Process-oriented Organization Modeling and Analysis

Viara Popova and Alexei Sharpanskykh

Vrije Universiteit Amsterdam, Department of Artificial Intelligence
De Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands

Abstract. This paper presents a formal framework for process-oriented modeling and analysis of organizations. The high expressivity of a sorted predicate logic language used for specification allows representing a wide range of process-related concepts (e.g., tasks, processes, resources), characteristics and relations, which are described in the paper. Furthermore, for every organization, structural and behavioral constraints on process-related concepts can be identified. Some of them should always be fulfilled by the organization (e.g., physical world constraints), whereas others allow some degree of organizational flexibility (e.g., some domain specific constraints). An organizational model is correct if it satisfies a set of relevant organizational constraints. This paper describes automated formal techniques for establishing correctness of organizational models w.r.t. a set of diverse constraint types. The introduced framework is a part of a general framework for organization modeling and analysis.

1 Introduction

Every organization achieves its goals by performing some set of tasks. The execution of tasks is often specified by dynamic structures called flows of control (or workflows), in which tasks are represented by processes. Usually control flows are based on a set of temporal ordering rules over processes. Tasks and processes are also related to other organizational concepts, such as resources, roles, agents (actors). Modern organizations are characterized by a great variety of such relations. To handle the high complexity of modern organizations, automated formal modeling and analysis techniques are indispensable.

To this end, this paper introduces a formal framework for process-oriented modeling and analysis. In this framework, tasks, processes, resources and other related concepts are specified in the formal language L_{PR}, based on the sorted first-order predicate logic [11]. The high expressivity of predicate logic allows including into L_{PR} a wide range of process-oriented concepts specified by sorts, sorted constants, variables, functions and predicates that represent relations on these concepts.

For every organization a set of structural and behavioral *constraints* expressed over its tasks and processes can de identified, which should be satisfied by the process-oriented model. In this paper the set of constraints is represented by the *logical theory* T_{PR} in L_{PR}, i.e., a set of sentences expressed in L_{PR}. It means that all concepts and relations defined in L_{PR} may be used for the specification of constraints. A process-oriented model specified in L_{PR} is *correct* if T_{PR} is satisfied by this model. The constraints in T_{PR} may be of different types: some are dictated by the restrictions of the physical world and should be satisfied by any process-oriented model; others depend on the application domain and may be changed by the designer. The classification of

Popova V. and Sharpanskykh A. (2007).
Process-oriented Organization Modeling and Analysis.
In *Proceedings of the 5th International Workshop on Modelling, Simulation, Verification and Validation of Enterprise Information Systems*, pages 114-125
DOI: 10.5220/0002422801140125

constraints is described in this paper. This paper also introduces automated techniques for establishing the correctness of a process-oriented model by verifying constraints. Interdependences that may exist in constraint sets are also handled by the proposed verification techniques. To our knowledge there exist no other frameworks that allow the simultaneous verification of different (interdependent) types of constraints based on the extensive set of concepts and relations as can be found in L_{PR}.

The proposed framework has some similarities with and distinctions from other process modeling approaches [5, 6, 8, 9, 18]. In particular, our framework realizes the most commonly used workflow patterns identified in [18] extended with time parameters (e.g., sequence and parallel execution, process synchronization, loops). At the same time, in comparison with other approaches [3, 9], the proposed framework provides a more extensive means for resource modeling (shared resources in particular), which will be addressed further in this paper.

A number of informal and semi-formal frameworks [5, 6, 8, 12] provide a rich ontological basis for processes-oriented modeling. However, the lack of formal foundations creates a significant obstacle for performing computational analysis in such frameworks. Some of the existing formal process-oriented analysis techniques are dedicated to the verification of constraints of a particular type only. For example, in [15] process algebra is used to model and analyze ordering constraints on processes, however it lacks the expressivity to represent global ordering constraints on processes. This deficiency is addressed in [10], however this approach does not allow the specification of (real) numbers (i.e., durations). A technique that provides means for reasoning on temporal numerical constraints on processes is described in [4]. An approach for resource constraints analysis is proposed in [9], however it does not address interdependences that may exist between constraints. The problem of establishing the correctness of a workflow with shared resources is considered in [3]. Some analysis techniques based on Petri-nets and their modifications take into account both ordering and resource-related constraints [17]. However it is difficult to express constraints over multiple objects, characteristics and relations of the organization (e.g., many physical world and domain-specific constraints considered in this paper) using Petri Nets. Furthermore, the proposed framework provides more expressive language for specifying constraints than the mentioned approaches.

