On-the-Fly Change Propagation for the Co-evolution of Business Processes

Shamila Mafazi, Georg Grossmann, Wolfgang Mayer, and Markus Stumptner

University of South Australia, Adelaide, SA, 5095, Australia
shamila.mafazi@mymail.unisa.edu.au,
firstname.lastname@unisa.edu.au

Abstract. In large organisations multiple stakeholders may modify the same business process. This paper addresses the problem when stakeholders perform changes on process views which become inconsistent with the business process and other views. Related work addressing this problem is based on execution trace analysis which is performed in a post-analysis phase and can be complex when dealing with large business process models. In this paper we propose a design-based approach that can efficiently check consistency criteria and propagate changes on-the-fly from a process view to its reference process and related process views. The technique is based on consistent specialisation of business processes and supports the data- and control flow perspective. This technique reduces the steps performed in the evolution of business processes by embedding the consistency checks and change propagation into the change enactment phase of the evolution.

1 Introduction

It is apparent that business processes evolve over time and have to be aligned with business rules, legislations, and law. Whereas, due to factors such as continuing change in regulations, policies, business compliances, and user demand the processes are subject to change. Since the economic success of any businesses relay on its ability to respond to the changes in its area swiftly, it is important for business to be able to update their processes in efficient and correct way [1].

Designing and updating process models in large organisations are usually performed by multiple stakeholders in parallel. Each stakeholder has a particular view on a process and performs changes independently from other stakeholders leading to the challenge that views may become out of sync and inconsistent with its reference process and other views. In this paper we propose a framework that offers generic methods and techniques for the efficient change propagation from a stakeholder to reference process and its process views on the process model level.

Gerth et al. [2] identified three steps for change management in process models: detection of differences, identification of change conflicts, and resolution of conflicts. We propose to propagate changes on the fly so stakeholders become immediately aware of changes made by other stakeholders which affect their

R. Meersman et al. (Eds.): OTM 2013, LNCS 8185, pp. 75–93, 2013.

view. By providing these capabilities, a separate step for conflict identification and resolution can be kept to a minimum or in optimal cases be eliminated. In this paper we focus on the mechanism on how to propagate changes on-the-fly and deal with changes made on the same process fragment at the same time in future work.

Related work often assumes that two corresponding process models are in strict refinement relation, whereas such a refinement relation is hardly noticed in practice [3]. We are applying existing techniques based on those strict process hierarchies but use this technique to propagate changes efficiently, i.e., we allow users to perform changes that violate the refinement relations and then propagate changes so consistency is established again. In particular, we are applying consistent specialisation of business processes to ensure consistency between views and reference process. By checking simple rules on the structure of process models and keeping relationships between elements in different processes, we are able to achieve co-evolution more efficiently.

The main contribution is twofold: In Section 3 we present our framework for on-the-fly change propagation in process co-evolution on the model level, and in Section 4 we provide a literature review and comparison criteria of state-of-the-art work in process (co-)evolution and change management.

The paper is structure as follows: the next section includes a motivating example which is used throughout the paper. Afterwards we explain our approach followed by a comparison of related work. The last section provides an outlook on future work and conclusion.

2 Motivating Example

Process models are widely used within organisations while they are mostly large and complicated. On the other hand dealing with a large and complicated process model is not always necessary, as preforming different business tasks need different parts of a process models to be highlighted. To fulfil this requirement, several methods to provide process views regarding different aspects of the process model have been proposed such as [4–6]. In previous work [7], we proposed a knowledge based framework to provide different views on a process model considering the user specified constraints. We have applied our approach on a real-world business process repository from the *product and system engineering* domain which contained software development processes with up to 260 activities in a single process. In some use cases, a view could be generated that reduced the complexity by up to 90%.

Different users may change their views over a reference model. To preserve the consistency between them change in one view needs to be propagated to the remaining views through the reference model. The co-evolution of process models refers to the change propagation between models either at the same or different abstraction hierarchy [8]. These changes may cause inconsistency in the process model views or the reference model. In this work we deal with inconsistent models in a model co-evolution scenario.

Fig. 1 indicates our motivating scenario, where every user has their own view over the reference model. As indicated in the figure users make changes in their process views. To preserve the consistency between these process models, change in one view needs to be propagated to the remaining views through the reference process model.

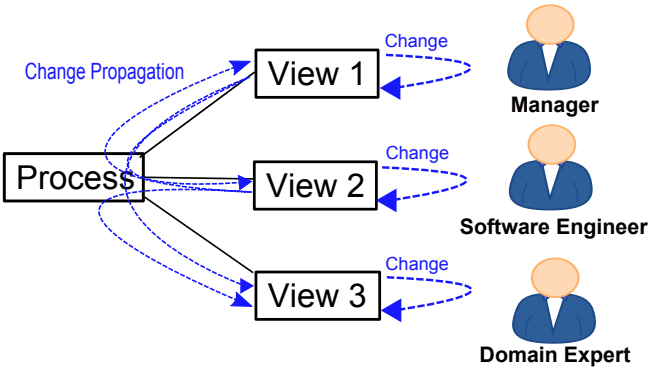

Fig. 1. Motivating Scenario

Fig. 2 shows an example for co-evolution of a reference process model and its views in BPMN. The model in the lower left depicts the reference model, the models in the top show two different views over the reference model. The grey and pink dashed lines indicate the correspondence relations for the tasks in process view 1 and view 2 respectively. In this figure the correspondence relations for gateways, control flows, and identical activities are left implicit. As illustrated in the figure, the mappings between views and the reference process model are different. For example tasks *Cancel Late* and *Send Cancellation Confirmation* from the reference model have been mapped to the task *Cancel* in process view 1 whereas, in process view 2, task *Send Cancellation Confirmation* has been removed by mapping it to a control flow in the view.

In the example, a new activity *Inform* is introduced by a stakeholder in View 2 which is highlighted by a bubble within View 2 in Fig. 2. This change is then propagated to the reference process by inserting *Inform* between *Cancel Late* and *Send Cancellation Confirmation*. In a last step, the changes are propagated to View 1 which consists only of an update of the properties in *Cancel* because the region including *Cancel Late* and *Send Cancellation Confirmation* is abstracted in View 1.

