Semantic Correctness in Adaptive Process Management Systems

Linh Thao Ly, Stefanie Rinderle, and Peter Dadam

Dept. DBIS, University of Ulm, Germany
{thao.ly, stefanie.rinderle, peter.dadam}@uni-ulm.de

Abstract. Adaptivity in Process Management Systems (PMS) is key to their successful applicability in pratice. Approaches have already been developed to ensure the system correctness after arbitrary process changes at the syntactical level. However, still errors may be caused at the semantical level. Therefore, the integration of application knowledge will flag a milestone in the development of process management technology. In this paper, we introduce a framework for defining semantic constraints over processes in such a way that they can express real-world application knowledge. On the other hand, these constraints are still manageable concerning the effort for maintenance and semantic process verification. This can be used, for example, to detect semantic conflicts when applying process changes (e.g., drug incompatibilities). In order to enable the PMS to deal with such semantic conflicts we also introduce a notion of semantic correctness and discuss how to (efficiently) verify semantic correctness in the context of process changes.

Keywords: Semantic Correctness, Semantic Process Verification, Semantic Constraints, Adaptive Process Management Systems.

1 Introduction

Due to steadily changing conditions at the global market, companies are forced to frequently adapt their business processes [1–4]. Therefore, adaptivity is the key factor for the successful application of process management technology in practice. Generally, process changes can take place at two levels – process type and instance level [5, 6]. Therefore, it is crucial for an adaptive process management system (PMS) to support both kinds of changes. However, it is still not sufficient to support process type and instance changes in an isolated manner. An adaptive PMS must also allow for the *interplay* between process type and instance changes [7]. A framework for the support of process type and instance changes as well as for their interplay (i.e., the support of change propagation to already individually modified instances) has been developed [3, 8]. Within this framework the structural (syntactical) correctness of the system is always preserved after arbitrary process changes. For example, it is automatically checked by the PMS whether process changes will lead to structural errors, like deadlock-causing cycles or not properly supplied input parameters, or to inconsistent instance states. However, the framework abstracts from semantical aspects. Thus, *semantic* errors may arise, especially in the context of process changes intitiated under time pressure. Consider, for example, process instance I reflecting the treatment process for patient Smith as depicted in Fig. 1. Assume that, due to suddenly arising headache, the drug Aspirin is administered to patient Smith. This is achieved by inserting activity `Administer Aspirin` into instance I in an ad-hoc manner by, for example, a nurse at her workplace.

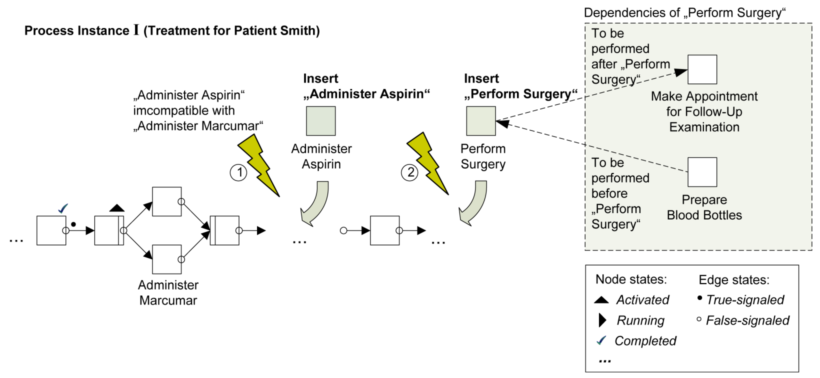
Process Instance I (Treatment for Patient Smith)
„Administer Aspirin" imcompatible with „Administer Marcumar"
Insert „Administer Aspirin"
Administer Aspirin
Insert „Perform Surgery"
Perform Surgery
1
2
Administer Marcumar
Dependencies of „Perform Surgery"
To be performed after „Perform Surgery"
Make Appointment for Follow-Up Examination
To be performed before „Perform Surgery"
Prepare Blood Bottles
Node states:
Activated
Running
Completed
Edge states:
True-signaled
False-signaled

Fig. 1. Semantic conflicts after process changes due to drug incompatibility and dependencies between activities

However, in this treatment process, the drug Marcumar, which is not compatible to Aspirin, is already administered some activities ahead (*semantic conflict*). Even if the process change is syntactically correct, it is not semantically. Especially when the process instance is often modified in an ad-hoc manner (for instance, `Administer Marcumar` was previously inserted as an ad-hoc modification) or when process changes at scheme level and at instance level occur together, it is likely that those conflicts remain undetected by users. If the PMS was aware of the incompatibility of these activities, it could prevent the user from causing semantic conflicts by, for example, warn the user accordingly. In Fig. 2, the user (a doctor with the appropriate authorization) still performs the change operation but has to document the reason for overriding the semantic constraint. Thus, it is possible to trace back semantic conflicts.

As motivated, it is crucial to be able to also integrate application knowledge (i.e., *semantic knowlegde*) within the process change framework in order to avoid semantic conflicts. In this context, many challenging questions arise:

– How to formalize and integrate application knowledge within an adaptive PMS?
– How to define a notion of semantic correctness of processes after changes?
– How to support the efficient verification of the semantic correctness?
– How to maintain the knowledge base?

