Conflict Resolution for On-the-fly Change Propagation in Business Processes

Shamila Mafazi[1] **Wolfgang Mayer**[2] **Markus Stumptner**[2]

[1,2] School of IT and Mathematical Sciences,
University of South Australia, Adelaide, SA, 5095, Australia,
[1] Email: shamila.mafazi@mymail.unisa.edu.au
[2] Email: firstname.lastname@unisa.edu.au

Abstract

Process models are widely used in organisations and can easily become large and complex. In the context of business process management, views are a useful technique to reduce complexity by providing only those process fragments that are relevant for a particular stakeholder. A key challenge in view management is the handling of changes that are performed concurrently by different stakeholders. Since the views may refer to the same process, the performed changes may affect the same region of a business process and cause a conflict.

Many approaches have been proposed for resolving conflicts in a post-analysis phase after all changes have been applied. They can be become costly when dealing with multiple changes that lead to multiple conflicts which cannot be resolved automatically and require an additional negotiation phase between stakeholders.

In this paper we propose a framework for the on-the-fly conflict resolution of changes that have been performed on views their underlying reference process. Different to existing approaches this framework applies behaviour consistency rules for business processes which consider the execution semantics and can be checked efficiently on the structure of processes without generating all possible execution traces or keeping track of change operations.

1 Introduction

Since size of a process model is a fundamental factor in its understandability (Rosa et al. 2010), different stakeholders of a process model are usually interested in individual, specific views over a detailed reference process model. This requires creating different process views considering the various requirements. However, models evolve and changes may cause discrepancies between the models.

There are several internal and external drivers for change in a process model. Once a view of a reference model is changed, it may no longer be consistent with the reference process model and other views. Inconsistencies and conflicts are two issues resulting from applying change operators (Barrett et al. 2008). Inconsistencies can arise from changes in the structure of a process model, such as addition or removal of a task, attribute, and data flow, as well as changes to attributes of said elements. Inconsistencies in processes and supporting systems can elevate costs and slow the development (Spanoudakis & Zisman 2001).

In general change may need to be propagated to other models in order to preserve consistency among related views and the reference process. Semi-automated change propagation techniques have been proposed Weidlich et al. (2012), Gerth (2013) but are usually performed in a separate step after all changes have been applied. However, a separate post-analysis step has the disadvantage that it may start another change cycle and can become costly. For example, if multiple changes performed on the same region of a process model by different stakeholders cannot be resolved automatically then the stakeholders must negotiate in a subsequent phase and resolve the conflicts manually.

In Mafazi et al. (2013) we proposed a view management framework that incorporates on-the-fly change propagation for non-conflicting changes. Based on simple rules for checking behaviour consistency (Schrefl & Stumptner 2002), we can efficiently identify inconsistencies caused by changes and point them out to users. Our view management framework supports all aspects of consistency management identified by Branco et al. (2013): defining consistency properties, matching model elements, checking, identifying, and fixing inconsistencies, but not conflict detection and resolution.

This paper focuses on conflict detection, propagation, and resolution of conflicting change operations applied in multiple views. The core contributions are (1) a framework for automatic conflict identification in views based on a common reference process, (2) an analysis of conflict resolution strategies that ensure behaviour consistency, and (3) an consistency restoration mechanism for conflicting attribute changes based on domain ontologies. The proposed approach has the potential to increase automation of conflict detection and resolution, support for control flow semantics without the need to analyse all possible execution traces, and support changing the process structure as well as task attributes.

The remainder of this paper is structured as follows: the next section introduces a motivating example, Section 3 discusses the view management framework and explains the main components, Section 4 covers the conflict resolution, Section 5 provides a literature review covering change propagation and conflict resolution for process modelling and model merging, and the last section provides an outlook on future work.

2 Motivating Example

Rich process models complement control flow information with a variety of entities and attributes that should be considered when making changes. For example, the fact that particular tasks in a process model are executed by a specific role or use the same data document may be used as a relevant criterion for abstracting a process model for a particular stakeholder or purpose. Figure 1 illustrates a conflicting change propagation scenario involving two

process views and a reference process model. In this process model, after issuing the renting request a car is assigned to the customer who then have the option to proceed with the booking or to cancel the request. *Process View 1* is for a stakeholder who is interested in observing those tasks which are executed by the role *Customer* which their execution duration exceeds 7 minutes. Likewise *Process View 2* captures only those tasks which are executed by *Customer* or *Receptionist*. Other tasks that do not satisfy the abstraction criterion are hidden in the respective process view. Each model element in a view is associated with the corresponding model element(s) in the reference process, as captured by a correspondence relation between the two models. For example, tasks *Cancel Late'* and *Send Cancellation Confirmation'* from the reference model are aggregated and mapped to the task *Cancel* in View 1.

Let us assume that *User A* replaces task *Use* in View 1 in Figure 1 with two consecutive tasks *Drive* and *Select Drop-off Loc*. Meanwhile, *User B* replaces this task with two alternate tasks *Drive* and *Postpone Reservation* in process View 2. Since both changes affect the same region in the reference model and they are in conflict, a conflict situation arises as applying one change renders the other one inapplicable. Section 4 provides the solutions for dealing with such conflicts.

3 View Management Framework

This section introduces the main components of our framework. We adopt a process life cycle model where processes transit between the stages *process design, generating views, changing views, on-the-fly change propagation, conflict detection* and *conflict resolution*. A novelty of our approach is that view generation is supported by specific constraints which define how the reference model is abstracted in a view. Using predefined abstraction patterns, we ensure that the abstraction constraints are satisfied and the resulting view generated views are observation consistent Schrefl & Stumptner (2002) with the reference model.

Views and the reference model are subject to change, as they must be aligned with new business rules, regulation, policies and user's demand. Since in large organisations changes are usually performed by multiple stakeholders in parallel, conflicting changes can arise easily. In order to resolve conflicts, suitable changes must be identified and propagate to relevant models.

3.1 Process Design

Although BPMN has become the de-facto standard for process modelling languages, a more formal representation is required for specifying the relationship between reference process and process views and for the verification of consistency rules. We base our framework on a labelled Petri net representation for process models, which is well-suited for expressing behaviour consistent specialisation and correspondences between models (Schrefl & Stumptner 2002), and leverage BPMN only for presentation. Formal mappings from BPMN to Petri net formalisms have been described by Dijkman, Dumas & Ouyang (2008). Figure 2 shows a Petri Net representation of the process model in Figure 1. The labels have been omitted for clarity of presentation.

