Appl Intell (2011) 34: 226–244
DOI 10.1007/s10489-009-0192-9

Formal analysis of executions of organizational scenarios based on process-oriented specifications

Viara Popova · Alexei Sharpanskykh

Published online: 19 September 2009

Abstract This paper presents various formal techniques for analysis of executions of organizational scenarios based on specifications of organizations. Organizational specifications describe (prescribe) ordering and timing relations on organizational processes, modes of use of resources, allocations of actors to processes, etc. The actual execution may diverge from scenarios (pre)defined by a specification. A part of techniques proposed in this paper is dedicated to establishing the correspondence between a formalized execution (i.e., a trace) and the corresponding specification. Other techniques proposed in this paper provide the analyst with wide possibilities to evaluate organizational performance and to identify bottlenecks and other inefficiencies in the organizational operation. For the proposed formal analysis the order-sorted predicate Temporal Trace Language (TTL) is used and it is supported by the dedicated software tool TTL Checker. The analysis approaches considered in this paper are illustrated by a case study in the context of an organization from the security domain.

Keywords Process executions · Process-oriented specifications · Formal methods · Organization modeling

V. Popova
Centre for Manufacturing, De Montfort University, The Gateway, Leicester, LE1 9BH, UK
e-mail: vpopova@dmu.ac.uk

A. Sharpanskykh (✉)
Department of Artificial Intelligence, Vrije Universiteit Amsterdam, De Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands
e-mail: sharp@few.vu.nl

1 Introduction

Management of processes in many modern organizations is supported by different dedicated software systems. A large class of such systems is the Workflow Management Systems (WfMS). Usually 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. Presently a number of formalisms are developed for representing such models: Petri-Nets, Workflow Nets, process algebra, logical specifications. Each of these formalisms has its own advantages and drawbacks, which are identified in [2]. Previously an expressive order-sorted predicate language for process-oriented modeling has been developed [20]. In this paper this language is used for the specification of a formal model, based on which actual execution of organizational scenarios is performed. The actual execution may diverge from scenarios (pre)defined by a model specification. Data about executions of organizational scenarios are recorded by most of WfMSs. Examples of data that are often registered are: starting and finishing time points of processes, types and amounts of resources used/consumed/produces/broken, names of actors, who perform processes. The type and the level of details of recorded data differ depending on a WfMS.

In order to guarantee the correct operation of an organization supported by a WfMS (1) a correct formal process-oriented specification(s) should be provided and (2) actual executions of organizational scenarios should correspond to this (these) formal specification(s).

For establishing the correctness of process-oriented (or workflow) specification a number of formal verification techniques exist [1–3, 20, 23, 25]. These techniques are

aimed at identifying errors and inconsistencies in specifications, irrespective of actual executions of these specifications and of the application domain. The verification techniques related to the model used in this paper are described in [20]. However, not many formal techniques and tools exist for performing validation of the process-oriented specification, i.e., establishing if the organization behaves as expected (i.e., as it is prescribed by the specification). In [5, 9] validation is performed by performing simulation of different scenarios of organizational behavior. Although simulation techniques can provide useful insights into relationships and dynamics of an organization, they often abstract from the complex dynamics of real organizations. To perform analysis based on the actual execution of an organizational scenario, data gathered by a WfMS during its operation can be used. For example, in [1] it is shown how the analysis based on linear temporal logic (LTL) can be used for establishing the correspondence between the observed organizational behavior recorded in a log-file and the expected behavior prescribed by a specification.

In this paper different types of formal analysis of actual executions based on organizational specifications will be described. These types include checking the conformity to a formal organizational (process-oriented in particular) specification, analysis of organizational emergent properties and organizational performance evaluation (estimation). The analysis is performed using an expressive predicate-based Temporal Trace Language (TTL), which allows more expressivity than LTL used in [1] (e.g., allows the specification of properties for checking over several traces). The proposed analysis techniques are supported by the software tool TTL Checker. Besides the checking of logical formulae, the TTL Checker allows the post-processing of checking results by constructing and evaluating arithmetical expressions.

The presentation is organized as follows. First, in Sect. 2 the proposed analysis framework is described in general. In the following sections different components of the framework are considered in detail. Sect. 3 briefly presents the language of process-oriented specifications, based on which actual executions are performed. A language used for formalizing these executions is introduced in Sect. 4. Section 5 describes the language TTL and considers the specification of TTL properties in the dedicated software environment TTL Checker. Different types of trace-based analysis are discussed in Sect. 6 and illustrated by examples. Section 7 presents a case study, in which it is demonstrated how the techniques proposed in this paper can be applied to the analysis of a real company's logistics. Finally, Sect. 8 concludes the paper.

2 Trace-based analysis: overview

For different types of analysis organizations can be considered from different perspectives. A general organization modeling and analysis framework has been described in [12, 18–20] that includes 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, *the process-oriented view* describes static structures of tasks and resources and the flow of control. Concepts and relations within every view are formally described using dedicated languages based on the expressive order-sorted predicate logic. The views are related to each other by means of sets of common concepts. This enables different types of analysis across different views.

This paper considers actual execution of organizational scenarios based on organizational specifications created using this framework. Such scenarios relate to all views of the framework but in particular to the process-oriented view as it describes the company's operations to which the recorded events are most often related. The specification of process-oriented models using the predicate-based language L_{PR} is considered in [20]. Section 3 of this paper gives a brief overview of the language L_{PR}.

Data about actual executions based on process-oriented specifications are structured in the form of a trace. A *trace* is 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 represented by atoms expressed in a sorted first-order predicate language L_{EX}. The constructs of L_{EX} will be described in more detail in Sect. 4.

The formal analysis of actual executions of organization scenarios is performed by checking (dynamic) organizational properties on traces using the dedicated software tool TTL Checker [6]. Properties to be checked are specified in Temporal Trace Language (TTL) [26], a variant of order-sorted predicate logic. 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). Note that the same formula may be checked on multiple traces at the same time. By doing this a property can be checked with respect to different executions of organizational scenarios. As a result the tool generates an 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. Furthermore, the tool allows performing statistical post-processing of multiple traces. More details on the language TTL and the tool will be given in Sect. 5.

Here we identify the types of trace analysis considered in this paper that can be performed using the TTL Checker.

Each organizational specification (pre)defines a set of correct scenarios of organization behavior. Depending on the type of an organization, the model may be specified at different levels of abstraction (with varying amount of details and precision) and may allow different degrees of freedom for agents in scenario executions. The actual execution of any organization may diverge from scenarios described by the specification. In some organizations a certain degree of deviation is allowed (e.g. as far as the principal organization goals are achieved and important requirements are satisfied), whereas other organizations require a strict adherence to the specification (e.g., incident management organizations, nuclear power plants). In the second case the verification of the conformity of an actual execution to a formal organizational specification is of special importance. This is the first type of analysis considered in this paper, which is discussed in detail in Sect. 6.1.

Every correct specification guarantees the satisfaction of a set of (global) constraints over organizational concepts and instances including processes, resources, roles and agents identified in the organization. These constraints are usually specified based on different organizational and general normative documents (e.g., an organizational mission statement, a strategy description, laws, organizational normative acts, different policies, job and procedure descriptions) and are formalized as predicate logic formulae that should be satisfied by corresponding process-oriented specifications. The classification of constraint types can be found in [20].

In general, if a trace conforms to the corresponding process-oriented specification (i.e., a trace is in the set of legal executions of the specification), then all constraints imposed on and satisfied by the specification are also satisfied by the trace. However, when the checking of the conformity of the trace to the specification 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., a set of organizational constraints) should be performed, which is the second type of analysis considered in this paper (see Sect. 6.2).

Often organizational models allow (different degrees of) autonomy of agents in the execution of organizational scenarios. For example, in many organic organizations processes are defined loosely in order to allow flexibility and rapid change of an organization. Although executions of such processes are not prescribed by the specifications, in order to investigate organizational performance, bottlenecks, error and inconsistencies, such executions (or traces) still need to be analyzed. This analysis type is called analysis of the emergent organizational behavior and will be investigated further in this paper in Sect. 6.3.

Finally, this paper proposes a method for the evaluation of organizational performance based on checking of the satisfaction of organizational goals related to processes and defined by specifications from the performance-oriented view on organizations (see Sect. 6.4).

The types of analysis described above may be performed both during the execution and after the execution of organizational scenarios. For example, the analyst may choose for the real time checking of the correspondence of a trace to some process-oriented specification to enable the immediate notification in case inconsistencies are identified. In the other case the recorded trace may be used by managers for (automated) post-analysis aimed at the improvement of organizational performance.

3 Process-oriented specification language L_{PR}

In this Section we briefly discuss the language L_{PR} for describing process-oriented specifications. For more details the reader is referred to [20]. The objects defined in the specification belong to one of the following main types (represented by sorts in the language L_{PR}):

- tasks describing functions that can be performed in the organization (sort TASK),
- processes which are instances of tasks and inherit their characteristics (sort PROCESS),
- resource types describing information and material artifacts that can be produced, used and consumed by tasks (sort RESOURCE_TYPE),
- resources are specific amounts of resource types that are produced at the same time by processes and can be used and consumed by other processes (sort RESOURCE),
- locations where resources can be present such as storage facilities (sort LOCATION),
- agents are individuals that can perform processes (sort AGENT related to the agent-oriented view of the general framework),
- roles are sets of functionalities that can be assigned to agents to perform (sort ROLE related to the organization-oriented view of the general framework),
- goals are organizational objectives that can be realized by performing instances of organizational tasks (sort GOAL related to the performance-oriented view),
- performance indicators are measures based on which the goals are defined and which evaluate specific aspects of the performance of processes in the workflow (sort PI related to the performance-oriented view of the framework).

