FORMAL ANALYSIS OF EXECUTIONS OF ORGANIZATIONAL SCENARIOS BASED ON PROCESS-ORIENTED MODELS

Viara Popova and Alexei Sharpanskykh
Vrije Universiteit Amsterdam
Department of Artificial Intelligence
De Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands
E-mail: {popova, sharp}@few.vu.nl

KEYWORDS
Process-oriented models, Formal analysis, Workflow Management

ABSTRACT

This paper presents formal techniques for analysis of executions of organizational scenarios based on process-oriented models of organizations. A part of these techniques is dedicated to establishing the correspondence between formalized executions (i.e., traces) and process-oriented models. Other techniques provide the analyst with wide possibilities to analyze organizational dynamics and to evaluate organizational performance. For the proposed formal analysis the order-sorted predicate Temporal Trace Language (TTL) is used. The analysis is supported by the dedicated software tool TTL Checker. The analysis approaches are illustrated by a case study in the context of an organization from the security domain.

INTRODUCTION

Process management in many modern organizations is supported by dedicated software systems, such as Workflow Management Systems (WfMS). WfMSs are used to guide/control the execution of organizational scenarios based on certain internal models. These models describe/prescribe ordering and timing relations on processes, modes of use of resources, allocations of actors to processes etc. WfMS models are expressed using different formalisms: Petri-Nets, Workflow Nets, process algebra, logical specifications. An approach proposed in this paper makes use of models specified in an expressive order-sorted predicate language L_{PR} described in (Popova and Sharpanskykh 2006). The actual execution of organizational scenarios may diverge from the dynamics (pre)defined by a process-oriented model. To capture this difference many WfMSs record data about actual executions (e.g., starting and finishing time points of processes, types and amounts of resources used/consumed/produced/broken, names of actors who perform processes).

To guarantee the correct operation of an organization supported by a WfMS (1) a correct formal process-oriented model should be provided and (2) actual executions of organizational scenarios should correspond to this formal model. For establishing the correctness of process-oriented (or workflow) models a number of formal verification techniques exist (e.g., Aalst and Hee 2002) aimed at identifying errors and inconsistencies in models, irrespectively of actual executions of these models. The verification techniques related to models used in this paper are described in (Popova and Sharpanskykh 2006). However, not many formal techniques and tools exist for establishing if the organization actually behaves as it is specified by the model (i.e., for validating a model). In (Barjis et al. 2002; Desel et al. 2003) validation is performed by simulation of organizational scenarios. Although simulation techniques can provide useful insights into relationships and dynamics of an organization, they often abstract from the complexity of dynamics of real organizations. To perform analysis based on the actual organizational execution, data gathered by a WfMS can be used. For example, in (Aalst et al. 2005) it is shown how the analysis based on linear temporal logic (LTL) can be used for establishing the correspondence between the observed and the expected organizational behavior. In this paper different types of formal, automatically supported analysis of actual executions based on process-oriented models will be described. These types include checking the conformity to a formal process-oriented model and to the formal organization, analysis of organizational emergent properties and organizational performance evaluation. The analysis is based on the predicate-based Temporal Trace Language (TTL), which allows more expressivity than LTL used in (Aalst et al 2005).

The presentation is organized as follows. First, the overview of the proposed analysis framework is given. Then, the specification of process-oriented models is briefly discussed and a language used for formalizing executions is introduced. Next, TTL and the dedicated software environment TTL Checker are considered. Finally, different types of trace-based analysis are discussed and illustrated by a case study from the security domain. The paper concludes with a discussion.

TRACE-BASED ANALYSIS: OVERVIEW

In (Popova and Sharpanskykh 2007a) a general organization modeling and analysis framework is introduced including different views on organizations. In particular, the performance-oriented view describes organizational goal structures, performance indicators structures, and relations between them. Within the organization-oriented view organizational roles, their authority, responsibility and power relations are defined. In the agent-oriented view different types of agents with their capabilities are identified and principles for allocating agents to roles are formulated. Finally, process-oriented view describes static structures of tasks and resources, the flow of control, and addresses the actual execution of organization processes. The views are related to each other by means of common concepts, which enables different types of analysis across views. This paper describes a part of the process-oriented view related to actual execution of organizational scenarios based on process-oriented models formalized in L_{PR}. Data about actual executions are structured in the form of a *trace* - a formal structure that consists of a time-indexed sequence of states. Each state is characterized by a set of organizational and environmental events that occur in the state. Events are specified by atoms in a sorted predicate language L_{EX}, described in this paper. The formal analysis of actual executions is performed by checking organizational properties expressed in TTL on traces using the TTL Checker tool. The TTL Checker has a graphical interface, using which TTL formulae can be inputted and traces that represent organization executions can be loaded and visualized (see for example Fig.1). The tool generates a positive answer, if the specified property is satisfied by the execution model (i.e., holds w.r.t. the loaded trace(s)). If a formula is not satisfied, a counterexample is provided. The tool also allows performing statistical analysis on multiple traces. More details on the TTL and the tool are given further in the paper. Here we identify the types of trace analysis that can be performed using the TTL Checker.

Each process-oriented model (pre)defines a set of scenarios of organization behavior. The actual execution of an organization may diverge from scenarios described by the model. In some organizations a certain degree of deviation is allowed, whereas other organizations require a strict adherence to the model (e.g., military organizations, nuclear power plants). In the second case the verification of the conformity of an actual execution to a formal organization model is of special importance. This is the first type of analysis considered in this paper.

