Conceptual Model for On Line Auditing

Wil van der Aalst, Kees van Hee, Jan Martijn van der Werf

Department of Mathematics and Computer Science, Technische Universiteit Eindhoven, P.O. Box 513, 5600 MB Eindhoven, The Netherlands

Akhil Kumar

Smeal College of Business, Penn State University, University Park, State College PA 16802.

Marc Verdonk

Deloitte, Netherlands

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 on line 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.

Key words: information assurance, auditing, architecture, conceptual model, constraints, business rules, conformance checking

Email addresses: w.m.p.v.d.aalst@tue.nl (Wil van der Aalst), k.m.v.hee@tue.nl (Kees van Hee), j.m.e.m.v.d.werf@tue.nl (Jan Martijn van der Werf), AkhilKumar@psu.edu (Akhil Kumar), mverdonk@deloitte.nl (Marc Verdonk)

1. Introduction

Organizations are executing business processes to reach their goals. 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, 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 working instructions for employees. As information systems become more and more complex, in many situations it becomes 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 stake holders. 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. All auditors should be independent in their research approach and in their judgement.

For financial statements we have the *financial audit* 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 thing that 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 conclude 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 *realtime* insights into violations of business rules. 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 realtime. One of the approaches used today is to embed *controls* in the information system. A control is an automated task in the information system aiming 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 principle that says that "two tasks for the same case should be handled by different agents".

Since the information systems become too complex to oversee them, it

may seem paradoxical that another information system is needed to check the first one. However, that is what we propose. 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 events conform to business rules. We call such a system an *On Line 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 looks as if the OLAT becomes part of the information system and therefore it could lose 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.

In this paper we sketch a "full blown" OLAT. Our description gives insight into the architecture and functionality of such a system. However, some components are ill-defined or even speculative. We present them here to give an impression of the ideal tool in the future and to sketch the context of the components we are concentrating on in this paper. These components are: the *database* and the *conformance checker*.

The organization of this paper is as follows. In Section 2 we give some

preliminaries, i.e., techniques for data modeling, process modeling and the language to express business rules. Note that we need these frameworks in order to express and check the business rules. We are not modeling the complete business processes of the organization, but only those aspects that are relevant for the business rules to be monitored. 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. There we 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. This model can be considered as an abstract design for the database of the OLAT. This database will record all data facts from the information system that are relevant to verify the business rules: both the relevant details of the business objects and the business processes. In Section 6 we study the business rules in detail. We have chosen to be as language independent as possible. Therefore we use more or less standard predicate calculus to express the rules. Here we also give templates of business rules, among which are several well-known rules, and we show how they can be expressed in the constraint language for the conceptual data model. 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. Full validation of the approach is out of scope. Section 9 discusses related work and finally, the last section gives a conclusion and our plan for future work.

2. Preliminaries

Here we explain the techniques for data modeling, for process modeling and the predicate language to express rules. Since we prefer to be as

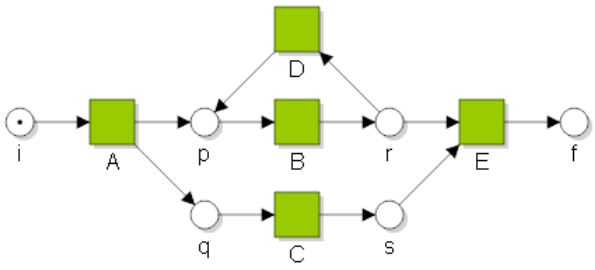

Figure 1: A simple Petri net

language independent as possible, we use plain predicate calculus for the business rules. They are expressed in terms of the data model. To facilitate that, we use a restricted form of the ER-model with some shorthand notations. For process models there are many options, but we have chosen Petri nets as a formalism that is well-understood, meaning it has clear behavioral semantics and it is very concise compared to more industrial process languages, like BPMN or UML-activity diagrams, which are very close to Petri nets. Note that we do not intend to present a new modeling approach, rather we need a consistent combination of different modeling frameworks. To be self-contained we present a more formal treatment of these modeling frameworks in Appendix B.

2.1. Petri Nets

For modeling of business processes we use Petri nets [26]. 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* when a transition can be executed. 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*

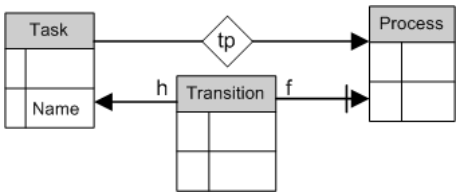

Figure 2: A simple Data model

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. Consider the example of Figure 1. Transition A is enabled, since i has a token. After firing of transition A, places p and q both have a token. Transitions B and C are enabled since place p and place q respectively have a token. After firing transition B, place r has a token, but as place s has no token, transition E is not enabled. Only after firing transition C, s has a token, thus enabling transition E. Transitions E and D are in conflict, as they share a marked input place. Firing of transition D removes the token from place r, and therefore, transition E is not enabled anymore.

2.2. Data Models

A database consists of *entities*, elements or records, stored in tables. The fields of the table are called the *attributes* of the entity. Between these entities, *associations* exist. Entities belong to an *entity type*, associations belong to a *relationship*. The entity type defines also the type of attributes of an entity. An Entity-Relationship diagram (ERD) [13] describes the type

of the 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 are traditionally drawn as diamonds and they are connected to entity types with arcs. We consider only binary relationships and most of them are functional relations. A functional relationship can be represented as a (possibly partial) function from one entity type to another. For functional relationships we drop the diamond notation and we represent them directly by an arc from the source entity type to the target entity type. If at the arrow head a vertical bar is drawn, the function is total. 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. Note that the names for the relationships are only unique for the source entity; from the context it is always clear which relation is meant. Consider the data model of Figure 2. In this data model, there are three entity types: 'Task', 'Process' and 'Transition', two functional relationships: 'h' and 'f', and a non-functional relationship 'tp'. The entity type 'Task' has a single attribute 'name'. The arrow on relation 'tp' indicates that $tp \subseteq \mathit{Task} \times \mathit{Process}$, i.e., 'tp' is a many-to-may relation from 'Task' to 'Process'. The relationship indicates that an entity of type 'Task' can have many associations to entities of type 'Process', and entities of type 'Process' can have many associations to entities of type 'Task'. Now consider the functional relation 'h'. 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' has source 'Transition' and target 'Process'. The vertical bar at the arrow head indicates that 'f' is a total function.