Different types of relations can be defined over objects of these sorts. Here only those are considered which are relevant for the analysis of execution traces.

Tasks have as characteristics minimal and maximal durations specified for example as t.min_duration = v for a task t and minimal duration v. Tasks can produce/use/consume resources of specific types which can be specified by task_uses(t:TASK, rt:RESOURCE_TYPE, v:VALUE) specifying that task t uses amount v of resource type rt and similarly task_produces(t:TASK, rt:RESOURCE_TYPE, v:VALUE) and task_consumes(t:TASK, rt:RESOURCE_TYPE, v:VALUE). A process is related to the task of which it is an instance by is_instance_of(t:TASK, p:PROCESS).

The set of specified processes together with the set of ordering relation defined on them form a workflow. Process ordering relations can be specified in the following ways:

starts_after(p1:PROCESS, p2:PROCESS) defines that process p1 starts after process p2;
starts_with
(p1:PROCESS, p2:PROCESS)—p1 starts simultaneously with p2;
starts_during(p1:PROCESS, p2:PROCESS)—p1 starts during the execution of p2;
finishes_with(p1:PROCESS, p2:PROCESS)—p1 finishes simultaneously with p2;
finishes_during(p1:PROCESS, p2:PROCESS)—p1 finishes during the execution of p2.

The beginning and the end of the workflow are designated by special zero-duration processes BEGIN and END.

Furthermore, three types of structures defining ways of executing processes can be defined: and-, or- and loop-structures. Branches of and-structures start simultaneously and are all executed. When the process after the structure should start is specified by the and-condition which can designate all, any or specific processes at the end of the branches that should finish before the workflow can continue. Only one branch of an or-structure can be executed depending on the or-condition which is an expression based on a decision variable (related to a decision process), state or a characteristic of an environmental object. Loop structures contain processes that can be repeated depending on the loop-condition within a maximum number of iterations. All these structures are defined in a similar way with special zero-duration processes designating the beginning and the end of the structures, for example: begin_and(id:AND_STRUCT) and end_and(id:AND_STRUCT) mark the beginning and the end of an and-structure with the name id. Conditions are specified such as and_cond(id:AND_STRUCT, e:COND_EXPRESSION) which means that the condition of and-structure id is defined by the expression e. Or-branches are defined as: or_branch(v:OR_COND_VALUE, p:PROCESS) for every specific condition value and first process p in a branch. Maximal number of iterations n of a loop-structure id is defined as: loop_max(id:LOOP_STRUCT, n:VALUE).

One of the characteristics of resource types is their expiration duration (e.g. rt.expiration_duration = v for a resource type rt with expiration duration v). Resource types that can be shared by several processes are specified in resource_sharable(rt:RESOURCE_TYPE, L:PROCESS_LIST). Resources that are related to resource types by is_resource_type(r:RESOURCE, rt:RESOURCE_TYPE). Resources are characterized by an amount (e.g., r.amount = v).

Processes can add or remove resource types from locations which can be specified using the predicates: process_adds_resource_type_to(p:PROCESS, rt:RESOURCE_TYPE, l:LOCATION, v:VALUE) and process_rem_resource_type_from(p:PROCESS, rt:RESOURCE_TYPE, l: LOCATION, v:VALUE) where the last argument specifies the amount of the added or removed resource type. Resources are considered removed at the starting time point of the corresponding process and are added at the ending time point of the process.

Relations between roles, agents and processes are defined as follows: role_performs_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.

4 Execution language L_{EX}

For the formalization of a trace a dedicated state language L_{EX} is used, which is based on L_{PR} briefly discussed in the previous Section. L_{EX} is based on an ontology specified by a number of sorts, sorted constants, variables, functions and predicates (i.e., a signature). Each sort included into this ontology is represented by 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). Note that in order to distinguish the names of sorts of L_{EX} from the names of sorts in the language L_{PR}, all sort names of L_{EX} finish with the EX postfix.

This ontology includes the following sorts:

PROCESS_EX—a set of all process names in a trace;
RESOURCE_EX—a set of all resource names;
RESOURCE_TYPE_EX—a set of all resource types names;
LOCATION_EX—a set of all location names;
ROLE_EX—a set of all role names;
AGENT_EX—a set of all agent names;
PI_EX—a set of all performance indicators names;
VALUE—an ordered set of numbers;
PROCESS_LIST_EX—a set of all names of process lists;

Table 1 Relations defined in L_{EX}

Predicate specification	Informal description
process_started: PROCESS_EX	The process specified as an argument has started
process_finished: PROCESS_EX	The process specified as an argument has finished
resource_used_by: RESOURCE_EX × PROCESS_LIST_EX × VALUE	Specifies that a certain amount of a resource is used by a process
resource_consumed_by: RESOURCE_EX × PROCESS_EX × VALUE	Specifies that a certain amount of a resource is consumed by a process
resource_produced_by: RESOURCE_EX × PROCESS_EX × VALUE	Specifies that a certain amount of a resource is produced by a process
resource: RESOURCE_EX × RESOURCE_TYPE_EX	Identifies a resource of a certain resource type
resource_expired: RESOURCE_EX	Specifies that a resource is expired
resource_invalid: RESOURCE_EX × VALUE	Specifies that a certain amount of a resource became invalid (cannot be used any more)
available_resource_amount: RESOURCE_EX × VALUE	Specifies the available amount of the resource
process_adds_res_to_location: PROCESS_EX × RESOURCE_EX × LOCATION_EX × VALUE	Specifies that a certain amount of the resource is added to the location
process_rem_res_from_location: PROCESS_EX × RESOURCE_EX × LOCATION_EX × VALUE	Specifies that a certain amount of the resource is removed from the location
available_res_amount_at_location: RESOURCE_EX × VALUE × LOCATION_EX	Specifies the available amount of the resource at the specific location
pi_has_value: PI_EX × VALUE	Identifies the value of a certain PI
agent_is_assigned_to_role: AGENT_EX × ROLE_EX	Specifies the assignment of an agent to a role
agent_performs_process: AGENT_EX × PROCESS_EX	Identifies that an agent performs a certain process
env_object_changed_state_into: ENV_OBJECT_EX × OBJ_STATE_EX	Specifies a changed state of an environmental object
env_object_changed_char_into: ENV_OBJECT_EX × OBJ_CHAR_EX × VALUE	Specifies the value of a certain characteristic of an environmental object
decision_taken: DECISION_VARIABLE_EX × DECISION_VAR_VALUE_EX	Identifies the value of a decision variable

DECISION_VARIABLE_EX—a set of all names of decision variables;
DECISION_VAR_VALUE_EX—a set of all values of decision variables;
ENV_OBJECT_EX—a set of all environmental objects names;
OBJ_STATE_EX—a set of all names of states of objects;
OBJ_CHAR_EX—a set of all names of object characteristics.

To define events, a number of relations are introduced into L_{EX} (see Table 1).

5 Language TTL and specification of dynamic properties

In this Section first the Temporal Trace Language (TTL) used for specifying dynamic properties for analysis, is introduced. Then, some peculiar aspects of the specification of TTL properties in the dedicated software environment TTL Checker are discussed.

TTL [26] is a variant of order-sorted predicate logic [15] and has some similarities with Situation Calculus and Event Calculus. Whereas the standard multi-sorted predicate logic is a language to reason about static properties only, TTL is an extension of such language with facilities for reasoning about the dynamic properties of arbitrary 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 (a set of all state names of a system), TRACE (a set of all trace names), STATPROP (a set of all state property names). Furthermore, for every sort S from the state language the following TTL sorts exist: the sort S^{VARS}, which contains all variable names of sort S; the sort S^{GTERMS}, which contains names of all ground terms, constructed using sort S; sorts S^{GTERMS} and S^{VARS} are subsorts of sort S^{TERMS}.

In TTL, formulae of the state language are used as objects. To provide names of object language formulae φ in TTL the operator $(*)$ is used (written as φ^*), which maps variable sets, term sets and formula sets of the state language to the elements of TTL sorts S^{GTERMS}, S^{TERMS}, S^{VARS} and

STATPROP. The state language and TTL define disjoint sets of expressions. Therefore, in TTL formulae we shall use the same notations for the elements of the object language (i.e., constants, variables, functions, predicates) and for their names in TTL without introducing any ambiguity. 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 our case an organization situated in the environment) in a trace is referred using a function symbol state of type TRACE $\times$ TIME $\rightarrow$ STATE.

The set of function symbols of TTL includes $\wedge, \vee, \rightarrow, \leftrightarrow$: STATPROP $\times$ STATPROP$\rightarrow$ STATPROP; not: STATPROP$\rightarrow$ STATPROP, $\forall, \exists$: S^{VARS} $\times$ STATPROP$\rightarrow$ STATPROP, which counterparts are Boolean propositional connectives and quantifiers. Further we shall use $\wedge, \vee, \rightarrow, \leftrightarrow$ in infix notation and $\forall, \exists$ in prefix notation for better readability.

The states of a system are related to names of state properties via the formally defined satisfaction relation denoted by the infix predicate $\models$ (or denoted by the prefix predicate holds): state(γ, t) $\models$ 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)$\models$ 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. In general, TTL terms are constructed by induction in a standard way from variables, constants and function symbols typed with all before mentioned TTL sorts.

