Handling concurrent changes in collaborative process model development: a change-pattern based approach

Selim Erol, Gustaf Neumann
Institute of Information Systems and New Media
Vienna University of Economics and Business
Vienna, Austria
serol@wu.ac.at, gneumann@wu.ac.at

***Abstract*— Business process modeling has gained increasing interest with the advent of business process management systems in organizational contexts. As business processes are subject to frequent change also respective models are complex and under continuous development with multiple modelers being involved.**

This paper presents an approach to integrate concurrent changes of multiple modelers equivalently into a resulting process model. The presented approach is based on the notion of change patterns which were used to capture the semantics of changes. These change patterns were identified by analyzing common change operations during model creation, modification, and refactoring. Together with the concept of conflict types change patterns are used as a conceptual means to implement the complexity of semi-automated merging of concurrent changes in modeling environments. Furthermore, it can be used to support modelers in manual merging of concurrent changes on a user-interface level. This approach enables smooth (incremental and iterative) development of models and provides a human-oriented approach to conceptualize change and conflict resolution in (concurrent) process model development.

***Keywords*-information systems design; business process design; business process modeling; collaborative modeling;**

I. Introduction

Business process modeling is an activity performed for various purposes. One purpose of process modeling in the context of organizational management is the documentation of actual ("as-is") processes for knowledge transfer among process stakeholders, the identification of weaknesses through process analysis and simulation and the documentation of desired future states ("to-be") of processes [1]. For the purpose of documentation and communication usually informal modeling techniques are sufficient. For the purpose of analysis and simulation semi-formal and formal modeling languages are used that enable the unambiguous formulation of a business process model and the quantification of input- and output variables, pre-/post-conditions and so forth. In recent years as well process models are increasingly used for the purpose of orchestrating business process related software services and applications. The latter purpose requires a valid specification of process models to enable their execution [2].

For the above mentioned purposes of process modeling several techniques have emerged in the last decades. For example, the concept of Event-driven Process Chains (EPC) [3] is targeted at capturing business processes from a business perspective and is considered a semi-formal approach [4] as it does not cover the semantics needed for an execution environment. Business Process Modeling Notation (BPMN) is a concept that extends the concept of EPCs with a well specified execution semantics [5] and has therefore been widely adopted in the business domain [6]. All these modeling techniques incorporate the concept of activities as the basic unit of modeling. The main goal of business process modeling is therefore to identify business activities, to arrange them in a certain order and provide contextual information like the resources needed, the output provided and the conditions under which an activity is performed.

In practice business process design is a highly collaborative activity that typically involves various stakeholders of a process. Although process models as one category of output are mostly created and maintained by a small group of dedicated experts, situations are likely to occur where changes to process models are intentionally or unintentionally performed in parallel. This is especially true within dynamic organizational environments where processes and respective models are frequently adapted to fit different contextual and situational needs or environments where a rather large community of modelers exists that is not necessarily aware of each others changes and access to a model collection. Such highly interactive modeling scenarios impose several challenges regarding the meaningful recognition of change intentions from a model revision and the integration of such change intentions to ensure the completeness and validity of the final model artifact.

In domains as software development such problematic concurrent situations are typically addressed through version control systems (VCS) that allow a developer to be aware of intermediate changes and to resolve potential conflicts. Most such concurrency handling approaches are based on text-based merging techniques [7] that assume a line of text as the smallest unit of comparison and merging [8]. Though, pure text based approaches do not hold for other

content types as for example tree-like or graph-like data structures. In research and practice, several approaches have been suggested that address problems resulting from the peculiarities of such data structures on a technical level. A comprehensive survey of graph-based model merging principles and techniques is given in [7], [9], [10]. For process model merging especially the work of [11]–[14] can be regarded as foundational. However, prior approaches in process model merging especially focus on syntactic conflicts and presume process models that are complete in the sense that they are syntactically correct. A shortcoming of current modeling environments is as well that they do not support modelers in understanding the rationale and context of conflicting changes but assume that a modeler is capable to merge changes on a technical level (e.g. data representation level like XML).

