information system. This requires that the business rules are specified in a precise language that allows to verify them over log files. In addition it should use a vocabulary that is familiar to business experts, and use terms such as cases, resources, agents and tasks. We introduce such a language and show that it can readily express many common types of business rules. We do not need the whole log but only an abstraction to those (aspects of) events that are relevant for the business rules. In addition we show that in many cases these rules can be translated to Petri nets which gives us not only an efficient way to detect violations of business rules, but also a way to analyze them. Such as Petri net can be seen as an abstraction of the business rules. The latter enables us to detect events that either may or must lead to violations later on. This way the monitor can report the risk of transgressions or even prevent them by intervening in the execution of the business information system.

## 1 Introduction and Motivation

In the past business information systems (BIS) were mainly used to support people in executing tasks in the business processes of an organization. Today we see that BIS have a much more responsible task: they execute large part of the business process autonomously. So organizations become more and more dependable

on their BIS. Business processes prescribe the order in which activities have to be executed. On top of that, there are many other *business rules* that should be met by the execution of business processes. Business rules can be required by any stakeholders, such as management, government (by law), shareholders and business partners (clients and suppliers).

It is practically impossible to verify if a BIS is satisfying the business rules. So we have to live with the fact that systems can always have some errors that may violate business rules. Normally *auditors* check periodically if the business rules are satisfied in the past period by inspection of logged data. In this paper we take it for granted that a BIS has some unknown errors and that the system is for all stakeholders more or less a black box. Instead of a human auditor we propose here an approach where the BIS is extended by an independent *monitor system* that checks the essential business rules on the fly and that reports violations of business rules (detective mode) or that interrupts the BIS to prevent the occurrence of a violation.

In Section 3 we give the necessary concepts for the approach and a set of characteristic business rules. In Section 4 we define a language for business rules and in Section 5 we express the set of characteristic rules from Section 3 in this language. In Section 6 we consider the relationship between *business rules* on the one hand and *business processes*, expressed as Petri nets, on the other hand. In fact we show how the business rules specify a process model that can be used to detect and prevent violations of the business rules. Here we use a process model not to control the business process like in workflow applications, but as a tool for runtime conformance checking.

## 2 Preliminaries

A labeled transition system is a 4-tuple  $(S, A, Tr, s)$  where  $S$  is the state space,  $A$  the set of *action labels*,  $Tr \subseteq S \times A \times S$  is the *transition relation* and  $s \in S$  is the initial state. A *run* of a transition system is a sequence  $(s_0, a_0, s_1, a_1, s_2, \dots, s_n)$  such that each triple  $(s_i, a_i, s_{i+1}) \in Tr$  with  $i \in \{0, 1, \dots, n-1\}$ . Next we introduce Petri nets, c.f. [5]. A *Petri net with inhibitors* is a 5-tuple  $(P, T, F, I, m_0)$ , where  $P \cap T = \emptyset$ ,  $P$  is the set of *places*,  $T$  the set of *transitions*,  $F \subseteq (P \times T) \cup (T \times P)$  the set of *normal arcs*,  $I \subseteq P \times T$  the set of *inhibitor arcs* and  $m_0 \in M$  the *initial marking* where  $M = P \rightarrow \mathbb{N}$  the set of all possible *markings*. We overload binary relations such as  $F$  and  $I$  also as the corresponding characteristic function, i.e.,  $F(x, y) = 1$  if  $(x, y) \in F$  and  $F(x, y) = 0$  if  $(x, y) \notin F$ . The *dynamics* of the Petri net are defined by a *labeled transition system*:  $(M, T, Tr, m_0)$  where  $M$  is the state space,  $T$  the set of action labels,  $m_0$  the initial marking and the transition relation  $Tr$  satisfies:

$$Tr = \{(m, t, m') \in M \times T \times M \mid \forall p \in P : \\ (p, t) \in F \Rightarrow m(p) > 0 \wedge (p, t) \in I \Rightarrow m(p) = 0 \wedge \\ m'(p) = m(p) - F(p, t) + F(t, p)\}$$

We let  $\bullet t$  and  $t\bullet$  denote input places and output places of  $t \in T$ , i.e.,  $\bullet t = \{p \mid (p, t) \in F\}$  and  $t\bullet = \{p \mid (t, p) \in F\}$ . Similarly, for a place  $p \in P$  we let  $\bullet p$  and  $p\bullet$  denote the producing transitions and consuming transitions, i.e.,  $\bullet p = \{t \mid (t, p) \in F\}$  and  $p\bullet = \{t \mid (p, t) \in F\}$ .