Often logic-based process analysis techniques [2] employ general-purpose methods for verifying models (e.g., model checking [7]), which have a high computational cost. This paper proposes more efficient algorithms dedicated for verifying particular types of constraints.

The framework introduced in this paper constitutes a part of a general formal framework for organization modeling and analysis in which organizations are considered from other perspectives (or views) as well. In particular, *the performance-oriented view* [14] 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. The views are related to each other by means of sets of common concepts. This enables different types of analysis across multiple views. An example of such analysis involving the process- and performance-oriented views is considered in [14].

The paper is organized as follows. Section 2 briefly introduces the language L_{PR}. Section 3 describes the classification of constraints. In Section 4, the methods for verification of constraints are given. The proposed approach is illustrated by an example in Section 5. Section 6 concludes the paper.

2 Process-oriented Modeling

Process-oriented models in the proposed framework are specified using the sorted predicate language L_{PR}. Due to the space limitation, only a general overview of L_{PR} is given in this Section. For all formal details of the language we refer to [13].

To illustrate different aspects of L_{PR} a simplified example is used that describes the operation of a 3PL (third-party logistics) provider (see Fig.1). In general, 3PL companies provide logistics services to other companies. The considered operation cycle begins with the customer order intake process, after which the order is processed and depending on the customer (company) is scheduled for some delivery type (for different companies different delivery regulations may be applied). During the delivery the assigned driver is supervised by the assigned fleet manager. After the delivery is finished, the delivery summary report is provided to the customer.

A *task* represents a function performed in the organization and is characterized by a name and by a maximal and a minimal durations. Tasks can be decomposed into more specific ones using AND- and OR-relations thus forming hierarchies.

A *workflow* is defined by a set of (partially) temporally ordered *processes*. Each process, except for the special ones with zero duration introduced below, is defined using a task as a template and all characteristics of the task are inherited by the process. Decisions are also treated as processes that are associated with decision variables taking as possible values the possible decision outcomes.

Definition 1 (A workflow): A workflow with the name w is defined by a tuple <w, P, C> with a set of processes P and a set of ordering relations C on processes from P.

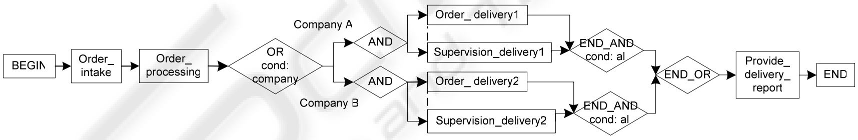

Fig. 1. The generalized workflow that illustrates the operation of a 3PL delivery company.

A workflow starts with the process BEGIN and ends with the process END; both have zero duration. Only one workflow can be defined in a process-oriented model. The (partial) order of execution of processes in the workflow is defined by sequencing, branching, cycle and synchronization relations (referred to as ordering relations) specified by the designer. Fig.1 is a graphical representation of the workflow built for the running example. A *sequencing relation* is specified by the predicate starts_after: PROCESS × PROCESS × VALUE expressing that the process specified by the first argument starts after the process specified by the second argument with the delay expressed by the third argument, e.g., starts_after(Order_processing, Order_intake, 0) represented graphically by solid arrows between the processes. *Synchronization relations* define temporal relations between processes that are executed in parallel (e.g., starts_with, finishes_with, starts_during). An example of such a relation is shown by a dashed line between the beginnings of the processes in Fig. 1, meaning that the connected processes should start simultaneously. Taken together, synchronization and sequencing relations allow specifying all cases of interval relations defined by Allen [1].

Branching relations are defined over and- and or-structures. An and(or)-structure with name id, starts with the zero-duration process begin_and(id) (begin_or(id)) and finishes by the zero-duration process end_and(id) (end_or(id)). These special processes are represented graphically by rhombuses. Our treatment of AND-structures is similar to the parallel split pattern combined with all types of the merge pattern from [18], represented in our case by an and-condition. In Fig.1 and-structures contain the condition value all, meaning that only when all processes in the and-structure are finished, the process specified after the end of the and-structure is allowed to start.

For every or-structure a condition is defined, based on which it is determined which branches of the or-structure will start. The condition may consist only of a condition variable. Our treatment of or-structures allows realizing both exclusive and multiple-choice patterns from [18]. The or-structure in the running example specifies the exclusive choice between two types of delivery depending on the company name.

Cycle relations are defined over *loop*-structures with conditions that realize cycle patterns from [18]. For every loop-structure a Boolean condition and the maximal number of times of the loop execution are specified.