A labelled Petri net is a Petri net that has labels attached to its arcs. Schrefl & Stumptner (2002) introduced the idea of labelling arcs, which is an analogy to the way different copies of a form are handled in business processes based on paperwork. Each task deals with a different copy of a form where it collects the copies from different sources and delivers them to various recipients. The labels determine which copies are used by which tasks.

Definition 1 (Labelled Petri Net) *A tuple $(N, F, A, D, \delta_A, L, l)$ is a labelled Petri Net model, where N is a finite set of nodes partitioned into disjoint sets of tasks 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 set of value domains, one for each data label. We further require that each $D_l \in D$ is a semi-lattice $(D_l, \preceq_l, \sqcup_l)$ with partial order $\preceq_l$ and least upper bound operator $\sqcup_l$. It is assumed that $u \in D_l$ provides less information about a data property than $v \in D_l$ iff $u \prec_l v$. $\delta_A : N_t \mapsto 2^A$ is a function mapping each task node to its set of variables. For brevity we write $n_t.A$ for $\delta_A(n_t)$. L is a finite set of labels, and $l : F \mapsto 2^L \setminus \emptyset$ is a function that assigns a label or set of labels to each control flow. Moreover, we write F^* to denote the transitive closure of the control flow relation F.*

Definition 2 (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: $\bullet$CancelLate=$\{s_2\}$ and CancelInvoice$'\bullet$=$\{s'_4\}$ in Figure 2.

The execution semantics of a labelled Petri Net is the same as that of a Petri Net where a task produces a token to each of its immediate post states and consumes a token from each of its immediate pre states. In this paper we assume that all models are labelled Petri nets satisfying the properties: *safe*, *activity-reduced*, *deadlock free*, *label preservation*, *unique label*, *common label distribution*. We refer the interested readers to (Schrefl & Stumptner 2002) for a complete description of these properties.

3.2 Generating Views

Mafazi et al. (2012) proposed a knowledge based framework for process views based on user-specified constraints and applied the approach to a real-world business process repository from the *product and system engineering* domain containing software development process models with up to 260 tasks in a single process model. In some use cases, a view could be generated that reduces the complexity by up to 90%. Since views were derived using the consistency rules proposed by Schrefl & Stumptner (2002), views can be seen as abstract supertypes of the more detailed reference model, which can be regarded as the intersection type of its parent views.

While there are several interesting techniques for identifying corresponding process model elements, such as different metrics and measures based on linguistic similarity of labels (Rosa et al. 2010, Pedersen et al. 2004), in our framework the correspondences between process model elements are captured while creating the views. We capture the relationship between elements in the reference process and a process view by a view-specific total mapping function h such that:

1. h maps each state, task, attributes and control flow relation of the reference process model to their corresponding entities in the view;
2. 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.

More formally, the correspondence between elements in the reference model and in a view can be captured as follows:

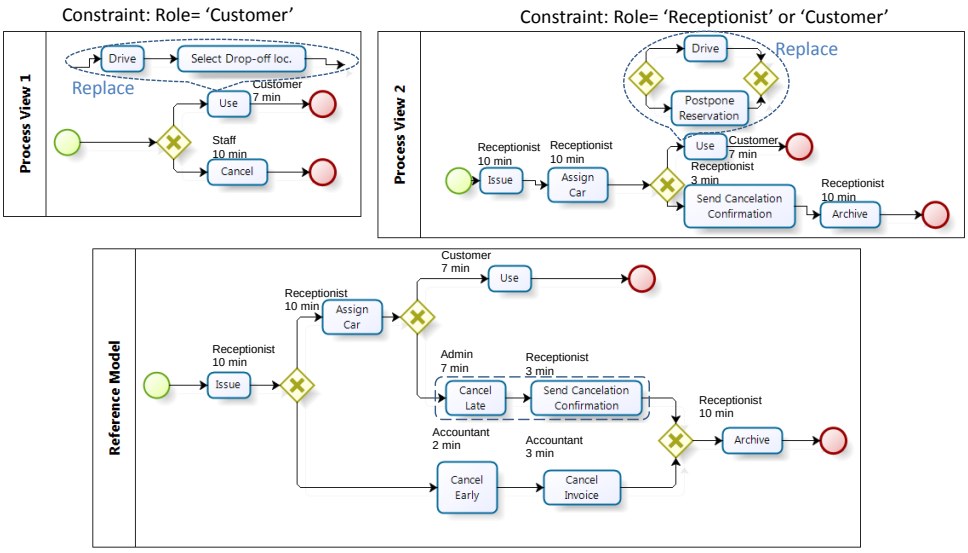

Figure 1: A conflicting situation example (BPMN)

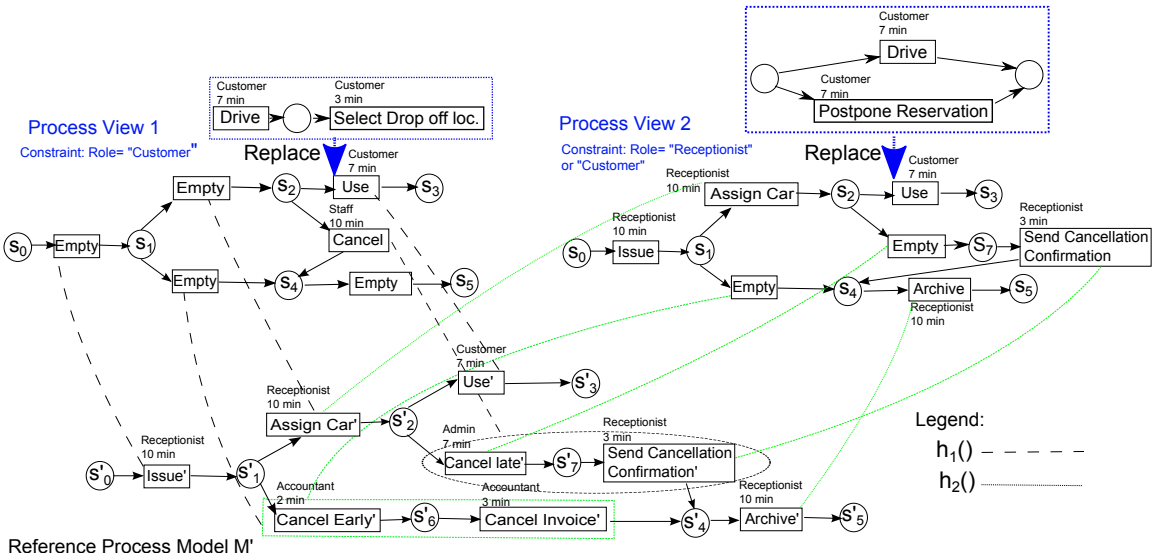

Figure 2: A conflicting situation example (Petri net)

Definition 3 (Process View) *Let M and M' be labelled Petri net models. The components of M and M' are understood as per Definition 1. Model M is a view of the underlying reference process model M' if a mapping function $h : N' \cup F' \cup L' \mapsto N \cup F \cup L$, satisfying the following properties:*

1. $n \in N' \cup F' \Rightarrow h(n) \in N \cup F$,
2. $l' \in L' \Rightarrow h(l') \in L$,
3. $a \in A' \Rightarrow h(a) \in A$,
4. $a' \in N'_s, a \in N_s, {\bullet}a = \emptyset, {\bullet}a' = \emptyset \Rightarrow h(a') = a$

For each view V_i there exists a separate mapping function h_i mapping the reference process elements to the corresponding view elements. The correspondences between the models are captured in a set of mapping functions $H = \{h_1, h_2, ..., h_n\}$.