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 $\tau_1 = \tau_2$ 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 (including the axiomatic basis) is given in [27].

An example of a well-formed TTL-formula is:

$\forall$t$\forall$p: PROCESS_EX state(γ, t) $\models$ process_started(p)

$\Rightarrow \exists$t' t' > t state(γ, t') $\models$ process_finished(p)

This property expresses that all processes started in trace γ should eventually finish in this trace.

Using a convenient graphical interface of the TTL Checker [6], TTL formulae can be inputted for the subsequent analysis on traces. The checking on traces is performed automatically and as a result the answer is generated identifying if a formula is satisfied by a trace(s). Note that even if a formula contains a universal quantifier(s), still only one answer will be generated. It also means that in case the checking fails, only one counterexample will be shown to the analyst, which does not reveal all other possible reasons for the failure of the formula. However, if the analyst demands more fine-grained analysis, it is still possible to replace automatically a formula that contains a quantified variable by a number of formulae, in which the quantified variable is replaced by particular instances from the domain of this variable, and then check each formula separately. If all obtained formulae are satisfied, then the original formula is satisfied as well. For example, the property described above can be replaced by the properties, in which the variable p is replaced by each individual from the sort PROCESS_EX.

To provide support for analysts who are not skilled in logics, the TTL Checker allows defining parameterized templates. Essentially, such templates are predefined logical formulae that can be referred by names with certain parameters. The designer instantiates a template by assigning certain values to these parameters.

Furthermore, the tool allows post-processing of the verification results by composing and evaluating arithmetical statements. These statements are formed from arithmetical relations and operations on numerical values. For this the following functions are used:

case(logical _formula, value1, value2)—meaning that if logical_formula is true, then the function is mapped to value1, otherwise—to value2.

sum([summation _variables], case(logical_formula, value1, 0))—the value of this function is calculated as follows: 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 (which is initially equal to 0).

Examples of analysis cases that include statistical post-processing will be given in Sect. 6.

Furthermore, the TTL Checker provides checking results for further more sophisticated statistical post-processing (e.g., using different hypothesis testing methods and automated tools such as SPSS).

6 Types of trace-based analysis

As discussed in Sect. 2, we consider four types of analysis over execution traces. They are described in more detail in

this section. First, Sect. 6.1 discusses the analysis of whether the trace agrees with the organizational specification. Section 6.2 discusses the analysis of whether the trace agrees with the formal organization (i.e., a set of organization-specific constraints). Section 6.3 describes the analysis of emergent organizational properties not prescribed by the organizational specification and/or the formal organization. Finally Sect. 6.4 discusses the analysis of organizational performance based on performance indicators from execution traces and organizational goals.

6.1 Trace conformity to an organizational specification

As described in Sect. 3 the process-oriented specification created by the designer consists of objects defined in the sorts of L_{PR} and characteristics and relations for these objects defined using the predicates and functions of L_{PR}. Every such specification can be translated to a set of constraints that should be satisfied by an actual execution trace of the specification. Since the specification is only a partial representation of a set of desirable behaviors of the (part of the) organization, it is possible that events unforeseen by the specification cause the actual trace to represent behavior outside of this set, thus, the actual execution does not agree with the specification. Therefore an important phase in the analysis of actual traces is to check whether they satisfy the constraints defined by the specification which will be discussed in detail in this section. Note that the traces are assumed to be recorded correctly with respect to the real execution and contain no syntactical mistakes and omissions.

First we describe the translation of the specification to constraints over execution traces. The constraints are represented as properties formulated in the Temporal Trace Language using the L_{EX} as a state language. These properties are defined over constants that represent the objects defined in the specification. Each property is based on one or a specific combination of language constructs such as ordering relations, and-/or-/loop-structures, characteristics of objects, etc. A combination of such constructs might generate more than one constraints representing different aspects. The set of properties generated by the translation process are meant to be checked together automatically on the trace and give feedback to the analyst which ones are violated.

In the following, we define general rules on how to translate a process-oriented specification to TTL properties that should be checked on actual execution traces.

The first property we consider represents the restriction that only processes defined in the specification are allowed to be performed. It is formalized in the TTL in the following way. For p1, ..., pn constants representing the names of the processes in the specification the following property should be checked:

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

The next properties represent the constraints that processes that are not part of an or-branch of any or-structure (should be performed in any condition) have indeed started and finished in the actual trace.

For p1 a process not in any or-branch:

C2: $\exists$t1 state(γ, t1) $\models$ process_started(p1)
C3: $\exists$t1 state(γ, t1) $\models$ process_finished(p1)

For processes that are part of an or-branch it is not known in advance whether they will be executed or not since that depends on the evaluation of the or-condition. Similarly for processes in loop-structures it is not known how many times they will be executed. Therefore for these processes it needs to be checked only for processes that have started whether they have finished in the actual trace which is expressed in the following property:

For p1 a process in an or-branch or in a loop-structure:

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

Additionally for processes not in loop-structures:

C5: $\exists$t1 state(γ, t1) $\models$ process_started(p1) $\Rightarrow$ ($\forall$t3 t3 $\neq$ t1 $\Rightarrow$ state(γ, t3) $\models$ $\neg$process_started(p1)

The next property checks whether the actual duration of a process is within the range defined by the corresponding task.

For a process p1, a task tk, min duration d1 and max duration d2 such that [is_instance_of(p, tk), tk.min_duration=d1, tk.max_duration=d2]:

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

The processes in the specification are synchronized by different types of ordering relations which are translated to constraints in the following way:

For p1, p2 such that starts_with(p1, p2):

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

For p1, p2 such that finishes_with(p1, p2):

C9: $\exists$t1 state(γ, t1) $\models$ process_finishes(p1) $\Rightarrow$ state(γ, t1) $\models$ process_finishes(p2)
C10: $\exists$t1 state(γ, t1) $\models$ process_finishes(p2) $\Rightarrow$ state(γ, t1) $\models$ process_finishes(p1)

For p1, p2 such that starts_during(p1, p2):

C11: $\exists$t1 state(γ, t1) $\models$ process_started(p1) $\Rightarrow$ $\exists$t2, t3 t2 $\leq$ t1 & t1 $\leq$ t3 $\wedge$ state(γ, t2) $\models$ process_started(p2) & state(γ, t3) $\models$ process_finished(p2)

For p1, p2 such that finishes_during(p1, p2):

C12: $\exists$t1 state(γ, t1) $\models$ process_finished(p1) $\Rightarrow$ $\exists$t2, t3 t2 t1 & t1 $\leq$ t3 state(γ, t2) $\models$ process_started(p2) & state(γ, t3) $\models$ process_finished(p2)

For p1, p2, d such that starts_after(p2, p1, d) except for beginning and ending of an and-, or-, or loop-structures:

C13: $\exists$t1 state(γ, t1) $\models$ process_finished(p1) $\Rightarrow$ $\exists$t2: state(γ, t2) $\models$ process_started(p2) & d = t2 − t1

Next, and-structures are considered. Firstly, specifications such as [starts_after(p, begin_and(id), d), starts_after(begin_and(id), p1), ..., starts_after(begin_and(id), pn)] are treated as [starts_after(p1, p, d), ..., starts_after(pn, p, d)]. Furthermore the end of the structure should be considered [starts_after(end_and(id), p1), ..., starts_after(end_and(id), pn), starts_after(p, end_and(id))] and it should be checked whether the order of execution at the end of the and-structure matches the specified and-condition.

For p1, ..., pn, p, d such that [starts_after(end_and(id), p1), ..., starts_after(end_and(id), pn), starts_after(p, end_and(id), d), and_cond(id, any)]:

C14: $\exists$t1 state(γ, t1) $\models$ [process_finished(p1) $\vee \cdots \vee$ process_finished(pn)] & ($\forall$t2: t2 $\leq$ t1 $\Rightarrow$ state(γ, t2) $\models$ [$\neg$process_finished(p1) $\wedge \cdots \wedge$ $\neg$process_finished(pn)] $\Rightarrow$ $\exists$t3 state(γ, t3) $\models$ process_started(p) & d = t3 − t1

For p1, ..., pn, p, d such that [starts_after(end_and(id), p1), ..., starts_after(end_and(id), pn), starts_after(p, end_and(id), d), and_cond(id, all)]:

C15: $\exists$t1, ..., tn, tp state(γ, t1) $\models$ process_finished(p1) & $\cdots$ & state(γ, tn) $\models$ process_finished(pn) & tp $\geq$ t1 & $\cdots$ & tp $\geq$ tn & (tp = t1 | $\cdots$ | tp = tn) $\Rightarrow$ $\exists$tt state(γ, tt) $\models$ process_started(p) & d = tt−tp

And-conditions with other expressions are treated similarly taking into account which processes should finish so that the next process can start, for example: [starts_after(end_and(id), p1), ..., starts_after(end_and(id), pn), starts_after(p, end_and(id), d), and_cond(id, finished(p1)$\wedge$ finished(p2))] can be checked as follows:

C16: $\exists$t1, t2, t state(γ, t1) $\models$ process_finished(p1) & state(γ, t2) $\models$ process_finished(p2) & t1 $\leq$ t & t2 $\leq$ t & (t = t1 | t = t2) $\Rightarrow$ $\exists$t3 state(γ, t3) $\models$ process_started(p) & d = t3−t

For or-structures it should be checked if exactly one of the branches is executed and that it matches the evaluation of the specified or-condition.