On data models we use predicate calculus with logical and set operations

and quantifiers over domains of entity types, relations and sets. 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:

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

Note that $h(t_1)$ indicates the unique entity that is related to t_1 by functional relation h. So $h(t_1)$ is a term that can be used to for a predicate, for instance $h(t_1) = h(t_2)$.

Most predicates can be translated into standard SQL (cf. [25]), all predicates can be checked using SQL augmented with stored procedures. This *constraint* can be checked by translating it into a standard SQL query and checking whether the result is *empty*. In this case, the query would look like: `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, alternatively, 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 in fact 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 real-time.

Since the audit applies to a business process, we briefly review the basic process terminology (cf. [3]). 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 *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, the case data of that case is shown. In the business process a task can be any kind of activity, however in the information system the *execution* of a task boils down 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. [3]). 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

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

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 possess this 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 dollars to his account.
2. *Corrective*: this type of control is like the detective control, but has the added functionality to (attempt to) correct the violation to the business rule directly. An example: an employee has just transferred 1 million dollars 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 there is 1 million dollars 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 Figure 3 the top-level architecture of the OLAT is presented. We distinguish data sets (displayed by drums) and program modules (displayed by rectangles). 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: *Runtime* data, *Dejure* models, and *Defacto* models. The Runtime data collects the data from the monitored information system. The Dejure models are the official models of the desired organization. In fact, the Runtime data should *conform* to the Dejure models. If not, there is a violation. The Defacto models are models derived from the Runtime data by discovery techniques and can differ from the Dejure models.

4.1. Runtime Data

The Runtime data contains the data from the information system. It is in fact the system log in which all (relevant) events of the information system are recorded. So the Runtime data concerns the events in the business

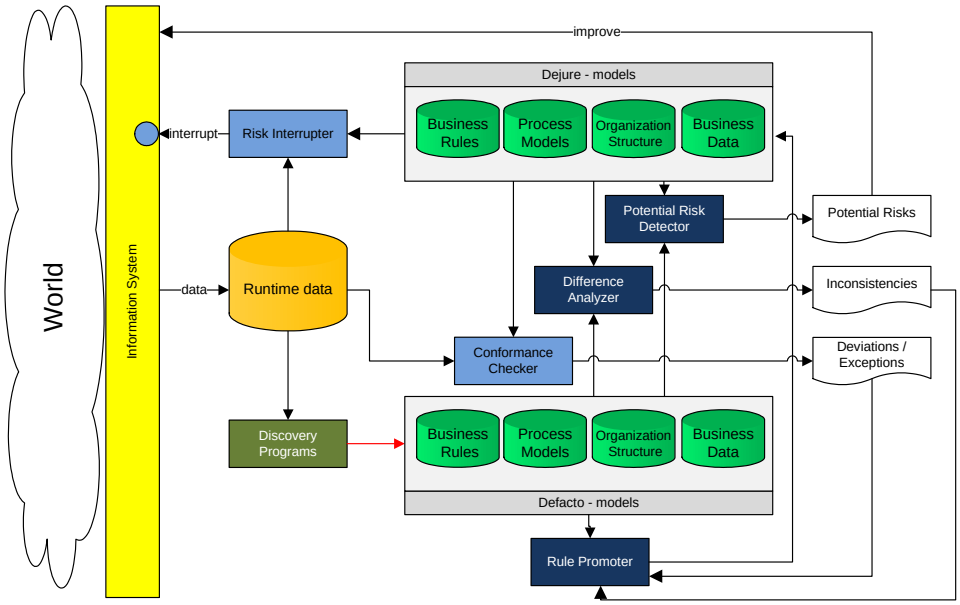

Figure 3: A top level architecture of an Online Auditing Tool

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.

4.2. Dejure and Defacto Models

The Dejure models describe the desired or official situation, whereas the Defacto models are derived from the Runtime data, and thus describe the actual observed situation / behavior. The Dejure models are made for the design of the information system. In the ideal situation these models are really implemented. Both the Dejure models and Defacto 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 examples of Dejure

and Defacto 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 Dejure models describe the desired or official behavior, business rules in the Dejure models should not be violated, whereas business rules in the Defacto models are discovered (as will be discussed later) in the Runtime data. The Dejure models are loaded from the information system or directly from the organization, while the Defacto models are obtained by discovery techniques. They may be less complete than the Dejure models. The Defacto and Dejure models share the same database schema as presented in the next section (see Figure 4).

4.3. Conformance Checker

The Conformance Checker checks whether the Runtime data conforms to the Dejure models, in particular the Dejure 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. [25]). The queries run on the database (i.e. the Dejure models plus the Runtime 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 / or 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.

4.4. Discovery Programs

Different from the conformance checker, discovery programs try to *derive* models out of the Runtime data. Many kinds of existing *data mining*

and *process mining* techniques and tools can be used to discover not only control flow, but also authorization rules, business data models, organizational models and business rules [4, 27, 28]. In general, mining techniques try to deduce patterns and rule from facts. In our case the facts are stored as events in the Runtime 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. [1]). For the structure of business data we could look at data in the form of forms that are used in the events to derive entities and relationships. For organizational models we could look at the permission events (in which an agent is granted a permission by another agent). As these rules are derived from the Runtime data, the models obtained by discovery are Defacto models. While detailed discussion of these techniques is beyond the scope of the current work, the kind of tools we have in mind are included in the well-known Process Mining toolset ProM [2].

4.5. Rule Promoter

The Rule Promoter represents functionality to convert a discovered Defacto model into a Dejure model, specifically if 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 Dejure models to the actual situation. Later on it may be part of a process of continuous improvement; e.g. when exceptions are discovered, analyzed and accepted, these exceptions are added to the Dejure models thus eliminating 'false positives' in the conformance checker.

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

4.6. Risk Interrupter

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

4.7. Difference Analyzer

The Difference Analyzer compares the Dejure and Defacto models. It also checks whether business rules, process models and organization structure are not conflicting between the Dejure and Defacto models. This check 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.