A *firing sequence* of a net is a sequence of transitions as they are encountered in a run.

### 3 Concepts

Organizations perform *business processes* in order to meet their *objectives* within the limits of a set of *business rules*. Business processes are sets of *tasks* with some precedence relationship. A business processes can have many instances concurrently. An instance of a business process is called a *case*. Tasks are executed for a case and they can be instantaneously, in which case the the task execution is an *event*, or the execution of a task takes time in which case we distinguish a *start event* and an *stop event*. Here we restrict ourselves for simplicity to the instantaneously case. Tasks are executed by *agents*, which can be humans or automated parts of the information system. An agent is allowed to execute a task if it is authorized by another agent that has the *right* to delegate the task. In reality agents have roles and tasks are specified by roles instead of agents. For simplicity we do not however consider the role concept here. The tasks manipulate objects called *resources*. Resources are created, deleted, used or transformed during a task execution. In the business process resources can be for instance materials, components, final products, energy, information or money. In a BIS the resources are always represented as data. Note that we follow the terminology that is used in *accounting information system* where the REA datamodel is used frequently. REA stands for Resource, Event and Agent, concepts with the same meaning as we gave them. The REA model is an entity relationship model for accounting information systems ([6], [4] and [3]).

Organizations become more and more dependent on their BIS. In fact the BIS *mirrors* in a sense the business processes internally. Due to many examples of mismanagement, the managers in organizations are held more and more accountable for executing the business processes according to business rules. So it is extremely important that the BIS satisfies the requirements implied by the business processes and the business rules. We speak of *process-aware information systems* c.f. [2] if the BIS actively controls the order of task execution. For the enforcement of the additional business rules we propose a *monitor system* that *detects* and reports violations of the business rules or that *prevents* violations by interrupting the BIS and triggering an exception handler. In order to do so the monitor system *records* relevant *events*, i.e. task executions of the original BIS. So the monitor is "hooked on" the original BIS and it intervenes the execution of it.

### 3.1 Nature of Business Rules

Examples of business rules are the famous Sarbanes-Oxley [1] rules and the General Accepted Accounting Principles(GAAP). The business rules that we encounter in practice mostly are of the following nature:

- *Task-order* related rules. Rules that prescribe or forbid certain task orders, such as: task *A* should always precede task *B* if task *C* has executed.
- *Agent-task-assignment* related rules. Rules that prescribe or forbid certain assignments of tasks to agents, such as: two specific tasks in the same process instance are not allowed to be executed by the same agent. This rule is known by auditors as the *four-eyes principle*.
- *Agent-task-delegation* rules. Rules that allow agents to give certain task execution rights to other agents, such as an agent can only be given a certain permission to execute a task by another agent that has that permission himself. Rules of this class are called *authorizations*.
- *Resource balancing* rules. Rules concerning the use of resources and in particular that these resources are always balanced in some form. An example is the *three-way matching rule* that prescribes that the number of items of a certain resource on an invoice should be equal to the number of items in a delivery and the number of items in the order. This rule is a well-known rule by financial auditors.

Combinations of them are possible as well.

### 3.2 Informal Examples of Business Rules

For illustrating the business rules, we adopt a simplified version of a purchase scenario in a bakery. The scenario concerns only the procurement of 3 products (i.e., resources): **butter, bread, salmon**. The procurement consists of the execution of the 5 following tasks: buy, packaging, packaging – grant, delivery, pay, use. The first and the last two tasks are carried out by the agent customer while the other ones are taken in charge by the agent baker. By packaging – grant, the agent baker can delegate the task packaging to his assistant (the agent assistant).

For this simple scenario, we give informal examples of each nature of business rules:

- *Task-order* related rules: “for every case, the tasks buy, packaging, delivery, pay and use happen in that order” or “no two distinct events for the same case can happen at the same moment”;
- *Agent-task-assignment* related rules: “for every case, the execution of the task buy is assigned to the agent customer”;
- *Agent-task-delegation* rules: “for every case, the agent baker can delegate the execution of the task packaging to the agent assistant”;
- *Resource balancing* rules: “for each case at each moment the total amount of used bread does not exceed the amount of previously delivered bread” or “if bread is bought then it has to be delivered in the same quantity”.

These examples will be used for illustrating the next sections.

## 4 Business Rule Language

In this section, we define the language BRL to formulate business rules. Business processes are described in terms of traces of tasks, which are defined in the following.