Example: Figure 2 shows two process views that have been generated from the reference process M' where, for example task *Cancel Late'* in the reference model has been mapped to state *Empty* in View 2. The relationship between the reference process and this view is specified as $h_2(CancelLate') = Empty$. Likewise the relations betweens the nodes of reference model and View 1 is captured by a function h_1. In the given example $h_1(CancelLate') = h_1(s'_7) = h_1(SendCancellationConfirmation') = Cancel$.

Since each element in the reference model has a corresponding element in each view, correspondences between view elements can be established indirectly via their mapping to the common reference model.

Example: In the given example in Figure 2 task *Cancel Late'* from the reference model is mapped to the task *Cancel* in View 1, that is $h_1(CancelLate') = Cancel$, and to task *Empty* in View 2. Moreover, task *Send Cancellation Confirmation'* from the reference model is mapped to the tasks *Cancel* and *Send Cancellation Confirmation* in View 1 and 2 respectively. Hence, it can be concluded that task *Cancel* from process View 1 corresponds to the tasks *Empty* and *Send Cancellation Confirmation* from process View 2.

3.3 Changing Views

Our framework supports the following operations in each model: insert, update, delete of model entities including tasks, places, arcs, task's attributes and labels. The changes can be applied to a process fragment which can be a single node or a group of nodes with their associated control flows. Among all the possible combinations of the changes, applying update-update, update-delete, and insert-insert of different process fragments in corresponding regions of different models may cause conflict.

Example: As illustrated in Figure 4, task *Use* has been replaced by the fragment containing the two consecutive tasks *Drive* and *Select Drop-off loc* in View 1, and by another fragment containing two alternative tasks *Drive* and *Postpone Reservation* in Process View 2. For replacing task *Use* by the fragment containing the two tasks *Drive* and *Select Drop-off loc* in View 1 in Figure 2, firstly the operator *RemoveProcessFragment(Use)* needs to be applied. Subsequently, *InsertProcessFragment(Drive,* s_8*, Select Drop-off Loc;* s_2*;* s_3*)* operator needs to be applied.

After applying the changes we verify that the resulting model is well-formed, safe, activity-reduced, deadlock-free, and satisfies the label properties. These properties can be checked efficiently using techniques such as SESE fragmentation (Fahland et al. 2009). If properties are violated then the process model must be revised before we continue with change detection and resolution. In our example, no issues arise, since the models remain safe, activity-reduced, and deadlock-free after replacing task *Use* in Process View 1 and 2.

3.4 On-the-Fly Change Propagation

Mafazi et al. (2013) proposed a design-based approach where consistency criteria are checked and *non-conflicting* changes are propagated on-the-fly from a process view to its reference model and related process views. Moreover, strategies for dealing with data flow inconsistencies after change propagation were presented. The work presented here complements the previous framework by focusing on conflict detection and resolution during change propagation. In the following we describe our framework in which conflicting changes are detected, possible resolutions are inferred, and resulting updates are propagated among models.

3.5 Consistency Criteria

Branco et al. (2013) define consistency management as

> *a set of methods and tools for establishing and maintaining consistency among software artifacts, such as models, code, documentation, and test cases which are usually created and maintained by different stakeholders*.

To ensure the consistency between the process models in our framework, we use the consistency criteria defined by Schrefl & Stumptner (2002).

Observation consistent specialisation allows us to identify model elements that are refined in one view and unrefined in another view. Moreover, elements that are defined in one view as view-specific extensions, for example view specific alternatives, that are not represented in other views can be present. Hence, the observation consistent specialisation framework provides *refinement* and *extension* rules. Informally, observation consistent specialisation guarantees that if features refined or added in the detailed model (denoted as the sub type) are ignored, any instance of the sub type can be observed identically from the point of view of the more abstract model (denoted as the super type). For precise formal definitions of extension and refinement rules we refer the interested readers to (Schrefl & Stumptner 2002, Mafazi et al. 2013).

3.6 Conflicts

According to Gerth et al. (2013) two change operations are conflicting if application of one operation renders the other one inapplicable. We follow the same definition in our approach to detect conflicts where the changes applied in different models conflict indirectly via their corresponding fragments in other views.

If a conflict arises, it is sufficient to notify the user about the conflict and to resolve the conflict by either combining the two conflicting changes or applying one of them. The rules of behaviour consistent extension and specialisation can guide the selection of suitable candidate operations. As a result, the effect of one of the conflicting changes is captured and reflected in the other relevant process models. On the other hand, if conflicting changes are combined, for example by inserting both change fragments as alternative branches, the effect of both of the changes can be reflected in the relevant models. In the following we discuss different types of conflicts and their identification and resolution strategies.

4 Conflict Detection and Resolution

Mens (2002) introduces five types of conflicts in the field of software merging: *textual, syntactic, semantic, structural* and *behavioural* conflicts. In the context of process models, we deal with structural conflicts including conflicts in the behaviour of the model before and after change. To deal with conflicting changes, Gerth et al. (2013) suggests three conflict resolution strategies: applying one, both, and none of the operators. We base our resolution mechanisms on the first and second strategies.

4.1 Type of Conflicts

Koegel et al. (2010) define two types of conflicts based on level of severity of the conflicting changes as *hard* and *soft* conflicts. For hard conflicts, the effect of only one of the changes can be reflected to the remaining model. For soft conflicts, both can be incorporated into the model. In our framework, the conflicting changes that can be combined, where one change contains the other one, or where the changes are identical in terms of structure and tasks are considered as soft conflicts. Others are considered hard conflicts, where only one change can be applied or conflicts must be resolved cooperatively.