4.8. Potential Risk Detector

This module is able to detect *potential* risks by analyzing the Runtime data, the Defacto and the Dejure models. For instance, if a Dejure model differs from a Defacto model, we could use it to see if a violation of a Dejure business rule could occur. This information is considered as a warning. In the ProM toolset ([2]) 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 of the information system and perhaps sends interrupts to the information system. 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 in fact involves the construction of a standard data-intensive application. For the other modules, like the discovery programs, we can use existing tools that use the database. So the implementation is a serious engineering effort but 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 datasets needed for the OLAT. Figure 4 depicts the conceptual model, using the techniques described in Section 2. The conceptual model consists of a data model, which is explained in Section 5.1, and *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 treated in Section 6.

5.1. Data Model

Figure 4 depicts the conceptual model. It is arranged into four components: the process definition, the business data definition, the organizational definition and runtime. We first explain the conceptual model, and

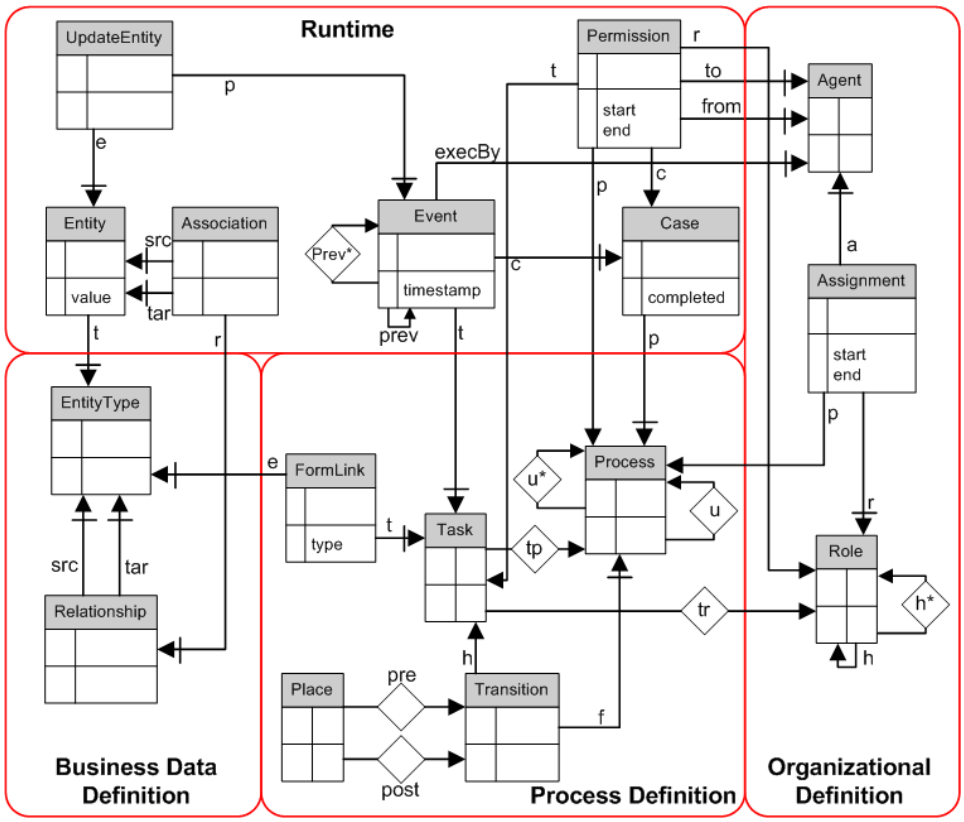

Figure 4: Conceptual model for OLAT

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 Dejure and Defacto models here: they share the same data model. Also, note that we sometimes introduce transitive closures of relations (i.e., h^*, u^* and $prev^*$). 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 to implement the conformance checker with a SQL engine.

5.1.1. Business Data Definition

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

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 sub processes. 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 sub process of x. (In the *instance* (see Appendix B) of the data model this means that $(x, y) \in u$.) 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 in the introduction, 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 this relation, 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. This way, places can be shared by two or more processes, thus providing the possibility to define compositions of processes, rather than only flat processes. The initial tokens of a place are an attribute of the place. Note that although we use labeled Petri nets, any other process notation could be used to define the order in which tasks can occur. Also note that we have no direct Runtime 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. [27, 28]).

5.1.3. Organizational Definition

Tasks can be executed by different *role*s. Roles 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. To indicate the time interval in which this assignment holds, the entity *Assignment* has two attributes: start and end, indicating the start and end times of the interval.

5.1.4. Runtime

The Runtime component stores all *events* and associated data from the information system. There are two type of events: events that indicate that something has been done for a specific task (the entity *Event* in the model) and 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 occurrences of events form a partial order. This partial order is represented by the relation *prev*. The relation $prev^*$ 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, 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 he delegates. Note that it is not always detectable 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. We first describe the constraints in natural language, their formalization can be found in Appendix A.

5.2.1. Consistency Constraints for the Process Definition

The conceptual model allows for sub processes. Although a process can be nested arbitrarily deep, cycles in the process hierarchy are of course not allowed. This can be expressed using two constraints. First, the relation u should be irreflexive, i.e. processes cannot depend on themselves. Secondly, as u^* is the transitive closure of u, and we do not want any 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, this task should be a task of that process.

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

To formalize p4, this statement is identical to stating that if for two transitions t_1 and t_2 the related task and process are the same, the transitions are the same:

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

The formalization of all the constraints can be found in Appendix A.

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. Permissions can be granted to act in a certain role, to perform a task, or to be involved in a process or case, or any combination therof. 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 transitive closure of relation h.

o2: Relation h^* is reflexive.

o3: The start time of an assignment is strictly smaller than its end time.

o4: The start time of a permission is strictly smaller than its end time.

o5: 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.

o6: 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 Runtime

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 $prev^*$ is the transitive closure of relation *prev*.

r2: If an event y occurs after event x, then the time stamp 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 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 Λ 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 use the same function names as in the conceptual model of Figure 4.

Task always precedes a task A task t_2 should always be performed before task t_1 in any case of process u.

$$\texttt{b1: TaskAlwaysBeforeTask}(\texttt{u} : \texttt{Process}, \texttt{t}_1, \texttt{t}_2 : \texttt{Task}) :=$$
$$\forall x_1 \in \mathit{Event} : (p(c(x_1)) = u \wedge t(x_1) = t_1) \Rightarrow$$
$$\exists x_2 \in \mathit{Event} : t(x_2) = t_2 \wedge c(x_1) = c(x_2) \wedge (x_2, x_1) \in \mathit{prev}^*$$

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.