For p, p1, ..., pn, d, decision variable dv and decision variable values val1, ..., valn such that [starts_after(begin_or(id), p, d), starts_after(p1, begin_or(id)), ..., starts_after(pn, begin_or(id)), or_cond(id, dv), or_branch(p1, val1), ..., or_branch(pn, valn)]:

C17: $\exists$t1 state(γ, t1) $\models$ process_finished(p) $\Rightarrow$ $\exists$t2 (state(γ, t2) $\models$ process_started(p1) & $\forall$t3 state(γ, t3) $\models$ [$\neg$process_started(p2) $\wedge \cdots \wedge$ $\neg$process_started(pn)] & $\exists$t4 state(γ, t4) $\models$ decision_taken(dv, val1) & t4 $\leq$ t2 & ($\forall$t5 t5 $\geq$ t4 & t5 $\leq$ t2 & state(γ, t5) $\models$ decision_taken(dv, val) $\Rightarrow$ val = val1) | $\cdots$ | (state(γ, t2) $\models$ process_started(pn) & $\forall$t6 state(γ, t6) $\models$ [$\neg$process_started(p1) $\wedge \cdots \wedge$ $\neg$process_started(pn-1)] & $\exists$t7 state(γ, t7) $\models$ decision_taken(dv, valn) & t7 $\leq$ t2 & ($\forall$t8 t8 $\geq$ t7 & t8 $\leq$ t2 & state(γ, t8) $\models$ decision_taken(dv, val) $\Rightarrow$ val = valn)) & d = t2 − t1

In a similar way, formulations can be given for the case of a condition based on the state of an environmental object or a characteristic of an environmental object.

Furthermore it should be checked that the processes in the branches that did not start also are not executed. For every or-branch such that [starts_after(p1, begin_or(id)), starts_after(p2, p1), ..., starts_after(pn, pn-1), starts_after(end_or(id), pn)] the following property should be checked:

C18: $\forall$t1 state(γ, t1) $\models$ $\neg$process_started(p1) $\Rightarrow$ $\forall$t2 state(γ, t2) $\models$ [$\neg$process_started(p2) $\wedge \cdots \wedge$ $\neg$ process_started(pn)]

For p1, ..., pn, p, d such that [starts_after(end_or(id), p1), ..., starts_after(end_or(id), pn), starts_after(p, end_or(id), d)]:

C19: $\exists$t1 state(γ, t1) $\models$ [process_finished(p1) $\vee \cdots \vee$ process_finished(pn)] $\Rightarrow$ $\exists$t2 state(γ, t2) $\models$ process_started(p) & d = t2 − t1

Next, loop-structures are considered. Specifications such as [starts_after(begin_loop(id), p1), starts_after(p2, begin_loop(id))] are treated as starts_after(p2, p1). Furthermore for every process in a loop-structure the corresponding sequencing relations are checked in a similar way. For the last process p2 in a loop-structure with a condition expression dv = val for a decision variable dv such that [starts_after(p1, begin_loop(id)), ..., starts_after(end_loop(id), p2, d2), starts_after(p3, end_loop(id), d3), loop_cond(id, dv = val), loop_max(m)] the following property should be checked:

C20: $\exists$t1 state(γ, t1) $\models$ process_finished(p2) $\Rightarrow$ $\exists$t2 (state(γ, t2) $\models$ process_started(p1) & $\exists$t3 state(γ, t3) $\models$ decision_taken(dv, val) & t3 $\leq$ t2 & ($\forall$t4 t4 $\geq$ t3 & t4 $\leq$ t2 & state(γ, t4) $\models$ decision_taken(dv, val1) $\Rightarrow$ val = val1) & $\neg$max_iter(p2) & t2 − t1 = d2) & (state(γ, t2) $\models$ process_started(p3) & t2 − t1 = d2 + d3 & ($\exists$t3 state(γ, t3) $\models$ decision_taken(dv, val1) & val1 $\neq$ val & t3 $\leq$ t2 & ($\forall$t4 t4 $\geq$ t3 & t4 $\leq$ t2 & state(γ, t4) $\models$ decision_taken(dv, val2) $\Rightarrow$ val2 = val1) | max_iter(p2)))

Property max_iter(p2) can be defined as follows where m is the maximal number of iterations:
$\exists t1, \ldots,$ tm $t1 \neq t2 \wedge \cdots \wedge t1 \neq tm \wedge \cdots \wedge tm - 1 \neq tm \wedge$ state(γ, t1) $\models$ process_started(p2) & $\cdots$ & state(γ, tm) $\models$ process_started(p2)

Different types of conditions are treated similarly taking into account the specific condition variable.

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_consumes(tk, rt, v)]:

C21: sum([r:RESOURCE_EX], case($\exists$t1, t2 state(γ, t1) process_started(p) & state(γ, t2) $\models$ process_finished(p) & $\exists$t3 t1 $\leq$ t3 & t3 $\leq$ t2 $\wedge$ state(γ, t3) $\models$ resource_consumed_by(r, p, v1) & $\exists$t4 state(γ, t4) $\models$ resource(r, rt), v1, 0)) = v

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:

C22: sum([L:PROCESS_LIST_EX], case($\exists$t1, t2 state(γ, t1) $\models$ process_started(p) & state(γ, t2) $\models$ process_finished(p) & t1 $\leq$ t & t $\leq$ t2 & state(γ, t) $\models$ resource_used_by(r, L, v1) & is_in_list(p, L) & $\exists$t4 state(γ, t4) $\models$ resource(r, rt), v1, 0)) = v

For resource type rt, task tk, amount v and process p such that [is_instance_of(p, tk), task_produces(tk, rt, v)]:

C23: $\exists$t1, t2 state(γ, t1) $\models$ process_started(p) & state(γ, t2) $\models$ process_finished(p) $\Rightarrow$ $\exists$t3 t1 $\leq$ t3 & t3 $\leq$ t2 & state(γ, t3) $\models$ resource_produced_by(r, p, v) & $\exists$t4 state(γ, t4) $\models$ resource(r, rt)

In the specification, the resources already available at the beginning of the workflow that will be used by its processes are represented as resources produced by the BEGIN process. It should therefore be checked if the available resource amount at the beginning of the execution trace matches the amount produced by the BEGIN process in the specification.

For resource r, resource type rt, amount v such that [process_output(BEGIN, r), is_resource_type(r, rt), r. amount=v]:

C24: sum([r:RESOURCE_EX], case(state(γ, 0) $\models$ [available_resource_amount(r, v1) $\wedge$ 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)]:

C25: $\exists$t1 $\exists$L1:PROCESS_LIST_EX state(γ, t1) $\models$ resource_used_by(r, L1, v) & $\exists$t2 state(γ, t2) $\models$ resource(r, rt) $\Rightarrow$ is_sublist_of(L1, L)

For resources it should also be checked if they are placed at the right location by processes. For resource type rt, process p, location l, value v such that [process_adds_resource_type_to(p, rt, l, v)]:

C26: $\exists$t1, t2 state(γ, t1) $\models$ process_started(p) & state(γ, t2) $\models$ process_finished(p) $\Rightarrow$ $\exists$t3, t1 $\leq$ t3 & t3 $\leq$ t2 & $\exists$r:RESOURCE_EX state(γ, t3) $\models$ process_adds_res_to_location(p, r, l, v) & $\exists$t4 state(γ, t4) $\models$ resource(r, rt)

For resource type rt, process p, location l, value v such that [process_rem_resource_type_to(p, rt, l, v)]:

C27: $\exists$t1, t2 state(γ, t1) $\models$ process_started(p) & state(γ, t2) $\models$ process_finished(p) $\Rightarrow$ $\exists$t3, t1 $\leq$ t3 & t3 $\leq$ t2 & $\exists$r:RESOURCE_EX state(γ, t3) $\models$ process_rem_res_from_location(p, r, l, v) & $\exists$t4 state(γ, t4) $\models$ resource(r, rt)

Finally it should be checked if role and process assignment to agents follow the specification:

For role r, agent a and process p such that [role_performs_process(r, p), agent_plays_role(a, r)]:

C28: $\exists$t1, t2 state(γ, t1) $\models$ process_started(p) & state(γ, t2) $\models$ process_finished(p) $\Rightarrow$ $\forall$t3 t1 $\leq$ t3 & t3 $\leq$ t2 & state(γ, t3) $\models$ [agent_performs_role(a, r) $\wedge$ agent_performs_process(a, p)]

The above listed properties are quite general and can be checked in any order on the actual execution trace. However in many cases it would be beneficial to enforce certain order in which the properties should be checked. For example it would often be useful to find efficiently the first time point at which the trace does not correspond to the specification and which constraint is violated. Often when one constraint is violated that causes the violation of many other constraints however finding all these constraints might not add much more information on what went wrong. For example when one process fails to produce a resource necessary for another process this might cause changes or even failures in the rest of the execution trace. However these are only consequences of the first failure—the production of the resource. It is therefore useful to alert the analyst of the first time point at which a violation of a constraint occurs. It is possible to find this point by checking all constraints, finding all violations and then finding the earliest one. However if we are only interested in the first one it would be more efficient to try to find the earliest violation first after which the search can be stopped.