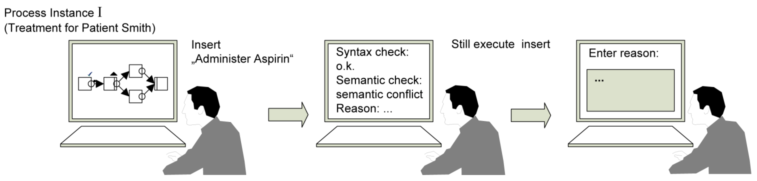

Fig. 2. Interaction scenario when a semantic conflict occurs

In this paper, we extend the framework presented in [3, 8] by integrating application knowledge into adaptive PMS. First of all, we provide a formalization for semantic constraints imposed on business processes. In particular, we introduce two fundamental kinds of semantic constraints (mutual exclusion constraints and dependency constraints) which serve as a basis for the following considerations. However, the set of semantic constraints can be easily extended. Based on the notion of semantic constraints, a general criterion for the semantic correctness of business processes (independently from the underlying process meta model) is provided. We show how to verify semantic correctness of processes based on this criterion, in particular in the context of process changes. For this, we exploit the semantics of the applied change operations, for example when applying single change operations (e.g., adhoc changes of process instances), or when applying concurrent changes (e.g., propagating process schema changes to biased process instances). Afterwards, we discuss different possibilities to realize verification of semantic process correctness in an efficient manner. One way based on exploiting certain process meta-model properties is discussed in more detail. Finally, we show how the semantic constraints can be organized within a domain repository.

This paper is structured as follows. In Sect. 2, a framework for the definition of semantic constraints and the notion of semantic correctness are introduced. In Sect. 3, we show how the semantic correctness of processes can be verified. In Sect. 4, we show the application of the criterion when, for example, a block-structured process model is used. In Sect. 5, a framework for organizing semantic constraints is introduced. Related work is discussed in Sect. 6. Finally, Sect. 7 concludes with a summary and an outlook on future research.

2 Semantic Constraints and Semantic Correctness in Adaptive Process Management Systems

As motivated in Sect. 1, it is desirable to integrate (semantic) application knowledge in the PMS in order to avoid semantic conflicts. It is in principle possible to integrate even very complex application knowledge in adaptive PMS. By connecting the PMS with a knowledge-based system or an expert system (e.g. [9, 10]), for instance, application knowledge maintained in the external system can be used by the PMS to avoid semantic conflicts. However, two important aspects influence the possibilities of integrating application knowledge in adaptive PMS. First of all, it is an important question how and by whom the knowledge base is maintained. The more application knowledge, and in particular the more complex the knowledge, the greater is the effort to keep the knowledge base up-to-date. Thus, there is a risk that the knowledge, according to which the semantic checks are performed, is outdated. In fact, this might be even more dangerous than not performing semantic checks at all. Users might rely on the semantic checks to ensure the semantic correctness of the process not knowing that the knowledge base is outdated. As a consequence, it seems reasonable to only integrate that kind of application knowledge which is really important and which will really be kept up-to-date. Second, the goal of integrating application knowledge is to enable the PMS to also perform process checks at the semantic level. However, the effort to perform these semantic checks must not lead to a bottleneck, especially when changes on process schemes are propagated to many running (and possibly ad-hoc modified) instances. This restricts the complexity of application knowledge to be integrated in adaptive PMS.

The two aspects mentioned above need to be kept in mind when thinking about integrating application knowledge in adaptive PMS. In future work, we will investigate the influence of these aspects in more detail. In this paper, we introduce two fundamental kinds of semantic constraints which can be imposed on processes: mutual exclusion constraints and dependency constraints. These constraints refer to activities and impose certain conditions on how these activities can be used in the process. By enabling the PMS to be aware of these fundamental constraints, many semantic errors, for example the ones depicted in Fig. 1, can be avoided. On the other hand, the introduced kinds of constraints are still manageable regarding the effort for maintenance and for semantic verification.

Mutual exclusion constraints express that two activities are not compatible and should not be executed together, for instance administering two incompatible drugs. Please note, that this does not mean that these activities must not occur in the same process. Due to the process structure, it depends on the position of the activities whether the constraint is satisfied or not. In Fig. 3, a semantic conflict occurs in the first process fragment while the second process fragment is semantically correct. Mutual exclusion constraints are symmetric.

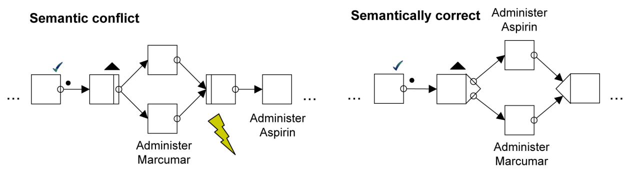

Fig. 3. Semantic conflict dependent of process structure

Dependency constraints express that an activity is dependent of another activity, i.e. these activities need to occur together in the process. In Fig. 1 for instance, activity `Perform Surgery` is added to the process. However, in the treatment process the activity `Prepare Blood Bottles` needs to be performed before and `Make Appointment for Follow-Up Examination` needs to be performed after `Perform Surgery`. These semantic dependencies of `Perform Surgery` cause a semantic conflict, when only `Perform Surgery` is inserted to the process.

Whether a process change can be applied to a concrete process is, therefore, not only a question of structural correctness or data flows but also a question of whether the semantic constraints over the process are violated by the process change. For our following considerations we assume the uniqueness of activities in a process (i.e. each activity may occur only once in a business process).

