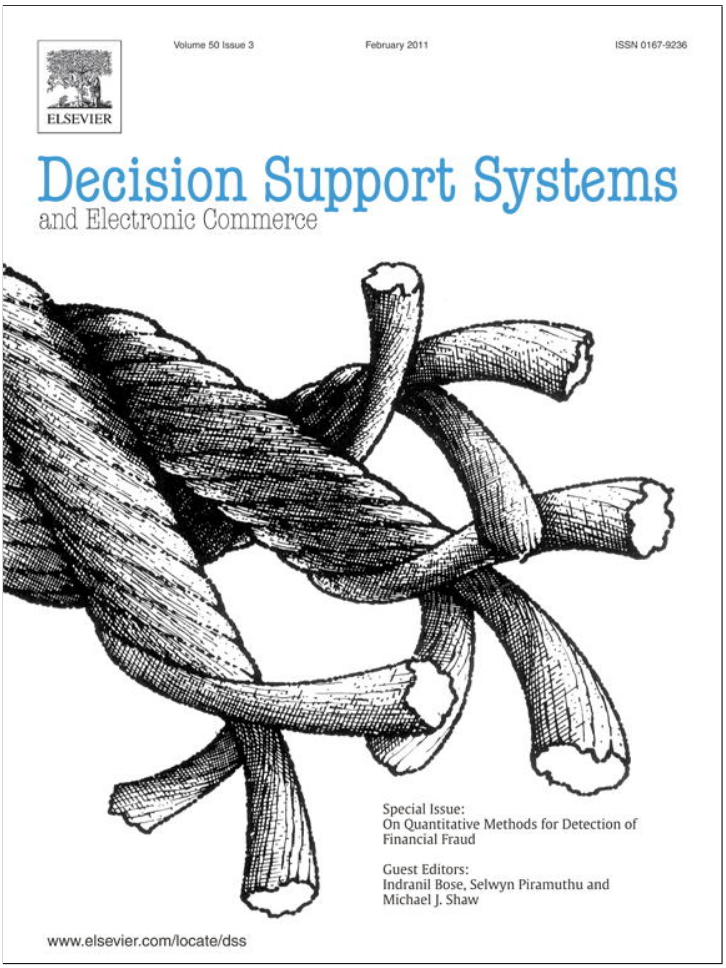


Decision Support Systems 50 (2011) 636–647

Contents lists available at ScienceDirect

Decision Support Systems

journal homepage: www.elsevier.com/locate/dss

# Conceptual model for online auditing

Wil van der Aalst [a,*], Kees van Hee [a], Jan Martijn van der Werf [a], Akhil Kumar [b], Marc Verdonk [c]

[a] Department of Mathematics and Computer Science, Technische Universiteit Eindhoven, P.O. Box 513, 5600 MB Eindhoven, The Netherlands
[b] Smeal College of Business, Penn State University, University Park, State College, PA 16802, United States
[c] Deloitte, The Netherlands

ARTICLE INFO

Available online 19 August 2010

*Keywords:*
Information assurance
Auditing
Architecture
Conceptual model
Constraints
Business rules
Conformance checking

ABSTRACT

The independent verification of the right applications of business rules in an information system is a task for auditors. The increasing complexity of information systems, and the high risks associated with violations of business rules, have created the need for Online Auditing Tools. In this paper we sketch a conceptual design for such a tool. The components of the tool are described briefly. The focus is on the database and the conformance checker, which are described in detail. The approach is illustrated with an example and some preliminary case studies from industry.



## 1. Introduction

Organizations are constantly executing business processes to achieve their goals [9,25]. These business processes need to be executed within certain *boundaries*. These boundaries are defined by business rules coming from different sources. Some rules are enforced by law and the authorities, and others by the shareholders. But contracts with business partners like customers and suppliers also create boundaries. Moreover, the board of an organization itself defines boundaries, e.g. in a code of conduct. Note that "staying within the boundaries" involves much more beyond avoiding fraud. So we consider fraud as an example of rule violation and therefore we do not treat it separately.

Information systems play a major role in executing the business processes, either in cooperation with employees or autonomously. This results in the need to implement business rules in both the information system and through operating instructions carried out by employees. As information systems become more and more complex, in many situations it gets very hard to manage the whole system. Since the management of an organization is responsible for the execution of the business processes, and accountable for staying within the boundaries, there is a need for checking whether the business rules are being followed on a continuous basis. Management has the prime responsibility to assess the operating effectiveness of "their" business rules, and must monitor the execution of the business processes closely. Independent verification is also needed. This is typically the job for auditors who provide assurance to stakeholders. Auditors can be either internal or external. An internal audit verifies adherence to both the internal and external boundaries (and can focus on both effectiveness and efficiency of the processes and the business rules), whereas an external audit typically only focusses on the adherence to external boundaries and the effectiveness of processes and business rules. Of course, all auditors should be independent in their research approach and in their judgement.

For financial statements a *financial audit* is performed by the CPAs (Certified Public Accountants). They verify if financial statements of organizations are in accordance with external boundaries like the Generally Accepted Accounting Principles (GAAP) and Sarbanes Oxley (SOX) legislation. But business rules concern much more than the financial reporting process and, therefore, there are numerous types of audits, e.g. ISO audits, food safety audits, Basel2 audits, information security audits, and operational audits. One aspect all audits have in common is that they often are very laborious and expensive. Moreover an audit always looks at a period in the past to determine if the business rules were adhered to in the period under review, while the management's main interest lies in the future.

In the ideal situation we would have a *continuous auditing process* that gives us *real time* insights into violations of business rules [13]. Clearly this is not feasible if done manually. Therefore, there is an urgent need for better techniques and software tools that make it possible to check arbitrary business rules automatically and in near real time. One of the approaches used today is to embed *controls* in the information system. A control is an automated task in the information system aimed at the prevention of violations of certain business rules. These controls are strongly related to the functions of the information system. Often business rules are generic, i.e. not bound to a specific business context. An example is the "four-eyes"

* Corresponding author.
*E-mail addresses:* w.m.p.v.d.aalst@tue.nl (W. van der Aalst), k.m.v.hee@tue.nl (K. van Hee), j.m.e.m.v.d.werf@tue.nl (J.M. van der Werf), AkhilKumar@psu.edu (A. Kumar), mverdonk@deloitte.nl (M. Verdonk).

0167-9236/$ – see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.dss.2010.08.014

principle that requires that "two tasks for the same case should be handled by different agents".

It may seem paradoxical that another information system is needed to check the first one. However, that is what we propose since the information systems themselves become too complex and thus require oversight. Our solution is not a type of theorem prover that verifies if the code of the information system correctly implements the business rules. Since people and organizations cannot be formally specified and may deviate at runtime, we envision a separate system that *monitors* the *relevant* activities of the information system and which independently checks if these activities conform to business rules. We call such a system an *Online Auditing Tool* or OLAT for short. Consequently, the information system should be equipped with a *logging* mechanism and the OLAT should be connected to the information system. The envisioned OLAT can work in two modes: it can *report* violations of business rules in the form of a report to the management of the organization, or it can send a *message* to the information system that can be used to exercise a *control*. Thus, the OLAT can also be considered as an external control mechanism for the information system. In the latter mode we have to be careful since it appears as if the OLAT becomes part of the information system, and therefore it could loose its independent status. However, the OLAT tool is only used to *detect* a (potential) violation and this information can be used in the information system to prevent the violation or to enact a compensation action. Although some techniques already exist to automate small parts of the audit process, a system integrating these techniques does not yet exist.

In this paper we sketch a "full blown" OLAT. This paper shows the possibilities and capabilities of such a tool, and gives insights into the architecture and functionality of such a system. Some components do not yet exist, and are ill-defined, or even speculative. Although we call the tool set "online", we do not mean they are all active in a real time mode. Only the conformance checking can be done in real time. However, the structure of the OLAT allows for reporting on a regular basis, thus providing near real time information.

At the outset it should be clarified that, rather than focussing on modeling techniques for data and processes, instead we focus on the architecture of a system that can handle any business process. Considerable previous work has already looked at approaches for modeling business processes. Therefore, we place more emphasis on development of a meta model that encompasses process, business data and organizational aspects along with the runtime issues. In particular, we are interested in modeling those aspects of an organization that are relevant for the business rules to be monitored. However, we need certain techniques or approaches in order to express and check the business rules. Therefore, we present in Section 2 some basic techniques for data modeling, process modeling and the language to express business rules. In Section 3 we define the concepts that are related to auditing in an informal way. In Section 4 we give a high-level architecture of an OLAT, and also describe software components for which we do not have a concrete solution yet. In Section 5 we describe a conceptual data model for the OLAT. In Section 6 we study the business rules in detail. We have chosen to be as language independent as possible. Therefore we use standard predicate calculus to express these rules. Section 7 gives a concrete example to illustrate our approach. In Section 8 we describe practical experience with the business rule evaluation in some real life cases. Section 9 discusses related work and finally, the last section gives a conclusion and our plan for future work.