In the following sections we will address the above described problem of concurrent changes to process models in detail. First, we will frame the notion of concurrency on a conceptual and formal level (section II). Second we will outline how make use of catalog of change patterns to describe process model changes on a semantic level and derive adequate strategies for resolving/merging conflicting/concurrent changes. Finally, we will illustrate how we implemented this approach in a highly interactive wiki-based process modeling environment.

II. Concurrent scenarios in collaborative process modeling

Collaboration in process model development involves both creative and communicative activities. Where creative activities are targeted on the incremental development of a model artifact, communicative activities aim at the coordination of such activities. These types of activities are involved to varying extent in the main stages of process model development [15] where different scenarios of collaboration may occur. The first activity targets at elicitation of domain knowledge and developing an informal model. The second activity is concerned with formalization of the informal description and the third main activity is the validation of a model against the initial requirements. Suffice to say that these activities are performed in an iterative manner until a stage is reached thatwhere a model is perceived as valid to be used for some purpose (see above).

Within these activities one can classify collaboration scenarios along the dimensions of timely and spatial distribution of interactions [16]. Same-time collaboration is a scenario where spatially distributed modelers seek to work concurrently at a dedicated point in time on a model artifact. Second, different-time collaboration is a scenario where modelers share a model artifact over a rather long period of time, repeatedly but more or less unpredictably – in terms of the point in time – interact with the model artifact. The spatial distribution of modelers plays a role as in the case of co-location creative activities is mainly coordinated through the physical presence of group members whereas in the case of a distributed scenario other means of awareness and coordination need to be considered to avoid breakdowns and interruptions of model development.

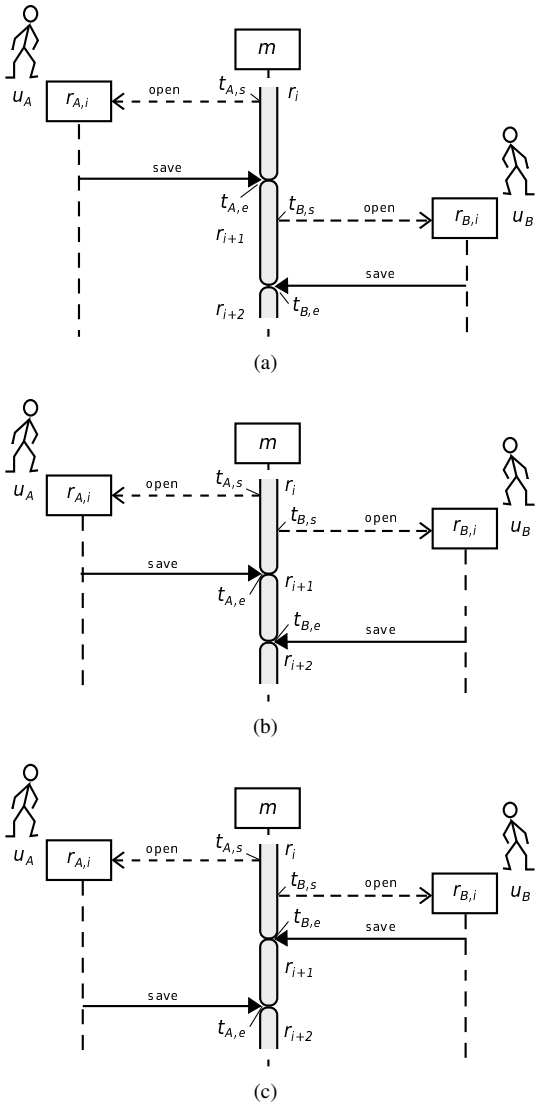


Figure 1. Scenarios

In the following we focus on distributed different time scenarios of collaboration where modelers' interactions with a model take place unpredictably over time and space. We outline three different scenarios that vary regarding the timely distributions of interactions and define a condition under which parallel development of a model leads to a breakdown in model development. Illustrations of several such scenarios are depicted in figure 1. The illustrations show a sequence diagram including users interacting with a content object. The vertical lines depict time lines of the involved users and the model under work. The latter time

line shows as well the life span of the revisions. Although the the illustrations show scenarios where only two users are involved it will hold also for an arbitrary number of users. Such scenarios can always be reduced to multiple two-user scenarios.