Tasks use, consume and produce resources of different types. Resource types include tools, supplies, components and other material or digital artifacts. Also data are considered as a special resource type. Resource types are characterized by: *name*; *category* – discrete or continuous; *measurement_unit*; *expiration_duration* – the length d of the time interval for which a resource type can be used. Specific resources represent instances of particular resource types and inherit their characteristics. The resources have, in addition to the inherited characteristics, also *name* and *amount.* Every resource in the workflow has to be produced by a process of this workflow or be available in the organization before the beginning of the workflow execution.

Some resources can be shared (used simultaneously) by a set of processes (e.g., storage facilities, transportation vehicles, some computers). The shared amount of the resource should be sufficient for the execution of every process in the set. Our representation of shared resources is different from [3] in several aspects: (1) the shared resource amount is used by processes simultaneously; (2) alternative sets of processes that are allowed to share a resource can be defined; (3) different amounts of a resource can be shared simultaneously; (4) specific conditions (requirements) for resource sharing can be defined. In the running example, for certain delivery processes certain trucks may be considered as shared resources.

The process-oriented view is related to the *organization-oriented* and the *agent-oriented views* through the sorts ROLE and AGENT. Each object of the sort ROLE describes a set of functionalities realized by organizational processes in a certain model, which are assigned together to individuals who will be performing them. These individuals are objects of the sort AGENT. An agent can be allocated to one or more roles if it satisfies the requirements for performing these roles. For example, role_performs_process(Driver,Order_delivery1) and agent_plays_role(Allan, Driver).

3 Constraints

Constraints are expressed as formulae in theory T_{PR} that are constructed from terms of L_{PR} in a standard way [11] using Boolean connectives and quantifiers over variables.

The constraints are divided in two groups: (1) *generic constraints* need to be satisfied by any model built using this framework; (2) *domain-specific constraints* are dictated by the application domain of the model. Two types of generic constraints are considered: (1) structural constraints used to ensure correctness of the workflow, task and resource hierarchies; (2) constraints imposed by the physical world. Both types of generic constraints are described in Sections 3.1. Section 3.2 discusses the domain-specific constraints.

3.1 Generic Constraints

The language allows building three types of structures: the workflow, the task hierarchy and the resource hierarchy. For each of them structural constraints are defined.

Workflow Structural Constraints. With respect to the workflow we define a set of structural constraints: structural correctness, temporal correctness and condition correctness constraints.

Structural correctness of the workflow
First let us introduce *reachability* and *complete reachablitity* relations.

Definition 2 (Reachability relation): The process p2 is reachable from the process p1 in the workflow w (reachable_from_in(p2, p1, w)) if there exists a sequence of processes constructed using the sequencing relations that starts at p1 and includes p2.

The algorithmic procedures for establishing the truth values of the reachability relation and all the following relations are given in [13].

Definition 3 (Complete reachability relation): The process p2 is completely reachable from the process p1 in the workflow w (completely_reachable_from_in(p2, p1, w)) if all process sequences built using the sequencing relations that start at p1 include p2.

For example, the process Provide_delivery_report from the running example in Section 2 is completely reachable from the process Order_intake.

Definition 4 (A well-formed and-structure): An and-structure with the name and_id defined in the workflow <w, P, C> is well-formed if the following constraints hold:

(1) ∃p∈P such that p = begin_and(and_id); ∃p∈P such that p = end_and(and_id);
(2) completely_reachable_from_in(end_and(and_id), begin_and(and_id), w) is true.

Well-formed or- and loop-structures are defined similarly. Both and- and or-structures defined in the running example in Section 2 are well-formed.

Now, the structural correctness property for a workflow can be introduced.

Definition 5 (A structurally correct workflow): A workflow <w, P, C> is structurally correct if the following constraints are satisfied:

(1) A workflow contains only one BEGIN (the first process), followed by one process and only one END (the last process) preceded by one process. Formally:
(a) ∃s∈P s=BEGIN; ∃s∈P starts_after(s, BEGIN) ∧ (∀s1∈P starts_after(s1, BEGIN) ⇒ s1=s)
(b) ∃s∈P s=END; ∃s∈P starts_after(END, s) ∧ (∀s1∈P starts_after(END, s1) ⇒ s1=s)
(c) ¬∃s∈P starts_after(BEGIN, s); ¬∃s∈P starts_after(s, END)

(2) For every process p, different from BEGIN, END, and the starting and ending processes for and- and or-structures, exactly two sequencing relations should be defined that identify the process that precedes p and the process that follows after p:
(a) ∃s∈P starts_after(p, s) ∧ (∀s1∈P starts_after(p, s1) ⇒ s1=s)
(b) ∃s∈P starts_after(s, p) ∧ (∀s1∈P starts_after(s1, p) ⇒ s1=s)