## 2. Preliminaries

Here we explain techniques for data modeling, process modeling and the predicate language to express rules. Although the focus of this paper is on the architecture and meta modeling of our OLAT tool, we need these techniques to illustrate the use of the tool. As we prefer to be as independent as possible from industry languages, we use plain predicate calculus for the business rules. Note that we do not intend to present a new modeling approach, rather we need a consistent combination of different modeling frameworks. For modeling approaches a large body of literature exists already, cf. [9,2].

### 2.1. Petri nets

For modeling of business processes we use Petri nets [2]. A Petri net consists of *transitions* (drawn as squares) which represent *tasks* that can be performed in a *process*, and *places* (drawn as circles), which define the *conditions* for a transition to be executed (or "fired"). Places and transitions are connected by arcs. Places that have an arc to (from) a transition $t$ are called the *input (output) places* of $t$. The state of a Petri net, also called a *marking*, is a distribution of objects, called *tokens*, over the places. A transition is *enabled* if in each of its input places there is at least one token. In that case it can *fire* which means that it consumes a token from each of its input places and produces a token in each of its output places. The behavior of a Petri net is characterized by the set (in fact graph) of markings that are reachable by transitions from an initial marking.

### 2.2. Data models

A database consists of *entities*, i.e., the elements or records stored in tables. All entities together form an instance of the database. The fields of a table are called the *attributes* of the entity, and are related by *associations*. On the schema level, entities belong to an *entity type*, associations belong to a *relationship*. The entity type also defines the type of attributes of an entity. An Entity-Relationship diagram (ERD) [14] describes the type of various entities and the relationships between them. Entity types are drawn as rectangles. Inside the rectangle, the entity type is given, together with its attributes. Relationships between pairs of entities are drawn by arcs connecting them, with a diamond in the middle. We consider only binary relationships and most of them are functional relations.

A *functional relationship* can be represented as a function from one entity type to another. For functional relationships we drop the diamond, and represent them directly by an arc from the source entity type to the target entity type. We also distinguish between a *total function* and a *partial function*. A total function is one in which every element of a domain has a mapping, while in a partial function some elements are not mapped. For notation purposes, if at the arrow head a vertical bar is drawn, the function is total, i.e. for all instances in the entity, the function returns an instance of the associated entity. Otherwise, it is a partial function. A *non-functional* relation is a many-to-many relation, or a set-valued function. For non-functional relationships we use the standard diamond notation where the arrow indicates the direction of the relation. A relationship is uniquely identified by the source entity type and the name of the relationship. That is, names for the relationships are only unique for the source entity; from the context it is always clear which relationship is intended.

Consider the data model of Fig. 1. Here, there are three entity types: 'Task' with attribute 'name', 'Process' and 'Transition'; two

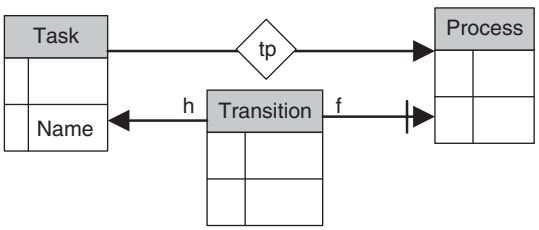



**Fig. 1.** A simple data model.

functional relationships: 'h' and 'f'; and a non-functional relationship 'tp'. The arrow on relation 'tp' indicates that $tp \subseteq Task \times Process$, i.e., 'tp' is a many-to-many relation between 'Task' and 'Process'. The functional relation 'h' is partial, and it points from 'Transition' to 'Task', indicating that each entity of 'Transition' is connected to at most one entity of 'Task'. The functional relation 'f' is total, i.e., every 'Transition' is connected to one 'Process'.

If in a database instance two entities $x,y$ are in a relationship $r$, we write $(x,y) \in r$. If $r$ is a functional relationship from x to y, we write $r(x) = y$ for $(x,y) \in r$. For example, we can formalize the constraint that states that if two entities of type 'Transition' have an association to the same entity of type 'Task' and to the same entity of type 'Process', these entities are identical to the following formula:

$$\forall t_1, t_2 \in Transition : (h(t_1) = h(t_2) \wedge f(t_1) = f(t_2)) \Rightarrow t_1 = t_2.$$

Note that $h(t_1)$ indicates the unique entity that is related to $t_1$ by the functional relation $h$. So $h(t_1)$ is a term that can be used in a comparison operation, as in $h(t_1) = h(t_2)$ above.

Most predicates can be translated into standard SQL (cf. [29]), all predicates can be checked using SQL augmented with stored procedures. In this case, the query would be: `SELECT * FROM Transition t1, Transition t2 WHERE t1.Task = t2.Task AND t1.Process = t2.Process AND t1.Id <> t2.Id`. If the result of this query is empty, then the constraint holds. This provides a practical approach to implement a conformance checker for business rules: translate the rules into SQL queries and if they evaluate to the empty set, the rule holds.

## 3. Concepts

An auditor will look for *assurance* that the business processes have performed within the boundaries determined by the business rules, by either auditing the *design*, i.e., the implementation and effectiveness of *controls*, or by looking *substantively* at the data generated by the system. The last approach is considered as very costly if done in the traditional way. We propose a new and efficient way for substantive data checking. Our proposal is to do it for specific business rules only, in an automated way and in near real time.

Since the audit applies to a business process, we briefly review the basic process terminology (cf. [2]). A *business process* is a collection of tasks with (potentially complex) coordination requirements among them. A *task* represents a set of activities in the real world that is considered as one atomic action performed by an *agent*, or it is automated. A task is uniquely associated to a *business form* which is a collection of entities.[1] An instance of a business process is called a *case*. A case has its own *case data* associated with it and is stored in a database. When a task is executed for a specific case, its case data is shown. In the business process a task can be any kind of activity, however in the information system the *execution* of a task is limited to reading, writing or updating these entities. As a task finishes, the coordination requirements determine the set of tasks that can be executed. Eventually, when no tasks are executable for a case, the case is closed. The modeling of business processes as Petri nets is very well understood and supported by tools (cf. [2]). Remember that we only model those aspects of business processes that are relevant for the business rules we are interested in.

Agents usually work in a certain role. A *role* is a generic identifier for a category of agents in an organization, e.g., a manager, director, vice-president, etc. are all generic roles. Thus, agents Joe and Mike might be managers, Sue a vice-president, and so on. We further assume that roles are organized in a hierarchy (i.e. a tree) in which the CEO is the top node, and each link between nodes represents a boss-employee relation. In general, every organization has a different hierarchy. There are different ways in which agents can be assigned to roles, but for now we will assume that an initial assignment of agents to roles is given. An agent $a_1$ can grant a permission to agent $a_2$ to perform (a) a specific task, (b) all tasks belonging to a process, (c) all tasks belong to a case, or (d) a specific task belonging to a specific case. The agent is only allowed to grant a permission if it has the permission itself, either by its role, or through a permission obtained from another agent.

Certain tasks are used to detect or prevent violations of business rules. These tasks are called *controls*. There are different types of controls and many different ways of classifying them. For the purpose of this research we will classify them in the way they are used to respond to an exception that occurs on a business rule.

1. *Detective*: this type of control is only able to detect that a violation to the business rule has occurred. An example: *an employee has just transferred $ 1 million to his account.*
2. *Corrective*: this type of control is like the detective control, but has the added functionality to (or attempt to) correct the violation to the business rule directly. An example: *an employee has just transferred $ 1 million to his account and the control is preparing to transfer it back.*
3. *Preventive*: this type of control prevents business rules from being violated. An example: *In the current payment run an amount of $ 1 million is going to be transferred to the bank account of an employee, but the payment run will not be processed for this reason.* A special case of preventive controls is a *prospective* control, which gives a warning if it is possible to break a business rule based on other actions performed.

We consider only two kinds of events: a *task* event and a *permission* event. The first is the execution of a task, the second is the granting of a permission.

## 4. Top-level architecture

In Fig. 2 the top-level architecture of the OLAT is presented. We distinguish data sets (displayed by drums) and program modules (displayed by rectangles). Some of these modules already have an implementation in tools like ProM [6]. However, to our knowledge, no tool exists that integrates these techniques into a single information system for auditing purposes. Note that we have more modules in the architecture than we actually will describe in detail. In this paper we only focus on the conformance checker and the risk interpreter. For the other modules we only give a high-level specification.