$\texttt{b2: RestrictUpdate}(\texttt{u} : \texttt{Task}, \texttt{x} : \texttt{EntityType}) :=$

$$\forall e_1, e_2 \in \mathit{Event} : \ c(e_1) = c(e_2) \ \wedge \ t(e_1) = u \ \wedge \ (e_1, e_2) \in \ \mathit{prev}^* \\ \wedge \neg(\exists y \in \mathit{UpdateEntity} \ : \ p(y) = e_2 \ \wedge \ t(e(y)) = x)$$

Limit number of repetitions of a task in cases of a process In any case of process P task u cannot be executed more than n times.

$\texttt{b3: LimitNrOfTasks}(\texttt{u} : \texttt{Process}, \texttt{z} : \texttt{Task}) :=$

$$\forall w \in \mathit{Case} : p(w) = u \Rightarrow |\{x \in \mathit{Event} \mid c(x) = w \wedge t(x) = z\}| \leq n$$

6.2. Examples of Agent Based Rules

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

4-eyes principle Two tasks t_1 and t_2 in the same case should always be executed by different agents.

$\texttt{b4: 4EyesPrinciple}(\texttt{t}_1, \texttt{t}_2 : \texttt{Task}) := \forall x, y \in \mathit{Event} \ :$

$$(c(x) = c(y) \ \wedge \ t(x) = t_1 \ \wedge \ t(y) = t_2) \Rightarrow \mathit{execBy}(x) \neq \mathit{execBy}(y)$$

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

$\texttt{b5: MutualExclusiveAgents}(\texttt{a}_1, \texttt{a}_2 : \texttt{Agent}) := \neg\exists u_1, u_2 \in \mathit{Event} \ :$

$$u_1 \neq u_2 \ \wedge c(u_1) = c(u_2) \wedge \ \mathit{execBy}(u_1) = a_1 \ \wedge \ \mathit{execBy}(u_2) = a_2$$

Task limit on an agent an agent a cannot do more than n tasks in any case of process u.

b6: $\texttt{TaskLimitOnAgent}(\texttt{u} : \texttt{Process}, \texttt{a} : \texttt{Agent}, \texttt{n} : \texttt{Nat}) :=$

$\forall w \in \mathit{Case} : (p(w) = u) \Rightarrow$

$|\{x \in \mathit{Event} \mid c(x) = w \wedge \mathit{execBy}(x) = a\}| \leq n$

Forbidden to write An agent a_1 is not allowed to update any entity in a process u.

b7: $\texttt{ForbiddenToWrite}(\texttt{a} : \texttt{Agent}, \texttt{u} : \texttt{Process}) := \forall x \in \mathit{Event}$:

$(\mathit{execBy}(x) = a \wedge p(c(x)) = u) \Rightarrow \neg(\exists y \in \mathit{UpdateEntity} : p(y) = x)$

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 then another value, or
- a value should be within some given set (i.e. within some limits).

Limit on 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: $\texttt{LimitEntAgent}(\texttt{a} : \texttt{Agent}, \texttt{b} : \texttt{EntityType}, \texttt{x} : \texttt{A}, \texttt{n} : \texttt{V}) :=$

$\forall z \in \mathit{Event}, y \in \mathit{UpdateEntity} : (\ p(y) = z\ \wedge\ t(e(y)) = b$

$\wedge\ \mathit{execBy}(z) = a\ ) \Rightarrow e(y).\mathrm{x} \leq n$

Limit on 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: $\texttt{LimitEntInCase}(\texttt{w} : \texttt{Case}, \texttt{b} : \texttt{EntityType}, \texttt{x} : \Lambda, \texttt{n} : \texttt{V}) :=$

$\forall y \in \mathit{Event}, z \in \mathit{UpdateEntity} : c(y) = w \wedge t(e(z)) = b \ \wedge p(z) = y$
$\wedge\ e(z).\mathrm{x} < n$

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

b10: $\texttt{ApprLim}(\texttt{a} : \texttt{Agent}, \texttt{u} : \texttt{Task}, \texttt{b} : \texttt{EntityType}, \texttt{x} : \Lambda, \texttt{n} : \texttt{V}) :=$

$\forall y \in \mathit{Event} : (\mathit{execBy}(y) = a \wedge t(y) = u)$
$\Rightarrow \texttt{LimitEntinCase}(c(y), b, x, n)$

Note that `LimitEntinCase` is defined in rule b9.

Three way match In each case of 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: $\texttt{ThreeWayMatch}(\texttt{u} : \texttt{Task}, \texttt{a}, \texttt{b}, \texttt{c} : \texttt{EntityType}) :=$

$\forall w \in \mathit{Case}, v \in \mathit{Event}, x, y, z \in \mathit{UpdateEntity} :$
$(\ t(v) = u \wedge\ \ c(p(x)) = c(p(y)) = c(p(z)) = c(v) = w\ )$
$\Rightarrow (\ e(x).\mathrm{value} = e(y).\mathrm{value} = e(z).\mathrm{value}$
$\wedge\ t(e(x)) = a\ \wedge\ t(e(y) = b \wedge\ t(e(z)) = c\ )$

7. Example

In the previous section we have shown how in our conceptual model business rules can be expressed in predicate logic, and thus can be checked by transforming these predicates into queries and running them on the database. As these predicates are parameterized, they can be filled in for a specific process, by specifying the parameters of the business rules. In this way, end users and process owners are not confronted with predicate logic, but rather express business rules on their processes by picking these predicates and assigning values to the parameters of the rules.

As an example to illustrate the framework, Figure 5 shows an *Administer Account Transfer* process. In this Petri net, if a transition is connected to a task, the transition is labeled with the name of that task. The Petri net has three unlabeled silent steps, which are needed for routing the process. The Petri net has 20 transitions, the business process consists of 17 tasks; three transitions are in fact "silent steps" only added for control flow purposes. The process starts with a *customer representative* receiving (task *t1*) an account transfer instruction 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 is higher than the limit for the customer, the process starts the *Authorization* sub process, in which the financial accountant requests an authorization, which is either authorized or not by the financial manager (tasks *t8a* and *t8b*). If the limit is not reached, or the transaction is authorized, the *banking specialist* checks the available funds. If this check fails, communication details

Table 1: Task-Role matrix

	Roles					
task	Customer Representative	Banking Specialist	Senior Financial Manager	Financial Manager	Financial Accountant	Financial Clerk
task t1	✓					
task t2	✓					
task t3						✓
task t4					✓	
task t5						✓
task t6		✓				
task t7					✓	
task t8a				✓		
task t8b				✓		
task t9						✓
task t10a					✓	
task t10b						
task t11						
task t15						✓
task t16			✓			
task t17						
task t18	✓					

are derived from the account unit (task *t9*). If the check is successful, the *Accounting Entry* sub process is started, which applies the accounting entry and calculates a fee for it. In all cases, the results are collected in a report (task *t15*), and after approving it (task *t16*), the customer is notified (task *t18*). If the report is not approved, it is changed (task *t17*).