(3) Loops should be introduced only by loop-structures, no other cycles are allowed:
$\forall$p1, p2$\in$P reachable_from_in(p1, p2, w) $\Rightarrow \neg$reachable_from_in(p2, p1, w)
(4) Processes, over which a synchronization relation is specified, should not belong to the same or-structure.
(5) All and-, or- and loop-structures in w are well-formed and each process p$\in$P can be reached from the BEGIN, and the END can be reached from p:
$\forall$p$\in$P reachable_from_in(p, BEGIN, w) $\wedge$ reachable_from_in(END, p, w)

The workflow defined by the running example in Section 2 is structurally correct.

Temporal correctness of the workflow
The duration of each process in a workflow may vary in actual executions and, because of the temporal ordering of processes in the workflow, each process may have different starting points in different executions. Among all starting points the earliest (est_p) and the latest starting time (lst_p) for each process p can be identified. The value for est_p (lst_p) is calculated under the assumption that all relevant processes (i.e., processes that may influence the starting time of p) have minimal (maximal) durations. A more detailed description of the calculation procedure is given in [13].

The earliest (latest) ending time point of the process p (eet_p (let_p)) is calculated as est_p + p.min_duration (lst_p + p.max_duration). Then, the earliest (latest) creation time of the resource r (ect_r (lct_r)) produced by p are defined as: $ect_r = eet_p$ and $lct_r = let_p$, and the earliest (latest) expiration time of r (eet_r (let_r)) is calculated as: $eet_r = ect_r$ + r.expiration_duration ($let_r = lct_r$ + r.expiration_duration).

Synchronization relations defined in a model may influence the starting time of processes in this model. Moreover, some sequencing/branching/cycle relations of the model may be in conflict with synchronization relations introduced by the designer. Let us define $[t1, t2] = [est_p, lst_p] \cap [est_s, lst_s]$ and $[t3, t4] = [eet_p, let_p] \cap [eet_s, let_s]$ for processes p and s. A *conflict* occurs in following cases: (a) if starts_with(p, s) is introduced and $[t1, t2] = \varnothing$; (b) if starts_during(p, s) is introduced and $[t1, t2] = \varnothing$; (c) if finishes_with(p, s) is introduced and $[t3, t4] = \varnothing$.

Definition 6 (A temporally correct workflow): A workflow <w, P, C> is temporally correct in the model M if the set of ordering relations in M is not conflicting.

If a workflow is temporally correct, starting points of processes influenced by the introduced synchronization relation are updated, using the values t1, t2, t3 and t4 defined above: (a) in case of starts_with(p, s) assign est_p=t1; est_s =t1; lst_p =t2; lst_s=t2; (b) in case of starts_during(p, s) assign est_p =t1 and lst_p=t2; (c) in case of finishes_with(p, s) assign eet_p=t3; eet_s=t3; let_s=t4; let_s=t4. Then, update the values of the earliest (latest) starting points for the processes reachable from p and for the processes reachable from s.

Condition correctness
This property concerns conditions specified for or- and loop-structures of a model.

A condition of the or-/loop-structure is *correct* iff:

(1) All values of condition variable(s) considered in the structure belong to the domain of this (these) variable(s);

(2) All elements from the domain of condition variable(s) are taken into consideration in the structure.

Tasks and Resource Inter-level Consistency Constraints. Tasks form hierarchies based on decomposition relations between them. When building such hierarchies, consistency should be maintained by making sure the set of inter-level constraints is satisfied. Here only some informal examples of inter-level constraints are given (for the complete list of these and other constraints see [13]):

“For every and-decomposition of a task, the minimal duration of the task is at least the maximal of all minimal durations of its subtasks.“

“If a task uses certain resource type as input then there exists at least one subtask in at least one and-decomposition of this task that uses this resource type.“

“For every and-decomposition of a task, if one subtask uses a data type as input which in not an input for the composite task then there exists another subtask that produces such data type.”

Functionally divisible resource or data types also form hierarchies. Due to the wide variety of possible situations, only one consistency constraint can be formulated, which should be satisfied for data types:

“If data type dt2 is a functional part of data type dt1, then the expiration duration of dt1 is at most the expiration duration of dt2.”

Physical World Generic Constraints. Generic constraints come from the physical world irrespective of the application domain. Here three examples of such constraints are given.