Every correct process-oriented model guarantees the satisfaction of a set of (global) constraints over processes, resources and agents identified in the organization. These constraints are usually specified based on different organizational and general normative documents (e.g., a strategy description, laws, policies, etc.). In general, if a trace conforms to the corresponding process-oriented model, then all constraints imposed on and satisfied by the model are also satisfied by the trace. However, when the checking of the conformity of the trace to the model fails, then the satisfaction of the constraints by the trace is not guaranteed any more. In this case the analysis of the conformity of a trace to a formal organization (i.e., organizational constraints) should be performed, which is the second type of analysis considered in this paper. Often process-oriented models allow (different degrees of) autonomy of agents in executing organizational scenarios. For example, in many organic organizations processes are defined loosely to ensure flexibility. To analyze the functioning of such organizations, an approach called analysis of the emergent organizational behavior is proposed in the paper.

Finally, the paper proposes a method for the evaluation of organizational performance based on checking the satisfaction of organizational goals related to processes. The types of analysis described above may be performed both during the execution and after the execution of organizational scenarios.

THE SPECIFICATION OF THE PROCESS-ORIENTED MODEL

Process-oriented models are expressed using the L_{PR} language, which is briefly described in this section. For more details see (Popova and Sharpanskykh 2006). The model describes the following objects (represented by sorts in L_{PR}): *tasks*, *processes* (particular instances of tasks in control flows), *resource types* describing information and material artifacts, *resources* (specific instances of resource types having specified amounts), *agents*, *roles* (sets of functionalities that can be assigned to agents), *goals*, *performance indicators* (measures based on which the goals are defined). Each object has a number of characteristics. For example, a task is characterized by a minimum duration (denoted by task_name.min_duration); a resource type has a characteristic expiration duration; resources are characterized by an amount. Furthermore, relations are defined over the objects. For example, the relation task_produces(t:TASK, rt:RESOURCE_TYPE, v:VALUE) specifies that task t produces amount v of resource type rt. Resource types that can be shared by several processes are specified in resource_sharable(rt:RESOURCE_TYPE, L:PROCESS_LIST).

The set of specified processes together with the set of ordering relation defined on them form a workflow. An example of an ordering relation is starts_after(p1: PROCESS, p2:PROCESS). It defines that process p1 starts after process p2. Furthermore, three types of structures specifying the flow of control between processes are defined: and-, or- and loop-structures. Branches of and-structures start simultaneously and are all executed. Only one branch of an or-structure can be executed depending on the or-condition. Loop structures contain processes that can be repeated depending on the loop-condition within a maximum number of iterations. Relations between roles, agents and processes are defined as follows: role_perfoms_process(r:ROLE, p:PROCESS) and agent_plays_role(a:AGENT, r:ROLE).

Relations to goals and PIs are defined as follows: is_realized_by(g:GOAL, L:TASK_LIST) defining that goal g can be realized by performing tasks in list L and measures(i:PI, p:PROCESS) specifying that performance indicator i is a measure over some aspect of the performance of process p.

EXECUTION LANGUAGE L_{EX}

For the formalization of a trace, a dedicated sorted predicate language L_{EX} is used, which is based on L_{PR}. Each sort included into L_{EX} represents a set of individual objects of a certain type that occur in the trace (e.g., the sort PROCESS_EX contains all names of processes that have been executed in the trace). To distinguish the names of sorts of L_{EX} from the names of sorts in L_{PR}, all sort names of L_{EX} finish with the EX postfix. To define events a number of relations are introduced into L_{EX} (see Table 1).

Table 1: Relations defined in L_{EX}

Predicate specification	Informal description
process_started: PROCESS_EX	A process has started
process_finished: PROCESS_EX	A process has finished
resource_used_by: RESOURCE_EX x PROCESS_LIST_EX x VALUE	A certain resource amount is used by a process
resource_consumed_by: RESOURCE_EX x PROCESS_EX x VALUE	A certain resource amount is consumed by a process
resource_produced_by: RESOURCE_EX x PROCESS_EX x VALUE	A certain resource amount is produced by a process
resource: RESOURCE_EX x RESOURCE_TYPE_EX	Identifies a resource of a certain resource type
resource_expired: RESOURCE_EX	A resource is expired
resource_invalid: RESOURCE_EX x VALUE	A certain resource amount became invalid (e.g. broken)
available_resource_amount: RESOURCE_EX x VALUE	Specifies the available amount of the resource
pi_has_value: PI_EX x VALUE	Identifies the value of a PI
agent_is_assigned_to_role: AGENT_EX x ROLE_EX	Specifies the assignment of an agent to a role
agent_performs_process: AGENT_EX x PROCESS_EX	Identifies that an agent performs a certain process
env_object_changed_state_into: ENV_OBJECT_EX x OBJ_STATE_EX	Specifies a changed state of an environmental object
env_object_changed_char_into: ENV_OBJECT_EX x OBJ_CHAR_EX x VALUE	Specifies the value of a certain characteristic of an environmental object
decision_taken: DECISION_VARIABLE_EX x DECISION_VAR_VALUE_EX	Identifies the value of a decision variable

LANGUAGE TTL AND TTL CHECKER TOOL

To analyze traces the language TTL is used. TTL is a variant of order-sorted predicate logic, which allows reasoning about dynamic properties of systems. TTL properties considered in this paper are specified based on state properties expressed as formulae in L_{EX}. For enabling dynamic reasoning, TTL includes special sorts: TIME (a set of linearly ordered time points), STATE (the set of all state names of a system), TRACE (the set of all trace names), STATPROP (the set of all state property names). In TTL, formulae of the state language (L_{EX} in this case) are used as objects. Further we shall use t with subscripts and superscripts for variables of the sort TIME; and γ with subscripts and superscripts for variables of the sort TRACE. A state of a system in a trace is denoted using a function symbol state of type TRACE x TIME $\rightarrow$ STATE. The set of function symbols of TTL includes:

$\wedge, \vee, \rightarrow, \leftrightarrow$: STATPROP x STATPROP$\rightarrow$ STATPROP,
not: STATPROP$\rightarrow$ STATPROP,
$\forall, \exists$: VARS x STATPROP$\rightarrow$ STATPROP,
which are counterparts to the Boolean propositional connectives and quantifiers.