4.2 Conflict Detection

Conflicting changes are identified based on their affected fragments in the underlying reference model. Affected regions in the reference model can be identified based on the correspondence relationship established by the mapping function h. Corresponding regions between views can be inferred based on their mappings to the shared reference model. In any case, conflicts arise if two changes affect corresponding regions in mutually incompatible ways.

Table 1 indicates different conflicting situations resulting from applying conflicting changes in two corresponding regions of two process models as well as the resolution strategy for dealing with each situation. The table outlines the type of conflict for given overlapping changes and states appropriate resolution strategies that may be applied in each situation.

Insert-Insert Conflicts: Insertion conflicts arise if different model elements are added in corresponding regions in different models. For example there would be a hard

conflict if one stakeholder inserts a parallel or a sequence fragment while another stakeholder inserts an alternative fragment such that the inserted fragments have identical tasks.

Update-Update Conflicts: Another type of conflict that can occur relates to updating the same attribute of two corresponding tasks in the two models. There would be a soft conflict if one change in a model can satisfy the constraints attached to both of the changed model. A hard conflict can also occur if a change violates the constraint attached to their models. For example, an update-update conflict arises if the owner of View 1 in 5 updates the role of task *Postpone Reservation* from *Customer* to *Admin*, and the owner of View 2 changes the execution role for the task *Postpone Reservation* from *Customer* to *Manager*.

Update-Delete Conflicts: Updating one task while deleting a corresponding task in another model causes this conflict. There would be a hard conflict if the update on the attribute of a task in a the view can satisfy the constraint attached to that view while its corresponding task is removed from another model. A soft conflict situation can occur if the update on the attribute of the view violate the constraint attached to that view. For example, assume that the owner of View 1 in Figure 5 removes task *Select Drop-off loc.* from View 1 while the owner of View 2 updates the attribute *role* of this task in their view to *Receptionist*.

4.3 Conflict Resolution

This section gives an overview on resolving operation-based conflicts between changed views.

Insert-Insert Conflict resolution: Table 1 suggests four of conflict resolution strategies:

1. Combine: applying the changes individually in different process models result in soft conflicts, and the result of both of the changes can be captured by combining the conflicting fragments. For example, if the conflicting fragments are two alternative fragments, there is a soft conflict that can be solved by combining the two fragments. Figure 3b and 3c show a soft conflict situation. In Figure 3b a single task is inserted whereas two tasks are inserted in the other view. Likewise, in Figure 3c a sequence of two task conflicts with the insertion a parallel fragment containing three tasks. This conflict can be resolved by inserting both fragments as alternative branches in both models.

2. Apply One: In case of soft conflicts when the changes on the models are identical, only one of the changes is considered and propagated. In case where one of the changes satisfies the constraints of both views, a soft conflict arises that can be resolved by propagating the satisfying change only.

3. Apply Specific changes: In case of a soft conflict, where one change contains the other one, the more comprehensive change is propagated. For example assume that one change inserts a parallel fragment while another one inserts a task in a parallel fragment. In this case the parallel fragment is propagated as it subsumes the other change.

4. User Decides: In case of a hard conflict, a change that satisfies the constraints of both views must to be selected and propagated to the relevant models. If both of the changes violate the constraints of both views, the changed views must be abstracted individually such that they satisfy their attached constraint. If after further abstraction the constraints are satisfied but the change in the views are conflicting then the owners of the views decide which change to propagate.

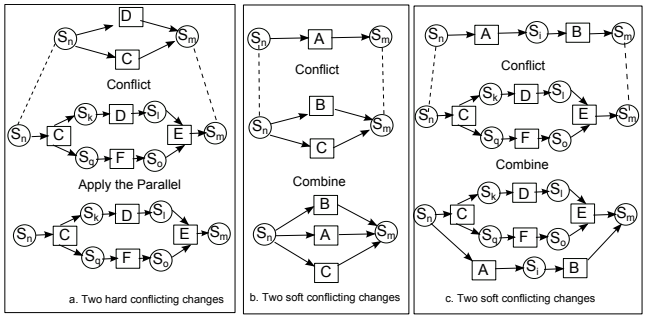

Figure 3: Conflict Resolutions

Soft conflicts arising from updating attributes of corresponding tasks in two models are resolved by propagating a change that satisfies the constraint of all affected models, possibly further generalising the views if necessary. For hard conflicts, the authors of the conflicting changes are notified of the conflict, and if they choose to proceed, the view must be abstracted further to resolve the inconsistency. **Example:** States s_2 in View 1 and 2 in Figure 2 are corresponding, and s_3 in View 2 and task *Use* in View 3 are corresponding. Given the changes applied to each view (*RemoveProcessFragment(Use)* and *InsertProcessFragment(Drive, s_8, Select Drop-off Loc.; s_2; s_3)* in View 1, and *RemoveProcessFragment(Use)* and *InsertProcessFragment(Drive, s_8, Select Drop-off Loc.; s_2; s_3)* in View 2), a soft conflict situation arises according to Table 1. This situation can be resolved by combining the conflicting fragments.

Update-Update Conflict Resolution:
In case of a soft conflict, an update that satisfies the constraints attached to each of the changed models is propagated to the relevant models. In case of a hard conflict where one or both of the changes violate the constraints, the owners of the views performing the changes are notified about the constraint violation induced by the change. If the owners intend to proceed with the change, the views that violate the constraints is abstracted further such that the constraints are satisfied. Subsequently, the structural changes are propagated to the relevant models.

Example: Continuing the example in Figure 5, since the change in the attributes of both of the views violate the constraint attached to the views, the owners of the views are notified about the violation of the constraint and if they intend to proceed with the changes the views are abstracted such that task *Postpone Reservation* is removed from both views.

Update-Delete Conflict Resolution: In case of a soft conflict, the changes removing a task are propagated. In case of hard conflict where both of the conflicting changes can satisfy the constraints attached to their individual view, if updating the attributes in one model can satisfy both of the constraints attached to the two changed views, the change that performs the *delete* needs to be discarded. In case the update of the attribute does satisfy the constraints attached to the two views, the change that applies *delete* is propagated to the relevant models.

Example: Removing task *Select Drop-off loc* in Figure 5 does not violate the constraint of View 1, whereas changing the attribute of this task in View 2 to *Admin* violates the constraint of View 2. Hence the deletion applied in View 1 is applied to the models.

Since our information about the intent of the owner of a view for applying a certain change is not captured in the change operators, the suggested resolution strategies for hard conflicts may not be acceptable and the stakeholders must decide which change to discard and which change to propagate.