The data sets form the database of the OLAT. The conceptual model of the database is presented in the next section. The database consists of three types of data: *Run time* data, de jure models, and de facto models. The run time data is collected from the monitored information system. The de jure models are the official models of the desired organization. In fact, the run time data should *conform* to the de jure models. Otherwise, it indicates a violation. The de facto models are derived from the run time data by discovery techniques and can differ from the de jure models.

### 4.1. Run time data

This data comes from the information system. All (relevant) events of the information system are recorded in the system log. So the run time data concerns the events in the business processes such as activities or tasks, and events in the authorization processes such as the granting of permissions. This data is needed to perform analysis by the conformance checker, difference analyzer and potential risk detector.

[1] Note that we only use the term "form" as a metaphor, we do not assume a particular form-based implementation.

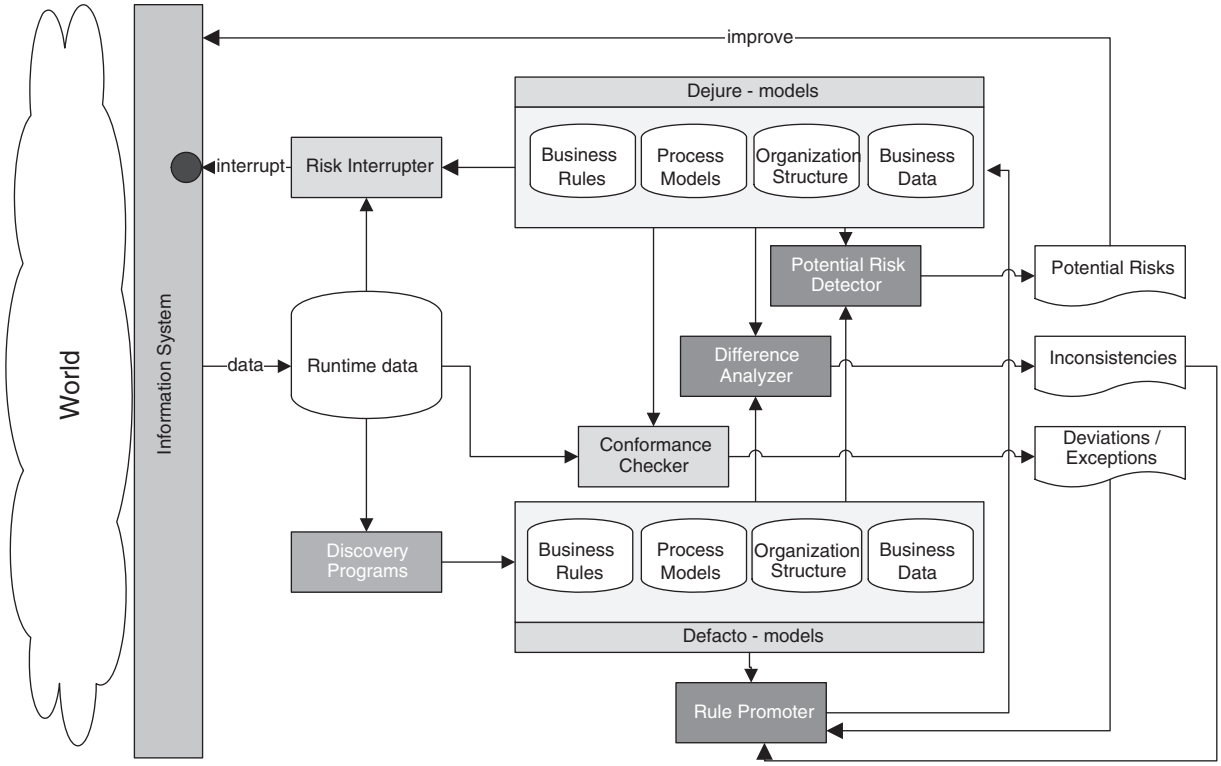



**Fig. 2.** A top-level architecture of an Online Auditing Tool.

### 4.2. De jure and de facto models

The de jure models describe the desired or official situation, whereas the de facto models are derived from the run time data, and thus describe the actually observed situation and behavior. The de jure models are made for the design of the information system. Both the de jure models and de facto models concern *process models* with tasks and their ordering, *business data* together with the forms data of the tasks, and the *organizational data* with the agents and their roles. Last, but not least, business rules are also a part of de jure and de facto models. *Business rules* are expressed in standard predicate logic. Except for the business rules, these other data sets are collected in one (relational) database as the de jure models describe the desired or official behavior, business rules in the de jure models should not be violated. On the other hand, business rules in the de facto models are discovered as will be discussed later. The de jure models are loaded from the information system, while the de facto models are obtained by discovery techniques. Hence, they may be less complete than the de jure models. The de facto and de jure models share the same database schema as presented in the next section (see Fig. 3).

### 4.3. Conformance checker

This module checks whether the run time data conforms to the de jure models, in particular the de jure business rules. This does not only include the control flow behavior, but also data flow, authorizations and business rules. Since the business rules are expressed in predicate logic, they can be translated into queries (cf. [29]). The queries run on the database (i.e. the de jure models plus the run time data). If the result of the query is the empty set, the rule is not violated. If a rule is violated, an exception report is generated based on the returned query containing the counter examples. This exception report needs to be analyzed by management and auditors, and can lead to either a remedial action, or to the conclusion that the situation should be allowed. In the latter case, the Rule Promoter can be used to add the newly discovered model to the de jure models. This conformance check, i.e. by executing the translated queries, can be run at any time, thus providing a way to continuously audit the system.

### 4.4. Discovery programs

In contrast to the conformance checker, discovery programs try to *derive* models out of the run time data. Many kinds of existing *data mining* and *process mining* techniques and tools can be used to discover not only the control flow, but also the authorization rules, business data models, organizational models and business rules [3,30,31]. In general, mining techniques try to deduce patterns and rules from facts. In our case the facts are stored as events in the run time data set. To discover a process model we look at the actual execution order of tasks for the cases, and from this we can infer a process structure (for example a Petri net) (cf. [5]). For the structure of business data we could look at data as business forms that are used in the events to derive entities and relationships. For organizational models, we can look at the permission events whereby an agent grants a permission to another agent. As these rules are derived from the run time data, the models obtained by discovery are de facto models. While detailed discussion of these techniques is beyond the scope of the current work, the kind of tools we have in mind is included in the well-known process mining toolset ProM [6].

### 4.5. Rule Promoter

This module represents functionality to convert a discovered de facto model into a de jure model, and in particular it concerns business rules. For this, it needs to be able to abstract from the specific instance information. In the first run, the module is used to tune the configuration of the de jure models to the actual situation. Later on it may be part of a continuous improvement process; e.g. when exceptions are discovered, analyzed and accepted, they are added to

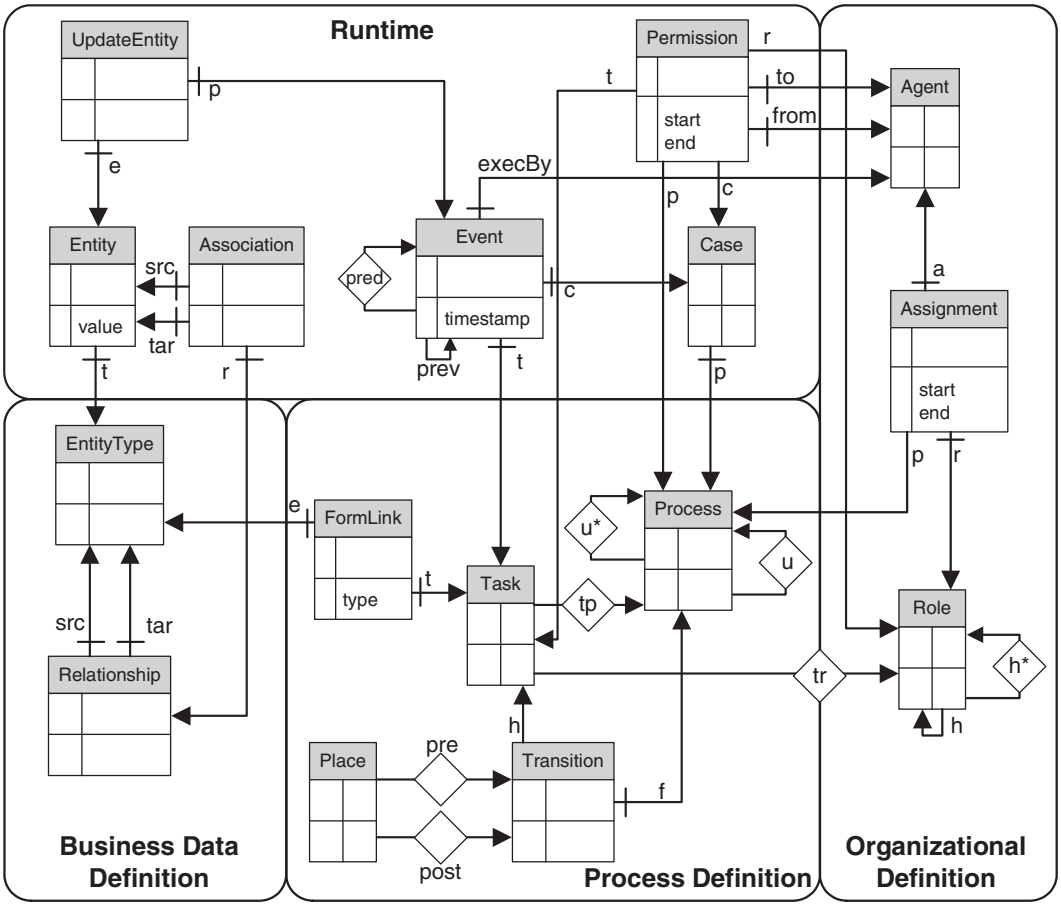



**Fig. 3.** Conceptual model for OLAT.

the de jure models, thus eliminating 'false positives' in the conformance checker.