### 4.1 Process Schema

A *process schema* is defined as  $S = (Ta, R, \rho, A, C, B)$  where:

- $Ta$  is a finite set of *tasks*;
- $R$  a finite set of *resource types* describing a property of events that indicates how much of a certain resource was involved in the event;
- $\rho \subseteq Ta \times R$  the relationship between tasks and resource types that indicates that a certain task requires a certain resource;
- $A$  a finite set of *agents*;
- $C$  a countably infinite set of *case identifiers*;
- $B$  is a countably infinite set of *basic values*.

The set  $\mathbb{Q}$  is not part of the schema but will be used as both the set of timestamps in events and as the domain of numbers used to indicate numbers of resources. Given such a process schema  $S$  we let  $P_S = \{\mathbf{time}, \mathbf{case}, \mathbf{task}, \mathbf{agent}\} \cup R$  which denotes the set of *properties* of events, and we let  $D_S = Ta \cup A \cup B \cup \mathbb{Q}$  denote the set of *denotable property values*, and let  $V_S = D_S \cup C$  denote the set of all possible *property values*.

Given a process schema  $S = (Ta, R, \rho, A, C, B)$  we define an *event* over  $S$  as a partial function  $ev : P_S \rightarrow V_S$  such that

1.  $\mathbf{dom}(ev) = \{\mathbf{time}, \mathbf{case}, \mathbf{task}, \mathbf{agent}\} \cup \{r \mid (ev(\mathbf{task}), r) \in \rho\}$ ,
2.  $ev(\mathbf{time}) \in \mathbb{Q}$  is the timestamp of the event,
3.  $ev(\mathbf{case}) \in C$  is the case of which the event is a part,
4.  $ev(\mathbf{task}) \in Ta$  is the performed task and
5.  $ev(\mathbf{agent}) \in A$  is the agent that performs the task,
6.  $ev(r) \in \mathbb{Q}$  is the number of resources involved of type  $r$ .

A *trace*  $\alpha$  over a process schema  $S$  is then defined as a finite set of events over  $S$ . We use variables such as  $\alpha$  and  $\beta$  to denote such traces. The *active domain* of a trace  $\alpha$  is the set of property values occurring in  $\alpha$  and formally defined as  $\Delta(\alpha) = \{v \mid ev \in \alpha, (p, v) \in ev\}$ .

In the example of the purchase scenario, the process schema is defined as followed:  $S = (Ta, R, \rho, A, C, B)$  with tasks  $Ta = \{\text{buy, packaging, packaging – grant, delivery, pay, use}\}$ , resource types  $R = \{\mathbf{butter}, \mathbf{bread}, \mathbf{salmon}\}$ , task resource relationship  $\rho = Ta \times R$  and agents  $A = \{\text{customer, baker, assistant}\}$ .

## 4.2 Business Rules

We now proceed with defining the language BRL (Business Rule Language). Given a process schema, BRL is the language in which we specify the business rules. A business rule expresses a property of the traces of the process schema.

We start with postulating a countably infinite set of variables  $\mathcal{X}$ . We now define the sets of formulas  $F$  and of expressions  $E$  by the following abstract syntax grammar:

$$\begin{aligned} F &::= \neg F \mid (F \wedge F) \mid \langle P_S = E, \dots, P_S = E \rangle \mid E = E \mid E < E. \\ E &::= \mathcal{X} \mid D_S \mid \Sigma(\mathcal{X} : F)E \mid (E + E). \end{aligned}$$

Note that in expressions we can denote all denotable property values in  $D_S = Ta \cup A \cup B \cup \mathbb{Q}$  which excludes the elements in  $C$ , the case identifiers.

The semantics of formulas  $\varphi \in F$  are defined by the proposition  $\alpha, \Gamma \models \varphi$  where  $\Gamma : \mathcal{X} \rightarrow V_S$  is a variable binding. This proposition states that  $\varphi$  holds for the trace  $\alpha$  under the variable binding  $\Gamma$ . For expressions  $e \in E$  the semantics are defined by the proposition  $\alpha, \Gamma \models e \Rightarrow v$  which states that the value of  $e$  is  $v$  under the trace  $\alpha$  and the variable binding  $\Gamma$ . For a variable binding  $\Gamma$  and variable  $x \in \mathcal{X}$  and value  $v \in V_S$  we let  $\Gamma[x \mapsto v]$  denote the variable binding  $\Gamma'$  that is equal to  $\Gamma$  except that  $\Gamma'(x) = v$ .