GC1: “No role executes more than one process at the same time”

$\forall$ r:ROLE, p1, p2:PROCESS, tp1, tp2, tp3, tp4:TIME_POINT role_performs_process(r, p1) $\wedge$ role_performs_process(r, p2) $\wedge$ esp_{p1} = tp1 $\wedge$ lep_{p1} = tp2 $\wedge$ esp_{p2} = tp3 $\wedge$ lep_{p2} = tp4 $\Rightarrow$ ((tp2 ≤ tp3) $\vee$ (tp4 ≤ tp1))

GC2: “Not consumed resources become available after all processes are finished”

GC3: “For every process that uses certain amount of a resource of some type as input, without consuming it, either at least that amount of resource of this type is available or can be shared with another process at every time point during the possible execution of the process”

3.2 Domain-specific Constraints

Domain-specific constraints are imposed by the application domain in which the specific model will be used and can be classified according to their sources. *Constraints imposed by the organization* have been chosen (e.g. by the management of the company) as necessary and need to be satisfied by any model for the particular organization. Such constraints can often be found in company policy documents, internal procedures descriptions, etc. For example, “certain information types cannot be used by certain tasks” (security/privacy). *Constraints coming from external parties* are enforced by an external party such as the society or the government and can contain rules about working hours, safety procedures, emissions, and so on. Sources for such constraints can be laws, regulations, agreements, etc. For example, “a driver should not drive more than 6 hours per day”. *Constraints of the physical world* come from the physical world w.r.t. the specific application domain and should be satisfied by any model in this domain. This is in contrast to the generic physical constraints which should be satisfied by any model irrespective of the application domain. For example, “there is always a break of at least 15 minutes between two consecutive lectures” (follows from the limitation of most humans to stay concentrated on a lecture for a very long time).

For all these types of constraints there are predefined templates [13], which can be selected and customized by the designer by assigning specific values to the parameters of the template. Examples of such templates with their parameters in brackets are:

DC1(p1:PROCESS, p2:PROCESS, d:VALUE): “If the same agent executes both processes, then there is a delay of duration at least d between the end of the first process and the beginning of the second one” (can also be formulated for a specific agent as additional parameter)

DC2(rt:RESOURCE_TYPE, min_am:VALUE): "At every time point the amount of resource of type rt available is at least min_am amount"
DC3(t:TASK, dr:VALUE): "For every agent performing processes of this task the sum of the durations of these processes should not exceed dr"

4 Correctness Verification of a Process-oriented Model

The verification of the correctness of a process-oriented model is performed during or at the end of the design of the model, depending on the verified types of constraints. In particular, some domain-specific constraints might not (yet) be satisfied for incomplete models. The designer can choose the moment when they should be checked. The syntactical check of the specification for a model and the verification of generic constraints are performed at every design step. Note that often only the set of relevant generic constraints is verified. This set is identified based on the type of the change made by the designer in the model. For example, if the minimal or maximal duration of a task or a decomposition relation between tasks is changed, then the corresponding task inter-level consistency constraint(s) expressed over these tasks should be checked. Changes in resource structures are dealt with similarly. If the designer changes the set of ordering relations or the (minimum or maximum) duration of a task of which an existing process is an instance, then first the structural constraints of the workflow are checked, and after that physical world and domain specific constraints.

For all other types of changes, the set of constraints that should be checked is formed from physical world and domain specific constrains from T_{PR} expressed over objects involved into relations affected by the change. For the checking, it is assumed that each process p in the workflow <w, P, C> can be active (executed) at any time point during the interval $[est_p, let_p]$. Therefore, it will be checked w.r.t. the whole interval $[est_p, let_p]$, even though the actual execution of p may take less time. Thus all possible intervals of the execution of p are taken into account. If the constraint is not satisfied in some possible execution, it can be discovered without checking all executions separately. This dedicated verification is computationally much cheaper than the general-purpose state-based analysis of all possible executions of a model (e.g., by model checking [7]), however still allows establishing correctness of the model.

Here two example algorithms are given for checking the satisfaction of constraints GC1 and GC3 defined in Section 3.1. The first algorithm is typical for the verification of constraints over processes, roles and agents, and the second one illustrates the verification of constraints over resource amounts related to processes.

Algorithm for verification of GC1. Let L be an empty queue and N be an empty set. Enqueue in L all roles defined in the model.

```
1. Until L is empty perform steps 3-5.
2. Dequeue L and assign the obtained value to the variable curr_role.
3. Put into N all processes assigned to curr_role in the model.
4. For each processes $p1, p2 \in N$ determine if they can be executed at the same time:
   if $\neg((let_{p1} \leq est_{p2}) \vee (let_{p2} \leq est_{p1}))$, then GC1 is not satisfied, exit; else empty N.
5. GC1 is satisfied.
```

Before describing an algorithm for checking GC3 let us introduce a definition of a workflow segment and a labeling procedure for workflow segments.