The approach proposed here is to consider the events of the trace in their natural temporal order starting from the beginning of the trace and processing them one by one. For each event that represents a starting or finishing point of a process only a selection of the relevant constraints are

checked. Here the general constraints (C1, ..., C28) can be used after some small adjustments coming from the fact that they are checked with respect to a specific time point when a specific event occurs.

In the following we define the set of relevant constraint with respect to the type of event occurring in the trace. The first constraints to be checked are C24 which checks the available resource at the first time point only and C2 (which checks if a process starts) for the first process(es) in the workflow that should start at the first time point unconditionally (not in any or-branch). 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 (a variation of C17). Afterwards the (partially) ordered list of starting and finishing points of processes is considered item by item. For every starting point the following types of constraints are considered (in this order):

1. the process is defined in the specification (C1),
2. the process has not been executed before in the part of the trace up to the current event for processes not in loop-structures (C5),
3. constraints with respect to the conditions for the end of and-structures (C14, C15, C16).
4. constraints related to synchronizations starts_with and starts_during (C7, C8, C11),
5. existence of a finishing point for the process (C3, C4),

For every finishing point the following types of constraints are considered (in this order):

1. resource-related constraints (C21, C22, C23, C25, C26, C27)
2. agent- and role-related constraints (C28)
3. process duration (C6)
4. constraints related to synchronizations finishes_with and finishes_during (C9, C10, C12)
5. constraints with respect to the process which should start next (C13, C17, C18, C19, C20).

From all types of considered constraints those are selected that refer to the specific process to which the starting or finishing point belongs. When two or more events coincide finishing points (in any order) are considered before the starting points (in any order).

The above described approach assumes the availability of the whole execution trace at the beginning of the analysis. In some situations however it might be necessary to perform such analysis at real time while the trace is being generated. This will give the possibility to react as soon as something in the execution deviates from the specification and take appropriate measures. With some adjustments, the generic properties can be used here as well. The analysis process works as follows. The information about events from the trace become available following the order of the time points at which they occur and all events happening at the same time point become available all at once. Depending on the type of the current considered event specific types of constraints are checked or assigned to be checked at specific time points in the future. The system gives a warning when a constraint is violated by the trace. At the first time point again the available resource (C24) and the starting of the first processes is checked (C2, C17). Then for every starting point of a process that appears in the trace the following types of constraints are considered:

1. the process is defined in the specification (C1),
2. the process has not been executed before in the part of the trace up to the current event (C5),
3. constraints with respect to the conditions for the end of and-structures (C14, C15, C16).
4. constraints related to synchronizations starts_with and starts_during (C7, C8, C11),
5. existence of a finishing point for the process (based on C3, C4)—as this information is not yet available in the trace, the corresponding properties are scheduled to be checked for every time point until such a finishing point occurs. When the maximal duration determined by the specification is passed and no finishing point has yet occurred a warning is given that the process exceeds its allowed duration. A warning is also generated if the process finishes before its minimal duration determined by the specification has passed.
6. resource-related constraints of the following types are checked for every time point until the process finishes: resource sharability (C25), resource used by a process (C22), resource produced (C23) or consumed (C21) by the process, resource at a location (C26, C27) up to the current time point is checked not to exceed the specified amount in the specification.
7. agent- and role-related constraints are checked for every time point until the process finishes.

For every finishing point the following types of constraints are considered (in this order):

1. resource produced (C23) or consumed (C21) by the process for its whole duration is checked to be equal to the pre-specified amount in the specification
2. constraints related to synchronizations finishes_with and finishes_during (C9, C10, C12)
3. constraints with respect to the process which should start next (C13, C17, C18, C19, C20)—since the necessary information is not yet available in the trace, the properties are scheduled to be checked in the following way. For every time point it is checked if the expected process has started until information about its starting point arrives. If this starting point is before the pre-specified delay a warning is issued. A warning is also issued when the delay has passed and the process has not started yet. C17 is

checked until the starting point of the first process in a branch of the or-structure. Afterwards it is checked for the rest of the incoming trace that none of the other first processes of other branches of this structure start at any later point (C18). At the end of the or-structure a property is scheduled for checking if the process after the or-structure starts for every time point until the process actually starts. For loop-structures a counter is kept for the current number of iterations. It is used to determine the next process together with the current evaluation of the condition. Based on that, the appropriate property is scheduled to be checked until the correct process starts.

For most types of constraints the following rule is used: when a specific constraint is checked or scheduled for checking it is marked and is not considered any more at the events occurring later. An exception is made for the loop-related constraints which might need to be considered multiple times.

6.2 Trace conformity to a formal organization

A formal organization is specified by a fixed set of rules that define (prescribe) organizational structure and behavior. These rules are usually described in different organizational and general normative documents (e.g., an organizational mission statement, a strategy description, laws, organizational normative acts, different policies, job and procedure descriptions) and are formalized as predicate logic constraints imposed on an organizational specification.

Some of these constraints are strict and should not be violated in any organizational scenario; e.g., "the number of working hours of an employee per week should not exceed a certain value", "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", "the maximum percentage of losses (e.g., due to breakage) of a certain resource type should be within a certain range".

In general, if a trace conforms to the corresponding organizational specification (i.e., a trace is in the set of possible executions of the specification), then all constraints imposed on and satisfied by the specification are also satisfied by the trace. However, when the checking of the conformity of a trace to an organizational specification 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 should be performed by checking organization-specific properties. Such properties are based on dependencies and characteristics defined in (implied by) an organizational specification, or correspond to different types of constraints (e.g., domain-specific, physical world constraints) defined for the specification.

Consider several examples:

P1: In the trace $\gamma 1$ the process p1 is executed (after some time) after the process p2 has finished:
$\exists$t1, t2 t1 $\leq$ t2 state(γ, t1) $\models$ process_finished(p2) & state(γ, t2) $\models$ process_started(p1)
For example, it is required that a product produced by an organization should be eventually delivered to the customer.
Other properties expressing ordering relations between processes (also including references to real time) are specified in a similar manner.

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($\exists$a':PROCESS_EX $\exists$am:VALUE_EX state(γ, t')$\models$ [resource_produced_by(r', a', am) $\wedge$ resource(r', r)], am, 0))/sum([γ :TR], case(true, 1, 0)) $\geq$ n,
here between(0, t) represents a set of all natural numbers in the interval [0, t].

P3: In the trace $\gamma 1$ the amount of loss of resources of type r caused by the consumption, usage, and invalidation evaluated at the time point t should be less than m.
sum([t': between(0, t), r':RESOURCE_EX], case($\exists$a': PROCESS_EX $\exists$am1:VALUE state($\gamma 1$, t')$\models$ [resource_produced_by(r', a', am1) $\wedge$ resource(r', r)], am1, 0)) $-$ sum([t':between(0, t), r':RESOURCE_EX], case($\exists$a': PROCESS_EX $\exists$am2:VALUE_EX state($\gamma 1$, t')$\models$ [resource_consumed_by(r', a', am2) $\wedge$ resource (r', r)], am2, 0)) $-$ sum([t':between(0, t), r':RESOURCE_EX], case($\exists$am4:VALUE_EX state($\gamma 1$, t')$\models$ [resource_invalid(r', am4) $\wedge$ resource(r', r)], am4, 0)) $-$ sum([r': RESOURCE_EX], case($\exists$l:PROCESS_LIST_EX $\exists$am2:VALUE_EX state($\gamma 1$, t)$\models$ [resource_used_by(r', l, am3) $\wedge$ resource(r', r)], am3, 0)) $<$ m

P4: In the trace $\gamma 1$ a resource r produced by an organization required by some other organizational processes should be used or completely consumed before its expiration date.
$\exists$t $\exists$p:PROCESS_EX $\exists$am:VALUE_EX state($\gamma 1$, t)$\models$ resource_produced_by(r, p, am) & [$\forall$t' t'>t state($\gamma 1$, t')$\models$ resource_expired(r) $\Rightarrow$ $\exists$t'' $\exists$pl: PROCESS_LIST_EX $\exists$am2 t'>t''& t''>t resource_used_by(r, pl, am2)]

P5: In the trace $\gamma 1$ 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 ($\gamma 1$, t')$\models$ [agent_performs_process(a, p') $\wedge$ process_finished(p')], t', 0)) $-$ sum([t'': between(0, t), p': PROCESS_EX], case(state($\gamma 1$, t')$\models$ [agent_performs_process(a, p') $\wedge$ process_started(p')], t'', 0))) $\leq$ n

P6: In the trace $\gamma 1$ no agent executes more than one process at the same time:
$\forall$p1:PROCESS_EX $\forall$t1 state($\gamma 1$, t1) $\models$ [agent_performs_process(a, p1) $\wedge$ process_started(p1)] $\Rightarrow$ $\exists$t2 state($\gamma 1$, t2) $\models$ process_finished(p1) & $\forall$t' t'$\leq$t2 & t'$\leq$t1 $\forall$p'$\neq$p1 state($\gamma 1$, t')$\models$ ($\neg$process_started(p') $\wedge\neg$agent_performs_process(a, p'))

P7: In the trace $\gamma 1$ at the time point t the amount of available resources of type r is at least a pre-specified minimum amount min.
sum([r':RESOURCE_EX], case($\exists$am1:VALUE_EX state($\gamma 1$, t)$\models$ [available_resource_amount(r', am1) $\wedge$ resource(r', r)], am1, 0)) > min

6.3 Analysis of organizational emergent properties

Emergent properties are not specified and not implied by an organizational specification and are related only to (result from) an actual execution(s) of an organization. Such properties may be checked for different reasons. For example, the analysis of emergent properties may be dedicated to the optimization of the organizational functioning (e.g., discovering and eliminating bottlenecks). Furthermore, emergent properties may be checked to test intuitions of the analytic about the operation of an organization. Many of these properties include the post-processing of the results of checking of dynamic properties by calculating the values of different statistical functions: e.g., sum, average, minimum, maximum etc., and are often expressed over multiple traces.