Definition 1 (Semantic constraint). *Let $\mathcal{A}$ be a set of activities*[1]*. A semantic constraint c is defined as a tuple (type,source,target,position,userDefined) whereas*

- $type \in \{Exclusion, Dependency\}$
- $source, target \in \mathcal{A}$, $source \neq target$
- $position \in \{pre, post, notSpecified\}$
- $userDefined$ *is a user-defined parameter*

The parameter *type* denotes whether the semantic constraint is a mutual exclusion constraint or a dependency constraint. The second parameter *source* denotes the source activity the constraint refers to while *target* denotes the target activity related to the source activity. Parameter *position* specifies the order the source and target activity are to be related to each other within the process (e.g., the surgery depends on the preparation of blood bottles and the bottles have to be prepared before (*pre*) the surgery). The last parameter $userDefined$ can be used for several purposes, for instance for additionally describing the constraint. Furthermore, it might also be used to indicate the importance of the constraint. For instance, to indicate whether a constraint is merely a recommendation or whether it is more severe. This information can be used by the PMS client to create an appropriate feedback for the user. As an example, the constraint mentioned above would look like this:

(`Dependency`, `Perform surgery`, `Prepare blood bottles`, `pre`,
`Blood bottles need to be prepared for the patient and stored in the surgery room before the surgery can take place`)

In Def. 2, the satisfaction of semantic constraints is defined taking the notion of execution trace as a basis. According to Def. 2, the constraint above, for example, is satisfied over a process if the source activity (`Perform surgery`) is not included in this process. In case it is, the constraint is satisfied, if `Prepare blood bottles` is always performed before `Perform surgery` in each possible execution trace of the process, in which `Perform surgery` appears.

Definition 2 (Satisfaction of semantic constraints). *Let $\mathcal{A}$ be a set of activities which can be used to specify a process p of type T. Let $\mathcal{Q}$ be the set of all possible execution traces of p. A trace $q \in \mathcal{Q}$ is defined by $q :=< e_1, \ldots, e_k >$ with events e_i = End(t)*[2]*, $t \in \mathcal{A}$. Then, we define the following functions:*

[1] Within the ADEPT framework, for example, $\mathcal{A}$ refers to the activity repository containing all relevant activities in the context of a certain process type T.

[2] We abstract from start events in the traces.

- *activities:* $\mathcal{Q} \mapsto \mathcal{A}$ *with activities(q):=* $\{t_1, \ldots, t_n\}$ *with* $q =< e_1, \ldots, e_k > \wedge \forall\, t_l\, \exists\, e_i$ *with* $e_i = End(t_l)$, *l = 1, ..., n; i = 1, ..., k (i.e., activities denotes a function that returns the set of all activities included in an execution trace q).*
- *processActs(p):=*$\{t_1, \ldots, t_n\}$ *with* $\forall\, t_l\, \exists\, q \in \mathcal{Q}$ *with* $t_l \in activities(q)$, *l = 1, ..., n (i.e., processAtcs returns all activities included in the process p).*
- *traceSucc:* $\mathcal{A} \times \mathcal{Q} \mapsto \mathcal{A}$ *with* $traceSucc(t, \sigma) := \{t_1, \ldots, t_n\}$ *with* $\sigma =< e_1, \ldots, e_k >$, $t_1, \ldots, t_n \in activities(\sigma) \wedge \forall\, t_l : \exists\, e_i,\ e_j \in \sigma$ *with* $e_i = End(t_l)$, $e_j = End(t)$, *l = 1, ..., n; i, j = 1, ..., k* $\wedge\ i < j$ *(i.e., traceSucc denotes a function which returns all direct or indirect successors of a given activity t within an execution trace* σ*).*
- *tracePred:* $\mathcal{A} \times \mathcal{Q} \mapsto \mathcal{A}$ *with* $tracePred(t, \sigma) := \{t_1, \ldots, t_n\}$ *with* $\sigma =< e_1, \ldots, e_k >$, $t_1, \ldots, t_n \in activities(\sigma) \wedge \forall\, t_l : \exists\, e_i,\ e_j \in \sigma$ *with* $e_i = End(t_l)$, $e_j = End(t)$, *l = 1, ..., n; i, j = 1, ..., k* $\wedge\ i > j$ *(i.e., tracePred denotes a function which returns all direct or indirect predecessors of a given activity t within an execution trace* σ*).*

Let $a_1,\ a_2 \in \mathcal{A}$ *be two activities,* $a_1 \neq a_2$*. Then, a semantic constraint c = (type, source, target, position, userDefined) with source=*a_1 *and target=*a_2 *is* `satisfied` *over process p (formally: satisfied(c, p) =* `True`*) iff one of the following conditions holds:*

- *type* $\in$ *{Exclusion,Dependendency} and* $a_1 \notin$ *processActs(p)*
- *type = Exclusion, position = pre and* $\forall$ *execution traces* $\phi \in \mathcal{Q}$*:* $a_1 \in activities(\phi) \Rightarrow a_2 \notin tracePred(a_1, \phi)$
- *type = Exclusion, position = post and* $\forall$ *execution traces* $\phi \in \mathcal{Q}$*:* $a_1 \in activities(\phi) \Rightarrow a_2 \notin traceSucc(a_1, \phi)$
- *type = Exclusion, position = notSpecified and* $\forall$ *execution traces* $\phi \in \mathcal{Q}$*:* $a_1 \in activities(\phi) \Rightarrow a_2 \notin traceSucc(a_1, \phi)$ *and* $a_2 \notin tracePred(a_1, \phi)$
- *type = Dependendency, position = pre and* $\forall$ *execution traces* $\phi \in \mathcal{Q}$*:* $a_1 \in activities(\phi) \Rightarrow a_2 \in tracePred(a_1, \phi)$
- *type = Dependendency, position = post and* $\forall$ *execution traces* $\phi \in \mathcal{Q}$*:* $a_1 \in activities(\phi) \Rightarrow a_2 \in traceSucc(a_1, \phi)$
- *type = Dependendency, position = notSpecified and* $\forall$ *execution traces* $\phi \in \mathcal{Q}$*:* $a_1 \in activities(\phi) \Rightarrow (a_2 \in tracePred(a_1, \phi)$ *or* $a_2 \in traceSucc(a_1, \phi))$