Definition 7 (A workflow segment): A segment SG of the workflow <w, P, C> is a set of processes from P ordered by C that are executed under the same set of values of or-conditions from w. This set of values is dynamically formed from the values of conditions of or-structures, from which the processes of SG can be reached. The set SEGMENTS contains all segments of the workflow.

Each segment has a label, assigned according to the following rules:

- the segment that contains processes that are executed independent of any condition values has the label "1".
- the label for a segment that corresponds to a branch of a certain or-structure is formed from three parts that follow each other:
  (1) the prefix defined by the label L of the segment, to which the beginning process of the or-structure belongs;
  (2) the index of the branch in the or-structure obtained incrementally starting from 1;
  (3) the sequential index of the or-structure in the segment with the label L, put in square brackets.

For example, the process Order_intake from the example introduced in Section 2 belongs to the segment labeled by "1", whereas the process Order_delivery2 belongs to the segment labeled by 1.2[1].

Further the algorithm is given for checking the satisfaction of GC3 with respect to the process p and the resource r. In this algorithm the following notations are used:

res_produced_by(r, p, am) for is_instance_of(p, t) $\wedge$ task_produces(t, r, am)
res_used_by(r, p, am) for is_instance_of(p, t) $\wedge$ task_uses(t, r, am)
res_consumed_by(r, p, am) for is_instance_of(p, t) $\wedge$ task_consumes(t, r, am)

Algorithm for verification of GC3. Identify the set of time points TP within the duration of p ($est_p \leq t < let_p$), at which the amount of some resource(s) of type r changes (i.e., time points at which other processes that use/consume/produce a resource of type r may start or finish). For every time point $t \in TP$ perform steps 2-7.

1. Determine the set RS of segments that contain finished before or executed simultaneously with p processes, which execution may influence the amount of resources of type r and which belong to the set of relevant processes PR(p) (defined in Section 3):
   RS={s $\in$ SEGMENTS | $\exists$a a$\in$s $\wedge$ a$\in$PR(p) $\wedge$ [$let_a < t \wedge$ [am1 > 0 $\vee$ am3 >0]] $\vee$ [$let_a > t \wedge est_a \leq t \wedge$ [am1 > 0 $\vee$ am2 > 0 $\vee$ am3 > 0], where am1, am2, and am3 are specificied in res_consumed_by(r, a, am1), res_used_by(r, a, am2) and res_produced_by(r, a, am3).
2. The labels of segments in RS that correspond to the branches belonging to the same or-structure are grouped.
3. The n-ary Cartesian product of all obtained groups is generated (n is the number of groups): $g_1 \times \ldots \times g_n$. Each tuple in the obtained product set corresponds to a possible combination of segments in the workflow. In such a way all possible execution of processes in the workflow, which use/produce/consume r and have latest ending time $\leq let_p$ are considered.
4. For every tuple in the product set identify the set of processes PS that corresponds to the tuple. If two or more processes from the same segment related by a sequencing relation may be executed at the same time in different instances of the workflow, replace PS by a number of sets, each of which will contain only one from these processes.
5. For every set of processes PS corresponding to the tuple, identify the set of resources RPS of type r produced by processes in PS.
6. For every process a $\in$ PS that consumes some amount of the resource of type r identify if this amount of not expired resource(s) from RPS is available. Update RPS after every iteration:

   7.1 Initial settings: Let temp_amount= am, where am is defined by res_consumed_by(r, a, am)

   7.2 Until temp_amount > 0 and RPS is not empty perform 7.3 and 7.4

   7.3 Identify resource res $\in$ RPS with the smallest earliest expiration time, which did not expire yet. It is assumed that such resource will be used first by a.