The proposition  $\alpha, \Gamma \models \varphi$  is defined by induction on the syntactic structure of  $\varphi$  by the following rules:

1.  $\alpha, \Gamma \models \neg\varphi$  iff it is not true that  $\alpha, \Gamma \models \varphi$
2.  $\alpha, \Gamma \models (\varphi \wedge \psi)$  iff  $\alpha, \Gamma \models \varphi$  and  $\alpha, \Gamma \models \psi$
3.  $\alpha, \Gamma \models \langle p_1 = e_1, \dots, p_k = e_k \rangle$  iff there exists an event  $ev \in \alpha$  such that  $\alpha, \Gamma \models e_i \Rightarrow ev(p_i)$  for all  $1 \leq i \leq k$
4.  $\alpha, \Gamma \models e_1 = e_2$  iff there exists a value  $v \in V_S$  such that  $\alpha, \Gamma \models e_1 \Rightarrow v$  and  $\alpha, \Gamma \models e_2 \Rightarrow v$
5.  $\alpha, \Gamma \models e_1 < e_2$  iff there exist  $v_1, v_2 \in \mathbb{Q}$  such that  $\alpha, \Gamma \models e_1 \Rightarrow v_1$  and  $\alpha, \Gamma \models e_2 \Rightarrow v_2$  and  $v_1 < v_2$

The proposition  $\alpha, \Gamma \models e \Rightarrow x$  is defined with induction on the syntactic structure of  $e$  by the following rules:

1.  $\alpha, \Gamma \models x \Rightarrow v$  iff  $\Gamma(x) = v$ , for  $x \in \mathcal{X}$
2.  $\alpha, \Gamma \models d \Rightarrow v$  iff  $d = v$ , for  $d \in D_S$
3.  $\alpha, \Gamma \models \Sigma(x : \varphi)e \Rightarrow w$  iff for some natural number  $n$  there exist  $n$  distinct  $v_1, \dots, v_n \in \Delta(\alpha)$  such that  $\{v_1, \dots, v_n\} = \{v \mid \alpha, \Gamma[x \mapsto v] \models \varphi\}$  and there exist  $w_1, \dots, w_n \in \mathbb{Q}$  such that  $\alpha, \Gamma[x \mapsto v_i] \models e \Rightarrow w_i$  for all  $1 \leq i \leq n$ , and  $w = \sum_{i=1}^n w_i$
4.  $\alpha, \Gamma \models e_1 + e_2 \Rightarrow w$  iff there exist  $v_1, v_2 \in \mathbb{Q}$  such that  $\alpha, \Gamma \models e_1 \Rightarrow v_1$  and  $\alpha, \Gamma \models e_2 \Rightarrow v_2$  and  $w = v_1 + v_2$ .

We also introduce the following short-hands. For logical disjunction we define  $(\varphi \vee \psi) \equiv \neg(\neg\varphi \wedge \neg\psi)$ . For logical implication:  $(\varphi \Rightarrow \psi) \equiv (\neg\varphi \vee \psi)$ . For existential quantification:  $(\exists x : \varphi) \equiv ((\Sigma(x : \varphi)1) > 0)$ . For universal quantification:  $(\forall x : \varphi) \equiv \neg(\exists x : \neg\varphi)$ . As usual we allow several quantifiers of the same type to be combined into one, e.g.,  $(\forall x : (\forall y : \varphi)) \equiv (\forall x, y : \varphi)$ .

## 5 Formal examples of Business Rules

To illustrate BRL we formally express some rules presented in Section 3.2.

The rule “for every case the tasks buy, packaging, delivery, pay and use happen in that order.” would be formulated in BRL as:  $\forall c, t_b, t_d, t_p, t_u : (\langle \mathbf{case} = c, \mathbf{task} = \text{buy}, \mathbf{time} = t_b \rangle \wedge \langle \mathbf{case} = c, \mathbf{task} = \text{packaging}, \mathbf{time} = t_b \rangle \wedge \langle \mathbf{case} = c, \mathbf{task} = \text{delivery}, \mathbf{time} = t_d \rangle \wedge \langle \mathbf{case} = c, \mathbf{task} = \text{pay}, \mathbf{time} = t_p \rangle \wedge \langle \mathbf{case} = c, \mathbf{task} = \text{use}, \mathbf{time} = t_u \rangle \Rightarrow (t_b < t_d < t_p < t_u))$ .