Related work mostly deals with schema evolution and migration of process instances, preserving the data flow consistency. To the best of our knowledge on-the-fly propagation of changes among different views at different abstraction levels has not been investigated yet. The following section presents our framework.

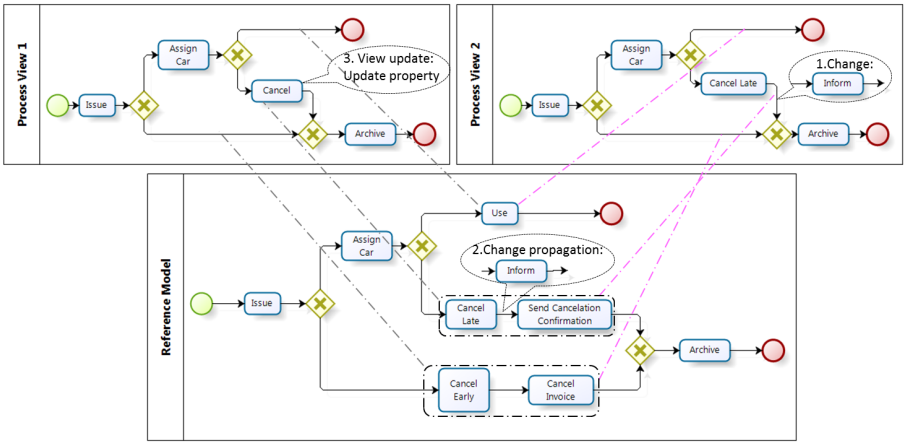

Fig. 2. Change Propagation Example

3 Consistent Co-evolution of Business Process Modelling

A typical software development life-cycle consists of several phases. Köhler et al. [9] introduced a cycle for the alignment of business and IT which consists of model-, develop-, deploy-, monitor- and analyse & adapt phases. We propose a framework for the consistent co-evolution of business processes in the modelling phase which consists of three steps: (1) in the *process modelling* step, a business process is created that represents the reference process, (2) in the *process view generation* step, views are generated from the reference process for stakeholders, and (3) in the *change and change propagation* step, different stakeholders perform changes on their views and their changes are propagated on the fly to reference processes and other views. The first step needs to be performed only once whereas steps 2 and 3 are usually repeated for each development cycle. This framework makes use of the *model driven engineering, MDE* paradigm. The advantages of MDE include increasing the productivity and compatibility by simplifying the process of design, and maximising reusability by lifting platform specific code to a platform independent model level. Our approach is set on the level of models which are the central artifact in MDE. The problem of consistent software evolution is also addressed by other communities such as semantic web comunity. However, these communities rather target a lower level of implementation and specific language whereas our approach is on the platform independent level.

We explain now each step in more detail and provide definitions for the formal notation and consistency constraints.

3.1 Process Modelling

There exist different languages for modelling business processes. BPMN has become the de-facto standard in recent years and we use this notation to represent the models. However, a more formal representation is required for specifying the relationship between reference process and process views, and for the verification of consistency.

Therefore we use a Petri net representation of the BPMN models, in particular labelled Petri nets which are helpful to express the consistent specialisation of processes [10]. Different approaches for mapping between the BPMN and Petri net notation exist. Since the discussion of such mappings are out of the scope of this paper we refer the interested users to [11].

A labelled Petri net is a Petri net that has labels attached to its arcs. As discussed in [10] the idea of labelling arcs corresponds to the mechanism by which different copies of a form are handled by each activity in business processes guided by paperwork, where different copies of a form have different colours. Each activity deals with different copies of a form where it can collect the copies from different input boxes and deliver them to different output boxes. The labels determine which copies have been used by which activities.

Definition 1 (Labelled Petri Net)
A tuple: $(N, F, D, D_{in}, D_{out}, \delta_{in}, \delta_{out}, L, l)$ *is a labelled Petri Net model, where* N *is a finite set of nodes partitioned into disjoint sets of activities* N_t *and states* N_s*,* $F \subseteq (N_s \times N_t) \cup (N_t \times N_s)$ *is the flow relation such that* (N, F) *is a connected graph,* D *is a finite set of data variables,* $D_{in} \subseteq D$ *and* $D_{out} \subseteq D$ *are the sets of input and output variables of the process, and* $\delta_{in} : N_t \mapsto 2^D$ *and* $\delta_{out} : N_t \mapsto 2^D$ *are functions mapping each activity node to its set of input and output variables, respectively. We write* $n_t.D_{in}$ *and* $n_t.D_{out}$ *for* $\delta_{in}(n_t)$ *and* $\delta_{out}(n_t)$*, respectively.* L *is a finite set of labels, and* $l : F \mapsto 2^L \setminus \emptyset$ *is a function that assigns a set of labels to each control flow. Moreover, we write* F^* *to denote the transitive closure of the control flow relation* F*.*

Different operations, such as initializing, updating, and verifying, performed by each activity in a process model, can be divided to read and write operations from an implementation point of view. In our framework, when an activity x reads or writes a data object, that data object is considered to be the input or the output of that activity denoted as $x.D_{in}$ and $x.D_{out}$ respectively.

The execution semantics of a labelled Petri Net is the same as a Petri Net where an activity produces a token to its immediate post states and consumes a token from its immediate pre states. For the reference process model and all its views, we assume a subset of labelled Petri Net which satisfy the certain properties:

- Safe: a labelled Petri Net is safe iff there is no execution trace in the model such that some activities on that trace can be completed while there exist some post states of those activities on that trace. This is a necessary property

as unsafe Petri Net contradicts the completeness and intention of a behaviour diagram.

- Activity reduced: this property is satisfied iff for every activity there is an execution trace that contains it. This property is necessary as the process model must not contain an activity which does not have any impacts on the process model.
- Deadlock: this property ensures that the execution must continue unless it reaches a final state. This is an important property as it prevents from blocking execution.
- Label preservation: requires that for every activity the incoming and outgoing arcs contain the same set of labels. In the real world scenario, this property ensures that once a specific form for a process instance has been developed, that form is neither overwritten nor are new copies of the form produced by other activities in the process model.
- Unique label: this property ensures that the label set of the incoming arc(s) of every joint or split activities be different from its outgoing label set. This property ensures that every activity can have some copies of a form only one input box and must deliver them to only one output box.
- Common label distribution: To preserve this property all the immediate arc(s) of a state must contain the same set of labels. This property ensures that each box contains the same copies for a form.