   7.4 If res.amount $\geq$ temp_amount,

```
        then update the amount of the resource as res.amount= res.amount- temp_amount and set
             temp_amount=0.
        else   update temp_amount = temp_amount - res.amount and delete res from the RPS.
    7.5 If      temp_amount > 0,
        then GC3 is not satisfied with respect to the process p and resource type r, exit.
7.  For every process a ∈ PS that uses a certain amount of the resource of type r at time point t
    identify, if this amount of not expired resource(s) from RPS is available. Update RPS after
    every iteration:
    8.1 Initial settings: Let temp_amount= 0.
    8.2 From the model identify process lists that may share a resource of type r and that contain
        as least one process from PS. For each process a ∈ PS at most one list will be chosen
        from the identified lists. Furthermore, any list that contains a may be selected, since the
        choice of the list does not influence the amount of available resources in RPS.
    8.3 For every chosen process list L update temp_amount = temp_amount+ am, where am is de
        fined in res_used_by(r, s, am) and s is some process from L. Delete all processes that
        belong to L from PS.
    8.4 For every process a ∈ PS update temp_amount = temp_amount+ am, where am is defined in
        res_used(r, a, am). Then perform the same calculations as on steps 7.2-7.5.
9. GP3 is satisfied with respect to the process p and resource type r.
```

In order to reduce the number of tuples generated at step 4 and to improve the computational properties, the introduced verification algorithm is extended with tuple reduction steps (the extended version of the algorithm can be found in [13]). This algorithm performs the local elimination of branches of and- and or-structures that do not contribute to the worst case situation. This allows reducing the number of execution paths that have to be checked significantly.

5 Example

In the context of the running example consider a particular delivery scenario: The logistics company performs shipments of goods between two departments of some enterprise, which are located in different regions. Depending on the type and the size of consignments three types of deliveries are distinguished: d1, d2 and d3. Goods for the delivery types d1 and d2 are located at the warehouse of department A and should be transported to department B. Goods assigned to d3 are stored at the warehouse of department B, and should be delivered to department A. For efficiency only trucks available at the starting location of the delivery can be used. The company owns five trucks of type tr1 and two trucks of type tr2. The delivery d1 can be fulfilled either by three tr1 trucks or by two tr2 trucks. The d2 can be accomplished by one truck of any type. The delivery d3 requires either four trucks tr1 or two trucks tr2. For simplicity, other combinations are not considered. One of the domain-specific constraints imposed by the company is to give a preference for using trucks of type tr1 (over tr2), when they are available (DC4). Formally,

∀a:TASK ∀p:PROCESS ∀v1:VALUE task_uses(a, tr2, v1) ∧ is_instance_of(p, a) ⇒ ∃p1:PROCESS ∃a1:TASK ∃v2:VALUE ∃id:OR_STRUCT is_instance_of(p1, a1) ∧ task_uses(a1, tr1, v2) ∧ starts_after(p1, begin_or(id)) ∧ starts_after(p, begin_or(id)) ∧ or_cond(id, resource_available(tr1) ≥ v2 ∧ or_branch(1, p1)

Furthermore, for deliveries of types d1 and d2 trucks may be shared (three tr1 trucks or two tr2 trucks), under the condition that the time difference between two deliveries is less than three hours, expressed by the domain-specific constraint DC5:

$\forall$p1, p2:PROCESS is_instance_of(p1, d1) $\wedge$ is_instance_of(p2, d2) $\wedge$ ((est$_{p1}$ $\geq$ est$_{p2}$ $\wedge$ (lst$_{p1}$ - est$_{p2}$) < 3) $\vee$ (est$_{p1}$ < est$_{p2}$ $\wedge$ (lst$_{p2}$ - est$_{p1}$) < 3)) $\Rightarrow$ $\exists$l:PROCESS_LIST resource_sharable(t1, l) $\wedge$ is_in_list(p1, l) $\wedge$ is_in_list(p2, l) (similarly for t2).

Initially all trucks are located at the base of dept. A. The delivery schedule includes processes p1, p4 (type d1), p2, p5 (type d2), and p3, p6 (type d3), ordered as in Fig.2.

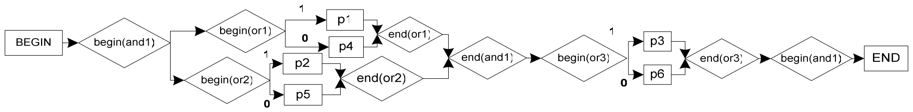

Fig. 2. The workflow that illustrates the delivery process considered in the example.

The constraints DC4 and DC5 are fulfilled by forcing the deliveries d1 and d2 to share three tr1 trucks (processes p1 and p2). However, the generic constraint GC3 defined previously is not satisfied by the model. The automatic verification identified that there are no sufficient resources provided for p3 (three tr1 trucks released by p1 and p2 are not sufficient for p3). To achieve the model correctness the designer may change either the model specification or the set of constraints imposed on the model. For example, without enforcing DC4 two tr2 trucks could be shared by p1 and p2, which would be further used by p3 without creating a resource problem.