The rule “no two distinct events for the same case can happen at the same moment” can be formulated as follows:  $\forall c, t, k_1, k_2, a_1, a_2, br_1, br_2, bt_1, bt_2, sm_1, sm_2 : ((\langle \mathbf{case} = c, \mathbf{time} = t, \mathbf{task} = k_1, \mathbf{agent} = a_1, \mathbf{bread} = br_1, \mathbf{butter} = bt_1, \mathbf{salmon} = sm_1 \rangle \wedge \langle \mathbf{case} = c, \mathbf{time} = t, \mathbf{task} = k_2, \mathbf{agent} = a_2, \mathbf{bread} = br_2, \mathbf{butter} = bt_2, \mathbf{salmon} = sm_2 \rangle) \Rightarrow (k_1 = k_2 \wedge a_1 = a_2 \wedge br_1 = br_2 \wedge bt_1 = bt_2 \wedge sm_1 = sm_2))$ .

The rule “if bread is bought then it has to be delivered in the same quantity” becomes:  $\forall c, x : (\langle \mathbf{case} = c, \mathbf{task} = \text{buy}, \mathbf{bread} = x \rangle \Rightarrow \langle \mathbf{case} = c, \mathbf{task} = \text{delivery}, \mathbf{bread} = x \rangle)$ .

The rule “for each case at each moment the total amount of used bread does not exceed the amount of previously delivered bread” can be formulated as:

$$\begin{aligned} \forall c, t : (\Sigma(t' : (t' < t))(\Sigma(br : \langle \mathbf{case} = c, \mathbf{time} = t', \mathbf{task} = \text{use}, \mathbf{bread} = br \rangle)br)) \\ \leq \\ (\Sigma(t' : (t' < t))(\Sigma(br : \langle \mathbf{case} = c, \mathbf{time} = t', \mathbf{task} = \text{delivery}, \mathbf{bread} = br \rangle)br)) \end{aligned}$$

Note that this formulation uses double aggregation, i.e., for each preceding moment we sum all the different amounts of bread at that moment and these results are in turn summed up again. It assumes that at each moment there is at most one amount of bread involved, because if at the same moment the amount 10 is delivered twice it will still be counted only once in this rule.

## 6 From Business Rules to Business Processes

In fact we have already a very rudimentary process model, namely a model where every task  $t \in Ta$  may be executed in any order. This process model can be expressed as a Petri system with only transitions and empty initial marking  $(\emptyset, Ta, \emptyset, \emptyset)$ . However then the process model is not taking care of any business rule since we can execute the tasks in any order with any agent and any resources. The business rules however will forbid some task orders. Now assume we have a set of business rules  $\Phi$ . Their conjunction should be true. We will construct a labeled transition system that exactly obeys the business rules. So if we let the monitor system execute this system, in the sense that each event of the BIS is the label of a transition that is executed, then we can discover violations as soon as an event occurs for which there is no transition enabled. We model this as follows. We start with an empty (event) trace  $\epsilon$ . In fact the trace so far is the *state* of the system. At some point in time the system is in state  $\alpha$ . We allow a

task to be executed, leading to a new event  $ev$  and to a new trace  $(\alpha; ev)$  if and only if

$$\forall \phi \in \Phi : (\alpha; ev) \models \phi$$

(Here we used a shorthand for insertion of an element in a set:  $(\alpha; ev) = \alpha \cup \{ev\}$ .) We call the set of correct behaviors of the system  $H$ . Given a process schema  $S = (Ta, R, \rho, A, C, B)$  and the set of all possible events  $E$  where  $E = \{ev | ev : P_S \rightarrow V_S\}$ . The *correct behavior* of a process schema  $S$  is the smallest set  $H$  such that:  $\epsilon \in H$

$$(\alpha \in H \wedge ev \in E \wedge \forall \phi \in \Phi : (\alpha; ev) \models \phi) \Rightarrow (\alpha; ev) \in H$$

In fact we have now defined a *labeled transition system*  $(H, E, Tr, \epsilon)$  where  $H$  is the state space,  $E$  the set of action labels,  $\epsilon$  the initial marking and

$$Tr = \{(\alpha, ev, \alpha') \in H \times E \times H | (\alpha; ev) = \alpha'\}$$

is transition relation. So the correct behavior is the set of all allowable sequences of events. If for some  $\alpha \in H$  and some  $t \in Ta$

$$\forall ev \in E : ev(task) = t \wedge \exists \phi \in \Phi : \neg((\alpha; ev) \models \phi)$$