We use a divide and conquer strategy for resolving

Changes in Process 1	**Alternative**		**Parallel**		**Sequence**		**A Task**	
Changes in Process 2	Ident. Tasks	Diff. Tasks	Ident. Tasks	Diff. Tasks	Ident. Tasks	Diff. Tasks	Ident. Tasks	Diff. Tasks
Alternative	Combine (S)	Combine (S)						
Parallel	User decides (H)	Combine (S)	Apply One (S)	Combine (S)				
Sequence	User decides (H)	Combine (S)	Combine (S)	Combine (S)	Apply One (S)	Combine (S)		
A Task	Apply Alternative (S)	Combine (S)	Apply Parallel (S)	Combine (S)	Apply Sequence (S)	Combine (S)	Apply One (S)	Combine (S)

Legend: (S) *soft conflict*, (H) *hard conflict*, Ident. *At least one identical tasks exist in the two fragments*, Diff. *Tasks in fragments are different*

Table 1: Conflict Resolution Strategies result of inserting conflicting fragments

conflicts caused by more than two stakeholders. For this purpose, firstly the conflicts are resolved between two models fragment by fragment. Once the two models are consistent, the conflicts between the third model and one of the consistent models are resolved. This process continues until all the models in conflict become consistent. In this way problem of resolving multiple conflicts between multiple models is reduced to resolving one conflicts between two views in each step. We employ behaviour consistent generalisation to ensure that the resulting views converge to mutually consistent processes and that the change propagation eventually terminates.

4.4 Consistency Restoration

Figure 4 continues the example given in Figure 2. After applying the changes in View 1 and 2, to restore consistency between the views, firstly the conflicts must be identified, resolved and then the changes need to be propagated to the reference model. The changed fragments in the views can be determined by the $h()$ function. If the applied changes are on the corresponding regions of the views based on the changes Table 1 suggest the type of conflict as well as the resolution strategy. Once the conflicts in the views resolved and propagated to the reference model, the two views need to be updated based on the applied conflict resolution strategy.

Example: As illustrated in the reference model from Figure 4 based on the correspondences, task *Use'* from the reference model is replaced by the combined fragment containing both changes made on View 1 and 2. Subsequently, as shown in Figure 5, the combined fragment is propagated from the reference model to the views. For this the *h* function is applied to map each affected element in the reference model to a corresponding update in the view.

4.5 Attribute Change Propagation

One may be interested in changing the label of a task or an attribute's value in a process view or in the reference model. Once a task's name changes, the name of all its corresponding tasks in the other process models must be changed accordingly in order to restore consistency. If the corresponding tasks in the two models are at the same level of details, then the same name can be used to update the name of the other corresponding node. Otherwise appropriate names must be selected. For this purpose we make use of domain ontologies such as meronymy relations (Smirnov et al. 2010) available in *MIT Process Handbook* (Malone et al. 2003). Likewise, domain knowledge including public resources, such as WORDNET[1], and

[1] http://www.wordnet-online.com

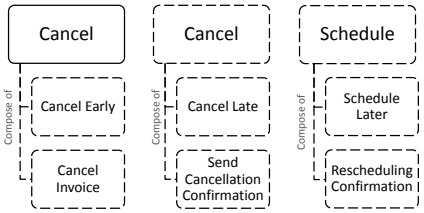

Figure 6: Task meronymy relation

clustering methods (Günther & Aalst 2007) can be used to assign appropriate attribute values to the corresponding tasks at different levels of detail. In this context, using domain knowledge and ontologies has the advantages that appropriate labels and attribute values can be assigned to the corresponding tasks which are at different levels of detail. For example, if the name of one of the two tasks in a refinement relation changes, the appropriate name can be assigned to the other task by using the domain knowledge. Similarly, ontological knowledge can be used to guide the view abstraction mechanism. Let us consider the meronymy relation for task *Cancel* represented in Figure 6. The relation shows that this task can be decomposed in two ways. If a stakeholder changes the name of the task *Cancel* to *Schedule* in View 1, the name of the corresponding tasks can be updated accordingly using the meronymy relations; that is, the label *Cancel Late* can be replaced by *Schedule Later* and *Send Cancellation Confirmation* by *Reschedule Confirmation*.

For other attributes, we employ a partial refinement relation that associates values with their more abstract parents in a hierarchy. We rely on a partial order defined over the set of domain values for the respective attributes and task labels for decomposition or generalisation of attribute's values including task's labels. Definition 1 provides a generic framework for their introduction, where suitable decomposition or generalisation relations can be defined by a partial order for each value domain associated with an attribute. Least upper bound and greatest lower bound on the attribute lattices reflect abstraction and decomposition of attributes. Figure 7 depicts an abstraction semi-lattice for attribute *Role*. The *is-a* relation connecting general roles with their specialisations forms a partial order over the set of role names.

5 Related Work

In the following we discuss related work directly addressing conflict resolution strategies in change propagation scenarios as well as approaches related to co-evolution