The states of a system are related to names of state properties via the satisfaction relation denoted by the infix predicate |= (or by the prefix predicate holds): state(γ,t)|= p (or holds(state(γ,t)), which denotes that the state property with a name p holds in trace γ at time point t. For example, state(trace1,10)|= process_started(p2) denotes that the process p2 has started in the trace1 at the time point 10. Both state(γ,t) and p are terms of TTL. All other TTL terms are constructed by induction in the standard predicate logic way.

Transition relations between states are described by dynamic properties, which are expressed by TTL-formulae. The set of *atomic TTL-formulae* is defined as:

(1) If v_1 is a term of sort STATE, and u_1 is a term of the sort STATPROP, then holds(v_1,u_1) is an atomic TTL formula.
(2) If τ_1, τ_2 are terms of any TTL sort, then τ_1=τ_2 is an atomic TTL formula.
(3) If t_1, t_2 are terms of sort TIME, then t_1<t_2 is an atomic TTL formula.

The set of *well-formed TTL-formulae* is defined inductively in a standard way using Boolean propositional connectives and quantifiers. TTL has semantics of the order-sorted predicate logic. A more detailed specification of the syntax and the semantics for the TTL is given in (Sharpanskykh and Treur 2006).

The analysis based on checking of TTL formulae on (one or more) traces is supported by the TTL Checker tool. Besides the logical analysis the tool allows statistical post-processing of the verification results. For this the following functions are used:

case(logical_formula, value1, value2): if logical_formula is true, then the case function is mapped to value1, otherwise – to value2.

sum([summation_variables], case(logical_formula, value1, 0)): logical_formula is evaluated for every combination of values from the domains of each from the summation_variables; and for every evaluation when the logical formula is evaluated to true, value1 is added to the resulting value of the sum function.

To provide support for analysts not skilled in logics, the tool allows defining parameterized templates (macros), which can be instantiated in different ways. Further details about the TTL Checker can be found in (Bosse et al. 2006). Examples of analysis cases that also include statistical processing will be given further in this paper.

TRACE CONFORMITY TO A MODEL

As described earlier the process-oriented model consists of objects, characteristics and relations defined in L_{PR}. Every such model can be translated to a set of constraints that should be satisfied by actual execution traces. The constraints are represented as properties in TTL using L_{EX} as a state language. Each property is based on a specific combination of language constructs (ordering relations, and-/or-/loop-structures, object characteristics, etc.) In the following we define rules on how to translate different parts of the model specification to TTL properties. Due to the space limitations only a part of the properties is given in this paper, for the rest of them we refer to (Popova and Sharpanskykh 2007b).

The first property we consider represents the restriction that only processes specified in the model are allowed to be performed. It is formalized in TTL as follows. For specific process names p1, ..., pn:

C1: $\forall$t, p:PROCESS_EX state(γ, t) |= process_started(p) $\Rightarrow$ p = p1 | ... | p = pn

The next properties represent the constraints that processes not part of any or-structure start and finish in the trace. For p1 a process not in any or-branch:

C2: $\exists$t1 state(γ, t1) |= process_started(p1)
C3: $\exists$t1 state(γ, t1) |= process_finished(p2)

The execution of processes in or- and loop-structures depends on the evaluation of conditions defined for these structures. In this case it needs to be checked whether the processes that have started also finish in the trace: For p1 a process in a loop-structure/or-branch:

C4: $\exists$t1 state(γ, t1) |= process_started(p1)
$\Rightarrow$ $\exists$t2: state(γ, t2) |= process_finished(p1)

Additionally for processes not in loop-structures:
C5: $\exists$t1 state(γ, t1) |= process_started(p1)
$\Rightarrow$ ($\forall$t3 t3 $\neq$ t1 $\Rightarrow$ state(γ, t3) |= $\neg$process_started(p1)

The next property checks if the actual duration of a process is within the range defined by the corresponding task. For a process p1, a task tk, durations d1 and d2 such that [is_instance_of(p, tk), tk.min_duration=d1, tk.max_duration=d2]:

C6: $\exists$t1, t2 state(γ, t1) |= process_started(p1) & state(γ, t2) |= process_finished(p2) $\Rightarrow$ d1 $\leq$ t2-t1 & t2-t1 $\leq$ d2

Ordering relations are translated to constraints in the following way. For p1, p2 such that starts_with(p1, p2):

C7: $\exists$t1 state(γ, t1) |= process_started(p1)
$\Rightarrow$ state(γ, t1) |= process_started(p2)
C8: $\exists$t1 state(γ, t1) |= process_started(p2)
$\Rightarrow$ state(γ, t1) |= process_started(p1)

Similarly for finishes_with and starts_during (C9, C10, C11). For p1, p2, d such that starts_after(p2, p1, d) except for beginning and ending of and-, or-, or loop-structures:

C12: $\exists$t1 state(γ, t1) |= process_finished(p1)
$\Rightarrow$ $\exists$t2: state(γ, t2) |= process_started(p2) & d = t2-t1

For and-structures it is checked if the order of execution of processes in these structures matches the specified and-conditions (C13, C14, C15). An and-condition designates all, any or specific processes at the end of the branches of an and-structure that should finish before the workflow can continue.