The process also involves the role of the *senior financial manager*, who is in charge of the *financial manager*. The financial manager is head of a team consisting of a financial accountant and a financial clerk. Table 1 shows the assignment of roles to tasks. Note that by the hierarchy, e.g., the senior financial manager can do everything a financial clerk 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. On this organization, we define the business rules that need to hold on

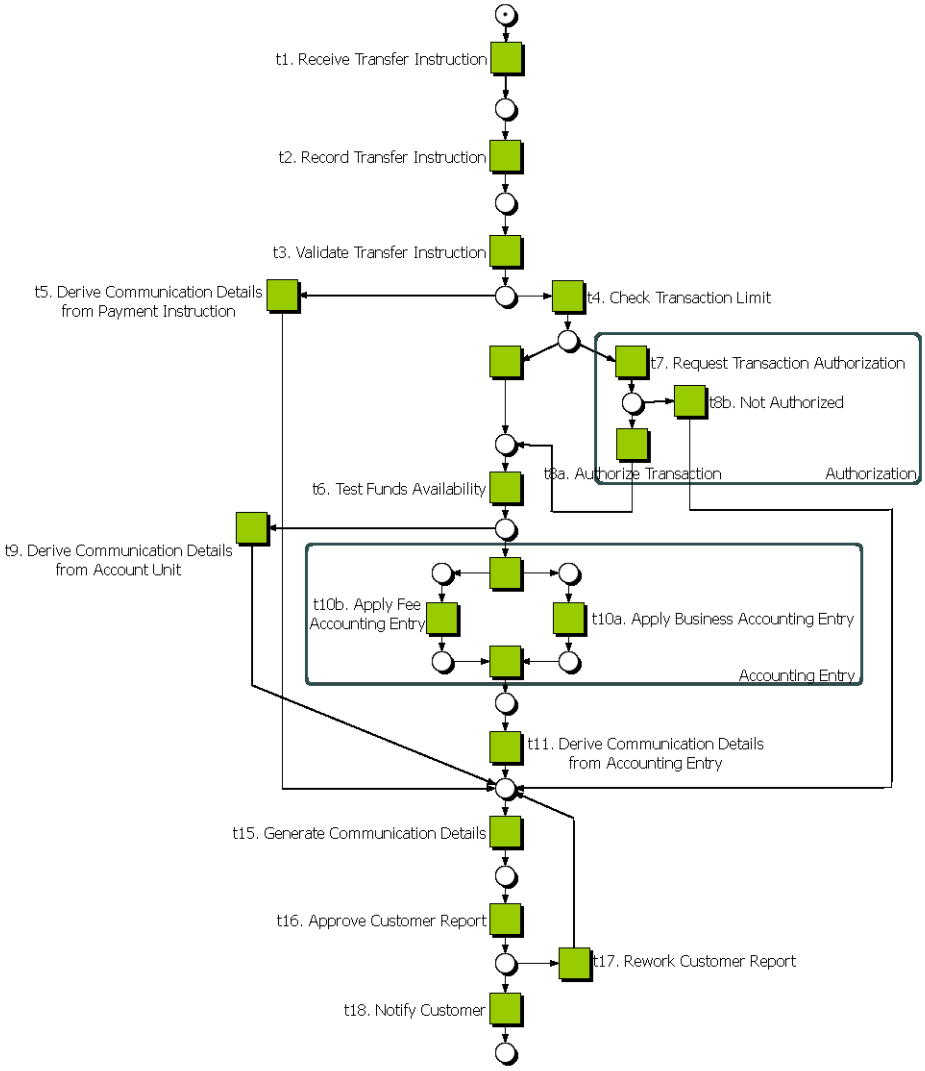

Figure 5: Example of an account transfer process

the process. Firstly, 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, and agent-beth cannot do more than one task. Last, tasks *t7* and *t8a* in a case may not be executed by the same agents, and this also applies to tasks *t7* and *t8b*, and for tasks *t10a* and *t10b*. 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: `TaskLimitOnAgent`(*agent-beth*, 1)

e5: `4EyesPrinciple`($t7, t8a$)

e6: `4EyesPrinciple`($t7, t8b$)

e7: `4EyesPrinciple`($t10a, t10b$)

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 specifically to a process or a subprocess.

8. Practical Experience with Business Rules

As *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 to

substantive data analytics, to 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, it fails and 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 a large log files from real information systems. We mention one example in each of the rule classes we have identified.

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. 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 business rule "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 ~~was~~ in a significant number of cases occurred after registration of the invoice.

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 "Forbidden to write" to distinguish between processes in business units (a.k.a "Company Code" in SAP) we found that on 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 the process of two 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.

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 and 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. However, there has been significant research interest 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 [4] 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 [6]. The alpha algorithm was the first technique able to discover concurrency [5]. Process mining is not limited to discovery. For example, in the context of ProM [2] several approaches to conformance checking were realized. The best developed technique is the Petri-net-based conformance checking technique by Rozinat et al. [27, 28]. 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. It is impossible to give a complete review of process mining techniques here, see `www.processmining.org` for more pointers to literature.

Some related research in this direction is discussed in [14, 15]. 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 [8, 23] 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 [9] is discussed in [29]. 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, 16] where the focus is on how to proceed when one party fails to meet its obligations. In such situations, the negligent party is obliged to perform some other actions in order to make certain amends for its failure as specified in BCL. A shortcoming of predicate logic is that it does not allow description of such scenarios easily. In [1], the authors have used temporal logic expressions to check whether a log corresponds to constraints. They express their constraints in Linear Time Logic (LTL) [19] and use a tool called LTL checker to verify if certain desired properties of the log are satisfied.