Otherwise, c is `violated` *over p (formally: satisfied(c, p) =* `False`*).*

For a process type (e.g., the treatment process), many constraints might be relevant. Even if the process was modelled semantically correct at buildtime, due to possible (unforeseen) process changes, activities might be deleted from or added to the process at runtime. Furthermore, mutual exclusion constraints cannot be modelled in the control-flow of a process. In these cases, the constraints imposed on the process will help to ensure a semantically correct execution. Now, based on the notion of satisfaction of constraints, a semantic correctness criterion for business processes can be defined.

Definition 3 (Semantic correctness of business processes). *Let T be a process type and let p be a process of type T. Let further C_p be the set of all semantic constraints defined over p. Process p is semantically correct $\Longleftrightarrow$*

$$\forall c \in C_p\text{: } satisfied(c,p) = \texttt{True}$$

Using Def. 1–3, it is possible to state for each business process whether the business process is semantically correct or not.

3 On Preserving Semantic Correctness of Processes

As specified in Def. 3, a process (no matter whether it is a process instance or a process schema) is semantically correct only if all of its semantic constraints are satisfied. Consequently, the semantic constraints of the process need to be analyzed when checking the process' semantic correctness. Not all the constraints on a process, however, are relevant. Depending on the situation in which the semantic check is initiated, it is possible to restrict the set of relevant constraints to be verified and thus to reduce the effort for semantic process verification. We now have a closer look on that.

In Sect. 3.1, we show how the semantic correctness of process schemes can be verified. In Sect. 3.2, we show how to ensure the semantic correctness of a process when ad-hoc process adaptations are carried out. In Sect. 3.3, we consider how to maintain the semantic correctness when schema evolution is performed. For the remainder of this section, let p be the process to be verified and let C_{vp} be the set of the constraints to be verified in the respective situation.

3.1 Semantic Correctness of Process Schemes

Basically, there are two ways of ensuring the semantic correctness of process schemes depending on the way the process models are constructed. If a process model is built by applying process changes to an "empty" schema the PMS might perform a semantic check each time a change operation is applied and check whether the semantic correctness of the process is still preserved after the change or not (cf. 3.2). The second possibility is to take an already existing process model[3] and to verify the correctness of the complete process schema at once. In this case, it is necessary to verify, whether the constraints imposed on the process are satisfied or not. However, constraints, for which the source activity is not included in the process, are always satisfied over this process by definition. Thus, these constraints need not be considered.

More formally: $C_{vp\ Schema} = \{c \in C_p;\ c(source)^4 \in processActs(p)\}$

3.2 Semantic Correctness After Applying Ad-Hoc Process Changes

In our framework, an ad-hoc process change is considered *semantically applicable* to a process if its application still preserves the semantic correctness of the

[3] This is, for instance, relevant when a process model is imported to the PMS or the process schema is obtained by applying process mining techniques.

[4] $c(source)$ denotes the source parameter of the constraint c.

process. The naive way of verifying the semantic correctness of a process after a process change is to verify the complete process model, as described in Sect. 3.1. However, this effort can be reduced by exploiting the semantics of the applied change operations (e.g., which activity has been inserted at which position). Thus, depending on which change operation is requested, only a smaller subset of constraints on the process needs to be verified. In the following, we discuss the interplay between change operations of type *Insert*, *Delete* and *Move* and the set of constraints to be verified.

When **inserting** an activity t into process p, all semantic constraints over p which have t as source parameter need to be verified since they might be violated. However, since dependency constraints which do not have t as source parameter cannot be violated by the addition of t, only mutual exclusion constraints with t as target parameter need to be considered. We can even further restrict the set of interesting exclusion constraints to those constraints whose source parameter is among the activities of p and whose target parameter corresponds to the inserted activity t. That is because all exclusion constraints, whose source parameter are not included in the process, are satisfied by definition.

More formally: $C_{vp\ Insertion}$=$\{c \in C_p$; $(c(source)$=$t)$ or $(c(type)^5$=Exclusion and $c(source) \in processActs(p)$ and $c(target)^6$=$t)\}$.

When **deleting** an activity t from process p, all semantic constraints over p with t as source parameter are satisified by definition. Similar to the insertion of activities, all constraints for which t occurs as target parameter are potentially interesting for correctness checks. However, mutual exclusion constraints with t as target parameter cannot be violated by the deletion of t. Only dependency constraints with t as target parameter and for which the source parameter is included in p might be violated by the deletion operation and therefore need to be verified.

More formally: $C_{vp\ Deletion}$=$\{c \in C_p$; $c(type)$=Dependency and $c(source) \in processActs(p)$ and $c(target)$=$t\}$.

The **moving** of an activity t from its original position within process p to a new position *pos* can be understood as being equivalent of deleting t and inserting t at *pos* afterwards[7]. Consequently, all constraints that which might be violated after applying deletion and insertion operations need to be verified.

More formally: $C_{vp\ Move} = C_{vp\ Deletion} \cup C_{vp\ Insertion}$.