Consider several examples:

P8: For the specified set of traces TR, determine a frequency of finishing the process p on time (duration should be within the interval [min_duration, max_duration]).
sum([γ :TR], case($\exists$t1,t2 state(γ, t1)$\models$ process_started(p) & state(γ, t2)$\models$ process_finished(p) & (t2 $-$ t1) $\leq$ max_duration & (t2 $-$ t1) $\geq$ min_duration], 1, 0)) / sum([γ :TR], case($\exists$t1 state(γ, t1)$\models$ process_started(p), 1, 0))

P9: In the trace $\gamma 1$ 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($\gamma 1$, t1) $\models$ [agent_performs_process(a', p') $\wedge$ process_finished(p')], t1, 0) $-$ sum([t2: between(0, t), p':PROCESS_EX, a':AGENT_EX], case(state($\gamma 1$, t2)$\models$ [agent_performs_process(a', p') $\wedge$ process_started(p')], t2, 0))) / sum([a':AGENT_EX], case(true, 1, 0))
Maximum and minimum workload is calculated in a similar manner.

P10: Maximum duration of a process p in all executions:
$\exists\gamma 1$, t1, t2 state($\gamma 1$, t1)$\models$ process_started(p) & state($\gamma 1$, t2)$\models$ process_finished(p) & $\forall\gamma' \neq \gamma 1\forall$t1', t2' [state($\gamma'$, t1') $\models$ process_started(p) & state(γ', t2')$\models$ process_finished(p) & (t2'$-$t1')$<$(t2 $-$ t1)]

P11: In all executions the delay between the end of the process p1 and the beginning of the process p2 should be less than n
$\forall\gamma\forall$t1, t2 state(γ, t1) $\models$ process_finished(p1) & state(γ, t2) $\models$ process_started(p2) $\Rightarrow$ (t2 $-$ t1)$<$ n

6.4 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 often concerning the whole organization to more specific goals often defined over separate departments or roles. 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 [18] as part of the performance-oriented view of the general framework. Example of goals can be: 'It is required to maintain high degree of product quality', 'It is required to achieve high customer satisfaction', 'It is required to maintain number of work-related accidents per year to less than 3', 'It is required to achieve productivity of n products per day per employee', etc.

Goals are formulated based on performance indicators (PIs), which are associated with certain organizational processes. PIs can also range from very abstract to very specific and can influence each other which can be specified by relations defined in the performance-oriented view. For more details the reader is referred to [19]. Examples of PIs can be: 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 (an evaluation period defined as a 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 [18], upwards in the goal hierarchy for determining the satisfaction of higher level goals. An example of this type of analysis is given in Sect. 7.4 in the frames of the case study.

7 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 (e.g., private property surveillance, safeguard) to different types of customers (individual, firms

and enterprises). The organization has well-defined structure with predefined (to a varying degree) job descriptions for employees. The total number of employees in the organization is approximately 230 000 persons. The global management of the organization (e.g., making strategic decisions) is performed by the board of directors, which includes among others the directors of the different divisions (regions). Within each region a number of areas exist controlled by area managers. An area is divided into several units, controlled by unit managers. Within each unit exist a number of locations, for which the contracts with customers are signed and security officers are allocated. The allocation of employees is performed based on plans created by planning groups.

The analysis techniques will be illustrated in relation to the planning process, which is described in the following in more detail.

The planning process consists of the forward (or long-term) planning and the short-term planning. The forward planning is a process of creation, analysis and optimization of forward plans that describe 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 a certain 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. Furthermore, 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. During the planning process short-term planners interact actively with forward planners (e.g., for consultations, problem solving). Furthermore, forward planners have a number of supervision functions with respect to short-term planners.

The position of forward planners in the organizational structure has changed as a result of the reorganization in the past. Before the reorganization each planning team had a forward planner as its member, who was mainly responsible for the creation of long-term plans for locations of the area. After the reorganization forward planners from area planning teams were combined into a centralized forward planning group, which now cooperates with all area planning teams.

A number of reasons for such a change in the organizational structure are identified in the reorganization reports. In the following it will be shown how the proposed analysis techniques could be used for the automated justification of the identified performance bottlenecks and other problems in the organization.

The company's reorganization reports specify the following motivations for the necessity of such reorganization:

(1) Uneven workload of forward planners in different area planning teams.
The truth of this statement can be established 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) $\models$ [agent_performs_process(a, p') $\wedge$ process_finished(p')], t1, 0)) − sum([t2: between(0, t), p':PROCESS_EX], case(state(γ1, t2)$\models$ [agent_performs_process(a, p') $\wedge$ process_started(p')], t2, 0))
If multiple traces are available, the average workload of every agent can be calculated as it is demonstrated in the property P9.
A side-effect of a high workload of a forward planner 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 of an organization to the corresponding specification.

(2) Certain planning tasks of a forward planner require collaboration with other forward planners. In the previous organization this has been achieved by informal (i.e., not determined by a formal organizational specification) 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 specification, 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 that started 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) $\models$ [agent_performs_process(a, p') $\wedge\neg$agent_performs_role(a1, r)], 1, 0))

If multiple traces (a set TR) are available, 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) $\models$ [agent_performs_process(a, p') $\wedge\neg$agent_performs_role(a1, r)], 1, 0)/$\sum$([γ :TR], case (true, 1, 0))

(3) Planning activities and data in each area were isolated from each other. Sometimes this lead to situations, when customer requests in one area were not satisfied because of the deficiency 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) $\models$ env_object_changed_state_into(r', active) & $\forall$t2 t2>t1 state(γ1, t2) $\models$ $\neg$env_object_changed_state_into(r', satisfied), 1, 0))

In the following subsections 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.

7.1 Organizational specification and correspondence of a trace to this specification

Based on company documents such as job descriptions, company policy, procedures, etc., a process-oriented specification was created for the planning departments. Part of this specification dedicated to the creation of daily plans and short-term plans is considered here. It describes the work performed with respect to these plans within one day. In the first half of the day security employees should provide their data change forms (containing requests for changes in the allocation schedule) to the unit manager (defined as process p3) who then checks and improves the data (process p4) and puts it in the system (p5) so that it becomes available to the planners. At the same time the planners are busy with other tasks, for example during the last week of the month they are busy with creating 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 (taking into account the available data change information in the system), inputting it in the system (p7) and informing all concerned (p8). Then they update the current-month short-term plan if necessary (p9) and so on. Part of the specification 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(end_and(and1), p5, 0)
starts_after(end_and(and1), end_or(or1), 0)
starts_after(and1, all)
starts_after(p6, end_and(and1), 0.5)
...
role_performs_process(sec_officer, p3)
role_performs_process(planner, p1)
role_performs_process(planner, p2)
role_performs_process(unit_manager, p4)
role_performs_process(unit_manager, p5)
...
is_instance_of(p1, t1)
task_produces(t1, STP, 1)
t1. min_duration = 3.5h
t1.max_duration = 4h
...

Based on this specification constraints can be generated as discussed in Sect. 6.1. For example the first few lines of the specification generate the following constraints for the first time point of any execution trace:

state(γ, 0) $\models$ process_started(p3) (based on C2)
state(γ, 0) $\models$ process_started(p2) & ($\forall$t3 state(γ, t3) $\models$ $\neg$process_started(p1)) & state(γ, 0) $\models$ $\neg$env_object_changed_state_into(week, last) | (state(γ, 0) $\models$ process_started(p1) & ($\forall$t3 state(γ, t3) $\models$ $\neg$process_started(p2)) & state(γ, 0) $\models$ env_object_changed_state_into(week, last) (based on C18)

$\forall$p:PROCESS_EX state(γ, 0) $\models$ process_started(p) $\Rightarrow$ p = p1 | p = p2 | p = p3 (based on C1)

Also based on company documents traces were created corresponding to this part of the specification. One of these traces is used to illustrate the analysis of whether an execution trace agrees with the specification. 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 which here consists of 12 hours. The time line shown is relative to the trace and not expressed in absolute date and time stamps. The absolute time line can always be calculated if necessary 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, process_started(p1) is only true for time point 0 and agent a1 performs process p1 for the whole duration of the process from time point 0 to time point 1.

The trace in Fig. 1 contains a process that is not in the specification, namely 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 therefore 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 (which were considered urgent) 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 specification prescribed.