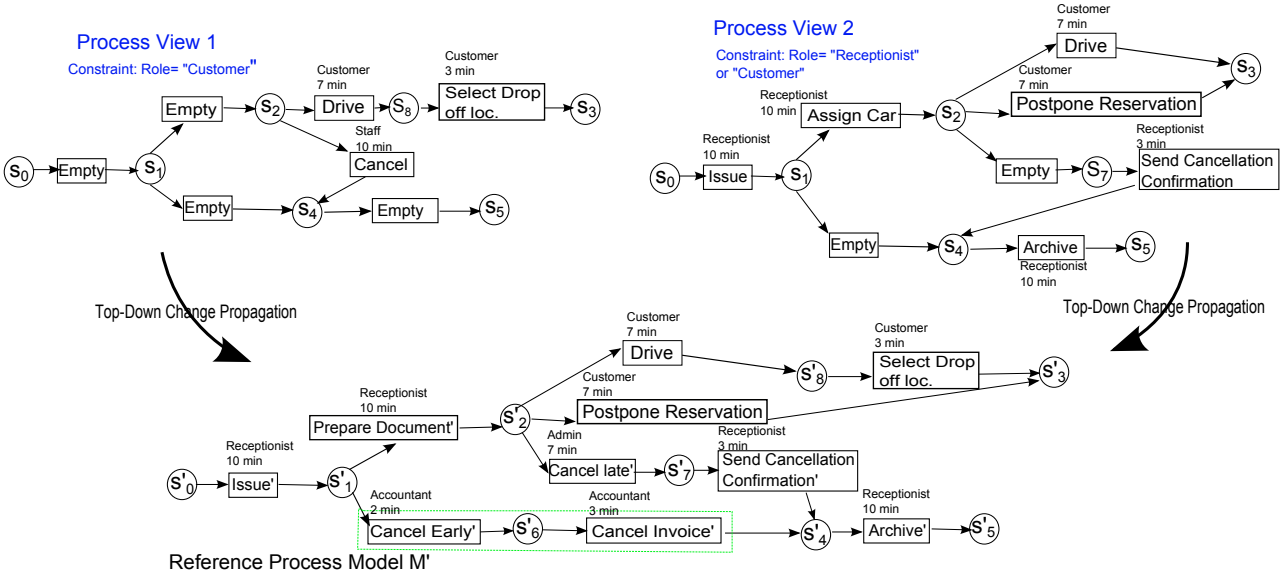

Figure 4: Propagating the changes from the views to the reference model

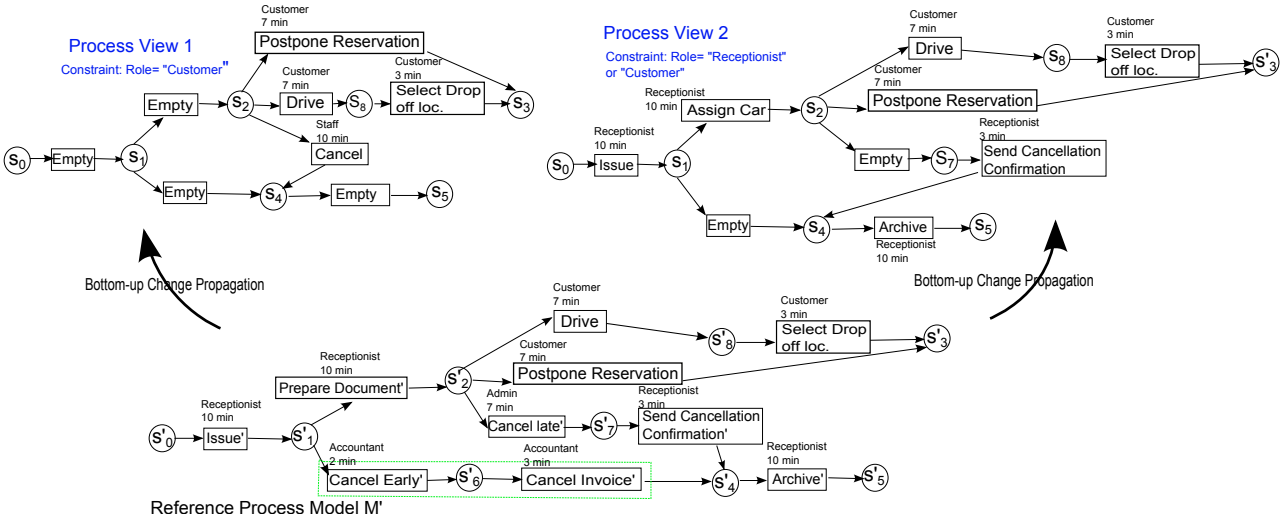

Figure 5: Propagating the changes from the reference model to the views

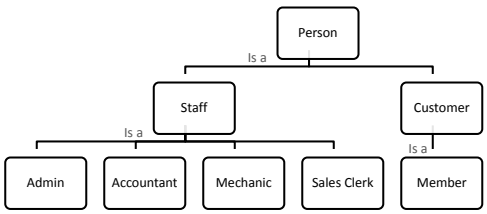

Figure 7: Role hierarchy

of process models. The mentioned approaches complement our approach in some aspects. We first discuss conflict resolution in change propagation scenarios, and subsequently investigate conflict resolution by merging process models and consistency criteria used in the literature. Table 2 summarises our findings. The table rows specify the papers and indicate their support towards the following criteria:

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

Support for attribute: This column indicates whether the related work considers changes in attributes and preserves their consistency. Since several real world process models have attributes attached to their tasks, and these attributes may be changed, it is important to have strategies to restore the attribute consistency. The possible values include $(+)$ supported, $(-)$ not supported, (N/A) the criterion is not in the scope of the paper, and $(n.d.)$ 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 have 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. Considering the large size of the process models, having different process views can facilitate not only the model comprehension but also aids business analysts.

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

Table 2: Comparison of Process Model Conflict Resolution in Change Propagation Approaches

Related Work	*Goal*	*Attribute Supp.*	*Bidirect. Propag.*	*Process View*	*Consist. Pres.*	*Conflict Res.*	*Behavioural Model Supp.*
Taentzer et al. (2012)	Conflict detection/ resolution	-	N/A	N/A	+	+	-
Brosch et al. (2010)	Conflict detection/ resolution	n.d.	N/A	N/A	n.d.	+	-
Mendling & Simon (2006)	View consolidation by merge	-	N/A	+	n.d.	-	+
Blanc et al. (2008)	Model inconsistency	n.d.	N/A	-	+	N/A	+
Gerth (2013)	Change management	-	-	-	n.d.	+	+
Küster et al. (2009)	Conflict resolution	-	+	N/A	+	+	+
Weidlich et al. (2012)	Change propagation	n.d.	+	N/A	+	N/A	+
Cicchetti et al. (2008)	Conflict resolution	+	N/A	n.d.	+	+	-

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

Conflict resolution: This column indicates whether the proposed conflict resolution strategy is structured-based, based on execution traces, or on change logs.

Support for behavioural models: This column discusses whether the related work supports behavioural model, such as BPMN and Petri nets, or if it only supports the structural models such as UML class diagrams. Considering the behaviour of the process model while dealing with conflicts helps identifying potential false-positive conflicts, that is semantically equivalent changes which are conflicting (Gerth et al. 2013).

5.1 Conflict Resolution in Change Propagation

It is important to identify the conflicts and dependencies of change operators in a process model before applying changes. That is if two change operators are conflicting then only one of them should be applied, or if two changes are dependent then one change needs the application of the second one. Küster et al. (2009) investigate dependencies and conflicts of changes in process models. They encode critical pairs, used for detecting dependent and conflicting transformations, as conditions for the change.

Gerth et al. (2013) represent a method for identifying *semantic* and *syntactic* conflicts in model versioning scenarios. For this purpose they use process model terms to capture the execution order of the elements in a process model and three-way merge. To identify conflicts they assume that all the changes are provided in the form of change operations, including *insert*, *delete* and *move*, stored in change logs.

Küster et al. (2009) use change logs to identify conflicting change operations. They suggest three options to resolve a conflicting situation: discarding one of the complete subsequence, combining the conflicting operations and modifying one or both of the conflicting operations. An appropriate option must be identified and selected by the user.