The first scenario as illustrated in figure 1(a) shows two model interactions that take place in sequence. The first modeler u_A opens (checks out) a model revision r_i from a repository at point in time $t_{A,s}$, changes the model, and submits (checks in) the changes to the repository at point in time $t_{A,e}$ which leads to model revision r_{i+1}. Subsequently, a second modeler u_B accesses the model revision r_{i+1}, opens it at point in time $t_{B,s}$, changes it and submits as well his new model revision at point in time $t_{B,e}$ which leads to a revision r_{i+2}. This scenario can be considered unproblematic as each model interaction is completed before a new model interaction starts.

In the second scenario as illustrated in figure 1(b) a revision r_i is checked out (opened for editing) by a user u_A at point in time $t_{A,s}$ from a repository (e.g. a centralized database). Shortly after the same revision r_i is checked out by user u_B at point in time $t_{B,s}$. Both users start to modify revision r_i in their local workspace (e.g. a browser memory). Thus, both users hold local revisions $r_{A,i}$ and $r_{B,i}$. Now (at time $t_{A,e} > t_{A,s}$) user u_A checks in (saves) his temporary revision to the central repository which will be assigned a revision number $i + 1 > i$. Next (at time $t_{B,e} > t_{A,e}$) user u_B tries to check in his local revision $r_{B,i}$ and will realize that an intermediate revision r_{i+1} exists. A situation occurs where the changes of user u_B would overwrite the changes of user u_A. This situation can be considered a potentially problematic situation as the time frames of two change operations targeted at the same object and performed by two different users have a temporal overlap which lead to situation where modeler u_A's changes will be unconsciously disregarded. The third scenario (fig. 1(c)) depicted shows one interaction completely nested with an other. It differs from scenario (b) only with regard to the order of check-out times but leads to the same potentially problematic situation as mentioned above. At least theoretically, a scenario can occur where either the start time or end times of two operations are equal $t_{,B} = t_{,A}$. The probability of such a scenario occurring increases with the number of involved users and their related activity. From a technical point of view the probability of such situations is also determined by the resolution/granularity of time-stamps to be used.

From the above a concurrent change situation can be defined as follows: Given two users u_A, u_B performing two independent change operations o_A, o_B on a shared object m. Both change operations are assigned a time frame characterized by a starting point $t_{.,s}$ and an end point $t_{.,e}$. A concurrent situation is defined as a situation where the end point $t_{.,e}$ of one change operation lies within the time frame of the other change operation. The concurrency condition for two concurrent change operations o_A, o_B therefore can be formalized as: $t_{A,s} \leq t_{B,e} \leq t_{A,e}$.

However, concurrent or overlapping interactions (check-out/check-in sequences) as defined above do not lead necessarily to a problem or conflict (thus we have called them so far potentially problematic). To be more accurate, concurrent changes to a process model constitute a conflict only if their "naive" sequential application leads to a model revision where changes stemming from one modeler are not incorporated anymore (discarded) or the final model revision reaches an invalid state with regard to the constraints of the modeling environment.

Conflict or not, concurrent interactions are usually undesirable and need to be recognized by an underlying software environment to allow for an appropriate handling of such situations be it automated, semi-automated or manually.

Recovering the semantics of model changes

Changes to process models usually follow a certain rationale. That is to say, a model is changed due to a certain business or technology driven requirement, e.g. a production process needs to be extended with an activity that deals with quality assessment of the final product. We argue that incorporating the rationale or semantics of a process model change into a collaborative modeling environment supports a modeler in conflict resolution. Moreover, we expect that having the rationale at hand cognitive effort to integrate (merge) changes is reduced and the overall maturity of a process model with regard to it's validity is increased.

To recover the rationale of a process model change from subsequent revisions of a process model it is necessary to have a meta-model that defines the semantics of the process modeling language used and on the other hand a meta-model that defines the semantics of change operations to a process model. For describing and explaining our approach we use a simplified meta-model which is derived from the BPMN [1] specification. The meta-model comprises the minimum set of semantic elements to describe the flow logic of a business process, namely activities gateways and events.