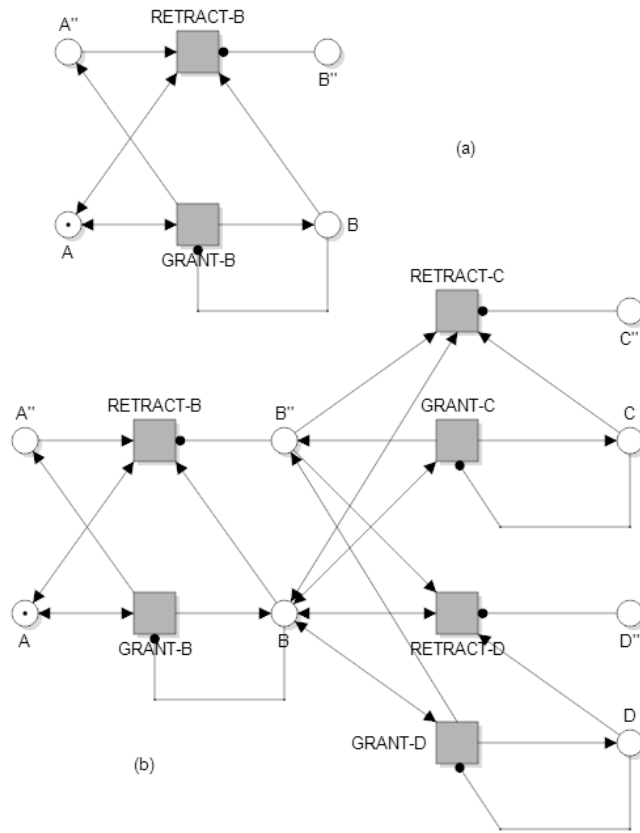
then we know that task  $t$  should not be executed if trace  $\alpha$  has occurred and we could try to generate an *interrupt* for the BIS to *prevent*  $t$  to execute. In that case the monitor system behaves as an external guard for task  $t$  in the BIS. If not, we can only detect and report the violation only after the execution of  $t$ . Note that the transition system  $(H, E, Tr, \epsilon)$  has an *infinite* state space  $H$ . Since the monitor system has to keep track of this system we need a finite representation of this transition system. In general this is not possible since we might have rules that need to keep infinite information of the past events, such as "the number of times task  $A$  has executed should be at most the number of times task  $B$  has executed for the same case". However there are (non trivial) subsets of the rule language for which finite aggregations of the event history are sufficient, which means that we can have a process model with a finite state space which allows for model checking. We will identify subclasses of the business rule language that allow us to construct a *history-dependent Petri nets* as introduced in [8] and [7]. There it is shown that each history dependent Petri net is equivalent (in fact *bisimilar*) to a Petri net with inhibitors. Under additional conditions [7] on the rules it even becomes a Petri net without inhibitors. A *history* of a Petri net is the sequence of transitions that have executed so far. A history dependent Petri net is an extension of Petri nets in which there is for each transition  $t$  a *guard*  $g(t)$  based on the *history* of the process. Like we have evaluated a business rule in the context of a trace, in history dependent Petri nets the guards are evaluated in the context of a history. The dynamics of a history dependent Petri net are defined by a labeled transition system  $(M \times T^*, T, Tr, (m_0, \epsilon))$  where  $T^*$

is the set of possible histories and  $Tr$  is defined by

$$Tr = \{((m, h), t, (m', (h; t))) \in (M \times T^*) \times T \times (M \times T^*) \mid h \models g(t) \wedge \forall p \in P : \\ (p, t) \in F \Rightarrow m(p) > 0 \wedge (p, t) \in I \Rightarrow m(p) = 0 \wedge \\ m'(p) = m(p) - F(p, t) + F(t, p)\}$$