Figure 3 shows the Petri net representation of the BPMN process model shown in Fig. 2. Data input and -output are indicated above each activity as label in form of $d*$ where $*$ is an ID indicating which activity consumes data from another activity. For example, the activity *Assign Car* in Process View 1 of Figure 3 has inputs $d1$–$d3$ which are produced by activity *Issue*.

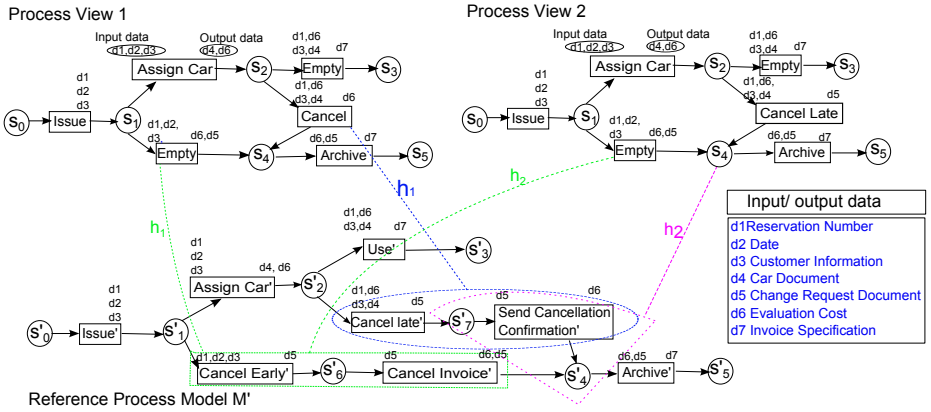

Fig. 3. Petri Net model for process models m in Fig. 2, for clarity's sake the control flow labels are omitted

To capture the relation of two activities regarding the execution sequence we base our formalization on the concept of *dominance* by Cytron et al. [12]. The binary *dominance* defines a relation between process model elements, considering all possible execution traces:

Definition 2 (Dominance Relation). *Let P be a labelled Petri Net model with nodes N and flow relation F. A non-empty set $D = \{n_1, ..., n_k\} \subseteq N$* dominates *a node $n \in N$, denoted as $D \triangleleft n$, if and only if in all execution traces from the start node to node n, a node in D precedes every execution of n.*

Example 1. Node *Issue* in Fig. 3 View 1 dominates all subsequent nodes; and node set {*Empty*, *AssignCar*} dominates node *Archive*.

Definition 3 (Immediate Pre-Node and Post-Node)
The set of pre-nodes of node $n \in N$ is defined as $\bullet n = \{n_1 \in N | (n_1, n) \in F\}$. Likewise the set of the post-nodes of node $n \in N$ is defined as $n\bullet = \{n_1 \in N | (n, n_1) \in F\}$.

Example 2. $\bullet$CancelLate=$\{s_2\}$, and CancelInvoice$'\bullet$=$\{s'_4\}$ in Figure 3.

After a reference process is modelled in BPMN and its equivalent representation as a labelled Petri net is generated, process views can be created from it.

3.2 Creating Process Views

In our previous work, we describe a knowledge-based framework for the purposeful abstraction of process models from constraints [7]. Constraints can be specified on activity properties and process views are generated automatically which fulfil the constraints. A constraint is specified according to the interests of a particular stakeholder, for example, a performance manager may only be interested in observing activities which take longer than a specified threshold. The framework is based on configuration techniques and can provide multiple views on different abstraction levels. The correspondences between a process model and its views are captured by a function while constructing the views. This allows checking for consistency and change propagation later in the modelling phase.

In our framework the main functions for abstracting a process model are *AggregateActivities()* and *RemoveActivity()*. By aggregating a group of activities from the reference model and mapping them to a composite activity in the view, a n:1 relationship is established.

Example 3. In Figure 3, the activities *Cancel late* and *Send Cancellation Confirmation* of Reference Process M' are aggregated to *Cancel* in Process View 1.

For removing irrelevant activities from a view, the *RemoveActivity()* function is applied. There are two possible outcomes: nodes are either removed or replaced by a *silent activity*. In the first case, a relationship between the deleted nodes in

the reference model and an arc representing a control flow in the view is established. The second case occurs if removing a node would violate the consistency between the reference process and its view. In this case, a relationship between the deleted nodes and a *silent activity* is established.

Definition 4 (Silent activity)
An empty activity $N_e \subseteq N_t$ is a silent activity [11] which has no effect in the process model but it is required to preserve the syntax of Petri net notation as well as the model consistency.

Example 4. In Figure 3, activity *Use* is mapped to a silent activity labelled "Empty" in both views. This is necessary because removing it would make states s_2 in both views a final state which is inconsistent with the reference process.

We capture the relationship between elements in the reference process and its views by a total mapping function $h()$ such that:

1. It maps each state, activity, data and control flow of the reference process model to the elements of the view.
2. Labels of the reference process are mapped to labels in the view. Different labels in the reference process model can be mapped to a single label in the view.
3. The initial state of the original model is mapped to the initial state of the changed model.

Definition 5 (Process View). *A labelled Petri Net diagram (for brevity we excluded the functions from the definition of the labelled Petri Net)* $M = (N, F, D, D_{in}, D_{out}, L)$ *is defined as a view of another labelled Petri Net diagram* $M' = (N', F', D', D'_{in}, D'_{out}, L)$ *by the function* $h_{M' \mapsto M}$.
$h : N' \cup F' \cup D'_{in} \cup D'_{out} \cup L' \mapsto N \cup F \cup D_{in} \cup D_{out} \cup L$

1. $e \in N' \cup F' \Rightarrow h(e) \in N \cup F$,
2. $e \in D'_{in} \cup D'_{out} \Rightarrow h(e) \in D_{in} \cup D_{out}$,
3. $l' \in L' \Rightarrow h(l') \in L$,
4. $a' \in N'_s, a \in N_s, {\bullet}a = \emptyset, {\bullet}a' = \emptyset \Rightarrow h(a') = a$

Example 5. Figure 3 shows two process views that have been generated from the reference process M' where, for example, the sequence $\{s'_7$, *Send Cancellation Confirmation*$', s'_4\}$ in the reference model is mapped to state s_4 in View 2. The relationship between the reference process and the view for this mapping is specified as $h(s'_7) = h($*Send Cancellation Confirmation*$') = h(s'_4) = s_4$.