For or-structures it should be checked if exactly one of the branches is executed and it matches the specified or-condition. An or-condition is an expression based on a decision variable (related to a decision process), state or a characteristic of an environmental object. For p, p_1,..., p_n, d, and a condition based on the decision variable dv such that [starts_after(begin_or(id),p,d),starts_after(p_1, begin_or(id)), ..., starts_after(p_n, begin_or(id)), or_cond(id, dv), or_branch(p_1, val_1),..., or_branch(p_n, val_n)] (similarly for other conditions):

C16: $\exists$t1 state(γ, t1) |= process_finished(p) $\Rightarrow$ $\exists$t2 (state(γ, t2) |= process_started(p_1) & $\forall$t3 state(γ, t3) |= [$\neg$process_started(p_2) $\wedge$ ... $\wedge$ $\neg$process_started(p_n)] & $\exists$t4 state(γ, t4) |= decision_taken(dv, val_1) & t4 $\leq$ t2 & ($\forall$t5 t5 $\geq$ t4 & t5 $\leq$ t2 & state(γ, t5) |= decision_taken(dv, val) $\Rightarrow$ val = val_1) | ... | (state(γ, t2) |= process_started(p_n) & $\forall$t6 state(γ, t6) |= [$\neg$process_started(p_1) $\wedge$ ... $\wedge$ $\neg$process_started(p_{n-1})] & $\exists$t7 state(γ, t7) |= decision_taken(dv, val_n) & t7 $\leq$ t2 & ($\forall$t8 t8 $\geq$ t7 & t8 $\leq$ t2 & state(γ, t8) |= decision_taken(dv, val) $\Rightarrow$ val = val_n)) & d = t2-t1

Furthermore it should be checked that the processes in the other branches are not executed (C17) and that the process after the or-structure starts correctly:

For p_1, ..., p_n, p, d such that [starts_after(end_or(id), p_1), ..., starts_after(end_or(id), p_n), starts_after(p, end_or(id), d)]:
C18: $\exists$t1 state(γ, t1) |= [process_finished(p_1) $\vee$ ... $\vee$ process_finished(p_n)] $\Rightarrow$ $\exists$t2 state(γ, t2) |= process_started(p) & d = t2-t1

For every loop-structure the correct execution order is checked w.r.t. a loop condition and a maximal number of iterations (C19).

The following properties concern resources and resource types and how they are used/consumed/produced/shared by processes. For resource type rt, task tk, amount v and process p such that [is_instance_of(p, tk), task_uses(tk, rt, v)] for every time point t in the trace it will be checked that the resource that is used matches the specification:

C21: sum([L:PROCESS_LIST_EX], case(∃t1, t2 state(γ, t1) |= process_started(p) & state(γ, t2) |= process_finished(p) & t1 ≤ t & t ≤ t2 & state(γ, t) |= resource_used_by(r, L, v1) & is_in_list(p, L) & ∃t4 state(γ, t4) |= resource(r, rt), v1, 0)) = v

Similarly, the properties C20 and C22 are defined for consumed / produced resources.

In the model, the resources available at the beginning of the workflow are represented as produced by the BEGIN process. Thus it should be checked if the available amount at the beginning of the trace matches the amount produced by the BEGIN process. For resource r such that [process_output(BEGIN, r), is_resource_type(r, rt), r.amount=v]:

C23: sum([r:RESOURCE_EX], case(state(γ, 0) |= [available_resource_amount(r, v1) ∧ resource(r, rt)], v1, 0)) = v

It should also be checked whether the resources are shared between lists of processes for which this is allowed. For resource type rt and list of processes L such that [resource_sharable(rt, L)]:

C24: ∃t1 ∃L1:PROCESS_LIST_EX state(γ, t1) |= resource_used_by(r, L1, v) & ∃t2 state(γ, t2) |= resource(r, rt) ⇒ is_sublist_of(L1, L)

Finally it should be checked if role/process assignments to agents are correct. For role r, agent a, process p such that [role_performs_process(r,p),agent_plays_role(a, r)]:

C25: ∃t1, t2 state(γ, t1) |= process_started(p) & state(γ, t2) |= process_finished(p) ⇒ ∀t3 t1 ≤ t3 & t3 ≤ t2 & state(γ, t3) |= [agent_performs_role(a, r) ∧ agent_performs_process(a, p)]

The above listed properties are general and can be checked in any order on the execution trace. However in many cases it would be beneficial to enforce certain order of checking. Often when one constraint is violated that causes the violation of others but finding all of them might not add much more information on what went wrong. It is therefore useful to alert the analyst of the first time point at which a violation of a constraint occurs. The approach proposed here is to consider the events of the trace in their natural temporal order. For each event that represents a starting or finishing point of a process only a selection of the relevant general constraints instantiated for a specific time point(s) and a specific event(s) are checked.

In the following we define the sets of relevant constraints w.r.t. the type of event occurring in the trace. The first constraints to be checked are C23 (available resource at the first time point) and C2 (checks if a process starts) for the first process(es) in the workflow that should start at the first time point unconditionally. If at the first time point an or-structure begins then it should be checked that only one branch is executed and it matches the evaluation of the condition (C16). Afterwards the (partially) ordered list of starting and finishing points of processes is considered. For every starting point the following types of constraints are considered (in this order): (1) the process is defined in the model (C1), (2) the process has not been executed before if not in loop-structures (C5), (3) constraints w.r.t. the conditions for and-structures (C13, C14, C15), (4) constraints related to starts_with and starts_during (C7, C8, C11), (5) the process finishes (C3, C4).