To the best of our knowledge, there are no methods or software for this task available today. Therefore, we assume this to be a human task.

### 4.6. Risk Interrupter

The Risk Interrupter takes input from the de jure models and the run time data in a way similar to the conformance checker. The difference is that this module interrupts the information system to prevent further processing of the case under consideration until issues are resolved and the risk is mitigated. Hence, it serves as an external guard for tasks in an information system. In fact, it can be seen as an external control based on the conformance checker.

### 4.7. Difference analyzer

The difference analyzer compares the de jure and de facto models. It also checks whether business rules, process models and organization structure are in conflict between the de jure and de facto models. This can be seen as a quality check for the models, and therefore a check for the functioning of the whole concept. Prototypes of such a tool have been designed [17].

### 4.8. Potential risk detector

This module is able to detect *potential* risks by analyzing the run time data, the de facto and the de jure models. For instance, if the de jure and de facto models differ, we could use it to see if a violation of a de jure business rule could occur. This information is considered as a warning. In the ProM toolset [6] several tools are available that could be used to realize this module.

### 4.9. Remarks on the implementation of the OLAT

We do not consider the implementation of the OLAT in detail in this paper. However we note that the heart of the OLAT is the database that contains all data. The conformance checker as well as the Risk Interrupter, can be based on a standard SQL engine. So the part of the system we focus on, can be realized using a standard database management system. Of course, the OLAT needs coupling with the information system to collect events from it, and also perhaps to send interrupts to it. It also needs a reporting facility. Since we aim at a generic OLAT we should be able to configure the OLAT for specific information systems, but this involves the construction of a standard data-intensive application. For the other modules, like the discovery programs, we might use existing tools that can query the database. Hence, the implementation is a serious engineering effort, but it does not require new scientific insights.

## 5. Conceptual model

The heart of the OLAT is the database. This section describes a conceptual model for all the data sets needed for the OLAT. The conceptual model shown in Fig. 3 is actually a meta model in that it integrates the high-level modeling elements of the organization definition, business data definition and process definition, along with a run time framework. These aspects are described at length next. In addition, there are *consistency* constraints that should hold for any organization. These constraints are explained in detail in Section 5.2. Note that if these constraints are violated, the database becomes

inconsistent, which is not the same as a violation of a business rule. Conformance of business rules is then treated in Section 6.

### 5.1. Data model

Fig. 3 depicts the conceptual model. It is arranged into four components: the process definition, the business data definition, the organizational definition and run time. We first explain the conceptual model, and then show how, using predicate logic, all kinds of business rules can be formulated on this model. Remember that we do *not* distinguish between the de jure and de facto models here: they share the same data model. Also, note that we sometimes introduce transitive closures of relations (i.e., $h^*$, $u^*$ and *pred* ). These transitive closures are assumed to be updated explicitly in the database, which is easy to perform. We use them to avoid recursive definitions in constraints (i.e. queries), thus allowing us to implement the conformance checker with an 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.

#### 5.1.2. Process definition

The process definition component describes the processes monitored in OLAT. Note that we store the process models in the form of a data model. A *process* contains *tasks* that can be executed for that process. Processes are often hierarchical. Parts of the process are either reusable, or are refined using subprocesses. In our conceptual model, this is modeled by relation $u$. If two processes $x$ and $y$ are related via $u$, then process $x$ *uses* process $y$, i.e. $y$ is a subprocess of $x$. To avoid recursion and to be able to use queries, we also store the irreflexive transitive closure of $u$ in a relation named $u^*$. Tasks can be shared by different processes. As stated earlier, a task is identified with a form providing the necessary data to execute that task.

A task typically reads and writes entities. The entity *FormLink* models the relation of the entity types that are used in a form to perform a task. Its attribute *type* defines whether the entity is read, written or both. To express conditions on the order in which tasks occur, we use labeled Petri nets. A *transition* is labeled with the process (relation $f$) and the task (relation $h$) it represents. The conceptual model allows that transitions which are connected via a place, do not need to be in the same (sub) process. However, we assume that all places connected to a transition belong to the same process. In this way, places can be shared by two or more processes, thus providing the possibility to define process composition, rather than only flat processes. The initial tokens of a place are an attribute of the place. It should also be noted that although we use labeled Petri nets, any other process notation could be used to define the order in which tasks can occur. Moreover, we have no direct run time information of the firing of transitions or the marking of a place. However it is possible to derive this information if $h$ is a bijection (see e.g. [30,31]).

#### 5.1.3. Organizational definition

Tasks can be executed by different *role*s that are placed in a hierarchy. If a role is *higher* in the hierarchy, it means that this role can execute all the tasks of its subordinates. The hierarchy is expressed using relation $h$: if $a$ and $b$ are related by $h$ (i.e. $(a,b)\in h$ in the instance) then $b$ is the supervisor of $a$. Again, we add the transitive and reflexive closure of the hierarchy relation, $h^*$.

*Agents* are assigned to roles via an *Assignment*. This assignment can be for all processes or for a single process, which is depicted by the optional relation $p$. The entity *Assignment* has start and end attributes to indicate the interval in which this assignment holds.

#### 5.1.4. Run time

The run time component stores all *events* and associated data from the information system. There are two types of events: events that indicate that something has been done for a specific task (the entity *Event* in the model) and the *granting* of permissions by agents (the entity *Permission*). The data associated with an event is business data, i.e. the content of the forms filled in. The entities *Entity* and *Association* store the business data definition. Each *Entity* belongs to an *entity type*. An *Association* associates two entities and belongs to some *Relationship*.

A *Case* is an instance of a process, and it proceeds through *Event*s that are raised whenever a task is executed. An event is always executed by some agent for a task in a process. The event occurrences form a partial order represented by the relation *prev*. The relation *pred* is the transitive closure of relation *prev*, and is used for formulating business rules. Typically, an event for a task in a case also involves entities in the business data which are created or updated. This information is stored in *UpdateEntity*. *Entity* contains the latest version of the entity, *UpdateEntity* stores the changes.

If an agent $A$ authorized another agent $B$ to perform a part of its work, then agent $B$ acquires a *Permission* from agent $A$ to perform some work. A permission is always for a time interval and it can apply to a role, a process, a case, a task, or any combination of thereof. By obtaining a role permission, agent $B$ can perform all tasks of that role, given that $A$ has that role in the first place. A permission can also apply to a specific process or case, indicating that agent $B$ can do anything $A$ can do for that process or case. If the permission is for a task, agent $B$ can execute that task as well. A permission is only allowed if agent $A$ has the proper permissions for the work it delegates. Note that one cannot always detect in which role an agent executes a task, only whether it has the right authorization.

### 5.2. Constraints on the data model

There are two types of constraints that can be defined on the process model: *logical consistency constraints* which do not depend on any business context, i.e. constraints to maintain the consistency of the data model, and *conformance constraints* which ensure the conformance of the data model within the business context. There is a simple distinction between the two: Consistency constraints do not use any specific attribute value, while business rules do. The latter are described in the next section. For the business data, there are no separate constraints, as it is a general schema for an ERD. In the remainder of this section we explain some of the most important consistency constraints. We classify the constraints according to the component of the entity types they address.