3.3 Semantic Correctness for Process Schema Evolution

In addition to ad-hoc changes at the instance level, adaptive PMS must support the modification of process schemes at the type level followed by the migration of running instances to the modified process schema as well. The semantic correctness of the process schema after applying the changes can be verified

[5] $c(type)$ denotes the type of the constraint c (Dependency or Exclusion).

[6] $c(target)$ denotes the target parameter of the constraint c.

[7] In conjunction with data flow aspects, moving is not always equivalent to deleting and inserting. However, this assumption can be used to derive statements about possible semantic conflicts here.

by using the considerations for ad-hoc changes made in Sect. 3.2. In case the schema change is semantically correct, it will also be semantically correct when being applied to *unbiased* instances (i.e., instances which still run according to the process schema they have been started on). However, the direct application of the schema change to *biased* instances (i.e., instances which have already been individually modified) might lead to semantic conflicts between type and instance changes. Assume that at instance level drug Marcumar has been administered for process instance I as an ad-hoc change. Afterwards, at process type level, activity `Administer Aspirin` is inserted into the associated process schema and is to be propagated to I. Migrating I to the modified process schema then causes a semantic conflict, even though the migration can be performed in a syntactically correct manner. Therefore, we have to check whether the process changes at type level are semantically applicable to the biased instances. We assume that the biased instances are semantically correct after the individually applied process instance changes. The propagation of changes at type level to a biased instance is semantically correct if the type changes are *semantically applicable* to the biased instance as ad-hoc instance change or vice versa (cf. Sect. 3.2).

Due to only considering *biased* instances, the number of instances to be checked is highly decreased. However, it is possible to further decrease the number of instances and relevant constraints to be verified. For example, if the change operations applied to a process instance constitute a superset of the change operations applied to the process schema (or vice versa), no semantic conflicts can occur. Due to space restrictions, we omit further details. For details on superset relations between change operations we refer to [8, 11].

For an efficient implementation of the considerations in Sect. 3, employing indexing techniques in order to easily access the relevant constraints in the respective situations seems very useful. After having considered, which constraints need to be verified in different situations, in the next section we consider how to verify those constraints.

4 On Optimizing Semantic Process Verification

The semantic correctness criterion for business processes defined in Sect. 2 is generic and can be applied to any process meta-model (e.g., Petri Nets [1] or BPEL4WS Nets [12]). For verifying the criterion, reachability analysis can be applied (i.e., by calculating all possible execution traces and checking them for certain order relations between activities according to the semantic constraints) which might be very costly. Therefore, we want to investigate different methods to ensure the semantic correctness criterion which are less expensive. In this paper, we present an approach which makes use of certain properties of the underlying process meta-model, namely block-structuring (e.g., WSM Nets [3]). However, we intend to also develop model-independent methods in future work.

4.1 Background Information

This section summarizes background information on WSM Nets [13, 14] as process description formalism in order to present an optimized verification method for semantic correctness.

A *process schema* is represented by a WSM Net which defines the *process activities* as well as the *control* and *data flow* between them. When using WSM Nets the control flow schema can be represented by attributed, serial-parallel graphs. In order to synchronize activities from parallel paths additional links can be used [15]. In this paper we abstract from cyclic structures within the process meta model in order to provide a fundament for an optimized semantic correctness verification. Further on, a WSM Net comprises a set of *data elements* and a set of *data edges*. A data edge links an activity with a data element and either represent a read access of this activity or a write access. The total set of data edges constitutes the data flow schema.

Definition 4 (WSM Net). *A tuple S = (N, D, NT, CtrlEdges, SyncEdges, DataEdges, BC) is called a WSM Net, if the following holds:*

- *N is a set of process activities and D a set of process data elements*
- *NT:* $N \mapsto$ {`StartFlow, EndFlow, Activity, AndSplit, AndJoin, XOrSplit, XOrJoin, StartLoop, EndLoop`}
  NT assigns to each node of the process schema a respective node type.
- *CtrlEdges* $\subset N \times N$ *is a precedence relation definining the valid order of activities (notation:* $n_{src} \rightarrow n_{dst} \equiv (n_{src}, n_{dst}) \in$ *CtrlEdges)*
- *SyncEdges* $\subset N \times N$ *is a precedence relation between activities of parallel branches*
- *DataEdges* $\subseteq N \times D \times \{read, write\}$ *is a set of read/write data links between activities and data elements*
- *BC:* $N \mapsto$ *Conds(D) where Conds(D) denotes the set of all valid transition conditions on data elements from D. BC(n) is undefined for nodes n with NT(n)* $\neq$ `XOrSplit`.

Which constraints have to hold such that a process schema S is well-structured is summarized in [15, 8] (e.g., absence of deadlock–causing cycles and correctly supplied input parameters). In the context of this paper, the block-structuring property is important, i.e., for all activities of node type `AndSplit` (`XOrSplit`) there is a unique activity of node type `AndJoin` (`XOrJoin`) and blocks (sequences as well as parallel and alternatives branchings can be nested but must overlap).

In this paper we abstain from defining process instances (see [3]) since this is not relevant for the following considerations.