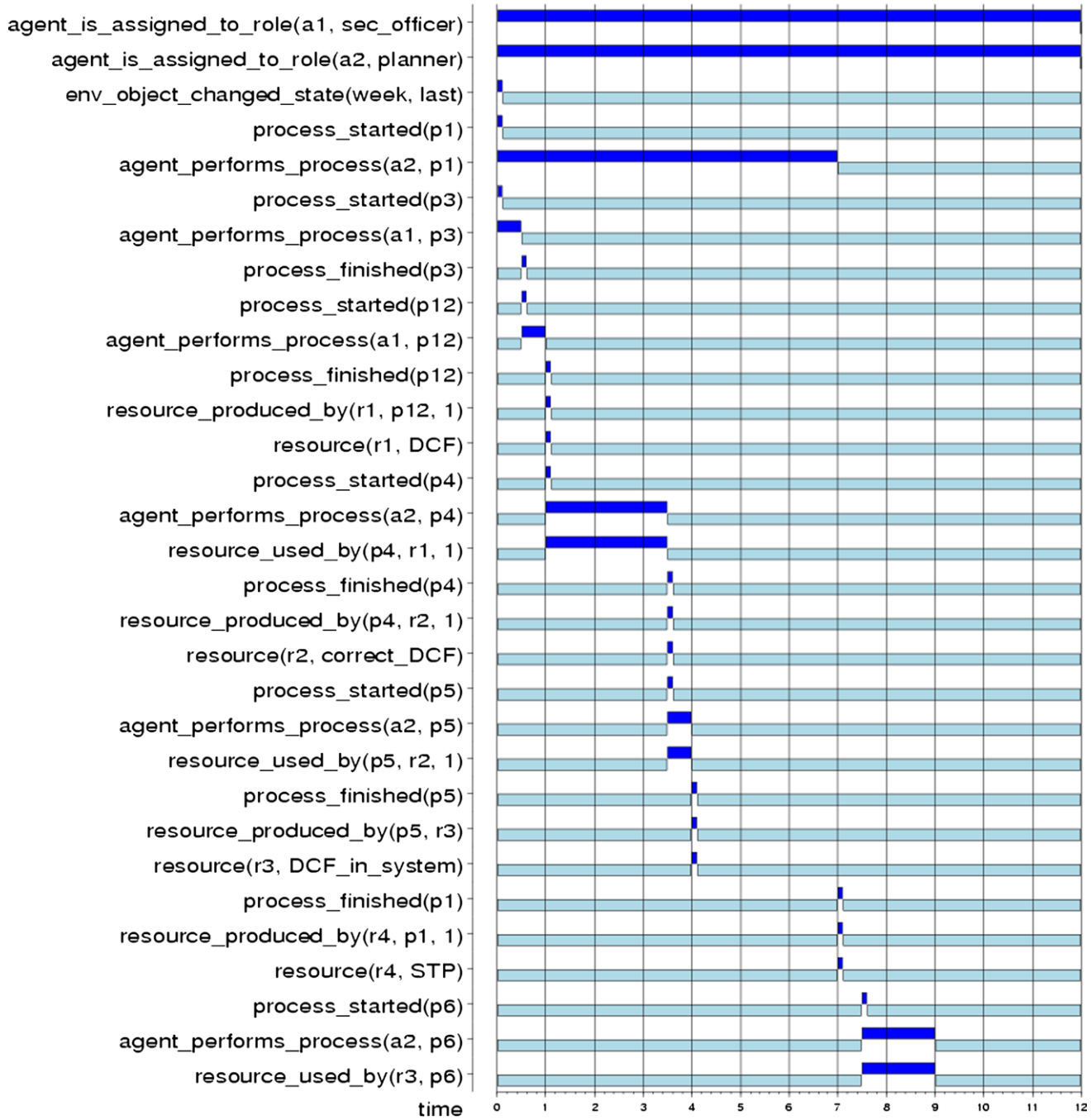

Fig. 1 The execution trace used for illustration

According to the approach presented in Sect. 6.1 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 so that only the part of the trace up to the current time point is available. At time point 0 first the three instantiated constraints given above are checked. They are all 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'. The analysis continues following the order defined in Sect. 6.1. Steps 3 and 4 from the list are not relevant because no such relations are defined in the specification. For step 5 the following properties are scheduled to be checked for every time point t until they are satisfied:

$\text{state}(\gamma, t) \models \text{process\_finished(p1)}$
$\text{state}(\gamma, t) \models \text{process\_finished(p3)}$

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 should be according to the specification:

state(γ, t) $\models$ process_finished(p1) $\Rightarrow$ t $\geq$ 3.5
state(γ, t) $\models$ process_finished(p1) $\Rightarrow$ t $\leq$ 4
state(γ, t) $\models$ process_finished(p3) $\Rightarrow$ t = 1

If any of these three properties fail, then the corresponding constraints are considered violated.

At step 6 resource-related constraints are scheduled. Here the only relevant resource is produced by p3 and is the collection of data change forms DCF which is considered as a whole and only one such collection can be produced. Therefore C23 is not relevant in this case.

For step 7, agent- and role-related constraint C26 is scheduled for checking for every time point t until the process finishes.

state(γ, t) $\models \neg$process_finished(p1) $\Rightarrow$ state(γ, t) $\models$ [agent_plays_role(a2, planner) $\wedge$ agent_performs_process(a2, p1)]
state(γ, t) $\models \neg$process_finished(p3) $\Rightarrow$ state(γ, t) $\models$ [agent_plays_role(a1, sec_officer) $\wedge$ agent_performs_process(a1, p3)]

From all the scheduled constraints one fails at time point 0.5—since at this time point process p3 finishes, its duration is below the specified minimal duration of 1 hour. At this step the analysis is stopped with the conclusion that the trace does not agree with the specification and the first process that violates the constraints is p3.

7.2 Formal organization properties

As it was discussed above, the trace from Fig. 1 does not agree with the specification. However this type of analysis does not elaborate on whether and which important organizational properties are satisfied. 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. It can be expressed as follows:

$\exists$t, p:PROCESS_EX, r:RESOURCE_EX
state(γ, t) $\models$ [resource_produced_by(r, p) $\wedge$ 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. The property reflects the comparative level of urgency of the two processes and can be expressed as follows:

$\exists$t1, t2, p:PROCESS_EX, r:RESOURCE_EX
state(γ, t1) $\models$ [resource_produced_by(r, p) $\wedge$ resource(r, daily_plan)] & state(γ, t2) $\models$ process_started(p9) $\Rightarrow$ t1 $\leq$ t2

This property is also satisfied.

7.3 Emergent Properties

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 (in this part of the model) is less than 3 hours. This can be checked by the following property:

sum([p:PROCESS_EX], case($\exists$t1, t2 state(γ, t1) $\models$ [process_started(p) $\wedge$ agent_performs_process(a, p) $\wedge$ agent_performs_role(a, unit_manager)] & state(γ, t2) $\models$ process_finished(p), t2 $-$ t1, 0)) $<$ 3

Various other properties might be meaningful to check depending on the situation, for example: how long does it take to produce a new short term plan where the sum of the durations of the processes producing STP resource is found over the traces for the days of the last week of the month and so on.

7.4 Performance evaluation

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) $\models$ pi_has_value(P1, v) $\Rightarrow$ v $\leq$ 8

For the example trace in Fig. 1 it will be calculated (see property 1 at the beginning of Sect. 7) and included at the end of the trace that pi_has_value(P1, 11) which is more than 8. Therefore goal G1.1.1 is not satisfied and it contributes negatively to the satisfaction of G1.1. The satisfaction values are propagated upwards in the goals structure according to the rules defined in [18].

8 Discussion

This paper introduces automated techniques for manifold formal analysis of actual executions based on specifications 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 (e.g., by identifying and eliminating bottlenecks). Furthermore, the results of the proposed analysis procedures may be used for proving the validity of process-oriented specifications.

For the proposed analysis techniques the TTL language and the dedicated 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 (e.g., situation and event calculus). The proposed analysis approaches and languages are related to the process-oriented view on organizations, which also includes a formal language for building process-oriented specifications and constraints imposed on these specifications. Properties to be checked on actual executions of organizational scenarios (traces) are obtained by the translation procedure of a process-oriented specification, described in this paper. The translation is performed only once and the resulting properties can be checked on every new-coming trace. The translation is based on clear translation rules that can easily be automated.

Also Petri-net-based approaches for checking conformance of a trace to a process-oriented specification have been developed. In particular, in the approach described in [22] to establish (the degree of) correspondence of a trace (event log) to a Petri-net-based process-oriented specification, the trace is executed in the model by following the transitions that belong to the logged events in the trace. Then, the measure of conformance is calculated based on the number of tokens that had to be created artificially and the number of tokens that had been left in the specification (which indicates the process not having properly completed). In contrast to this approach, the technique described in this paper does not require execution of a trace on a specification every time when correspondence is checked. On the other hand, the Petri-net-based approach does not perform a specification translation into properties to be checked on traces, as the proposed method does. However, by checking the properties obtained as the result of a translation, the proposed method allows precise identification of inconsistency sources instead of a generalized conformance measure offered by the Petri-net-based approach. The central ideas of the approach from [22] are very similar to the research described in [8].

Approaches based on state machines are also used to check correspondence between a trace and a process-oriented specification. In such approaches state machines encode sets of possible executions of organizational scenarios and state transitions correspond to events [7]. Thus, the problem of establishing correspondence of a trace to a process-oriented specification is reduced to the problem of determining, whether a trace represents an execution of the state machine (i.e., process-oriented model). One of the approaches to achieve this is by processing the trace from the beginning and checking whether each states and transition of the trace are in accordance with the process-oriented specification. This type of approaches is similar to the Petri-net-based approaches considered above. Another approach is based on formulating properties over states (usually expressed in standard modal temporal logics CTL and LTL) and checking whether they are satisfied by the trace [17]. In this case properties to be checked stem from the process-oriented specification using some translation/abstraction mechanism. It should be noted that the above mentioned approaches concentrate exclusively on process-oriented concepts while the approach presented in this paper is based on a multi-faceted modeling framework including also performance-, organization- and agent-related concepts and relations. This allows more extensive analysis, an example to which is the performance evaluation analysis described in this paper.