#### 5.2.1. Consistency constraints for the process definition

The conceptual model allows for subprocesses. Although a process can be nested arbitrarily deep, cycles in the process hierarchy are not allowed. This can be expressed using two constraints. First, the relation $u$ should be irreflexive, i.e. a process should not depend on itself. Secondly, as $u^*$ is the transitive closure of $u$, and we disallow cyclic references, $u^*$ needs to be irreflexive as well. For the purpose of discovery algorithms, we require that the task and process uniquely identifies a transition. This gives rise to the following constraints:

p1: Relation $u^*$ is the transitive closure of relation $u$.
p2: Relations $u$ and $u^*$ are irreflexive.
p3: If a transition belongs to a certain process and represents a task, the task should also belong to that process.

p4: The combination of a task and a process uniquely identifies a transition.

For example, p4 is equivalent to stating: if, for two transitions $t_1$ and $t_2$, their related task and process are the same, the transitions are the same. Formally:

$$\forall t_1, t_2 \in Transition : (h(t_1) = h(t_2) \wedge f(t_1) = f(t_2)) \Rightarrow t_1 = t_2.$$

*5.2.2. Consistency constraints for the organizational definition*

Consistency constraints for the organizational definition are related to the definition of the role hierarchy and the granting of permissions. A permission may be granted to an agent to act in a certain role, to perform a task, or to be involved in a process or case, or any combination thereof. An agent is only allowed to give a permission to another agent for a role if that agent has the proper authorization. The agent has this authorization if either it is allowed to assume that role, or it possesses the permission explicitly. This leads to the following (non-exhaustive) set of constraints.

O1: Relation $h^*$ is the reflexive transitive closure of relation $h$.
O2: The start time of an assignment is strictly smaller than its end time.
O3: The start time of a permission is strictly smaller than its end time.
O4: An agent can only grant a permission for a role, if it is assigned to that role, or if it has a permission for that role itself.
O5: An agent can only raise an event for a task in a case, if it has a role assignment to execute that task, or it has a permission to execute it.

*5.2.3. Consistency constraints for the run time*

The main consistency constraints for the run time are concerned with the correctness of events: the events should happen in the right order, i.e. the timestamp of events in the relation *prev* should conform to the ordering. Also, the storage of business data should be according to the schema. This leads to the following set of constraints.

r1: The relation *pred* is the transitive closure of relation *prev*.
r2: If event $y$ occurs after event $x$, then the timestamp of $x$ should be at most the time stamp of $y$.
r3: The source and target entities an association relates to, should be of the correct type specified by the relationship the association belongs to.
r4: If an event in a case occurs, the task related to the event should be in the process of which the case is an instance.
r5: If an entity is updated by an event, it should be of an entity type that is in the form of the task the event is of.
r6: If a permission is both for a process and a case, the process of the case should be the same process as the permission is for.
r7: If an agent performs a task, and it is authorized by an assignment, this assignment is unique.

## 6. Business rules

In this section we present business rules. Since it is in principle impossible to list all possible business rules, we only consider some characteristic examples that occur frequently. Remember that a business rule is a constraint on the data model involving business data as *parameters*. Therefore, we are able to express business rules as parameterized constraints. Further, note that we can check them by query processing. So the implementation of the conformance checker could be based on a standard database engine. It is not only possible to express business rules for a single process or case, but it is also possible to express business rules involving several processes or cases.

In general, business rules concern the following aspects:

- *ordering based*, i.e. about the execution order of tasks in cases;
- *agent based*, i.e. about the involvement of a role or agent in cases and processes;
- *value based*, i.e. in forms belonging to a task.

In business rules these aspects may be combined. In this section, we show examples for each of the aspects. In some examples we need the set of attributes $\Lambda$ and the set of values $V$. We use the notation $e.a = v$ to express that attribute $a$ of entity $e$ has value $v$.

*6.1. Examples of ordering based rules*

Ordering based rules express constraints concerning the ordering of events and tasks in processes. Below we refer to the conceptual model of Fig. 3 for the function names which denote the relationships between various entities.

O1: Task precedence. A task $t_2$ should always be performed before task $t_1$ in any case of process $u$.

$$b1 : TaskAlwaysBeforeTask(u : Process, t_1, t_2 : Task) := \\ : \forall x_1 \in Event : (p(c(x_1)) = u \wedge t(x_1) = t_1) \Rightarrow \\ \exists x_2 \in Event : t(x_2) = t_2 \wedge c(x_1) = c(x_2) \wedge (x_2, x_1) \in pred$$

O2: Restrict update operation. After task $u$ is performed in a case, no entity of type $x$ can be updated anymore in that case. For example, an employee cannot change the travel expense form (or entity) after it has been approved.

$$b2 : RestrictUpdate(u : Task, x : EntityType) := \\ \forall e_1, e_2 \in Event : c(e_1) = c(e_2) \wedge t(e_1) = u \wedge (e_1, e_2) \\ \in pred \wedge \neg(\exists y \in UpdateEntity : p(y) = e_2 \wedge t(e(y)) = x)$$

O3: Maximum repetitions of a task in a case. In any case of process $P$ task $u$ cannot be executed more than $n$ times.

$$b3 : LimitNrOfTasks(u : Process, z : Task) := \\ \forall w \in Case : p(w) = u \Longrightarrow |\{x \in Event | c(x) = w \wedge t(x) = z\}| \leq n$$

*6.2. Examples of agent based rules*

Role or agent based business rules express constraints about the involvement of roles and agents in processes.

A1: 4-eyes principle. Two tasks $t_1$ and $t_2$ in the same case should always be executed by different agents. Below *execBy* is a function that returns who performed an event.

$$b4 : 4EyesPrinciple(t_1, t_2 : Task) := \\ \forall x, y \in Event : (c(x) = c(y) \wedge t(x) = t_1 \wedge t(y) = t_2) \Rightarrow \\ execBy(x) \neq execBy(y)$$

A2: Mutually exclusive agents. Two agents $a_1$ and $a_2$ should never appear together in a case.

$$b5 : MutualExclusiveAgents(a_1, a_2 : Agent) := \\ \neg\exists u_1, u_2 \in Event : u_1 \neq u_2 \wedge c(u_1) = c(u_2) \wedge execBy(u_1) = a_1 \wedge \\ execBy(u_2) = a_2$$

A3: Maximum tasks by an agent. An agent $a$ cannot do more than $n$ tasks in any case of process $u$.

$$b6 : TaskLimitOnAgent(u : Process, a : Agent, n : Nat) := \\ \forall w \in Case : (p(w) = u) \Rightarrow \\ |\{x \in Event | c(x) = w \wedge execBy(x) = a\}| \leq n$$

A4: Forbid agent to write. An agent $a_1$ is not allowed to update any entity in a process $u$.

$$b7 : ForbiddenToWrite(a : Agent, u : Process) := \forall x \in Event : (execBy(x) = a \wedge p(c(x)) = u) \Rightarrow \neg(\exists y \in UpdateEntity : p(y) = x)$$

### 6.3. Examples of value based business rules

Value based business rules concern the values of business data. Typically, these constraints can have the following form:

- two values should be equal,
- one value should be larger than another value, or
- a value should be within some given set (i.e. within some limits).

V1: Restrict entity-attribute-value for an agent. An agent $a$ is not allowed to write an entity of type $b$ with value of attribute $x$ larger than $n$.

$$b8 : LimitEntAgent(a : Agent, b : EntityType, x : \Lambda, n : V) := \forall z \in Event, y \in UpdateEntity : (p(y) = z \wedge t(e(y)) = b \wedge execBy(z) = a) \Rightarrow e(y).x \leq n$$

V2: Restrict entity-attribute-value for a case. For each entity of type $b$ written in case $w$, the value of attribute $x$ is lower than $n$.

$$b9 : LimitEntInCase(w : Case, b : EntityType, x : \Lambda, n : V) := \forall y \in Event, z \in UpdateEntity : c(y) = w \wedge t(e(z)) = b \wedge p(z) = y \wedge e(z).x < n$$

V3: Agent approval limit. An agent $a$ can only perform task $u$ for a case, if for each entity of type $b$ written in that cas, attribute $x$ is less than $n$. E.g., a bank vice-president can approve a loan up to a limit of $500,000.

$$b10 : ApprLim(a : Agent, u : Task, b : EntityType, x : \Lambda, n : V) := \forall y \in Event : (execBy(y) = a \wedge t(y) = u) \Rightarrow LimitEntinCase(c(y), b, x, n)$$

Note that LimitEntinCase is defined above in rule b9.