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 relates to role based access control (RBAC) [30]. The basic RBAC framework consists of three entities: roles, permissions and users. Roles (such as manager, director, etc.) are assigned permissions or rights (to hire an employee, approve a purchase, etc.) and users (Joe, Lin, Sue) are associated with roles. Thus, users acquire certain permissions to perform organizational tasks by virtue of their membership in roles. The notion of separation of duties [20, 31], 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. There are two types of separations of duty: static and dynamic. In recent years, RBAC has become the preferred access control model for most business enterprises. This framework allows association of roles with tasks, and only users that belong to a certain role can perform certain tasks. 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. The authors develop algorithms to check whether, given a combination of tasks and users, it is possible to find a task assignment that will satisfy the collection of constraints and available users. 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 [32]. A system architecture for enforcing RBAC in a Web-based workflow system is given in [7]. 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 [22]. 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, 24, 17, 18].

10. Conclusion

We introduced the need for the on-line auditing of the business processes of an organization and proposed an On-line Auditing Tool (OLAT). Such an

OLAT is connected to the organizations 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 affairs in the business processes. Based on this image auditing processes can run continuously. We have given a high level architecture of such an OLAT and we studied in more detail the database and the conformance checker. In fact we 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 on 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, however in an off-line mode.

There are several aspects of this work that need elaboration. First of all we would like to build a prototype and to perform some on-line experiments with it. Secondly we should have the ability to insert business rules from a library of business rules, i.e a set of predefined predicates like the ones presented in Section 6. This would make it feasible for controllers and other business experts to add business rules for conformance checking without help of computers scientist, by just filling in the parameters. Thirdly, we plan to modify 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 instance OLAT for financial departments or for health care systems. Finally there are several unexplored components in the OLAT architecture, such as the Risk Interrupter, the Potential Risk detector and the 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.

A. Consistency Constraints of the Data Model

In this section, the constraints given in Section 5 are formalized. Let $\texttt{SubPeriod}(\texttt{x},\texttt{y}) := x.\text{start} \geq y.\text{start} \wedge x.\text{end} \leq y.\text{end}$, and $\texttt{InPeriod}(\texttt{x},\texttt{y}) := x.\text{start} \leq y.\text{timestamp} \leq x.\text{end}$.

A.1. Consistency Constraints for the Process Definition

p1: $\forall p_1, p_2 \in \mathit{Process} \ : \ (p_1, p_2) \in u^* \ : \ (p_1, p_2) \in u$
$\quad \vee \ (\exists p_3 \in \mathit{Process} : \ (p_1, p_3) \in u^* \ \wedge \ (p_3, p_2) \in u)$

p2: $\forall p \in \mathit{Process} : (p, p) \notin u \ \wedge \ (p, p) \notin u^*$

p3: $\forall x \in \mathit{Transition} : (h(x), f(x)) \in tp$

p4: $\forall t_1, t_2 \in \mathit{Transition} : (h(t_1) = h(t_2) \ \wedge \ f(t_1) = f(t_2)) \ \Rightarrow \ t_1 = t_2$

A.2. Consistency Constraints for the Organizational Definition

o1: $\forall r_1, r_2 \in \mathit{Role} \ : \ (r_1, r_2) \in h^* \ :$
$\quad (r_1, r_2) \in h \ \vee \ (\exists r_3 \in \mathit{Role} : \ (r_1, r_3) \in h^* \ \wedge \ (r_3, r_2) \in h)$

o2: $\forall r \in \mathit{Role} \ : \ (r, r) \in h^*$

o3: $\forall x \in \mathit{Assignment} \ : \ x.\text{start} < x.\text{end}$

o4: $\forall x \in \mathit{Permission} \ : \ x.\text{start} < x.\text{end}$

o5: $\forall x \in \mathit{Permission} \ : \ (\exists z \in \mathit{Role} \ : \ r(x) = z) \ :$
$\quad (\exists y \in \mathit{Assignment} : \ \mathit{from}(x) = a(y) \ \wedge \ (r(x), r(y)) \in h^*$
$\qquad \wedge \ p(x) = p(y) \ \wedge \ \texttt{SubPeriod}(x, y))$
$\quad \vee \ (\exists y \in \mathit{Permission} \ : \ \mathit{from}(x) = \mathit{to}(y) \ \wedge \ t(x) = t(y)$
$\qquad \wedge \ r(x) = r(y) \ \wedge \ p(x) = p(y) \ \wedge \ c(x) = c(y)$

$\wedge$ $\texttt{SubPeriod}(x, y)$)

o6: $\forall\ x\ \in\ Event\ :$

($\exists y \in Assignment\ :\ execBy(x) = a(y)\ \wedge\ \texttt{InPeriod}(y, x)$

$\wedge\ (\exists z \in Role\ :\ (z, r(y)) \in h^*\ \wedge\ (t(x), z) \in tr)$

$\wedge\ (\exists z \in Process\ :\ p(y) = z)\ \Rightarrow\ p(y) = p(c(x))\ )\ \vee$

($\exists y\ \in Permission\ \texttt{InPeriod}(y, x)$

$\wedge\ (\exists z \in Task :\ t(y) = z)\ \Rightarrow\ t(y) = t(x)$

$\wedge\ (\exists z \in Case :\ c(y) = z)\ \Rightarrow\ c(y) = c(x)$

$\wedge\ (\exists z \in Process : p(y) = z)\ \Rightarrow p(y) = p(x)$

$\wedge\ (\exists z \in Role :\ r(y) = z)\ \Rightarrow$

$(\exists z \in Role :\ (z, r(y)) \in h^*\ \wedge\ (t(x), z) \in tr)$

)

A.3. Consistency Constraints for the Runtime

r1: $\forall e_1, e_2 \in Event :\ (e_1, e_2) \in prev^*\ :\ (e_1, e_2) \in prev$