In contrast to our approach, these works assume that all the changes are recorded in change logs while building a version from the original model. Recording and reasoning about change logs can be expensive in large process models especially if composite operators such as replacing or swapping process fragments are applied. Moreover, building process model terms to capture the execution order of the elements in a process model to establish correspondences between model elements can be expensive considering the large size of process models in the real world scenarios.

Dam et al. (2010) propose a semi-automated change propagation technique for UML class diagrams based on *Alloy facts*. The technique provides repair plans for resolving conflicts and inconsistencies. Although conflicting situations and resolutions are not directly discussed, such are included in their proposed repair plan. Hermann et al. (2012) propose a semi-automated conflict resolution approach in synchronising the concurrent model modifications based on rules expressed as *Triple Graph Grammars*.

Cicchetti et al. (2008) propose a domain specific language to control the conflicts from cooperative changes over the same process model elements. This semi-automated approach is based on a meta model representation which enables detection of semantic and syntactic conflicts. To represent the conflicting modifications, the changed models are merged and the appropriate conflict resolutions are recommended.

Altmanninger (2008) introduces the conceptual design of the *SMoVer* version control system. The system provides information about the models' execution semantics which facilitates better conflict identification than what purely syntactic and structural approaches can offer. The focus of the framework is on detection of conflicts; their resolution is left to the user.

Küster et al. (2012) propose an approach to synchronise process views at different abstraction levels through a shared process model. The shared process model is initialised by a single process view, as it is assumed that different process views evolved from a single model. If one of the views changes, it is checked into the shared process model to update other views. This approach synchronises the views but not the reference model, and the approach assumes that changes to different views do not happen concurrently.

Taentzer et al. (2010, 2012) define two syntax-based conflict notions, *operation and state-based* conflicts, in model versioning based on graph theory. Operation-based conflicts include conflicts such as adding a node in one version and removing it in another version. By giving the priority to the *add* this conflict is resolved. State-based conflicts occur where the final state of the graph after applying the changes contradicts consistency rules specified in the modelling notation. This categorisation has been elaborated by Taentzer et al. (2012) for *EMF-based* models, while the modelling features of the models such as controlling, multiplicity, and ordered features have been formalised by graph constraints. Conflict patterns are presented which illustrate the conflicting operations, such as *delete-use, delete-move, delete-update, update-update, move-move,* and *insert-insert*. For resolving any of these conflicts a strategy has been presented. However it is not clear how changes are set to a pre-defined change operation. This can cause difficulties in the identification of the potential operations especially where the investigated versions differ significantly.

Weidlich et al. (2012) use *behavioural profiles* for

identifying changed regions in a model and propagate the changes between models at different levels of abstraction. In this approach data flow and other task properties are not considered. Moreover, the approach depends on the existence of execution traces of a process model.

5.2 Conflict resolution in model merging

Brosch et al. (2010) introduce a model versioning system called *AMOR*. This tool provides an exact conflict report in model merging as well as semi-automatic executable resolution strategies. For this purpose the predefined resolution patterns based on the UML class diagram recommend a resolution for conflicts. In this framework consistency preservation is not discussed, and neither the execution semantic of the modelling notation nor its behaviour can be captured. In contrast, our approach is language specific and considers the execution semantics and behaviour of the process model.

Mendling & Simon (2006) propose a method for integration of model views expressed in conceptual modelling notations such as EPC. Firstly the semantic relations between views are derived based on the relations between the model elements, where two elements are either *equivalent* or *in sequence*. Secondly a merge operator creates the merged model by taking the semantic relationship and the EPC models as input. Lastly a set of restructuring rules are applied on the merged model to eliminate any unnecessary model elements. In contrast, we assume that the semantic relationship has been established earlier when the views are created. Moreover in our approach the observation consistency across the models is preserved in all the stages.

5.3 Consistency management in process modelling

Weidlich & Mendling (2012) discuss the notion of *view consistency*, which is concerned with achieving and preserving a certain degree of uniformity and consistency between views created by multiple stakeholders as well as a reference process model. In this context, models are consistent if they cover exactly the same part of a scenario and there are no behavioural contradictions between them. As the process models are changed frequently, having an appropriate consistency notion can help propagating changes from one model to another.

Dijkman, Quartel & van Sinderen (2008) address issues arising in information system design where several stakeholders collaborate to create a system. Their framework establishes and supports a common terminology for stakeholders and consistency management mechanisms between different views. The paper considers observation and weak invocation consistency, and deals with models that are either in refinement or overlap relations.

Blanc et al. (2008) represent an approach to detect (structural and methodological) state-based model inconsistencies expressed as logic formula. Structural consistency rules enforce the relationship between model elements of two models without considering how models have been constructed. Methodological consistency rules enforce an appropriate order of editing operations on a model. The consistency rules are based on domain-specific use cases and compare the previous state of the model with the state resulting from changes. Mens et al. (2006) address inconsistencies resulting from change propagation by introducing a framework for inconsistency management for UML class diagrams based on graph transformation rules. After applying changes, the model is evaluated against the graph transformation rules to determine conflicts, and resolution rules are applied to eliminate conflicts. The approach can identify syntactical conflicts such as contradicting changes on the meta models and violation of *OCL* constraints, although semantic conflicts of the models and behaviour inconsistency are left aside.

Perry et al. (2001) notes that configuration management systems are only able to identify conflicts resulting from applying changes in one version which overlaps with some other changes in another version. However, no criteria for view abstraction or change resolution have been proposed.

Table 2 illustrates a comparison of the discussed approaches. Although our framework can support all the criteria stated in this table, it is limited in terms of supporting certain change patterns discussed by Weber et al. (2007). In particular moving and swapping process fragments are not supported, as these patterns violate support the observation consistency rules underlying our framework.

6 Conclusion and Future Work

We considered the detection and resolution of conflicting changes in view management of business process models. We identified shortcomings of existing approaches in resolving conflicts such as relying on execution traces analysis, support for static models only rather than dynamic models, and dependence on change logs in a post-analysis phase. To overcome these shortcomings, we presented a framework for on-the-fly change propagation and extended it with selected conflict resolution strategies. We showed how behaviour consistency rules for checking the consistency can be used for conflict resolution and restoration of behavioural consistency among a set of related views. Our initial efforts have revealed that considering the purpose and intention of view can help to resolve soft- and some hard conflicts automatically. The validation of our approach by integrating the conflict resolution and change propagation with our tool for generating views from a reference model is subject to future work.

References

Altmanninger, K. (2008), Models in conflict-towards a semantically enhanced version control system for models, *in* 'Proc. of MODELS Workshop', LNCS 5002, Springer, pp. 293–304.