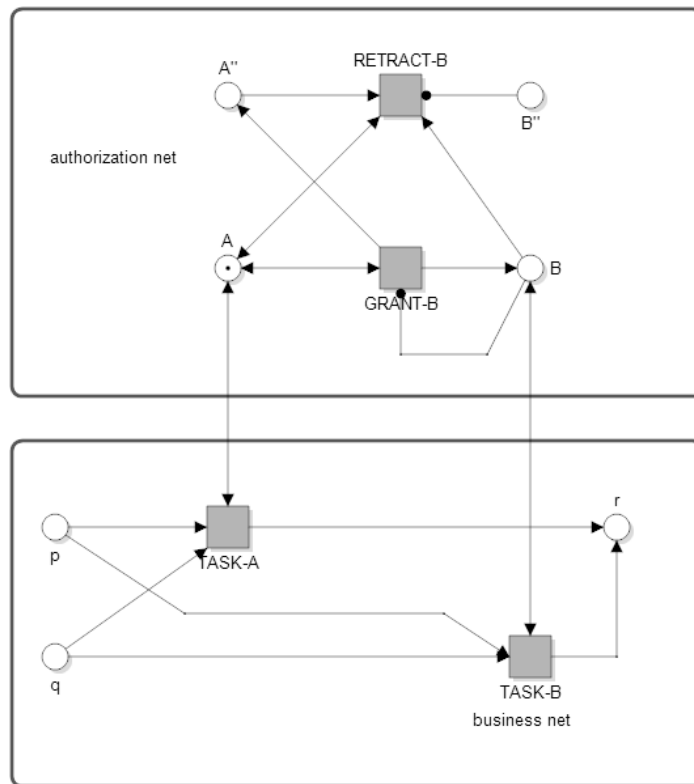
So far we have not reached much, since the history-dependent Petri net also has an infinite state space. However, under certain restrictions of the language the state space for the evaluation of the formulas becomes finite. An example of this is if the only allowed quantifiers are the universal and existential quantifiers, but not the summation quantifier, and in event formulas of the form  $\langle p_1 = e_1, \dots, p_n = e_n \rangle$  we require  $e_i$  to be a variable if  $p_i = \mathbf{time}$  and a constant otherwise. In that case the language is essentially that of first order logic over traces which is known to be strictly weaker than finite state machines in the sense that for each formula there is a finite state machine that determines the truth value of the formula for a certain trace. As stated above, we can transform history-dependent net into a bisimilar Petri net with inhibitors. In case the state space is finite, we can analyze the behavior by state space exploration.

Next we extend this approach to incorporate more elements of the business rules. First of all distinguish two kinds of events: events with a "real" task, correspond to real work in the organization and events with a *delegation* task. Note that if we can split the business rules  $\Phi$  into disjoint subsets  $\Phi = \Phi_{task} \cup \Phi_{agent} \cup \Phi_{resource}$  where  $\Phi_{task}$  only involves real tasks and no information about agents or resources,  $\Phi_{agent}$  only about tasks and agents and finally  $\Phi_{resource}$  about the rest. Then we can form with  $\Phi_{task}$  the history-dependent Petri net that we can transform in a Petri net with or without inhibitors [7]. We call it the *business net*. Next we consider the rules in  $\Phi_{agent}$ . So we assume we have a Petri net to describe the rules in  $\Phi_{task}$  and now we add another Petri net that takes care of the agents. Consider Fig. 1. Here we see two nets (a) and (b). In fact (a) is a *pattern* that we have repeated in (b). First we consider (a). The meaning of (a) is that we are considering one task, say  $t \in Ta$  and two agents A and B. The places in the nets have these names as labels and there are also places labeled A" and B". The meaning is that if there is a token in a place for agent X then X has the right to perform task  $t$  and if not then there is a token in place X". Place X" is marked with the number of "activities" (either delegations or task executions) in which the agent is involved. There is an invariant property for the places X and X"; if the number of tokens in place X" is greater than zero, then place X has one token. In the initial state only place A has one token. In the initial marking only place A has one token. The meaning is that A can grant B the right to perform task  $t$  and that A can also retract that right. In net (b) we see that A can delegate to B and B can delegate to C and D. Note that we have taken care of the "generic" business rule that an agent can only delegate a task if he is authorized to do the task himself and that he can only retract a delegation from an agent if this one has no pending delegations of that task himself. So for the Agent-task-delegation rules we are able to construct a Petri net using the pattern of Fig. 1 to implement the standard task-agent rules. Note that we