The *process view generation* step ensures that a generated view is consistent with the reference process. In the following section we describe the consistency criteria between references process and views, and how they can be checked using simply design rules.

3.3 Consistency Criteria

The problem of inconsistency between multiple views of a reference model is well-studied [13]. Consistency criteria help to identify contradictions between views and their reference process model once different stakeholders change the views. In our framework we apply *consistent specialization of business processes* to ensure the consistency between process views and their respective reference process model.

Specialisation consists of *refinement* and *extension* where refinement refers to refining an activity into more detail and extension refers to adding new elements to a process model. A process model P' is an extension of process model P if it possesses new features in comparison to P. Likewise, process model P' is a refinement of a process model P if the inherited features are considered in more detail.

Schrefl et al. [10] investigated three criteria for the consistent refinement and extension of business processes: *observation consistency*, *weak-* and *strong invocation consistency*. Informally, observation consistent specialization guarantees that if features added at a process are ignored and features refined at a changed process model are considered unrefined, any instance of that changed process can be observed as correct from the point of view of the original process.

Weak invocation consistency captures the idea that instances of a changed process can be used the same way as instances of the original process. An extended version of this property, strong invocation consistency, guarantees that one can continue to use instances of a changed process the same way as instances of a original model, even after activities that have been added in the changed process model have been executed.

These criteria relate to other methods that investigated consistency between business processes. Protocol- and projection inheritance introduced by Basten et al. [14] corresponds to weak invocation consistency and observation consistency respectively. The advantage of the work by Schrefl et al. is a set of rules that allow to efficiently check for consistency without the necessity to analyse all possible traces.

Fig. 4 shows the consistency rules by which the consistency of a process model is assessed. According to Schrefl et al. [10] preserving the extension rules E_1 to E_3, on the left side of Fig. 4 guarantees the observation consistency between the two models, whereas refinement rules R_1 and R_2 from Fig. 4 assure observation and invocation consistency. A process model A' is the observation consistent with another model A if, by ignoring the new features of A', the instances of A' can be observed seamlessly by A. Two process models are invocation consistent if all the instances of process model A' can be invoked by the process model A.

The rules shown in Fig. 4 are used to check for consistency of control flows between reference process and views. As mentioned above, there are two types of consistency rules, rules for extension and rules for refinement, depicted on the left and right hand side in Fig. 4. Which consistency rule needs to be checked depends on the applied change. For example if an activity is inserted into or

Extension Rules:

E1. Partial Inheritance
(a) $\alpha' = \alpha, L \subseteq L'$
(b) $t \in N_t' \wedge t \in N_t \wedge (s,t) \in F$
$\Rightarrow (s,t) \in F' \wedge l(s,t) \subseteq l'(s,t)$
(c) $t \in N_t' \wedge t \in N_t \wedge (t,s) \in F$
$\Rightarrow (t,s) \in F' \wedge l(t,s) \subseteq l'(t,s)$
E2. Immediate definition of pre/post-states
(a) $(s,t) \in F' \wedge s \in N_s \wedge t \in N_t$
$\Rightarrow (s,t) \in F$
(b) $(t,s) \in F' \wedge s \in N_s \wedge t \in N_t$
$\Rightarrow (t,s) \in F$
(c) $(s,t) \in F' \wedge s \in N_s \wedge t \in N_t \wedge$
$x \in L \wedge x \in l'(s,t) \Rightarrow x \in l(s,t)$
(d) $(t,s) \in F' \wedge s \in N_s \wedge t \in N_t \wedge$
$x \in L \wedge x \in l'(t,s) \Rightarrow x \in l(t,s)$
E3. No label deviation
(a) $(s,t) \in F' \wedge t \in N_t' \wedge t \notin N_t$
$\Rightarrow l'(s,t) \cap L = \oslash$
(b) $(t,s) \in F' \wedge t \in N_t' \wedge t \notin N_t$
$\Rightarrow l'(t,s) \cap L = \oslash$
(c) $(s,t) \in F' \wedge s \in N_s' \wedge s \notin N_s$
$\Rightarrow l'(s,t) \cap L = \oslash$
(d) $(t,s) \in F' \wedge s \in N_s' \wedge s \notin N_s$
$\Rightarrow l'(t,s) \cap L = \oslash$

Refinement Rules:

R1. Pre and post-state satisfaction
(a1) $t \in N_t, s \in N_s, \hat{s} \in N_s, t' \in N_t', s' \in N_s' : h(t') = t \wedge$
$h(s') = s \wedge (s,t) \in F \wedge (s',t') \in F' \wedge (\hat{s},t) \in F$
$\Rightarrow \exists \hat{s'} \in S' : h(\hat{s'}) = \hat{s} \wedge (\hat{s'},t') \in F'$
(a2) $t \in N_t, s \in N_s, t' \in N_t', s' \in N_s' :$
$h(t') = t \wedge h(s') = s \wedge (s,t) \in F \wedge (s',t') \in F'$
$\wedge x \in l(s,t) \wedge x'' \in h^{-1}(x)$
$\Rightarrow \exists s'' \in N_s' : h(s'') = s \wedge (s'',t') \in F' \wedge x'' \in l'(s'',t')$
(b1) $t \in N_t, s \in N_s, \hat{s} \in N_s, t' \in N_t', s' \in N_s' :$
$h(t') = t \wedge h(s') = s \wedge (s,t) \in F \wedge (t',s') \in F' \wedge (t,\hat{s}) \in F$
$\Rightarrow \exists \hat{s'} \in N_s' : h(\hat{s'}) = \hat{s} \wedge (t',\hat{s'}) \in F'$
(b2) $t \in N_t, s \in N_s, t' \in N_t', s' \in N_s' :$
$h(t') = t \wedge h(s') = s \wedge (t,s) \in F \wedge (t',s') \in F'$
$\wedge x \in l(t,s) \wedge x'' \in h^{-1}(x)$
$\Rightarrow \exists s'' \in N_s' : h(s'') = s \wedge (t',s'') \in F' \wedge x'' \in l'(t',s'')$
R2. Pre and post-state refinement
(a1) $s' \in N_s' \wedge t' \in N_t' \wedge (s',t') \in F' \wedge h(s') \neq h(t')$
$\Rightarrow (h(s'),h(t')) \in F$
(a2) $s' \in N_s' \wedge t' \in N_t' \wedge (s',t') \in F' \wedge h(s') \neq h(t')$
$\wedge x' \in l(s',t')$
$\Rightarrow h(x') \in l(h(s'),h(t'))$
(b1) $s' \in N_s' \wedge t' \in N_t' \wedge (t',s') \in F' \wedge h(s') \neq h(t')$
$\Rightarrow (h(t'),h(s')) \in F$
(b2) $s' \in N_s' \wedge t' \in N_t' \wedge (t',s') \in F' \wedge h(s') \neq h(t')$
$\wedge x' \in l(t',s')$
$\Rightarrow h(x') \in l(h(t'),h(s'))$