$\vee\ (\exists e_3 \in Event :\ (e_1, e_3) \in prev^*\ \wedge\ (e_3, e_2) \in prev)$

r2: $\forall (x, y) \in prev :\ x.\text{timestamp} \leq y.\text{timestamp}$

r3: $\forall a \in Association :\ t(src(a)) = src(r(a))\ \wedge\ t(tar(a)) = tar(r(a))$

r4: $\forall e \in Event :\ (t(e), p(c(e))) \in tp$

r5: $\forall u \in UpdateEvent :\ \exists f \in FormLink :\ t(e(u)) = e(f)\ \wedge\ t(p(u)) = t(f)$

r6: $\forall x \in Permission :\ (\exists y \in Process, z \in Case :$

$p(x) = y\ \wedge\ c(x) = z) :\ p(x) = p(c(x))$

r7: $\forall y \in Event, x_1, x_2 \in Assignment\ :$

$(\ c(y) = a(x_1) = a(x_2)\ \wedge\ (t(y), r(x_1)) \in tr\ \wedge\ (t(y), r(x_2)) \in tr$

$\wedge \texttt{InPeriod}(x_1, y) \wedge \texttt{InPeriod}(x_2, y)\ ) \Rightarrow x_1 = x_2$

B. Formalization of Modeling Framework

B.1. Petri Nets

A Petri net [26] is a 3-tuple $N = (P, T, F)$ where (1) P and T are two disjoint sets of *places* and *transitions* respectively; (2) $F \subseteq (P \times T) \cup (T \times P)$ is a *flow relation*. We call the elements of the set $P \cup T$ *nodes* of N, elements of F are called *arcs*. Places are depicted as circles, transitions as squares. for each element $(n_1, n_2) \in F$, an arc is drawn from n_1 to n_2. Let $N = (P, T, F)$ be a Petri net. Given a node $n \in P \cup T$, we define its *preset* ${}^\bullet n = \{n' \mid (n', n) \in F\}$, and its *postset* $n^\bullet = \{n' \mid (n, n') \in F\}$. Markings are states of a net. A *marking* m of N is defined as a function $P \to N$. A pair (N, m) is called a *marked Petri net*. A transition $t \in T$ is *enabled* in a marking m if and only if $\forall p \in {}^\bullet t : m(p) > 0$. Enabled transitions may *fire*. A transition firing results in a new marking m', denoted by $(N, m)\,[t\rangle\,(N, m')$, with $\forall p \in P : m'(p) = m(p) - \chi_F(p, t) + \chi_F(t, p)$, where χ_F is the characteristic function of F.

B.2. Data Model

A database consists of *entities*, elements or records, stored in tables. Between these entities, *associations* exist. Entities belong to an *entity type*, associations belong to a *relationship* between entity types. An Entity-Relationship diagram (ERD) [13, 25], describes the type of the entities and the relationships between them. Without loss of generality, we only consider binary relationships, since non-binary relations can be mapped onto new entities. The cardinality between a relationship r and an entity type E defines the number of associations of type r an entity of E can have. In this paper we limit the cardinality to the set of ranges $C = \{[0..*), [1..*), [0..1], [1..1]\}$.

Let Λ be a label set. An *Entity-Relationship Diagram* (ERD) S is a 4-tuple $S = (\mathcal{E}, \mathcal{A}, R, C_R)$, where (1) $\mathcal{E}$ is a set of *entity types*, (2) $\mathcal{A} : \mathcal{E} \to \mathcal{P}(\Lambda)$ defines the attribute types for each entity type, (3) $R \subseteq \mathcal{E} \times \Lambda \times \mathcal{E}$ is a set of *relationships*, for a relationship $(x, r, y) \in R$ we call x the source of r and y to target of r. and (4) $C_R \colon R \to C \times C$ defines the cardinalities for the relationships, where for a relation $r \in R$ and $C_R(r) = (x, y)$, we call x the *source cardinality* and y the *target cardinality*.

The current state of a database is called an *instance*. Let $\mathcal{I}$ denote the universe of entities and V the set of possible (attribute) values. An instance consists of entities belonging to an entity type, and associations between these entities. An instance I_S of a schema $S = (\mathcal{E}, \mathcal{A}, R, C_R)$ is a 3-tuple $I_S = (I_E, I_A, I_R)$ where (1) $I_E \colon \mathcal{E} \to \mathcal{P}(\mathcal{I})$, returns for each entity type the entities present; (2) $I_A \colon \mathcal{I} \times \Lambda \to V$ returns for each attribute the attribute values of each entity; and (3) $I_R \colon R \to \mathcal{P}(\mathcal{I} \times \mathcal{I})$ returns for each relationship the set of associations.

An instance is *consistent* if it satisfies the ERD, i.e., the ERD satisfies all the constraints in the ERD, including the cardinality constraints. This is expressed as follows. Let $S = (\mathcal{E}, \mathcal{A}, R, C_R)$ be an ERD, and $I_S = (I_E, I_A, I_R)$ be an instance of this ERD. The instance is *consistent* if:

- for all $A, B \in \mathcal{E}$ such that $A \neq B$ holds $I_E(A) \cap I_E(B) = \emptyset$;
- for all $(i, l) \in \mathrm{dom}(I_A)$ there exists an entity A such that $i \in I_E(A)$ and $l \in \mathcal{A}(A)$; and
- for each relationship $r = (A, l, B) \in R$ holds $\pi_1(I_R(r)) \subseteq I_E(A)$, $\pi_2(I_R(r)) \subseteq I_E(B)$, for all $a \in I_E(A)$ that $|\{b \mid (a, b) \in I_R(r)\}| \in \pi_1(C_R(r))$ and for all $b \in I_E(B)$ that $|\{a \mid (a, b) \in I_R(r)\}| \in \pi_2(C_R(r))$.

The constraints are denoted by standard predicate calculus with logical operators $\neg, \wedge, \vee, \Rightarrow$ and quantors $\forall, \exists$ with domains such as $\forall x \in A : \phi(x)$ where A is an entity type, a relationship or a defined set. Formulas are built in a standard way with term symbols from the data model such as entity types, relationships and attributes, variables like x, y, z and set theory operations like $f(x) \in A$, $f^{-1}(y) \subseteq B$, $g(f(x)) = h(x)$ and $f(x).a = g(y).b$, where a and b are attributes of the result of entity $f(x)$ and $g(y)$ respectively[2].

References

[1] W.M.P. van der Aalst, H.T. de Beer, and B.F. van Dongen. Process Mining and Verification of Properties: An Approach Based on Temporal Logic. In *On the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE: OTM Confederated International Conferences, CoopIS, DOA, and ODBASE*, number 3760 in lncs, pages 130–147. Springer, 2005.