For every finishing point the following constraints are checked (in this order): (1) resource-related constraints (C20, C21, C22, C24), (2) agent-/role-related constraints (C25), (3) durations (C6), (4) constraints related to finishes_with (C9, C10), (5) constraints on the next process (C12, C16, C17, C18, C19). From all types of considered constraints those are selected that refer to the specific process to which the starting or finishing point belongs. When more events coincide finishing points are considered before starting points.

The above described approach assumes the availability of the whole execution trace at the beginning of the analysis. In some situations it might be necessary to perform such analysis while the trace is being generated. This gives the possibility to react as soon as an event in the execution deviates from the model and take appropriate measures. With some adjustments, the generic properties can be used here as well, as described in (Popova and Sharpanskykh 2007b).

CONFORMITY TO A FORMAL ORGANIZATION

A formal organization is specified by a fixed set of rules that define (prescribe) organizational structure and behavior and are formalized as predicate logic constraints imposed on a process-oriented model.

In (Popova and Sharpanskykh 2006) different types of constraints are described (e.g., domain-specific, physical world constraints). Some of these constraints are strict and should not be violated in any organizational scenario; e.g., “all employees involved in a certain process, which has a risk factor for human health, should be provided with the necessary safety means”. Other rules are less strict and can be (temporally) violated; e.g., “the average amount of a certain resource produced by an organization is required to be greater than a certain number”.

In the following several examples of formal organization properties that can be checked on traces are considered.

P1: In the trace γ1 the process p1 is executed (after some time) after the process p2 has finished:
∃t1, t2 t1≤t2 state(γ, t1) |= process_finished(p2) & state(γ, t2) |= process_started(p1)

P2: For the specified set of traces TR the average overall amount of resources of type r produced by an organization up to a time point t should be at least n:
sum([γ:TR, t':between(0, t), r':RESOURCE_EX], case(∃a':PROCESS_EX ∃am:VALUE_EX state(γ, t')|= [

resource_produced_by(r', a', am) ∧ resource(r', r)], am, 0)) / sum([γ:TR], case(true, 1, 0)) ≥ n,
here between(0, t) represents a set of all natural numbers in the interval [0, t].

P3: In the trace γ1 the overall amount of working hours of an agent a at time point t (e.g., a time point in the end of some working period) should not exceed n:
(sum([t': between(0, t), p':PROCESS_EX], case(state(γ1, t')|= [agent_performs_process(a, p') ∧ process_finished(p')], t', 0)) – sum([t'': between(0, t), p': PROCESS_EX], case(state(γ1, t')|= [agent_performs_process(a,p') ∧ process_started(p')], t'',0))) ≤ n

ANALYSIS OF EMERGENT PROPERTIES

Emergent properties are not specified and not implied by an organizational model and are related only to (result from) an actual execution(s) of an organization. Such properties may be checked for different reasons: e.g., to optimize the organizational operation by discovering and eliminating bottlenecks. Many emergent properties include a post-processing of the checking results by applying different statistical functions: e.g., sum, average, minimum, maximum, and are often expressed over multiple traces. Consider several examples:

E1: For the specified set of traces TR, determine a frequency of finishing the process p on time (i.e., duration should be within the interval [min_duration, max_duration]).
sum([γ:TR], case(∃t1,t2 state(γ, t1)|= process_started(p) & state(γ, t2)|= process_finished(p) & (t2-t1) ≤ max_duration & (t2-t1) ≥ min_duration], 1, 0)) / sum([γ:TR], case(∃t1 state(γ, t1)|= process_started(p), 1, 0))

E2: In the trace γ1 at the time point t calculate the average workload of agents of an organization:
(sum([t1: between(0, t), p':PROCESS_EX, a':AGENT_EX], case(state(γ1, t1) |= [agent_performs_process(a', p') ∧ process_finished(p')], t1, 0) – sum([t2: between(0, t), p':PROCESS_EX, a':AGENT_EX], case(state(γ1, t2)|= [agent_performs_process(a', p') ∧ process_started(p')], t2, 0))) / sum([a':AGENT_EX], case(true, 1, 0))

E3: Maximum duration of a process p in all executions:
∃γ1, t1, t2 state(γ1, t1)|= process_started(p) & state(γ1, t2)|= process_finished(p) & ∀γ'≠γ1 ∀t1', t2' [state(γ', t1') |= process_started(p) & state(γ', t2')|= process_finished(p) & (t2'-t1')<(t2-t1)]

PERFORMANCE EVALUATION

The performance of an organization at a certain time point (for a certain period) is evaluated by determining the satisfaction of key organizational goals. These goals range from high-level abstract goals to very specific ones. High-level goals are decomposed to more specific goals which are easier to measure, thus, forming goal decomposition structures. Goals are defined and discussed in (Popova and Sharpanskykh 2006) as part of the performance-oriented view on organizations. Example of goals are: 'It is desired to maintain high degree of product quality', 'It is desired to achieve high customer satisfaction', 'It is desired to maintain number of work-related accidents per year to less than 3', etc.

Goals are formulated based on performance indicators (PIs), which are associated with certain organizational processes. Examples of PIs are: product quality, customer satisfaction, number of accidents, productivity, etc. The values of these PIs are measured (directly or indirectly) during or after the process execution depending on the goal evaluation type and in the end or during a certain period of time (goal horizon). Then, by comparing the measured values with the corresponding goal expressions, the satisfaction of the goals is determined. Further, the obtained goal satisfaction measure is propagated by applying the rules defined in (Popova and Sharpanskykh 2006), upwards in the goal hierarchy for determining the satisfaction of high level goals. An example of this type of analysis is given further in the frames of the case study.