V4: Three-way match. In each case of a process $n$, if task $u$ is executed, then entities of types $a$, $b$ and $c$ belonging to the case should have the same value. E.g., the price of the invoice should match the price on the quotation and on the delivery notice.

$$b11 : ThreeWayMatch(n : Process, u : Task, a, b, c : EntityType) := \forall w \in Case, v \in Event, x, y, z \in UpdateEntity : (c(w) = n \wedge t(v) = u \wedge c(p(x)) = c(p(y)) = c(p(z)) = c(v) = w \wedge t(e(x)) = a \wedge t(e(y) = b \wedge t(e(z)) = c) \Rightarrow e(x).value = e(y).value = e(z).value)$$

## 7. Example

Above we showed how parameterized business rules are expressed in predicate logic, and checked by further transforming predicate logic into queries and running them against a database. In this way, end users and process owners are not confronted with details of predicate logic.

As an example to illustrate the framework, Fig. 4 shows the Petri net for an *Administer Account Transfer* process. The process starts with a *customer representative* receiving an account transfer instruction (task $t1$) from a client, who records the transfer instruction (task $t2$). Next, a *financial clerk* validates the instructions (task $t3$). If the validation reveals a problem, communication details of the invalid instruction are extracted (task $t5$). Otherwise, a *financial accountant* checks the transaction limit of the transaction (task $t4$). If the transaction amount exceeds the limit for the customer, the process starts the *Authorization* subprocess consisting of tasks $t7$, $t8$a and $t8$b. If the limit is not reached, or the transaction is authorized, the *banking specialist* checks the available funds. If this check fails, communication details are derived from the account unit (task $t9$); if it passes, the *Accounting Entry* sub process is started, which applies the accounting entry and calculates a fee for it. In allcases, the results are collected in a report (task $t15$), and after it is approved (task $t16$), the customer is notified (task $t18$). If the report is not approved, it is reworked (task $t17$), and tasks $t15$ and $t16$ are repeated.

This process also involves the role of the *senior financial manager*, who supervises the *financial manager*, and heads a team also including a financial accountant and a financial clerk. Table 1 shows the assignment of roles to tasks. Note that based on the role hierarchy, the senior financial manager inherits the permission to do everything her subordinate can do.

The organization has the following agents: agent-joe, agent-sue, agent-eric and agent-beth. These agents fulfill the roles within the organization. In this organization, we next define the business rules that must hold for the process. First, it is not allowed to update the entity *cust-account* after task $t11$ has been executed. Secondly, agent-joe and agent-sue are not allowed to work together in any case. Agent-eric is not allowed to execute more than 4 tasks. Last, tasks $t7$ and $t8$a in a case may not be executed by the same agents, and this also applies to tasks $t7$ and $t8$b, and for tasks $t10$a and $t10$b. To set up the conformance checker of OLAT, we need to implement these business rules in the system. Given the set of predefined business rules in the previous section, the process owner only has to specify the following functions:

e1: *RestrictUpdate*($t11$,cust-account)
e2: *MutualExclusiveAgents*(agent-joe,agent-sue)
e3: *TaskLimitOnAgent*(agent-eric,4)
e4: *4EyesPrinciple*($t7$,$t8$a)
e5: *4EyesPrinciple*($t7$,$t8$b)
e3: *4EyesPrinciple*($t10$a,$t10$b).

Most of these rules apply to all processes in the system; however, it is also possible to associate a process parameter with a rule in order to apply it to a specific process or subprocess.

## 8. Practical experience with business rules

As computing prices fall and *data analytics* becomes more affordable, there are more applications of it in auditing. The Big Four audit firms are all venturing into this space and embedding their principles into the audit approach. In recent years we have seen a shift from introducing more *controls* in the information system towards *substantive* data analytics and validation of business rules. The main benefit of this type of audit is that there is a shift from identifying the *risk* from violation of a business rule towards *detection* of the violation. In practice we still see a combination of both: a control is tested; if it fails, then the whole population of data has to be validated against the business rules. While we have not yet developed a full blown OLAT, Deloitte Netherlands used a preliminary version of it in off-line mode for the validation of several business rules on large log files from real information systems. We mention one example in each of the rule classes we have identified. In all cases, we could feed the log to the application and execute the queries to check the business rules in a small amount of time, thus providing an efficient approach to audit a complete business process. The outcome of these cases shows that it is feasible to check compliance on a regular basis without much effort.

### 8.1. Ordering based rule

A utility company introduced the rule that invoices could only be paid if there was a valid purchase order present in the system. This rule was applicable for 3 months and was configured in their system as an automated control, which we verified to work correctly.

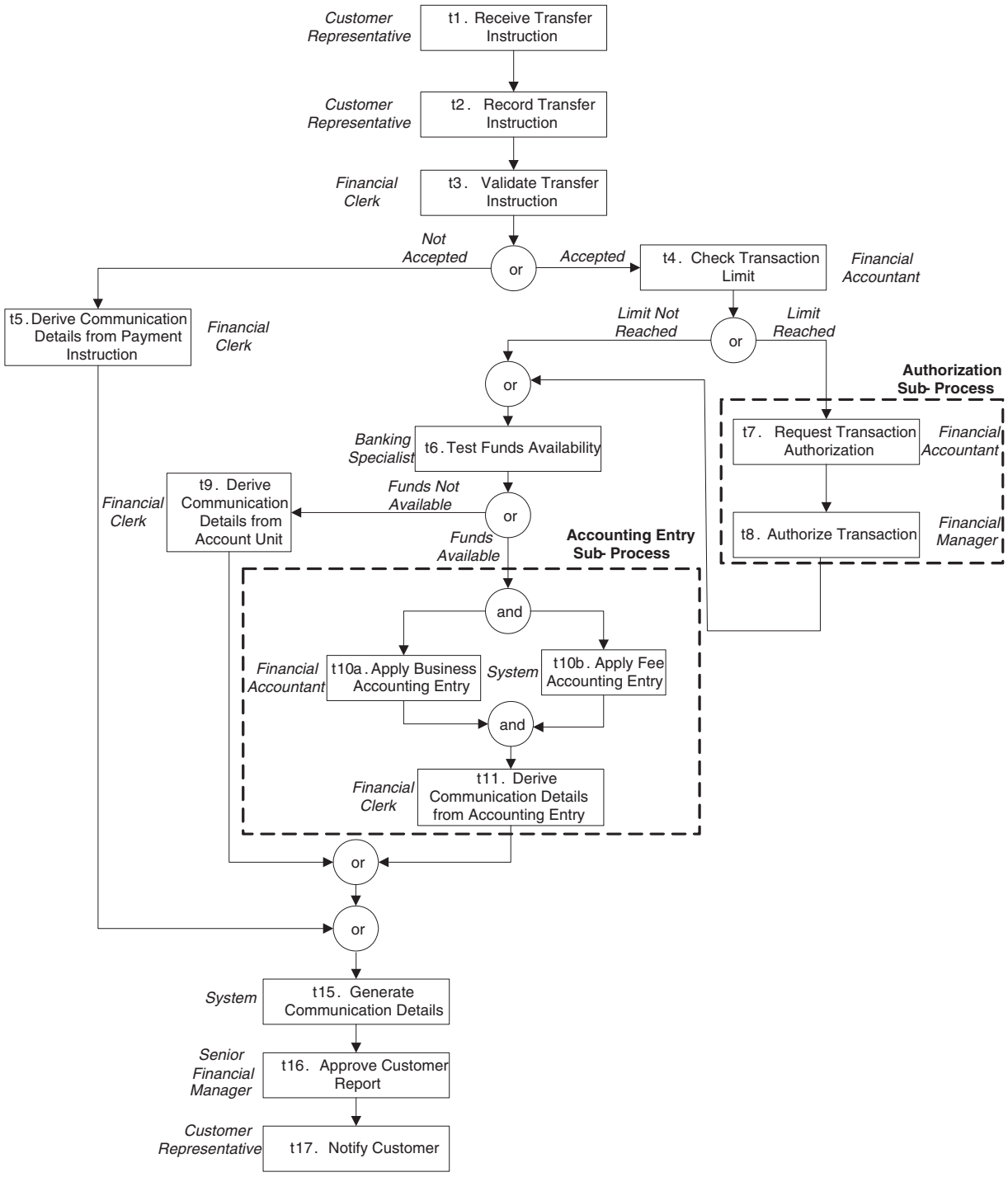



**Fig. 4.** Example of an account transfer process.

However in the process an invoice was registered in the system just before it was paid and the essence of the rule was that the company wanted to prevent placing orders that were not approved through the formal process. Therefore it was decided to run the *task precedence* business rule O1 in Section 6.1 ("Task $t1$ always precedes task $t2$"), with $t1$ = "PO approval" and $t2$ = "Invoice registration", against the complete population of invoices of these 6 months. We found that in the first 3 months, a significant number of invoices were paid without a PO approval being present at all. In the last 3 months we noted that for all invoices paid a PO had been approved, but that this approval in a significant number of cases occurred after registration of the invoice.