**Fig. 1.** Authorization process as Petri net

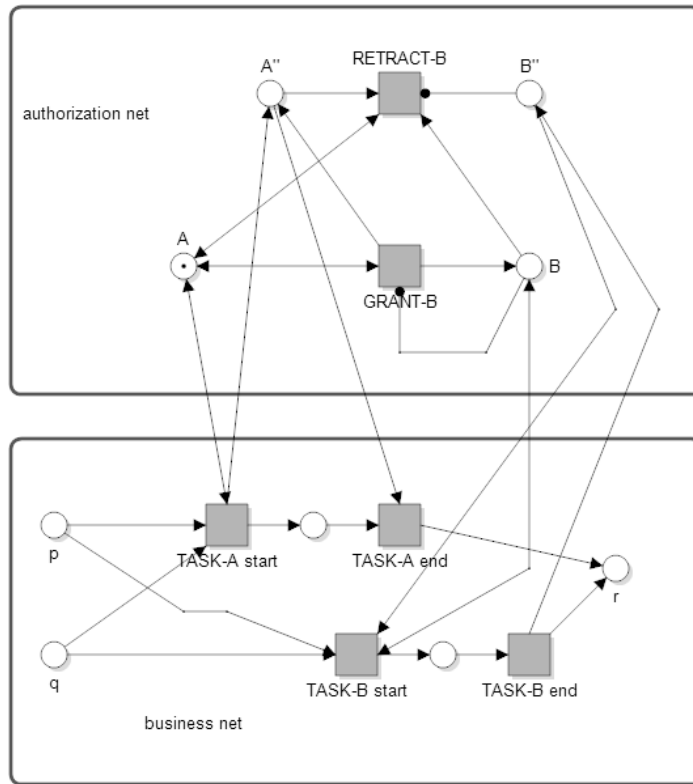
have such a net for each task. This can become quite a big net, but we will not construct it by hand, but we generate it from the business rules. We call this net the *authorization net*. The next question is how integrate this authorization net to the business process net. In Fig. 2 we see two fragments of a business net and an authorization net. We consider here the case that tasks are instantaneously. The business net is supposed to represent all the rules in  $\Phi_{task}$ . The business net is already modified, because the original net has one task (transition), called TASK with two input places  $p$  and  $q$  and one output place  $r$ . Since both agents A and B are potentially allowed to perform the TASK we have duplicated the task into TASK-A and TASK-B. Duplication means that they both have the same input and output places in the business net. In Fig. 3 we modified the the model to deal with the case where tasks in the business net have a start and an end transition.



**Fig. 2.** Integration of authorization net and business process net

From our business rules given in Section 3.2, the business rule “for every case, the agent baker can delegate the execution of the task packaging to the agent assistant” can illustrate the Petri net of Fig. 2 where the agent A refers to the agent baker and the agent B refers to the agent assistant.

The general rules for authorization are represented now. The rest of the rules can be taken into account by making the system *history-dependent* again, but now for the remaining rules only. So we can use the other rules as *guards* for transitions in our *monitor net*, the process model of our monitor system. So we make again a history-base Petri net and the question is what we have gained in this way. The answer is that we have reduced the amount of *rule evaluation* because many of the rules are transformed into the *execution rule* for Petri nets which can be evaluated easily at run time with a Petri net engine or workflow engine.



**Fig. 3.** Integration for the case with start and end transitions

## 7 Conclusion and future work

We have presented an approach to build a monitor system that is able to check business rules on the fly and in parallel to a business information system (BIS). The assumption is that we can not trust the BIS. We have seen how we can translate parts of the evaluation of business rules into the execution of a Petri net. This is an efficient way of checking rules because it can be done in an incremental way (i.e. event by event) using a Petri net engine. If the Petri net can not execute a transition, then a rule is violated and this can be reported or it may generate an interrupt for the BIS and trigger an exception handler in the BIS. In the future we will also try to transform the other types of business rules, in particular resource related rules into Petri nets.

## References

1. D. Berg, *Turning sarbanes-oxley projects into strategic business processes*, Sarbanes-Oxley Compliance Journal (2004).
2. M. Dumas, W.M.P. van der Aalst, and A. H. ter Hofstede, *Process aware information systems: Bridging people and software through process technology*, Wiley Interscience, 2005.
3. W.E. McCarthy, *The rea accounting model: a generalized framework for accounting systems in a shared dat environment*, The accounting review (1982).
4. ———, *An ontology analysis of the primitives of extended rea enterprise information architecture*, International Journal of Accounting Information Systems **3** (2002).
5. W. Reisig, *Petri Nets: An Introduction*, Monographs in Theoretical Computer Science: An EATCS Series, vol. 4, Springer-Verlag, Berlin, 1985.
6. P.J. Romney, M.B.;Steinbart, *Accounting information systems*, 11 ed., Pearson International Editions, 2009.
7. K. M. van Hee, A. Serebrenik, and N. Sidorova, *Token history nets*, Fundamenta Informaticae **85** (2008).
8. K. M. van Hee, A. Serebrenik, N. Sidorova, and W. M. P. van der Aalst, *History-dependent Petri nets*, Petri Nets and Other Models of Concurrency - ICATPN 2007 (Jetty Kleijn and Alex Yakovlev, eds.), Springer, 2007.