CASE STUDY

The application of different types of analysis will be illustrated in the context of an organization from the security domain. The main purpose of the organization is to deliver security services to different types of customers. The organization has well-defined multi-level structure that comprises several areas serving groups of locations (security objects) and has predefined (to a varying degree) job descriptions for employees (approx. 230.000 persons). The allocation of employees to security objects is based on plans created by planning groups.

The planning process consists of the forward (or long-term) planning and the short-term planning. The forward planning is a process of creation of plans describing the allocation of security officers within the whole organization for a long term (4 weeks). Forward plans are created based on customer contracts by forward planners. During the short-term planning, plans that describe the allocation of security officers to locations within an area for a short term (a week) are created and updated based on the forward plan and up-to-date information about the security employees. Based on short term plans, daily plans are created. Within each area the short-term planning is performed by the area planning team that consists of planners and is guided by a team leader.

The position of the forward planners in the organizational structure has changed as a result of a reorganization in the past. Before the reorganization each planning team had a forward planner who was mainly responsible for the creation of long-term plans for the area. After the reorganization the forward planners were combined into a centralized forward planning group. A number of reasons for such a change are identified in the reorganization reports. In the following it will be shown how the proposed analysis techniques could be used for automated justification of the identified performance bottlenecks and other problems in the organization.

(1) Uneven workload of forward planners in different area planning teams.
This statement can be checked by calculating the workload for the forward planners in different areas and comparing the results. For this the following property can be used with a – the agent name, for whom the workload is calculated, and t – the time point up to which the workload is calculated:

sum([t1: between(0, t), p':PROCESS_EX], case(state(γ1, t1) |= [agent_performs_process(a, p') ∧ process_finished(p')], t1, 0)) - sum([t2: between(0, t), p':PROCESS_EX], case(state(γ1, t2)|= [agent_performs_process(a, p') ∧ process_started(p')], t2, 0)),
here a is an agent name and

If multiple traces are available, the average workload of every agent can be calculated as it is demonstrated in property E2. A side-effect of high workload could be the undue execution of some processes assigned to the forward planner. This can be established by verifying the correspondence of the actual execution to the model.

(2) Certain forward planning tasks require collaboration with other forward planners. In the previous organization this has been achieved by informal (i.e., not specified by a formal organizational model) cooperation between forward planners from different areas.

This statement can be justified in two steps. First by performing the analysis of the correspondence of a trace to the model, it can be established that in the trace exist processes performed by agents that are not allocated to the roles, to which these processes are assigned. Then, the number (or frequency) of such processes until the time point t for each role r can be calculated as follows:

sum([p':PROCESS_EX], case(∃t1<t ∃a:F_PLANNER
state(γ1, t1) |= [agent_performs_process(a,p') ∧
¬agent_performs_role(a1, r)], 1, 0))

For multiple traces (a set TR), the average number of such processes for role r can be calculated as follows:
sum([γ:TR, p':PROCESS_EX], case(∃t1<t ∃a:F_PLANNER state(γ, t1) |= [agent_performs_process(a, p') ∧ ¬agent_performs_role(a1, r)], 1, 0)/sum([γ:TR], case(true, 1, 0))

(3) Planning activities within each area were isolated from each other. Sometimes this led to situations, when customer requests in one area were not satisfied due to lack of security officers, whereas in other areas available employees were in plenty.

Such situations could be identified by calculating the (average) number of customer requests that were not accomplished by the organization until the time point t:
sum([t1: between(0, t)), r': CUSTOMER_REQUEST], case(state(γ1, t1) |= env_object_changed_state_into(r', active) & ∀t2 t2>t1 state(γ1, t2) |= ¬env_object_changed_state_into(r', satisfied), 1, 0))

In the following section we illustrate in more detail the different types of analysis of execution traces using the activities of the short-term planners after the reorganization of the planning departments.

EXAMPLES OF TRACE ANALYSIS

Based on company documents such as job descriptions, company policy, procedures, etc., a process-oriented model was created for the planning departments. Part of this model dedicated to the creation of daily plans and short-term plans within one day is considered here. In the first half of the day security employees should provide their data change forms (requests for changes in the allocation schedule) to the unit manager (defined as process p3) who then checks and improves the data (p4) and puts it in the system (p5). At the same time the planners are working on other tasks, for example during the last week of the month they create a new short-term plan (STP) for the next month (p1). In the second half of the day they work on creating a daily plan (p6) for the next day (using the data change information in the system), inputting it in the system (p7) and informing all concerned (p8). Then they update the current short-term plan if necessary (p9) and so on. Part of the specification of the model is shown below:

```
starts_after(begin_and(and1), BEGIN, 0)
starts_after(begin_or(or1)
begin_and(and1), 0)
starts_after(p3, begin_and(and1), 0)
starts_after(p4, p3, 0)
starts_after(p5, p4, 0)
starts_after(p2, begin_or(or1), 0)
or_cond(or1,week_state)
or_branch(last,p1)
or_branch(other,p2)
starts_after(end_or(or1), p1, 0)
starts_after(end_or(or1), p2, 0)
starts_after(begin_and(and1), p5, 0)
starts_after(begin_and(and1), end_or(or1), 0)
and_cond(and1, all)
starts_after(p6, end_and(and1), 0.5)
...
role_performs_process(sec_officer, p3)
role_performs_process(planner, p1)
...
is_instance_of(p1, t1)
task_produces(t1, STP, 1)
t1. min_duration = 3.5h
t1.max_duration = 4h
...
```