### 8.2. Agent based rule

At a large consumer products company we found that authorizations in their SAP system allowed for booking and approval of purchase orders across business units. This was against company policy and also posed a risk for the reliability of their financial statements. Using an extension of the business rule A4 in Section 6.2, "Forbid agent to write", to distinguish between processes in business units, we found that in the total population of 1892 purchase orders there were 140 agents involved in 5 business units. The business rule held for all but one agent that was involved in a process across two

**Table 1**
Task-role matrix.

| task | Roles | | | | | |
|---|---|---|---|---|---|---|
| | Customer representative | Banking specialist | Senior financial manager | Financial manager | Financial accountant | Financial clerk |
| Task *t*1 | ✓ | | | | | |
| Task *t*2 | ✓ | | | | | |
| Task *t*3 | | | | | | ✓ |
| Task *t*4 | | | | | ✓ | |
| Task *t*5 | | | | | | ✓ |
| Task *t*6 | | ✓ | | | | |
| Task *t*7 | | | | | ✓ | |
| Task *t*8a | | | | ✓ | | |
| Task *t*8b | | | | ✓ | | |
| Task *t*9 | | | | | | ✓ |
| Task *t*10a | | | | | ✓ | |
| Task *t*10b | | | | | | |
| Task *t*11 | | | | | | |
| Task *t*15 | | | | | | ✓ |
| Task *t*16 | | | ✓ | | | |
| Task *t*17 | | | | | | |
| Task *t*18 | ✓ | | | | | |

business units. Further inquiry about this exception with the agent confirmed that our assessment was correct, but that there was a plausible explanation for this fact.

### 8.3. Value based rule

At a chemical company we found that the invoice verification option in SAP (which implements the 3-way match) was set to optional. A quick sample drawn on the population showed that indeed the option had been disabled for certain purchase orders that were in the selected sample. Overruling this option poses the risk that invoice amounts, goods received and goods ordered are not in accordance, but the actual impact of this risk is hard to quantify. We used the business rule "3-way match" to verify the whole population of purchase orders based on the amount and monetary value. In this way, we were able to assess the invoices that did not pass the 3-way match criteria. These invoices were followed up, some corrections were made and credit notes requested from suppliers.

## 9. Related literature

Most business process modeling tools do not provide adequate support for information assurance and this is often added in a piecemeal and rather ad hoc manner. To the best of our knowledge there are few efforts to develop a comprehensive architecture and conceptual model for online auditing, which is an important part of our contribution. A promising AI-based approach for detecting procurement fraud is presented in [13]. A workflow is described in terms of pre- and post-conditions that must be satisfied, and a violation of post conditions raises a flag for further investigation. While the basic objectives are similar, our goals are more ambitious since the OLAT architecture also includes organizational and data models, as well as a more extensive discovery and corrective capability.

While there are few OLAT-like holistic architectures, there has been significant research interest focussed on various vocabularies and logic-based methods for expressing business rules in the modeling of processes. Since the mid-nineties several groups have been working on techniques for process mining, i.e., discovering process models based on observed events. In [3] an overview is given of the early work in this domain. The idea to apply process mining in the context of workflow management systems was introduced in [7]. The Alpha algorithm was the first technique able to discover concurrency [4]. Process mining is not limited to discovery. For example, in the context of ProM [6] several approaches to conformance checking were realized. The best developed technique is the Petri net-based conformance checking technique by Rozinat et al. [31]. Here an event log and a process model are compared and deviations are measured and highlighted in both the model and log. Metrics such as fitness, appropriateness, etc. quantify conformance and the diagnostics allow for drilling down the problem.

Conformance checking is related to checking fitness, and measuring the quality of a process mining technique. In [20] negative events are inserted to turn process mining into a classification problem, thus addressing problems related to appropriateness [31]. Also related is the work by Cook [16], where the event streams of a process model and a log are compared based on string distance metrics. Recently, several process mining techniques have been adapted to provide operational support, i.e., process mining is not done off-line but online. Examples are the recommendations provided in [34] and the predictions given in [1]. These papers illustrate that existing process mining techniques can be used in a real time setting. However, it is impossible to give a complete review of process mining techniques here, see http://www.processmining.org for more pointers to the literature.

Further related research is discussed in [18,19]. Here the authors have developed a declarative approach for process modeling using the SBVR (Structured Business Vocabulary and Rules) vocabulary and created a new framework. The vocabulary is supported by a model and allows process modeling and specification of access constraints in an English-like language. They also support defeasible logic [28] which is a non-monotonic logic and can work with a set of inconsistent constraints. Another approach for handling compliance inspired by defeasible logic and deontic logic [8] is discussed in [32]. These logics are more advanced than predicate logic, and are based on notions of permissions, obligations and prohibitions. They are applied in the context of the Business Contract Language (BCL) [21,26] where the focus is on how to proceed when one party fails to meet its obligations. In [5], the authors have used temporal logic expressions to check whether a log corresponds to constraints.

Prior research has looked at the issue of information security from various perspectives, e.g. at the network and operating system levels. However, our focus is on security at the application level, and the stream of security related research that is relevant here pertains to role based access control (RBAC) [33]. The notion of separation of duties [24,35], although it preexisted in accounting and control systems, also reemerged in the context of RBAC as the idea that if task 1 is performed by role A, then task 2 must be performed by role B, and membership of these roles must not intersect. This is a useful framework that has now been widely adopted in popular database management systems from IBM and Oracle.

Some related work on specification and enforcing role-based authorizations in workflow systems is discussed in [11]. The main focus of this work is on enforcement of constraints at run time. A formal model called W-RBAC for extending RBAC in the context of workflows using the notions of case and organizational unit is described in [36]. The approach in [12] is based on the notions of conflicting roles, permissions, users and tasks. More sophisticated algorithms for enforcing separation of duties in workflows are developed in [27]. Finally, another stream of prior work that informs our research is the literature on basic financial control principles, particularly as it relates to the recent Sarbanes–Oxley legislation [10,15,22,23].

## 10. Conclusion

Currently, the work of an auditor is mostly manual, and thus very laborious. Many existing tools that can be used for auditing only focus on a small part of the actual work of an auditor. In this paper, we argued for the need for online auditing of the business processes of an organization and proposed an Online Auditing Tool (OLAT). Such an OLAT is connected to the organization's information system but is not a part of it. The assumption is made that all relevant events in the information system are passed to the OLAT. In this way, the OLAT can build an independent image of the state of the business processes and information systems

executing day to day operations. Based on this image auditing processes can run continuously. Although some tools and techniques exist, these techniques are not well integrated into a single information system.

We presented a high-level architecture of such an OLAT and studied in more detail the database and the conformance checker. We also designed a conceptual data model with a set of consistency constraints in predicate logic. The business rules are designed to realize this part of the OLAT by a standard database management system in such a way that each business rule is translated in a straightforward way into a query that can be executed against the database. For the other components of the OLAT we have referred to process mining techniques and tools. We have performed some real-life case studies with the approach using a preliminary tool, although in an off-line mode. The studies performed so far demonstrate the realizability of the approach. Together with an auditor firm, we are building a prototype of such an OLAT tool by integrating the currently available off-the-shelf components.

There are several aspects of this work that need elaboration. First of all, we would like to build a prototype and perform online experiments with it. Secondly we should have the ability to insert business rules from a library of predefined business rule like the ones given in Section 6. This would make it feasible for controllers and other business experts to add business rules for conformance checking without the help of computers scientists, by just filling in the parameters. Thirdly, we plan to refine the conceptual model in order to make the delegation of roles easier. We also intend to extend the conceptual model to incorporate domain specific knowledge, for, say, financial departments or health care systems. Finally there are several unexplored components in the OLAT architecture, such as the Risk Interrupter, potential risk detector and difference analyzer. We have some rough ideas for them, but there are many open questions. However, the most urgent activity is experimentation with a prototype, because the proof of the pudding is in the eating.

## Acknowledgements

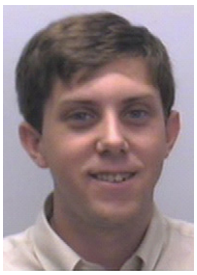
Some part of the work on this paper was done while Akhil Kumar was visiting the Information Systems Department at the Technical University of Eindhoven. He appreciates the hospitality of the hosts. His research was funded in part by the Smeal College of Business at Penn State.