Fig. 4. Rules for checking behaviour consistency of extension (on the left) and refinement (on the right) [10]

deleted from a view, the extension rules are relevant. In the following we discuss data flow consistency which is checked locally in a view once changes are applied.

Data Flow Consistency Criteria: We discuss three data-flows anomalies investigated by Reichert et al. [1] which are *missing data*, *unnecessary data*, and *lost data*.

In the following firstly the rules for identifying the violation of data flow consistency after propagating all the required changes in a process model are presented. Secondly, the repair option(s) for each violation are presented.

Missing data refers to the situation where a data object o is read by an activity x before it has been written. In our framework, the rule to ensure the absence of missing data in a model, can be expressed formally as:
$\forall x \in N_t : o \in x.D_{in} \Rightarrow \exists Y \subset N_t : Y \triangleleft x \wedge o \in \bigcap_{y \in Y} y.D_{out}$
This states that for each reader x of o, there must be a dominating set Y where each member writes o.

$\forall x \in N_t : o \in x.D_{in} \rightarrow \exists y \in N_t : (y,x) \in F^* \wedge o \in y.D_{out}$
If a process model contains missing data, i.e. the above rule gets violated, to repair the inconsistency, the process modeller has two options:

1. The missing data object is added to a preceding activity, i.e. $AddOutputDataFlow(y,o)$
2. The activity causing the missing data inconsistency situation is removed from the process model, i.e. $RemoveActivity(x)$

An *unnecessary data* situation occurs where a data object o is written by an activity x, but is not read by any activity (such as y) afterwards in the process model. Formally the rule is expressed as:
$\forall x \in N_t : o \in x.D_{out} \Rightarrow \exists y \in N_t : (x, y) \in F^* \wedge o \in y.D_{in}$

In case of violation of the rule, the data object o which causes the inconsistency needs to be removed from the output data of the activity x, that is $RemoveOutputDataFlow(x, o)$. The other option is to map the data object o to the input data set of a successor activity. However considering this option, requires change in the internal functionality of that successor activity which cannot be considered at the model level.

Lost data happens when a data object o is written twice by two consecutive activities x and y without being read in between. Formally:

$$\forall x, y \in N_t : o \in x.D_{out}, o \in y.D_{in}, (x, y) \in F^* \Rightarrow \nexists Z \subset N_t \setminus \{x\} : Z \lhd y \bigwedge_{z \in Z} [o \in z.D_{out} \wedge (x, z) \in F^*]$$

A data object o written by x is lost data if any potential reader y of o reachable from x is dominated by a set of writers Z whose members are reachable from x. Hence, o written by x is not lost if no such set Z exists. To repair the lost data flow inconsistency, in case of violation of the mentioned rule, Sadiq et al. [15] provided two options for the process modeller in order to resolve the inconsistency as follows:

1. The modeller selects one activity to write data object o and that data object is removed from the output data set of the other consecutive activity(ies) writing that object.
2. The other option is to change the data object o in one of the consecutive activities.

The consistency of control- and data flow is checked after changes are made to a process view. In the following we discuss applying changes to views in general and then how the consistency criteria are used in the change propagation step.

3.4 Performing Changes

Our library of change operators currently contains two types of operators: structural and entity change operators. For change propagation we address two types of change operations, adding and deleting activities. As indicated by related work, a majority of change patterns can be composed out of those operators [16]. An example for entity change operators are also *add* and *remove* which are applicable on the data inputs and outputs of activities. Table 5 indicates a list of operators currently defined in our library.

3.5 Change Propagation

If one of the views changes, the changes need to be propagated to restore consistency between the views and the reference model. However in some cases there

Operator	Type
Remove Node/Flow	Structural
Add Node/Flow	Structural
Remove Data flow	Entity
Add Data flow	Entity

Examples for Logic Representation of Operators

RemoveActivity(x):
$\forall x \in N_t, e \in N_e \rightarrow$
$h(x) = e \wedge e.D_{in} \leftarrow \oslash \wedge e.D_{out} \leftarrow \oslash$
AddOutputDataFlow(x, i):
$x.D_{out} \leftarrow i$
RemoveOutputDataFlow(x, o):
$x.D_{out} \leftarrow o$

Fig. 5. Change Operators

is an issue, regarding preserving the consistency and privacy of a process view. On the one hand one can imagine that some changes applied by a stakeholder are private or sensitive which cannot be shared with other views or added to a reference model. On the other hand to preserve the consistency, every change in a process view needs to be propagated to the other views as well as the reference model regardless of the sensitivity or privacy of those changes. One possible compromise is that the private changes in a view can be propagated anonymously i.e. as an empty task, such that the changes are not visible to other stakeholder yet the changes have been propagated hence the consistency is preserved. Fig. 6 illustrates a change propagation scenario where activity *Inform* needs to be inserted in Process View 2 after activity $CancelLate$. The scenario includes the following steps:

Apply Changes: A stakeholder applies changes on a view using available operators.

Example 6. As illustrated in Fig. 3, activity *Inform* is inserted in View 2.

Check Process Properties: A first correctness check includes checking for local properties: safe, activity-reduced, deadlock-free, and label properties as explained in Section 3.1 and data inconsistencies. The process properties can be checked efficiently using techniques such as SESE fragmentation [17]. If properties or data inconsistencies are violated then the process model must be changed in a way so they are satisfied and no data flow violation occurs.

Example 7. After activity *Inform* is inserted, the process view remains safe, activity-reduced and deadlock-free, and no data flow inconsistencies were introduced.