In [10] an approach is proposed for enacting distributed business workflows using BPEL4WS on multi-agent platforms. In this approach Lightweight Coordination Calculus (LCC) [21] is used to interpret workflow specifications. These specifications describe executions of organizational scenarios, which are actually realized by agents. This is an essential distinction from our work, in which agents may demonstrate behavior deviating from the dynamics described by a process-oriented specification. The main contribution of our work is a repertoire of analysis means to identify such behavioral deviations of agents and to establish their consequences for the organization (e.g., for its performance). Furthermore, LCC is an imperative language for representing dialogues between agents that comprise sending and receiving messages. The process-oriented language described in this paper allows specifying not only flows of control, but also more specific process-oriented aspects of organizations (e.g., resources and their relations to processes and actors), not captured by LCC.

In the area of multi-agent systems, norms are employed often to regulate and coordinate the behaviors of autonomous agents interacting in open environments [14, 16]. Norms are essentially rules that can be categorized as permissions, obligations and prohibitions and describe respectively what may be done, what should be done, and what should not be done. Normative systems are formalized often using deontic logic [11, 14]. The language TTL as a variant of reified order-sorted predicate logic, used for the formalization of properties in this paper, has a higher expressive power than any variant of deontic logic. Furthermore, the types of properties being checked on traces in the proposed approach are not restricted to permissions, obligations and

prohibitions only. In particular, diverse causal relations on states of processes can be specified. Moreover, as shown in the paper, a set of properties to check the trace conformity to an organizational specification is identified based on an organizational specification in a systematic manner and its completeness can be ensured. On the contrary, the frameworks from normative systems literature known to the authors are not obtained from formal organizational specifications (e.g., workflow models) and the issue of completeness of sets of norms is not discussed in this literature.

Monitoring and verifying of violation of norms is done usually using automata- and graph-like structures [13, 16]. This type of verification differs from temporal interval checking performed by the TTL Checker tool. For more details about the computational complexity we refer to [6].

The analysis techniques introduced in this paper can be applied to both mechanistic and organic organizations [24]. In particular, since many mechanistic organizations are characterized by a high stability and a large number of routine processes that can be specified with high precision, the verification of the conformity of actual executions of such organizations to a formal organizational specification is of special importance. At the same time organic organizations are highly dynamic and their processes are very flexible, variable and often unpredictable. Therefore, models for such organizations can be specified only at a high abstraction level, sometimes defining only interface states (i.e., inputs and outputs) of high-level processes, and then the analysis techniques for the evaluation of emergent organizational processes and performance can be applied.

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 based on recorded information about environmental events; (2) if data about the execution are represented in the form of informal logs created based on a process-oriented specification 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 the specification based on which an original trace is generated can be related to an equivalent specification in L_{PR}.

Traces can also be generated based on a process-oriented specification by performing simulations. Such traces can be used for diagnosis of inconsistencies, redundancies and errors in organizational structure and behavior. For this type of analysis dedicated software is provided [6], in which different scenarios of organizational behavior based on process-oriented specifications are built and traces are generated. Further analysis of these traces is performed in the TTL Checker as it is described in this paper. This type of analysis is discussed in more detail in [9].

In the cases when a process-oriented specification is not (or partially) known or when a trace represents a significant extension of the specification based on which it was generated, process mining techniques may be used [1]. Using these techniques information about the original process model can be derived from execution traces.

Event logs (traces) can also be used for determining the equivalence of process-oriented specifications—in [4] such an approach in the context of Petri-nets is presented.

In the future it will be investigated how the proposed techniques can be applied for the analysis of inter-organizational processes. Also more analysis cases supported by the proposed techniques will be performed in the context of real organizations.

References

1. van der Aalst W, Beer H, van Dongen B (2005) Process mining and verification of properties: an approach based on temporal logic. In: Meersman R, et al (eds) On the move to meaningful Internet systems 2005: CoopIS, DOA, and ODBASE: OTM confederated international conferences, CoopIS, DOA, and ODBASE 2005, vol 3760. Springer, Berlin, pp 130–147
2. van der Aalst W, van Hee KM (2002) Workflow management: models, methods, and systems. MIT Press, Cambridge
3. Adam NR, Atluri V, Huang W-K (1998) Modeling and analysis of workflows using petri nets. J Intell Inf Syst 10(2):131–158. Special Issue on Workflow and Process Management
4. Alves de Medeiros AK, van der Aalst WMP, Weijters AJMM (2008) Quantifying process equivalence based on observed behavior. Data & Knowl Eng 64(1):55–74
5. Barjis J, Shishkov B, Dietz J (2002) Validation of business components via simulation. In: Proceedings of the 2002 summer computer simulation conference
6. Bosse T, Jonker CM, Meij L, van der Sharpanskykh A, Treur J (2009) Specification and verification of dynamics in agent models. Int J Coop Inf Syst 18(1):167–193
7. Cook J, He C, Ma C (2001) Measuring behavioral correspondence to a timed concurrent model. In: Proceedings of 2001 international conference on software maintenance, pp 332–341
8. Cook JE, Wolf AL (1999) Software process validation: quantitatively measuring the correspondence of a process to a model. ACM Trans Softw Eng Methodol 8(2):147–176
9. Desel J, Juhas G, Lorenz R, Neumair C (2003) Modelling and validation with VipTool. In: Lecture notes in computer science, vol 2678. Springer, Berlin, pp 380–389
10. Guo L, Robertson D, Chen-Burger Y-H (2005) A novel approach for enacting the distributed business workflows using BPEL4WS on the multi-agent platform. In: IEEE international conference on e-business engineering (ICEBE'05), pp 657–664
11. Horty JF (2001) Agency and deontic logic. Oxford University Press, Oxford
12. Jonker CM, Sharpanskykh A, Treur J, Yolum P A Framework for formal modeling and analysis of organizations. Appl Intell 27(1) 49–66

13. Kyas M, Prisacariu C, Schneider G (2008) Run-time monitoring of electronic contracts. In: 6th international symposium on automated technology for verification and analysis (ATVA'08). Springer, Berlin, pp 397–407
14. Krogh C (1996) The rights of agents. In: Wooldridge M, Muller JP, Tambe M (eds) Agent theories, architectures, and languages II, vol 1037. Springer, Berlin, pp 1–16
15. Manzano M (1996) Extensions of first order logic. Cambridge University Press, Cambridge
16. Modgil S, Faci N, Meneguzzi F, Oren N, Miles S, Luck M (2009) A framework for monitoring agent-based normative systems. In: Proceedings of the eighth international conference on autonomous agents and multi-agent systems (AAMAS'09), Budapest, Hungary. ACM, New York
17. Pfeiffer J-H, Rossak WR, Speck A (2004) Applying model checking to workflow verification. In: Proceedings of the 11th IEEE international conference and workshop on the engineering of computer-based systems (ECBS'04), pp 144–152
18. Popova V, Sharpanskykh A (2007) Formal modelling of goals in agent organizations. In: Proceedings of AOMS@IJCAI workshop, pp 74–86
19. Popova V, Sharpanskykh A (2007) Modeling Organizational Performance Indicators. In: Barros F, Frydman C, Giambiasi N, Zeigler B (eds) Proceedings of international modeling and simulation multiconference, pp 165–170
20. Popova V, Sharpanskykh A (2008) Process-oriented organisation modelling and analysis. Enterp Inf Syst J 2(2):157–176
21. Robertson D (2005) A lightweight coordination calculus for agent social norms. In: Proceedings of the autonomous agents and multi-agent systems workshop on declarative agent languages and technologies. Lecture notes in computer science, vol 3476. Springer, New York, pp 183–197
22. Rozinat A, van der Aalst WMP (2008) Conformance checking of processes based on monitoring real behavior. Inf Syst 33(1):64–95
23. Saake G (eds) (1998) Logics for databases and information systems. Kluwer Academic, Dordrecht, pp 117–166
24. Scott WR (2001) Institutions and organizations, 2nd edn. SAGE Publications, Thousand
25. Singh MP (1996) Synthesizing distributed constrained events from transactional workflow specifications. In: Proceedings of the 12th IEEE international conference on data engineering, pp 616–623
26. Sharpanskykh A, Treur J (2006) Verifying interlevel relations within multi-agent systems. In: Proceedings of the 17th European conference on artificial intelligence, ECAI'06. IOS Press, Amsterdam
27. Sharpanskykh A (2008) On computer-aided methods for modeling and analysis of organizations. PhD Dissertation, VU University Amsterdam

Viara Popova is a Research Fellow at the Centre for Manufacturing, De Montfort University, UK. She received her MSc degree in Computer Science at Sofia University, Bulgaria, and a PhD degree at Erasmus University Rotterdam in the area of knowledge discovery. Subsequently she worked as a post-doctoral researcher at Vrije Universiteit Amsterdam in the area of modeling and analysis of multi-agent and human organizations with a focus on logistics and incident management. Her current research interests include organization modeling, prediction of demand in manufacturing, knowledge discovery and natural language processing.

Alexei Sharpanskykh received his PhD degree at the Vrije Universiteit Amsterdam in the area of Artificial Intelligence. Currently he is working as a post-doctoral researcher at the same university. He is doing research in modeling and analysis of multi-agent organizations in the context of a number of projects in the areas of logistics, incident management and ambient intelligence.