## References

[1] W.M.P. van der Aalst, Using process mining to generate accurate and interactive business process maps, BIS 2009 Workshops, vol. 37 of LNBIP, Springer, 2009, pp. 1–14.
[2] W.M.P. van der Aalst, K.M. van Hee, Workflow Management: Models, Methods and Systems, The MIT press, Cambridge, Massachusetts, 2002.
[3] W.M.P. van der Aalst, B.F. van Dongen, J. Herbst, L. Maruster, G. Schimm, A.J.M.M. Weijters, Workflow mining: a survey of issues and approaches, Data & Knowledge Engineering 47 (2) (2003) 237–267.
[4] W.M.P. van der Aalst, A. Weijters, L. Maruster, Workflow mining: discovering process models from event logs, IEEE Transactions on Knowledge and Data Engineering 16 (9) (2004) 1128–1142.
[5] W.M.P. van der Aalst, H. Beer, B. Dongen, Process mining and verification of properties: an approach based on temporal logic, CoopIS 2005, No. 3760 in LNCS, Springer, 2005, pp. 130–147.
[6] W.M.P. van der Aalst, B.F. van Dongen, et al., ProM 4.0: comprehensive support for real process analysis, ICATPN 2007, vol. 4546 of LNCS, Springer, 2007, pp. 484–494.
[7] R. Agrawal, D. Gunopulos, F. Leymann, Mining process models from workflow logs, Sixth International Conference on Extending Database Technology, 1998, pp. 469–483.
[8] G. Antoniou, N. Dimaresis, G. Governatori, A system for modal and deontic defeasible reasoning, AI 2007: Advances in Artificial Intelligence, No. 4830 in LNCS, Springer, 2007, pp. 609–613.
[9] A. Basu, R.W. Blanning, A formal approach to workflow analysis, Information System Research 11 (1) (2000) 17–36.
[10] D. Berg, Turning Sarbanes–Oxley projects into strategic business processes, Sarbanes–Oxley Compliance Journal (2004).
[11] E. Bertino, E. Ferrari, V. Atluri, The specification and enforcement of authorization constraints in workflow management systems, ACM Transactions on Information and System Security 2 (1) (1999) 65–104.
[12] R.A. Botha, J.H.P. Eloff, Separation of duties for access control enforcement in workflow environments, IBM Systems Journal 40 (3) (2001) 666–682.
[13] K. Chari, J. Perols, An AI-based approach for procurement fraud detection, Proceedings of the Workshop on Information Technologies and Systems, 2005.
[14] P.P. Chen, The entity-relationship model: towards a unified view of data, ACM Transactions on Database Systems 1 (1976) 9–36.
[15] Committee of Sponsoring Organizations, Internal Control-Integrated FrameworkURL, http://www.coso.org/publications/executivesummaryintegratedframework.htm.
[16] J.E. Cook, A.L. Wolf, Software process validation: quantitatively measuring the correspondence of a process to a model, ACM Transactions on Software Engineering and Methodology 8 (2) (1999) 147–176.
[17] B.F. van Dongen, R.M. Dijkman, J. Mendling, Measuring similarity between business process models, CAiSE, 2008, pp. 450–464.
[18] S. Goedertier, J. Vanthienen, Declarative process modeling with business vocabulary and business rules, OTM 2007 Workshops, No. 4805 in LNCS, Springer, 2007, pp. 603–612.
[19] S. Goedertier, C. Mues, J. Vanthienen, Specifying process-aware access control rules in SBVR, Advances in Rule Interchange and Applications, No. 4824 in LNCS, Springer, 2007, pp. 39–52.
[20] S. Goedertier, D. Martens, B. Baesens, R. Haesen, J. Vanthienen, Process mining as first-order classification learning on logs with negative events, BPM 2007 Workshops, vol. 4928 of LNCS, Springer, 2008, pp. 42–53.
[21] G. Governatori, Z. Milosevic, A formal analysis of a business contract language, International Journal of Cooperative Information Systems 15 (4) (2006) 659–685.
[22] S. Green, Manager's Guide to the Sarbanes–Oxley Act: Improving Internal Controls to Prevent Fraud, Wiley, 2004.
[23] D.A. Haworth, L.R. Pietron, Sarbanes–Oxley: achieving compliance by starting with ISO 17799, Information Systems Management 23 (1) (2006) 73–87.
[24] D.R. Kuhn, Mutual exclusion of roles as a means of implementing separation of duty in role-based access control systems, RBAC 97, ACM, New York, NY, USA, 1997, pp. 23–30.
[25] A. Kumar, J.L. Zhao, Dynamic routing and operational controls in workflow management systems, Management Science 45 (2) (1999).
[26] P.F. Linington, Z. Milosevic, J. Cole, S. Gibson, S. Kulkarni, S. Neal, A unified behavioural model and a contract language for extended enterprise, Data & Knowledge Engineering 51 (1) (2004) 5–29.
[27] D.-R. Liu, M.-Y. Wu, S.-T. Lee, Role-based authorizations for workflow systems in support of task-based separation of duty, The Journal of Systems and Software 73 (3) (2004) 375–387.
[28] D. Nute, Defeasible logic, handbook of logic in artificial intelligence and logic programming, Nonmonotonic Reasoning and Uncertain Reasoning, vol. 3, 1994, pp. 353–395.
[29] J. Paredaens, P. De Bra, M. Gyssens, D. van Gucht, The Structure of the Relational Database Model, Springer-Verlag New York, Inc., New York, NY, USA, 1989.
[30] A. Rozinat, W.M.P. van der Aalst, Conformance testing: measuring the fit and appropriateness of event logs and process models, BPM 2005 Workshops, vol. 3812 of LNCS, Springer, 2006, pp. 163–176.
[31] A. Rozinat, W.M.P. van der Aalst, Conformance checking of processes based on monitoring real behavior, Information Systems 33 (1) (2008) 64–95.
[32] S. Sadiq, G. Governatori, K. Namiri, Modeling control objectives for business process compliance, Business Process Management, No. 4714 in LNCS, Springer, 2007, pp. 149–164.
[33] R. Sandhu, E. Coyne, H. Feinstein, C. Youman, Role-based access control models, IEEE Computer 29 (2) (1996) 38–47.
[34] H. Schonenberg, B. Weber, B.F. van Dongen, W.M.P. van der Aalst, Supporting flexible processes through recommendations based on history, BPM 2008, vol. 5240 of LNCS, Springer, 2008, pp. 51–66.
[35] R.T. Simon, M.E. Zurko, Separation of duty in role-based environments, Computer Security Foundations Workshop, 1997, Proceedings., 10th, 1997, pp. 183–194.
[36] J. Wainer, A. Kumar, P. Barthelmess, DW-RBAC: a formal security model of delegation and revocation in workflow systems, Information Systems 32 (3) (2007) 365–384.

**Wil van der Aalst** is a full professor of Information Systems at the Technische Universiteit Eindhoven (TU/e). Currently he is also an adjunct professor at Queensland University of Technology (QUT) working within the BPM group there. His research interests include workflow management, process mining, Petri nets, business process management, process modeling, and process analysis. For more information about his work visit: http://www.workflowpatterns.com, http://www.workflowcourse.com, http://www.processmining.org, http://www.yawl-system.com, or http://www.wvdaalst.com.

**Kees M. van Hee** is a full professor of Information System at the Technische Universiteit Eindhoven since 1984. He was 16 years managing director of several consultancy firms, including Deloitte. In 1999 he became partner at Deloitte as national director of consultancy until 2004. Since 2004 he is full professor again. He published over 120 articles on the following topics: Markov decision processes, applications of queuing theory, decision support systems, specification methods and tools, Petri nets, database systems and workflow management systems. He published five books and he is the originator of the software tools ExSpect and Yasper. He conducted over 20 PhD-projects and over 130 master thesis projects. Five of his PhD students became full professors. He presented over 150 lectures at conferences for scientists, practitioners or managers.



**Jan Martijn van der Werf** is a PhD candidate in the Architecture of Information Systems group at the Technische Universiteit Eindhoven. He obtained his M.Sc. in Business Information Systems (2006) at the same university. His research interests include modeling and verification of information systems, their architectures, and the use of process mining in the monitoring of such systems.

**Akhil Kumar** is a professor of information systems at the Smeal College of Business at the Pennsylvania State University. He received his Ph.D. from the University of California at Berkeley, and has previously been on the faculties at Cornell University and University of Colorado. He has done pioneering work in data replication and XML based workflows. His research interests are in workflow systems, e-services, distributed information systems and intelligent systems. He has published more than 80 scientific papers in academic journals and international conferences, and also held many editorial positions.

**Marc Verdonk** is a senior manager and IT auditor with Enterprise Risk Services at Deloitte Nederland as well as a PhD candidate in the AIS group at Technische Universiteit Eindhoven. He obtained his M.Sc. in Computer Science at Universiteit Utrecht and is both Certified Information System Auditor (CISA) as Certified Information System Security Professional (CISSP). His main interest is to make the audit profession future-proof by designing and applying technology-based approaches that makes auditing more efficient, effective, value added and maybe even fun, for the auditee as well as the auditor.