[2] W.M.P. van der Aalst, B.F. van Dongen, C.W. Günther, R.S. Mans, A.K. Alves de Medeiros, A. Rozinat, V. Rubin, M. Song, H.M.W. Verbeek, and A.J.M.M. Weijters. ProM 4.0: Comprehensive Support for Real Process Analysis. In *Application and Theory of Petri Nets and Other Models of Concurrency (ICATPN 2007)*, volume 4546, pages 484–494, 2007.

[2] A function could be a partial function. If it is applied outside its domain, the value $\perp$ (bottom) is returned.

[3] W.M.P. van der Aalst and K.M. van Hee. *Workflow management: models, methods and systems*. The MIT press, Cambridge, Massachusetts, 2002.

[4] W.M.P. van der Aalst, B.F. van Dongen, J. Herbst, L. Maruster, G. Schimm, and A.J.M.M. Weijters. Workflow Mining: A Survey of Issues and Approaches. *Data and Knowledge Engineering*, 47(2):237–267, 2003.

[5] W.M.P. van der Aalst, A.J.M.M. Weijters, and L. Maruster. Workflow Mining: Discovering Process Models from Event Logs. *IEEE Transactions on Knowledge and Data Engineering*, 16(9):1128–1142, 2004.

[6] R. Agrawal, D. Gunopulos, and F. Leymann. Mining Process Models from Workflow Logs. In *Sixth International Conference on Extending Database Technology*, pages 469–483, 1998.

[7] G.-J. Ahn, Sandhu. R.S., M.H. Kang, and Park.J.S. Injecting rbac to secure a web-based workflow system. In *ACM Workshop on Role-Based Access Control*, pages 1–10, 2000.

[8] G. Antoniou, D. Billington, G. Governatori, and M.J. Maher. Representation results for defeasible logic. *ACM Trans. Comput. Logic*, 2(2):255–287, 2001.

[9] G. Antoniou, N. Dimaresis, and G. Governatori. A System for Modal and Deontic Defeasible Reasoning. In *AI 2007: Advances in Artificial Intelligence*, number 4830 in LNCS, pages 609–613. Springer, 2007.

[10] D. Berg. Turning Sarbanes-Oxley Projects into Strategic Business Processes. *Sarbanes-Oxley Compliance Journal*, November 2004.

[11] E. Bertino, E. Ferrari, and V. Atluri. The specification and enforcement of authorization constraints in workflow management systems. *ACM Trans. Inf. Syst. Secur.*, 2(1):65–104, 1999.

[12] R.A. Botha and J.H.P. Eloff. Separation of duties for access control enforcement in workflow environments. *IBM Syst. J.*, 40(3):666–682, 2001.

[13] P.P. Chen. The Entity-Relationship Model: Towards a unified view of Data. *ACM Transactions on Database Systems*, 1:9–36, Jan 1976.

[14] S. Goedertier, C. Mues, and J. Vanthienen. Specifying Process-Aware Access Control Rules in SBVR . In *Advances in Rule Interchange and Applications*, number 4824 in LNCS, pages 39–52. Springer, 2007.

[15] S. Goedertier and J. Vanthienen. Declarative Process Modeling with Business Vocabulary and Business Rules. In *On the Move to Meaningful Internet Systems 2007: OTM 2007 Workshops*, number 4805 in LNCS, pages 603–612. Springer, 2007.

[16] G. Governatori and Z. Milosevic. A formal analysis of a business contract language. *Int. J. Cooperative Inf. Syst.*, 15(4):659–685, 2006.

[17] S. Green. *Manager's Guide to the Sarbanes-Oxley Act: Improving Internal Controls to Prevent Fraud.* Wiley, 2004.

[18] D.A. Haworth and L. R Pietron. Sarbanes-Oxley: Achieving compliance by starting with ISO 17799. *Information Systems Management*, 23(1):73–87, 2006.

[19] G. Holzmann. *Spin Model Checker.* Addison Wesley, 2003.

[20] D.R. Kuhn. Mutual exclusion of roles as a means of implementing separation of duty in role-based access control systems. In *RBAC '97: Proceedings of the second ACM workshop on Role-based access control*, pages 23–30, New York, NY, USA, 1997. ACM.

[21] P.F. Linington, Z. Milosevic, J. Cole, S. Gibson, S. Kulkarni, and S. Neal. A unified behavioural model and a contract language for extended enterprise. *Data Knowl. Eng.*, 51(1):5–29, 2004.

[22] D.-R. Liu, M.-Y. Wu, and S.-T. Lee. Role-based authorizations for workflow systems in support of task-based separation of duty. *J. Syst. Softw.*, 73(3):375–387, 2004.

[23] D. Nute. Defeasible logic. *Handbook of logic in artificial intelligence and logic programming, volume 3: Nonmonotonic reasoning and uncertain reasoning*, pages 353–395, 1994.

[24] Committee of Sponsoring Organizations. Internal control - integrated framework.

[25] J. Paredaens, Paul De Bra, M. Gyssens, and D. van Gucht. *The structure of the relational database model*. Springer-Verlag New York, Inc., New York, NY, USA, 1989.

[26] W. Reisig. *Petri Nets: An Introduction*, volume 4 of *Monographs in Theoretical Computer Science: An EATCS Series*. Springer-Verlag, Berlin, 1985.

[27] A. Rozinat and W.M.P. van der Aalst. Conformance Testing: Measuring the Fit and Appropriateness of Event Logs and Process Models.

In *BPM 2005 Workshops (Workshop on Business Process Intelligence)*, volume 3812, pages 163–176, 2006.

[28] A. Rozinat and W.M.P. van der Aalst. Conformance Checking of Processes Based on Monitoring Real Behavior. *Information Systems*, 33(1):64–95, 2008.

[29] S. Sadiq, G. Governatori, and K. Namiri. Modeling control objectives for business process compliance. In *Business Process Management*, number 4714 in lncs, pages 149–164. Springer, 2007.

[30] R. Sandhu, E. Coyne, H. Feinstein, and C. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, 1996.

[31] R.T. Simon and M.E. Zurko. Separation of duty in role-based environments. *Computer Security Foundations Workshop, 1997. Proceedings., 10th*, pages 183–194, Jun 1997.

[32] J. Wainer, A. Kumar, and P. Barthelmess. DW-RBAC: A formal security model of delegation and revocation in workflow systems. *Inf. Syst.*, 32(3):365–384, 2007.