6 Conclusions

This paper introduces a formal framework for process-oriented modeling and analysis. The framework is based on an expressive formal sorted predicate logic language and includes efficient dedicated analysis techniques for checking the correctness of process-oriented models w.r.t. a set of constraints defined in an organization. The constraints classified in the paper may express both local (i.e., related to individual objects) and global (i.e., related to multiple objects) properties of an organization.

The proposed approach differs from constraint satisfaction methods developed in [16]. Whereas the main focus of the latter techniques is on finding (optimal) solutions given a consistent and stable set of constraints, our approach addresses both design of a model and of constraints that should be satisfied by the model. The designer is free to vary both the model and the constraint specifications. The designer is supported by the automated tool that allows identifying sources of inconsistencies and mistakes both in the model and the constraint specifications.

The developed approach allows scalability by performing compositional design of models. Using task hierarchies models can be built at different levels of abstraction. General constraints defined for high level processes are refined into more specific ones that should be satisfied by processes of lower levels. In such a way, to decrease complexity models of different abstraction levels can be analyzed separately keeping relations with each other through task hierarchies and the constraint refinement.

Furthermore, although the introduced predicate language is very intuitive, still a graphical interface for specifying models would be of help. Such an interface is currently being developed. However, graphics would provide only a little help in the specification of constraints. For this property templates can be used as shown in [13].

The formal methods discussed in the paper are dedicated for the verification of process-oriented models, however, also a number of formal techniques for the analysis of actual execution based on the introduced process-oriented model, have been developed. These techniques will be discussed elsewhere.

References

1. Allen, J.F.: Maintaining knowledge about temporal intervals. *Communications of the ACM* 26 (1983) 832-843.
2. Attie, P., Singh, M., Sheth, A., Rusinkiewicz, M.: Specifying and enforcing intertask dependencies. In Proceedings of the 19th VLDB Conference (1993)
3. Barkaoui, K., Petrucci L.: Structural Analysis of Workflow Nets with Shared Resources, In: Aalst, W.M.P. van der, Michelis G. De, Ellis, C. A.:Workflow Management: Net-based Concepts, Models, Techniques and Tools, Vol.98. (1998) 82-95
4. Bettini, C., Wang, X., Jajodia, S.: Temporal Reasoning in Workflow Systems. Distributed and Parallel Databases 11(3) (2002) 269 – 306.
5. Business Process Modeling Language (BPML).http://www.bpmi.org.
6. CIMOSA – Open System Architecture for CIM; ESPRIT Consortium AMICE, Springer-Verlag, Berlin (1993)
7. Clarke, E.M., Grumberg, O., Peled, D.A.: Model Checking. MIT Press (2000)
8. Fox, M.S.: The TOVE project: towards a common-sense model of the enterprise. In Petrie, C.J., Jr.(Ed): Proceedings of ICIEMT'92, MIT Press (1992) 310-319
9. Li, H., Yang, Y., Chen, T.Y.: Resource constraints analysis of workflow specifications. Journal of Systems and Software 73(2) (2004) 271-285
10. Lu, R., Sadiq, S., Padmanabhan, V., Governatori, G.: Using a Temporal Constraint Network for Business Process Execution. In: Proc. of 17th Australasian Database Conference (2006)
11. Manzano, M.: Extensions of First Order Logic, Cambridge University Press (1996)
12. Menzel, C., Mayer, R.J.: The IDEF family of languages. In: Bernus, P. et al. (eds.): Handbook on Architectures of Information Systems, Springer-Verlag, Heidelberg (1998) 209-241
13. Popova, V., Sharpanskykh, A.: Process-Oriented Organization Modeling and Analysis Based on Constraints. Technical Report 062911AI, VUA, http://hdl.handle.net/1871/10545
14. Popova, V., Sharpanskykh, A.: Modelling Organizational Performance Indicators. In: Barros, F. et al. (Eds): Proc. of IMSM'07 conference (2007) 165–170
15. Singh, M. P.: Synthesizing distributed constrained events from transactional workflow specifications. In: Proc. of the 12th IEEE Intl. Conf. on Data Engineering, (1996) 616–623
16. Tsang, E.: Foundations of Constraint Satisfaction. Academic Press. (1993)
17. Van der Aalst, W. M. P.: The application of Petri Nets to workflow management. The Journal of Circuits, Systems and Computers 8(1) (1998) 21–66
18. Van der Aalst, W.M.P., Ter Hofstede, A.H.M., Kiepuszewski, B., Barros, A.P.: Workflow patterns. Distributed and Parallel Databases 14(3) (2003) 5–51