4.2 On Exploiting Process Meta Model Properties

From the general constraint satisfaction criteria presented in Sect. 2 we derived meta-model specific conditions on WSM Nets. Using these meta-model specific criteria the satisfaction of semantic constraints and thus the semantic correctness of a process can be verified in an optimized way. For all semantic constraints in

Def. 1, such meta-model specific criteria can be derived. Due to space restrictions, however, we abstain from presenting all the criteria. Instead, as an example, we show how a particular meta-model specific criterion can be derived. Consider the following semantic constraint over the treatment process from Sect. 1:

c_1: (`Dependency`, `Perform surgery`, `Prepare blood bottles`, `pre`, `...`)

If `Perform surgery` does not occur in the treatment process, then c_1 is satisfied by definition and consequently not of further interest for semantic process verification (cf. Sect. 3). In case `Perform surgery` occurs in the process, it is necessary that `Prepare Blood Bottles` is a direct or indirect predecessor of `Perform surgery` in the treatment process for c_1 to be satisfied. Otherwise, it is not possible that `Prepare Blood Bottles` is performed before `Perform Surgery`, each time `Perform Surgery` is performed. However, this is not sufficient, since this execution order is not guaranteed. When verifying semantic constraints, it is necessary to also take the process structure into account. If `Prepare Blood Bottles` is contained in the inner part of an XOR-block while `Perform Surgery` is not, `Prepare Blood Bottles` is not sure to be performed each time `Perform Surgery` is performed. Therefore, c_1 is not satisfied over the process depicted in Fig. 4.

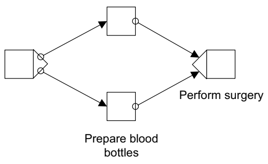

Fig. 4. `Prepare blood bottles` is not sure to be performed each time `Perform surgery` is performed

From this example we conclude the following conditions for the satisfaction of that kind of dependency constraints over block-structured meta models:
A semantic dependency constraint c_{dep}= $(Dependency, source, target, pre, ...)$ over a process p represented by a WSM Net $S = (N, D, NT, ...)$ with $source \in N$ (i.e. $source \in processActs(p)$) is satisfied (i.e., $satisfied(c_{dep}, p)$ = `True`) if and only if the two following conditions hold:

– $target \in pred^*(S, source)$ (*necessary condition*)
– $\forall s \in N$ with $NT(S)$ = `XOrSplit`: $target \in inBlock(S, s) \Rightarrow source \in inBlock(S, s)$ (*sufficient condition*), where:
  - $pred^*(S, n)$ ($succ^*(S, n)$) denotes the set of all direct and indirect predecessors (successors) of n in S[8]
  - $inBlock(S, s) := succ^*(S, s) \cap pred^*(S, join(S))$
  - $join(S, s)$ yields the unique associated join for split node s

[8] Note that $pred^*$ ($succ^*$) refers to structural predecessors (successors) whereas $tracePred$ ($traceSucc$) refers to predecessors (successors) within execution traces.

In the following we show that these conditions ensure the semantic correctness.

Proof sketch. Let c_{dep}= $(Dependency, source, target, pre, userDefined)$ be a semantic dependency constraint over a process p represented by a WSM Net $S = (N, D, NT, ...)$ with the set of all execution traces Φ for which $source \in N$ holds. Then, the following proposition $\bowtie$ is to be proven (cf. Def. 2):

$satisfied(c_{dep}, p) = \texttt{True} \Longleftrightarrow$
$(target \in pred^*(S, source)) \wedge$
$(\forall s \in N$ with $NT(S) = \texttt{XOrSplit}$: $target \in inBlock(S, s) \Rightarrow$
$source \in inBlock(S, s))$
$\bowtie \Longleftrightarrow$
$\forall \phi \in \Phi$: $source \in activities(\phi) \Rightarrow target \in tracePred(source, \phi)$ (i) $\Longleftrightarrow$
$(target \in pred^*(S, source)) \wedge$
$(\forall s \in N$ with $NT(S) = \texttt{XOrSplit}$: $target \in inBlock(S, s) \Rightarrow$
$source \in inBlock(S, s))$ (ii)
"$\Longrightarrow$": Proof by contradiction (i.e. ((i) $\Longrightarrow$ (ii)) $\Longleftrightarrow$ ($\neg$(ii) $\Longrightarrow$ $\neg$(i)))

Let us assume that (ii) does not hold (i.e. $\neg$(ii) holds). Let us first assume that the necessary condition does not hold, i.e. $target \notin pred^*(S, source)$. This means that there is no path from *source* to *target* in p. Then, there are four possibilities:

1. $target \notin N \Longrightarrow source \notin tracePred(target, \Phi)$
2. $target \in succ^*(S, source) \Longrightarrow source \notin tracePred(target, \Phi)$
3. *target* and *source* are in an parallel block
4. *target* $\in N$ and *target* in an XOR-path while *source* is in the other XOR-path

Possibilities 1 and 2 are clear. If the third possibility is true, then c_{dep} is also violated since, due to the interleavings of parallelly executed activities, there might be at least one trace, where *target* and *source* do not occur in the required ordering relation. If the fourth possibility is true, then either *source* or *target* are executed during an process execution. Thus c_{dep} is violated as well. As shown, all possibilites that are left when the necessary condition is not true lead to the violation of c_{dep} ($\neg$(i)).

Now let us assume that the necessary condition holds, but not the sufficient condition. This means: $\exists s, NT(S) = \texttt{XOrSplit}$ with $target \in inBlock(S, s) \wedge source \notin inBlock(S, s)$. Since $target \in pred^*(source))$ holds (necessary condition), we can construct an execution trace of p by not chosing the XOR-path which *target* is on while still executing *source*. This leads to $\neg$(i). □

The reverse direction "$\Longleftarrow$" can be proven analogously.