Check Consistency Rules: In a following step the consistency rules shown in Fig. 4 are checked for observation, weak- and strong invocation consistency. If the criteria are violated then the stakeholder is informed that the applied changes will have an impact on reference process and other views. This provides an option to undo certain changes in case the impacts of the changes were unintentional.

Example 8. In the example Views 2 in Fig. 3 and 6 are not consistent after activity *Inform* is inserted because it violates the observation consistency. As the

post-states of *Cancel Late* in the two views are different, the post-state satisfaction, rule b_1 in Fig. 4 is violated. Hence View 2 in Figure 6 is not a behaviour consistent specialisation of View 2 in Fig. 3. The stakeholder receives a warning about this and proceeds because the change was intentional and has to be propagated.

Propagate Changes to Reference Model: The same change operators that have been applied on the view are applied on the reference process with the difference that the arguments of the change operators are mapped according to the relationship between the elements of the view and the reference process. For this the *h()* function is used which relates each element from one process to the other. Further the relationship between the reference process and the view needs to be updated in the implementation of the *h()* function.

Example 9. As illustrated in reference model from Fig. 6 activity *Inform* is added to the reference process. The relationship between the reference process and View 2 is updated with $h(s'_6) = \{s_6\}$ and $h(Inform') = \{Inform\}$.

Propagate Changes to Views: First, for each view, its consistency with the reference model is checked. If the process and the view are consistent and the changes in the reference process affect a region that is abstracted in the view then no changes need to be propagated. Otherwise the same change operator as on the reference process is applied on the view where the arguments of the operator are translated by using the mapping function *h()* that maps elements in the reference process to the view.

Example 10. In Fig. 6 the reference model is not a behaviour consistent refinement of View 1. Checking the changed region in the reference model from Fig. 6 and $h()$ function, it is obvious that no change needs to be applied in the structure of the process View 1. To keep View 1 consistent with the reference model, only the activity $Inform'$ and its pre-state s'_6 from the reference model need to be mapped to the activity *Cancel* from process View 1, that is: $h(s'_6) = h(Inform') = \{Cancel\}$ between reference process and View 1.

4 Related Work

The following subsections discuss the related work directly addressing change propagation between process models as well as the approaches which are potentially useful in the co-evolution of process models such as finding corresponding process model elements. All the mentioned approaches are compliment to one aspect presented in this paper. The following sub sections discuss the three main aspects in the change management as, *process model alignment*, *co-evolution* and *variability*. The results of comparision of related work have been captured in tables 1 and 2 where the first table compares the related work in the first area and

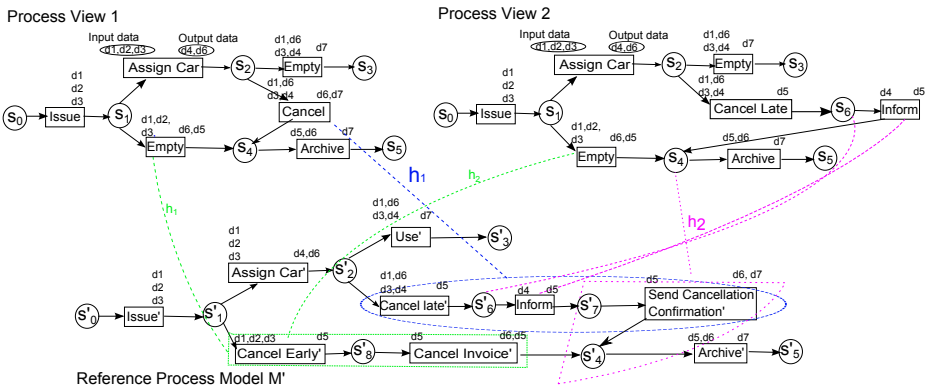

Fig. 6. Change propagation steps following changes made to Fig. 3, for clarity sake the control flow labels are omitted

the second table the two former areas. In these tables rows specify the papers and indicate their support towards the following criteria:

Goal: This column classifies the related work based on their goal.

Data Flow (Consistency): The data flow column indicates whether the related work considers data flow consistency in a change propagation scenario. This criteria is important as the robustness of a process model is fundamentally depended on the correctness of the data flow [1]. The possible values include supported, not supported, the criterion is not in the scope of the paper, and the criterion is in the scope of the paper but it has not been discussed.

Bidirectional Change Propagation: This column indicates the support of the approach for a top down and bottom up change propagation where different levels of abstraction hierarchy has been considered by the approach. This is an important aspect since the change cannot be restricted to any particular abstraction levels.

Support for Process Views: This column indicates the support of the approach for different process views. This column holds the same possible values as the previous column.

Consistency Preservation: The column discusses the consistency criteria proposed by the related work in the change propagation scenario. This is an important criterion as in practice there is no single process model but a group of process models. In order to preserve the uniformity between such models, the consistency between them must be evaluated against formal criteria.

Change Operators: This column indicates whether the change propagation technique is based on change operators or not. Using an operator based approach has different advantages such as identifying and enabling the traceability of the changed region.

Abstraction Hierarchy: This column discusses whether a process model at different abstraction levels are considered or not in the related work, as in practice process models are usually at different levels.

4.1 Process Model Alignment

One important aspect in propagating changes from one process model to another, is finding corresponding elements in the two process models. Finding corresponding elements in two structures means that for each entity in one structure, a corresponding entity in the other structure, with the same intended meaning can be found. To find the correspondences between models at different abstraction levels, Dijkman et al. [18] used two methods as lexical matching of element pairs and graph matching.

Brockmans et al. [19] proposed a technique for semantic alignment of Petri Net models at different levels of abstraction by translating the process model to OWL. For this purpose they consider the syntactic and semantic similarity between model elements.

Branco et al. [20] deals with management of consistency between models at different levels of abstraction. They proposed an algorithm which establishes corresponding nodes between the models belong to different abstraction levels, based on the type and name of the nodes.

Weidlich et al. [21] represent a framework for matching process model elements. The focus is mainly on addressing the problem of computational complexity of an element from one model which corresponds to multiple elements from another model. Although the approach deals with behavioural consistency between process models, the focus is on inheritance of single process model. We extend their approach to a group of related process models.