Based on this specification constraints are generated (as discussed earlier). For example, the first few lines of the specification generate the following constraints for the first time point of an execution trace:

state(γ, 0) |= process_started(p3) (based on C2)
state(γ, 0) |= process_started(p2) & (∀t3 state(γ, t3) |= ¬process_started(p1)) & state(γ, 0) |= ¬env_object_changed_state_into(week, last) | (state(γ, 0) |= process_started(p1) & (∀t3 state(γ, t3) |= ¬process_started(p2)) & state(γ, 0) |= env_object_changed_state_into(week, last) (based on C17)
∀p:PROCESS_EX state(γ, 0) |= process_started(p) ⇒ p = p1 | p = p2 | p = p3 (based on C1)

Also based on company documents traces were created corresponding to this model. One such trace is used to illustrate the analysis of whether an execution trace agrees with the model. The trace represents a day from the last week of the month. Part of this trace is shown in Fig. 1. In the left part the atoms are listed and in the right part the time line is shown consisting of 12 hours. The time line is relative to the trace and not expressed in absolute date and time stamps. The absolute time line can always be calculated given the time stamp of the beginning of the trace. For each atom, the time interval for which it is true is displayed by a dark-grey bar while a light-grey bar designates that the value is false. For example for the whole duration of the trace agent a1 is assigned to play the role of a security officer and process_started(p1) is only true for time point 0.

The trace in Fig. 1 contains a process that is not in the model, p12. It is executed instead of process p3. According to p3, the security officers should deliver the change forms to the unit manager however on that day the unit manager was unavailable and the forms were brought directly to the planners (p12) who then had to check and improve them and input them in the system. These extra tasks prevented the planners from finishing their work on creating a short-term plan on time. Therefore all other processes during the rest of the day were shifted later than the model specified.

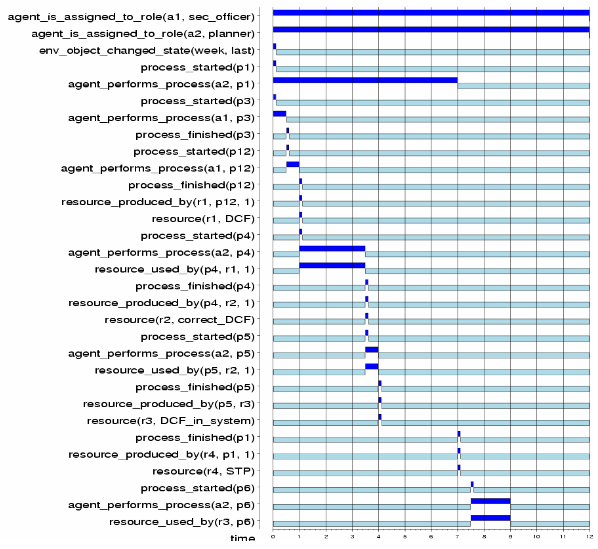

Figure 1: The execution trace used for illustration

The trace is considered time point by time point taking into account the starting and finishing points of processes. We assume that the analysis is performed in real time, i.e. only the part of the trace up to the current time point is available. At time point 0 the three constraints given above are checked. They are satisfied since the only two processes starting are p3 and p1 and at this time point the state of the object week is indeed 'last'. Next the following properties are scheduled to be checked at every time point t until satisfied:

state(γ, t) |= process_finished(p1)

state(γ, t) |= process_finished(p3)

If that does not happen before the end of the trace then it is considered that this constraint is violated. Also the minimal and maximal duration of the processes should be according to the model:

state(γ, t) |= process_finished(p1) ⇒ t ≥ 3.5
state(γ, t) |= process_finished(p1) ⇒ t ≤ 4
state(γ, t) |= process_finished(p3) ⇒ t = 1

Next resource-related constraints are considered. The only relevant resource is the collection of data change forms DCF which is considered as a whole and only one collection can be produced. Thus C22 is not relevant.
Also agent-/role-related constraint C25 is scheduled for checking at every time point t until the process finishes.

state(γ,t) |= ¬process_finished(p1)
⇒ state(γ,t) |= [agent_plays_role(a2,planner) ∧
agent_performs_process(a2, p1)]
state(γ,t) |= ¬process_finished(p3)
⇒ state(γ,t) |= [agent_plays_role(a1,sec_officer) ∧
agent_performs_process(a1, p3)]

From all the scheduled constraints one fails at time point 0.5 when process p3 finishes – its duration is below the specified minimal duration of 1 hour. At this step the analysis stops – the trace does not agree with the model and the first process that violates the constraints is p3. Then, at this point it can be checked whether and which important organizational properties are satisfied (i.e., conformity to the formal organization). One of the properties extracted from the organizational documents of the company is that a daily plan for the next day is available before the end of the current working day, expressed as follows:

∃t, p:PROCESS_EX, r:RESOURCE_EX
state(γ,t) |= [resource_produced_by(r, p) ∧
resource(r, daily_plan)]
This property is satisfied by the trace.

Another property says that if the planners need to update the short-term plan then this should be performed only after the daily plan is available:

∃t1, t2, p:PROCESS_EX, r:RESOURCE_EX
state(γ, t1) |= [resource_produced_by(r, p) ∧ resource(r, daily_plan)] & state(γ, t2) |= process_started(p9) ⇒ t1 ≤ t2
This property is also satisfied.