The satisfaction criterion for dependency constraints for block-structured process meta-models presented above can be verified very efficiently. Special constructs of the meta-models, for instance references to the split and join nodes, can be exploited by the PMS in order to find out whether the respective constraint is satisfied or not. However, using the meta-model specific criterion it is also possible to leave the verification to an external reasoning system (e.g. RACER [9]). In this case, information about the process structure need to be mapped to rules in the reasoning system in order enable it to apply inference techniques. We intend to further investigate these implementation alternatives in future work.

5 A Framework for Semantic Constraints

In our approach, a set of semantic constraints is assigned to a process. However, several processes may share constraints. In this section, we present a framework for organising semantic constraints such that they can be reused easily.

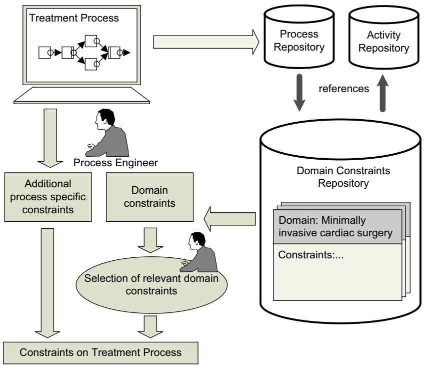
Treatment Process
Process Repository
Activity Repository
references
Process Engineer
Additional process specific constraints
Domain constraints
Domain Constraints Repository
Domain: Minimally invasive cardiac surgery
Constraints:...
Selection of relevant domain constraints
Constraints on Treatment Process

Fig. 5. Organisation of constraints in a domain constraints repository

The three main components of the framework are the *domain repository*, the *process repository*, and the *activity repository* (cf. Fig. 5). Semantic constraints are organised in the domain repository. In particular, constraints are assigned to domains, for instance the domain *Minimally invasive cardiac surgery*. Thus, a domain contains a set of constraints that are typical of this domain. The constraints presented in this paper refer to activities which are organized in an activity repository. For future work, we also plan to introduce constraints that refer to other abstraction levels, for instance abstraction levels in the activity repository. Process types (process schemes) are organized in a process repository. Each process type is assigned a domain of the domain repository. Thus, it is possible to assign a default set of domain constraints to a process. However, processes that are assigned to the same domain might still have different semantic constraints that are not captured in the domain. Therefore, for each process type, the process designer can specify additional semantic constraints for the process or leave out some unnecessary domain constraints.

6 Related Work

The issue of integrating semantics in process management systems has often been adressed in literature. In particular, there are interesting approaches from the clinical domain concerning the formalization of clinical guidelines in a computer-readable way, e.g. [16,17] and GLIF3 [18] or GUIDE [19]. However, so far, this information cannot be directly used for automatical analyses by the PMS.

Current approaches on adaptive PMS mainly focus on structural aspects (e.g., [3,4,8,20]) or have a different notion of semantic correctness (e.g., [21, 1]). Many related approaches focus on the aspect of integrating heterogenous resources. In particular, activities and their parameters are often described using ontologies, e.g. [22–24] and also many approaches concerning semantic web service composition, for instance [25,26]. When a process is composed, the PMS can check, whether the activities and their parameters semantically fit together. However, these approaches do not consider semantic constraints over processes, for instance mutual exclusion constraints, the way we do.

As discussed in Sect. 2, approaches from the field of *Artificial Intelligence*, in particular knowledge-based systems, e.g. [27,10,9], can be used to integrate application knowledge in PMS. This is also closely related to approaches concerning the integration of business rules in PMS. Application knowledge can be sourced out into a *Business Rule Engine*, e.g. commercially available systems like ILOG [28]. Thus, decision processes, for instance, which outgoing paths of an activity to follow, can be supported taking also background knowledge into account. This approach, however, is not directly suitable for situations like the one outlined in our example scenario since this situation concerns not only the occurrence of an activity in the process but also the relations between activities (cf. Fig. 3). In [29,30], an approach to ensure the integrity of processes is introduced. Rules, realized as database triggers, are applied, when certain data conditions occur. Using the change framework presented in [3,8] the process is adapted in an ad-hoc manner according to the triggered rule. Our approach, however, goes further since for the semantic verification, structural information about the process is needed.

In [31] van der Aalst et al. introduced an approach for verifying given properties of past processes by applying process mining techniques. This approach can help detecting constraint violations. However, the approach introduced in [31] is orthogonal to our work because it aims on analysing past processes on certain aspects while our intention is to ensure the semantic correctness of running processes. Thus, these two approaches can complement each other.

We consider our approach to be orthogonal to the approaches mentioned in this section.

7 Conclusion and Outlook

In this paper, we introduced a framework for the integration of application knowledge within an adaptive PMS by using semantic constraints. Based on these constraints, a generic criterion for semantic correctness of processes has been provided. We have shown how this criterion can be generally ensured. Furthermore, we have addressed the issue of verifying semantic correctness after process changes. Exemplarily for block-structured process meta-models, we have shown how semantic process verification can be realized in an efficient manner.

Finally, an architecture for the integration of semantic constraints within an adaptive PMS has been presented.

Using our approach, all semantic conflicts caused by violation of dependency and mutual exclusion constraints can be avoided. However, the expressiveness of the presented constraints is limited. Therefore, in future work we will extend our framework, e.g. by introducing context restrictions on constraints concerning their validity (e.g., time or location) or by introducing constraints on other levels of granularity than the activity level (e.g. data). Furthermore, we want to develop further methods to efficiently verify semantic correctness within an adaptive PMS. For example, we want to analyze how the information referred to by semantic constraints can be organized (e.g., within an ontology) in order to decrease evaluation effort. All considerations are to be implemented within the adaptive PMS ADEPT (e.g. [15]).