Table 1. Comparison of Process Model Change Propagation Approaches in Process Model Alignment

Related Work	*Goal*	*Data Flow*	*Bidirect. Propag.*	*Process View*	*Consist. Pres.*	*Oper.*	*Abstr. Hier.*
Dijkman [18]	Process Alignment	N/A	N/A	+	N/A	N/A	+
Brockmans [19]	Inter-operability	-	N/A	N/A	N/A	N/A	+
Branco [20]	Matching	-	N/A	+	n.d.	-	+
Weidlich [21]	Compatibility	-	N/A	N/A	+	N/A	-
Liu [22]	Create View	-	N/A	+	+	N/A	+
Eshuis [5]	Create View	-	N/A	+	+	N/A	+
Bobrik [23]	Create View	-	N/A	+	+	N/A	+
Zhao [24]	Create View	-	N/A	+	+	N/A	+

Legend: + *supported*, − *not supported*, N/A *not applicable*, n.d. *not discussed.*

Finally, several approaches deal with producing different process views considering different requirements such as, [5, 22–24] while there is no focus on change propagation between these views. Table 1 shows the comparison of the above mentioned approaches.

4.2 Process Co-evolution

Weidlich et al. [3] used *behavioural profile* for identifying the changed regions in a model to propagate the changes between other models at different levels of abstraction. Since any changes in the structure of a process model impacts the relation of nodes, which is behavioural profile, comparing the behavioural profile of the source model and the changed model, the changed region can be identified. In this approach data and properties are not considered. Also the approach depends on the existence of the execution traces of a process model while the computation of the behavioural profile is a costly task.

Fdhila et al. [25] proposed a generic approach to propagate changes in collaborative process scenario. In this scenario different business partners have their own private processes by which a public view is provided for other partners. The proposed approach deals with propagating the changes from a changed view to others preserving the invocation consistency. In compare to our work, in this approach if the consistency of a changed view is not preserved with other views, the change is abandoned. There is no repair plan by which the identified inconsistency can be resolved.

Weidmann et al. [26] has proposed a synchronization approach for process models at different levels of abstraction assuming the model element correspondences exist. The approach focuses on alignment of tasks only and do not consider gateways or tasks properties including data flows.

Dam et al. [27] proposed an agent oriented framework to fix the inconsistencies resulted from change propagation. For this purpose they have created a library including the plans for fixing the constraints violation during the run time. Although the completeness and correctness of generated plans are discussed and guaranteed, since all the possible repair plan choices are considered, the approach is not scalable.

Ekanayake et al. [28] presented a semi-automated technique for change propagation across process model versions. To keep the versions synchronised, for changing a fragment, that fragment is locked until the change is propagated to all the other versions containing that changed fragment.

Gerth et al. [29] proposed a language independent framework for change management. The framework uses workflow graphs as an intermediate representation for the process model, by which the list of differences, dependencies, and conflicts are computed. This framework is applicable in the versioning scenarios where different versions must be refinement of the source model. By merging different versions to get the source model, the changes in the views are identified and propagated to the source model. The possibility that change in one version

may impact other versions is not considered and also the changes are always propagated from the views to the source model not vice versa.

The process co-evolution approach mentioned above are compared in Table 2.

4.3 Process Model Variability

Configuration is a kind of design activity which has been widely used to manage process model variability in literature. The aim is to design a process model from a set of predefined components which can be connected together in a certain way.

Hallerbach et al. [30] introduces *Provop* framework to tackle with the challenges in the management of process model variability, such as the challenge relates to designing a reference model and its predefined adjustments, such that different variants can be derived by configuring the reference model. The configuration is applied on a collection of change operations for control- and dataflow. The framework can ensure the soundness of process variants belong to a process family from data flow perspective but not the control flow. Becker et al. [31] produces different variants by projecting the model elements from a reference process model.

A comparison of the two approaches mentioned above can be found on the bottom of Table 2.

Table 2. Comparison of Process Model Change Propagation Approaches

Related Work	*Goal*	*Data Flow*	*Bidirect. Propag.*	*Process View*	*Consist. Pres.*	*Oper.*	*Abstr. Hier.*
Process Co-Evolution							
Kurniawan [32]	Refinement Preservation	-	+	-	+	+	+
Küster [33]	Conflict resolution	-	+	N/A	+	+	+
Weidlich [3]	Change Propagation	-	+	N/A	+	+	+
Fdhila [25]	Compatibility	-	-	+	+	+	+
Weidmann [26]	Change Propagation	-	+	-	+	+	+
Dam [27]	Change Propagation	-	+	N/A	+	+	-
Gerth [29]	Change Management	-	-	-	n.d.	+	+
Our Approach	Change Propagation	+	+	+	+	+	+
Process Model Variability							
Hallerbach [30]	Process Configuration	+	+	N/A	+	+	+
Ekanayake [28]	Change Propagation	-	+	N/A	n.d.	n.d	+

Legend: + *supported*, − *not supported*, N/A *not applicable*, n.d. *not discussed*.

5 Conclusion

We presented a framework for process co-evolution that applies the notion of consistent specialisation for on-the-fly change propagation. We showed on an example how rules developed for checking consistent refinement and extension of business processes can be used to propagate changes efficiently and re-establish consistency among a reference process and its views. So far the framework supports change propagation on the model level and the rules have been implemented in a process abstraction framework. For the next step in our research agenda we plan to implement those rules in a collaborative process design environment and investigate conflicts and resolution of changes that have been performed on the same process fragment and at the same time.