Barrett, S., Chalin, P. & Butler, G. (2008), Model merging falls short of software engineering needs, *in* 'Proc. of MoDSE Workshops'.

Blanc, X., Mounier, I., Mougenot, A. & Mens, T. (2008), Detecting model inconsistency through operation-based model construction, *in* 'Proc. of ICSE', pp. 511–520.

Branco, M. C., Xiong, Y., Czarnecki, K., Küster, J. & Volzer, H. (2013), 'A case study on consistency management of business and it process models in banking', *Software & Systems Modeling* pp. 1–28.

Brosch, P., Kappel, G., Seidl, M., Wieland, K., Wimmer, M., Kargl, H. & Langer, P. (2010), Adaptable model versioning in action, *in* 'Modellierung', LNI 161, GI, pp. 221–236.

Cicchetti, A., Ruscio, D. & Pierantonio, A. (2008), Managing model conflicts in distributed development, *in* 'Proc. of MDELS', LNCS 5301, Springer, pp. 311–325.

Dam, H. K., Le, L.-S. & Ghose, A. (2010), Supporting change propagation in the evolution of enterprise architectures, *in* 'Proc. of EDOC', IEEE Computer Society, pp. 24–33.

Dijkman, R. M., Dumas, M. & Ouyang, C. (2008), 'Semantics and analysis of business process models in BPMN', *Information & Software Technology* **50**(12), 1281–1294.

Dijkman, R. M., Quartel, D. A. & van Sinderen, M. J. (2008), 'Consistency in multi-viewpoint design of enterprise information systems', *Information and Software Technology* **50**, 737–752.

Fahland, D., Favre, C., Jobstmann, B., Koehler, J., Lohmann, N., Völzer, H. & Wolf, K. (2009), Instantaneous Soundness Checking of Industrial Business Process Models, *in* 'Proc. of BPM 2009', LNCS 5701, Springer, pp. 278–293.

Gerth, C. (2013), *Business Process Models. Change Management*, LNCS 7849, Springer.

Gerth, C., Küster, J., Luckey, M. & Engels, G. (2013), 'Detection and resolution of conflicting change operations in version management of process models', *Software & Systems Modeling* **12**(3), 517–535.

Günther, C. & Aalst, W. (2007), Fuzzy mining adaptive process simplification based on multi-perspective metrics, *in* 'Proc. of BPM', LNCS 4714, Springer, pp. 328–343.

Hermann, F., Ehrig, H., Ermel, C. & Orejas, F. (2012), Concurrent model synchronization with conflict resolution based on triple graph grammars, *in* 'Proc. of FASE', LNCS 7212, Springer, pp. 178–193.

Koegel, M., Herrmannsdoerfer, M., von Wesendonk, O. & Helming, J. (2010), Operation-based conflict detection, *in* 'Proc. of the 1st International Workshop on Model Comparison in Practice', IWMCP '10, ACM, pp. 21–30.

Küster, J., Gerth, C. & Engels, G. (2009), Dependent and conflicting change operations of process models, *in* 'Proc. of ECMDA-FA', LNCS 5562, Springer, pp. 158–173.

Küster, J., Völzer, H., Favre, C., Branco, M. C. & Czarnecki, K. (2012), Supporting different process views through a shared process model, Technical report, Generative Software Development Laboratory, University of Waterloo, Canada.

Mafazi, S., Grossmann, G., Mayer, W. & Stumptner, M. (2013), On-the-fly change propagation for the coevolution of business processes, *in* 'Proc. of: OTM Conferences', LNCS 8185, Springer, pp. 75–93.

Mafazi, S., Mayer, W., Grossmann, G. & Stumptner, M. (2012), A knowledge-based approach to the configuration of business process model abstractions, *in* 'Proc. of Knowledge-intensive Business Processes Workshop'.

Malone, T. W., Crowston, K. & Herman, G. A., eds (2003), *Organizing Business Knowledge: The MIT Process Handbook*, The MIT Press.

Mendling, J. & Simon, C. (2006), Business process design by view integration, *in* 'Proc. of BPM Workshops', LNCS 4103, Springer, pp. 55–64.

Mens, T. (2002), 'A state-of-the-art survey on software merging', *Software Engineering, IEEE* **28**(5), 449–462.

Mens, T., Straeten, R. & DHondt, M. (2006), Detecting and resolving model inconsistencies using transformation dependency analysis, *in* 'Proc. of MODELS', LNCS 4199, Springer, pp. 200–214.

Pedersen, T., Patwardhan, S. & Michelizzi, J. (2004), Wordnet::similarity: measuring the relatedness of concepts, *in* 'Demonstration Papers at HLT-NAACL', HLT-NAACL–Demonstrations '04, Association for Computational Linguistics, pp. 38–41.

Perry, D. E., Siy, H. P. & Votta, L. G. (2001), 'Parallel changes in large-scale software development: an observational case study', *ACM Trans. Softw. Eng. Methodol.* **10**(3), 308–337.

Rosa, M. L., Dumas, M., Uba, R. & Dijkman, R. M. (2010), Merging business process models, *in* 'Proc. of OTM', LNCS 6426, Springer, pp. 96–113.

Schrefl, M. & Stumptner, M. (2002), 'Behavior-consistent specialization of object life cycles', *ACM Trans. Softw. Eng. Methodol.* **11**(1), 92–148.

Smirnov, S., Dijkman, R., Mendling, J. & Weske, M. (2010), 'Meronymy-based aggregation of activities in business process models', *Conceptual Modeling–ER 2010* pp. 1–14.

Spanoudakis, G. & Zisman, A. (2001), *Inconsistency management in software engineering: survey and open research issues*, World Scientific Publishing.

Taentzer, G., Ermel, C., Langer, P. & Wimmer, M. (2010), Conflict detection for model versioning based on graph modifications, *in* 'Graph Transformations', LNCS 6372, Springer, pp. 171–186.

Taentzer, G., Ermel, C., Langer, P. & Wimmer, M. (2012), 'A fundamental approach to model versioning based on graph modifications: from theory to implementation', *Software & Systems Modeling* pp. 1–34.

Weber, B., Rinderle, S. & Reichert, M. (2007), Change patterns and change support features in process-aware information systems, *in* 'Advanced Information Systems Engineering', LNCS 4495, Springer, pp. 574–588.

Weidlich, M. & Mendling, J. (2012), 'Perceived consistency between process models', *Information Systems* **37**(2), 80–98.

Weidlich, M., Mendling, J. & Weske, M. (2012), 'Propagating changes between aligned process models', *Journal of Systems and Software* **85**(8), 1885–1898.