Analyzing this trace it can be seen that the reason why the planners get overloaded is because the unit manager was not available to perform the processes assigned to him. Based on this, the analyst might decide to check in what percentage of the traces it happens that the work load of the unit manager is less than 3 hours. This can be checked by the following emergent property:
sum([p:PROCESS_EX], case(∃t1, t2 state(γ, t1) |= [process_started(p) ∧ agent_performs_process(a, p) ∧ agent_performs_role(a, unit_manager)] & state(γ, t2) |= process_finished(p), t2-t1, 0)) < 3

Also it can be determined if the events specified in the trace had an impact on the organizational performance. One of the high-level goals of the organization considered in the case study is the goal G1: 'It is required to maintain good level of satisfaction of the employees'. This general goal is decomposed into more specific goals among which is the goal G1.1: 'It is required to maintain that the level of work load is moderate'. This is again decomposed into even more specific goals among which is the goal G1.1.1: 'It is required to achieve that the number of working hours per day for each employee is not more that 8'. This goal is based on the performance indicator P1: 'working hours per day per employee' which can be evaluated for every trace for the last point t of the trace.

$\forall$v:VALUE state(γ, t) |= pi_has_value(P1, v) $\Rightarrow$ v $\leq$ 8

For the trace in Fig. 1 it will be calculated and included at the end of the trace that pi_has_value(P1, 11) which is more than 8. Thus goal G1.1.1 is not satisfied and contributes negatively to the satisfaction of G1.1 which is propagated upwards in the goals structure.

DISCUSSION

This paper introduces automated techniques for manifold formal analysis of actual executions based on process-oriented models of organizations. On the one hand these techniques allow identifying errors and inconsistencies in executions of organizational scenarios, on the other hand they provide means for the evaluation and improving of organizational performance. For the proposed analysis techniques the TTL language and the environment TTL Checker are used, which allow high expressivity in specification of properties, including precise timing relations, references to multiple states (execution histories), arithmetical operations and checking properties on multiple traces. All these possibilities make TTL more expressive language than the standard modal logics (e.g., LTL, CTL, ATL) and calculi. Although TTL is an intuitive, close to the natural language, to define complex properties some skills in logics are needed. To support designers (e.g., managers) not skilled in logics, the used tool allows defining parameterized templates (macros) for TTL formulae, which can be instantiated in different ways which can also be used.

In the proposed approach traces are based on the actual execution of organizational scenarios. Such traces can be obtained in different ways: (1) automatically generated by a WfMS; (2) if data about the execution are represented in the form of informal logs obtained based on a process-oriented model in L_{PR}, they can be formalized (manually or automatically) using the language L_{EX}; (3) in case data about the execution are represented in some other formal language, the translation between this language and L_{EX} (if possible) is performed. Note that the translation and further analysis of traces obtained by (3) is possible only if a model based on which an original trace is generated can be related to an equivalent model in L_{PR}. Traces can be also generated based on a process-oriented model by performing simulations. Such traces can be used for diagnosis of inconsistencies, redundancies and errors in organizational structure and behavior. This type of analysis and the dedicated software are described in (Broek et al. 2006).

REFERENCES

Aalst, W. van der; Beer, H.; and Dongen, B. van. 2005. "Process Mining and Verification of Properties: An Approach based on Temporal Logic". In *On the Move to Meaningful Internet Systems*. Springer-Verlag, Berlin.

Aalst, W. van der and Hee, K.M van. 2002. *Workflow Management: Models, Methods, and Systems*. MIT press, Cambridge, MA.

Barjis, J; Shishkov, B and Dietz, J. 2002. "Validation of Business Components via Simulation". In *Proceedings of the 2002 Summer Computer Simulation Conference*.

Bosse, T.; Jonker, C.M.; Meij, L. van der; Sharpanskykh, A. and Treur, J. 2006. "Specification and Verification of Dynamics in Cognitive Agent Models". In *Proceedings of the Sixth International Conference on Intelligent Agent Technology*, IAT'06. IEEE Computer Society Press.

Broek, E.; Jonker, C.; Sharpanskykh, A.; Treur, J. and Yolum, P. 2006. Formal Modeling and Analysis of Organizations. In *Coordination, Organization, Institutions and Norms in Agent Systems I*, LNAI 3913, Springer

Desel, J.; Juhas, G.; Lorenz, R. and Neumair, C. 2003. „Modelling and Validation with VipTool". LNCS 2678, 380-389.

Popova, V. and Sharpanskykh, A. 2006. "Process-Oriented Organization Modeling and Analysis Based on Constraints". Technical Report 062911AI, VUA, http://hdl.handle.net/1871/10545

Popova, V. and Sharpanskykh, A. 2007a. "Formal Modelling of Goals in Agent Organizations". In *Proceedings of the AOMS Workshop (joint with IJCAI 2007)*.

Popova, V. and Sharpanskykh, A. 2007b. "Formal analysis of executions of organizational scenarios based on process-oriented models". Technical Report 071601AI, VUA, http://hdl.handle.net/1871/10643

Sharpanskykh, A. and Treur, J. 2006. "Verifying Interlevel Relations within Multi-Agent Systems". In *Proceedings of the 17th European Conference on Artificial Intelligence, ECAI'06*. IOS Press.

Alexei Sharpanskykh is a PhD student at the Vrije Universiteit Amsterdam. He received his Master degree in Computer Science at the Zaporizhzhya National Technical University (Ukraine). Currently he is doing research in modelling and analysis of multi-agent organizations in the context of a number of projects in the areas of logistics, incident management and air traffic control.

Viara Popova received her MSc degree in Computer Science at Sofia University, Bulgaria, and a PhD degree at Erasmus University Rotterdam in the area of Machine Learning and Data Mining. Subsequently she worked as a post-doctoral researcher at the Vrije Universiteit Amsterdam in the area of modeling and analysis of multi-agent organizations with a focus on logistics and incident management.