References

1. Reichert, M., Weber, B.: Enabling Flexibility in Process-Aware Information Systems. Springer (2012)
2. Gerth, C., Küster, J., Engels, G.: Language-independent change management of process models. In: Schürr, A., Selic, B. (eds.) MODELS 2009. LNCS, vol. 5795, pp. 152–166. Springer, Heidelberg (2009)
3. Weidlich, M., Mendling, J., Weske, M.: Propagating changes between aligned process models. Journal of Systems and Software 85(8), 1885–1898 (2012)
4. Smirnov, S., Reijers, H., Weske, M., Nugteren, T.: Business process model abstraction: a definition, catalog, and survey. Distributed and Parallel Databases 30, 63–99 (2012)
5. Eshuis, R., Grefen, P.: Constructing customized process views. Data & Knowledge Engineering 64, 419–438 (2008)
6. Reichert, M., Kolb, J., Bobrik, R., Bauer, T.: Enabling personalized visualization of large business processes through parameterizable views. In: Proc. ACM SAC, pp. 1653–1660. ACM Press (2012)
7. Mafazi, S., Mayer, W., Grossmann, G., Stumptner, M.: A knowledge-based approach to the configuration of business process model abstractions. In: Int'l Workshop on Knowledge-intensive Business Processes (2012)
8. Weidlich, M., Weske, M., Mendling, J.: Change propagation in process models using behavioural profiles. In: Proc. IEEE SCC, pp. 33–40 (2009)
9. Küster, J.M., Koehler, J., Ryndina, K.: Improving Business Process Models with Reference Models in Business-Driven Development. In: Eder, J., Dustdar, S. (eds.) BPM 2006 Workshops. LNCS, vol. 4103, pp. 35–44. Springer, Heidelberg (2006)
10. Schrefl, M., Stumptner, M.: Behavior-consistent specialization of object life cycles. ACM Trans. Softw. Eng. Methodol. 11(1), 92–148 (2002)
11. Dijkman, R.M., Dumas, M., Ouyang, C.: Semantics and analysis of business process models in BPMN. Information & Software Technology 50(12), 1281–1294 (2008)
12. Cytron, R., Ferrante, J., Rosen, B.K., Wegman, M.N., Zadeck, F.K.: Efficiently computing static single assignment form and the control dependence graph. ACM Trans. Program. Lang. Syst. 13, 451–490 (1991)
13. Weidlich, M., Mendling, J.: Perceived consistency between process models. Information Systems 37(2), 80–98 (2012)
14. Basten, T., van der Aalst, W.M.: Inheritance of behavior. The Journal of Logic and Algebraic Programming 47(2), 47–145 (2001)
15. Sadiq, S., Orlowska, M., Sadiq, W., Foulger, C.: Data flow and validation in workflow modelling. In: Proc. Australasian DB Conf., pp. 207–214 (2004)

16. Weber, B., Rinderle, S., Reichert, M.: Change patterns and change support features in process-aware information systems. In: Krogstie, J., Opdahl, A.L., Sindre, G. (eds.) CAiSE 2007. LNCS, vol. 4495, pp. 574–588. Springer, Heidelberg (2007)
17. Fahland, D., Favre, C., Jobstmann, B., Koehler, J., Lohmann, N., Völzer, H., Wolf, K.: Instantaneous Soundness Checking of Industrial Business Process Models. In: Dayal, U., Eder, J., Koehler, J., Reijers, H.A. (eds.) BPM 2009. LNCS, vol. 5701, pp. 278–293. Springer, Heidelberg (2009)
18. Dijkman, R., Dumas, M., Garcia-Banuelos, L., Kaarik, R.: Aligning business process models. In: Proc. of EDOC, pp. 45–53 (2009)
19. Brockmans, S., Ehrig, M., Koschmider, A., Oberweis, A., Studer, R.: Semantic alignment of business processes. In: Proc. of ICEIS, pp. 191–196 (2006)
20. Castelo Branco, M., Troya, J., Czarnecki, K., Küster, J., Völzer, H.: Matching business process workflows across abstraction levels. In: France, R.B., Kazmeier, J., Breu, R., Atkinson, C. (eds.) MODELS 2012. LNCS, vol. 7590, pp. 626–641. Springer, Heidelberg (2012)
21. Weidlich, M., Dijkman, R., Weske, M.: Deciding behaviour compatibility of complex correspondences between process models. In: Hull, R., Mendling, J., Tai, S. (eds.) BPM 2010. LNCS, vol. 6336, pp. 78–94. Springer, Heidelberg (2010)
22. Liu, D.R., Shen, M.: Workflow modeling for virtual processes: an order-preserving process-view approach. Information Systems 28(6), 505–532 (2003)
23. Bobrik, R., Reichert, M., Bauer, T.: View-based process visualization. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 88–95. Springer, Heidelberg (2007)
24. Zhao, X., Liu, C., Sadiq, W., Kowalkiewicz, M.: Process view derivation and composition in a dynamic collaboration environment. In: Meersman, R., Tari, Z. (eds.) OTM 2008, Part I. LNCS, vol. 5331, pp. 82–99. Springer, Heidelberg (2008)
25. Fdhila, W., Rinderle-Ma, S., Reichert, M.: Change propagation in collaborative processes scenarios. In: Proc. CollaborateCom 2012, pp. 452–461 (2012)
26. Weidmann, M., Alvi, M., Koetter, F., Leymann, F., Renner, T., Schumm, D.: Business process change management based on process model synchronization of multiple abstraction levels. In: Proc. SOCA. IEEE Computer Society (2011)
27. Dam, K.H., Winikoff, M.: Generation of repair plans for change propagation. In: Luck, M., Padgham, L. (eds.) AOSE 2007. LNCS, vol. 4951, pp. 132–146. Springer, Heidelberg (2008)
28. Ekanayake, C.C., La Rosa, M., ter Hofstede, A.H.M., Fauvet, M.-C.: Fragment-based version management for repositories of business process models. In: Meersman, R., Dillon, T., Herrero, P. (eds.) OTM 2011, Part I. LNCS, vol. 7044, pp. 20–37. Springer, Heidelberg (2011)
29. Gerth, C.: Business Process Models. LNCS, vol. 7849. Springer, Heidelberg (2013)
30. Hallerbach, A., Bauer, T., Reichert, M.: Capturing variability in business process models: the Provop approach. Journal of Software Maintenance and Evolution: Research and Practice 22(6-7), 519–546 (2010)
31. Becker, J., Delfmann, P., Knackstedt, R.: Adaptive reference modeling: Integrating configurative and generic adaptation techniques for information models. In: Reference Modeling, pp. 27–58. Physica-Verlag HD (2007)
32. Kurniawan, T.A., Ghose, A.K., Dam, H.K., Lê, L.-S.: Relationship-preserving change propagation in process ecosystems. In: Liu, C., Ludwig, H., Toumani, F., Yu, Q. (eds.) ICSOC 2012. LNCS, vol. 7636, pp. 63–78. Springer, Heidelberg (2012)
33. Küster, J., Gerth, C., Engels, G.: Dependent and conflicting change operations of process models. In: Paige, R.F., Hartman, A., Rensink, A. (eds.) ECMDA-FA 2009. LNCS, vol. 5562, pp. 158–173. Springer, Heidelberg (2009)