Acknowledgement. We thank Michael Nahler for the valuable results of his Master thesis ([32]) which have partially been incorporated in this paper.

References

1. v.d. Aalst, W., Basten, T.: Inheritance of workflows: An approach to tackling problems related to change. Theoret. Comp. Science **270** (2002) 125–203
2. Casati, F., Ceri, S., Pernici, B., Pozzi, G.: Workflow evolution. Data and Knowledge Engineering **24** (1998) 211–238
3. Rinderle, S., Reichert, M., Dadam, P.: Flexible support of team processes by adaptive workflow systems. DPD **16** (2004) 91–116
4. Weske, M.: Formal foundation and conceptual design of dynamic adaptations in a workflow management system. In: HICSS-34. (2001)
5. Kochut, K., Arnold, J., Sheth, A., Miller, J., Kraemer, E., Arpinar, B., Cardoso, J.: IntelliGEN: A distributed workflow system for discovering protein-protein interactions. DPD **13** (2003) 43–72
6. Reichert, M., Rinderle, S., Dadam, P.: On the modeling of correct service flows with BPEL4WS. In: EMISA'04. (2004) 117–128
7. Rinderle, S., Reichert, M., Dadam, P.: Correctness criteria for dynamic changes in workflow systems – a survey. DKE **50** (2004) 9–34
8. Rinderle, S.: Schema Evolution in Process Management Systems. PhD thesis, University of Ulm (2004)
9. Haarslev, V., Möller, R.: Description of the racer system and its applications. In: Proceedings International Workshop on Description Logics (DL-2001), Stanford, USA, 1.-3. August. (2001) 131–141
10. Hayes-Roth, F.: Rule-based systems. Commun. ACM **28** (1985) 921–932
11. Rinderle, S., Weber, B., Reichert, M., Wild, W.: Integrating process learning and process evolution - a semantics based approach. In: BPM'05. (2005)
12. Andrews, T., Curbera, F., Dholakia, H., et al., Y.G.: BPELWS - Business Process Execution Language for Web Services. (2003) BEA Systems, International Business Machines Corporation, Microsoft Corporation, SAP AG, Siebel Systems.
13. Rinderle, S., Reichert, M., Dadam, P.: On dealing with structural conflicts between process type and instance changes. In: BPM'04. (2004) 274–289
14. Rinderle, S., Reichert, M., Dadam, P.: Disjoint and overlapping process changes: Challenges, solutions, applications. In: CoopIS'04. (2004) 101–120

15. Reichert, M., Dadam, P.: ADEPT$_{flex}$ - supporting dynamic changes of workflows without losing control. JIIS **10** (1998) 93–129
16. Maviglia, S., Zielstorff, R., Paterno, M., Teich, J., Bates, D., Kuperman, G.: Automating complex guidelines for chronic disease: Lessons learned. Journal of American Medical Inf. Ass. **10** (2003) 154–165
17. Blaser, R., Schnabel, M., Heger, O., Opitz, E., Lenz, R., Kuhn, K.: Improving pathway compliance and clinician performance by using information technology. In: MIE'05. (2005)
18. Boxwala, A., Peleg, M., Tu, S.: GLIF3: a representation format for sharable computer-interpretable clinical practice guidelines. Biomed Inform. **37** (2004) 147–61
19. Quaglini, S., Stefanelli, M., Cavallini, A., G, G.M., Fassino, C., C, C.M.: Guideline-based careflow systems. Artif Intell Med **20** (2000) 5–22
20. Weske, M.: Flexible modeling and execution of workflow activities. In: Proc. Hawaii Int'l Conf. on System Sciences, Hawaii (1998) 713–722
21. van der Aalst W. M. P., Basten, T., Verbeek, H.M.W., Verkoulen, P.A.C., Voorhoeve, M.: Adaptive workflow: On the interplay between flexibility and support. Interprise Information Systems (2000) 63–70
22. Pathak, J., Caragea, D., Honovar, V.: Ontolgy-extended component-based workflows: A framework for constructing complex workflows from semantically heterogeneous software components. In: SWDB'04. (2005) 41–56
23. Bowers, S., Lin, K., Ludäscher, B.: On integrating scientific resources through semantic registration. In: SSDBM'04. (2004)
24. Kim, J., Gil, Y., Spraragen, M.: A knowledge-based approach to interactive workflow composition. In: ICAPS 04. (2004)
25. Cardoso, J., Sheth, A.: Semantic e-workflow composition. JIIS. **21** (2003) 191–225
26. Zhang, R., Arpinar, I.B., Aleman-Meza, B.: Automatic composition of semantic web services. In: Intl. Conf. on Web Services, Las Vegas NV, June 2003. (2003)
27. Hayes-Roth, F., Jacobstein, N.: The state of knowledge-based systems. Commun. ACM **37** (1994) 26–39
28. Ader, M.: Ilog components for business process management solutions (2002)
29. Greiner, U., Ramsch, J., Heller, B., Löffler, M., Müller, R., Rahm, E.: Adaptive guideline-based treatment workflows with adaptflow. In: CGP 2004. (2004) 113–117
30. Müller, R., Greiner, U., Rahm, E.: Agentwork: A workflow system supporting rule-based workflow adaption. DKE **51** (2004) 223–256
31. v. d. Aalst, W., de Beer, H., van Dongen, B.: Process mining and verification of properties: An approach based on temporal logic. In: CoopIS'05. (2005) 130–147
32. Nahler, M.: Semantical conflicts in adaptive process managament systems (2005) (in german).