Based on this simple meta-model of a business process model we introduce a typology of change operations that are targeted at a process model as whole or it's constituting flow elements. We distinguish between three fundamental types of atomic change operations (or primitives). Namely, $add(\dot{m})$, and $modify(\dot{m}, args)$, and $delete(\dot{m})$ where $\dot{m}$ is an instance of a base element's concrete subtypes) and $args$ is a set of arguments that is passed to the modify operation. A modify operation's arguments may refer to an element's attribute name and value. A $delete$ operation is actually the inverse of a add operation. Thus, we can say that a process model change consists of a set of atomic change operations where at least one atomic change operation needs

[1] Business Process Model and Notation, see http://www.bpmn.org

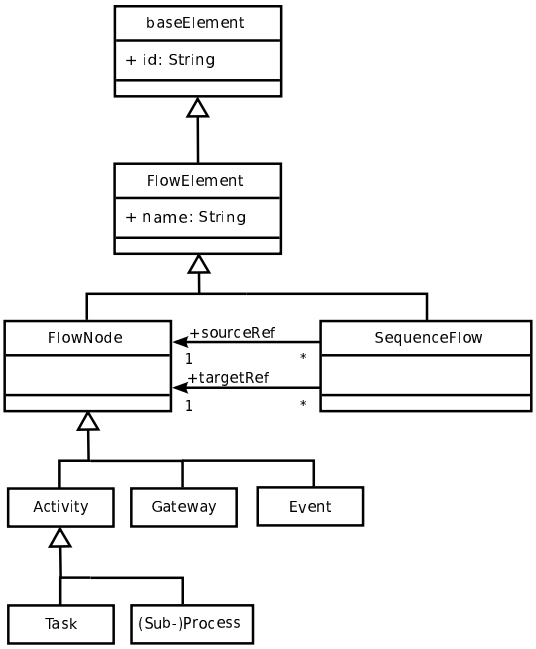
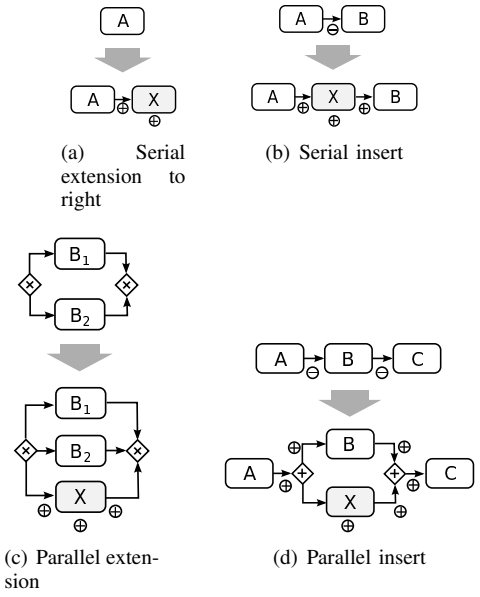


Figure 2. Simpel meta-model based on BPMN 2.0



(a) Serial extension to right

(b) Serial insert

(c) Parallel extension

(d) Parallel insert

Figure 3. Illustraions of selected change patterns

to exist. This set of change operations can be computed from subsequent revisions of a process model.

Having a set of change operations at hand one cannot simply reconstruct the semantics of a change. Therefore, based on the above defined types of atomic change operations we introduce so called change patterns as sets of related atomic change operation types (compound change operations) that reflect a particular change rationale. We have chosen the notion of pattern as it implies a certain form of abstraction from a concrete change and at the same time reflects recurring typical changes. For example, a parallelization of activities might involve two, three or an arbitrary number of activities. A change pattern description then only refers to parallelization as an act of logically arranging activities in parallel. As all possible change patterns can be hardly predefined on a theoretical basis we have established an open catalog that can be extended as knowledge about typical change patterns grows. Also the adaption and extension of change patterns according a particular context of use is possible. The catalog of change patterns raises the detection and handling of conflicting changes from a modeling language level (meta-model level) to a domain language level (ontological level) where changes are represented in a more expressive way.

As a starting point for the specification of such patterns we used the results from a prior study of process model adaption patterns [17] together with our own experience from experiments and a subsequent analysis of change patterns [18]. In contrast to [17] we also included change patterns that refer to changes that do not necessarily lead to syntactical correct process models. This is grounded in the insight that during process model development also incomplete revisions of models may be submitted to a repository. In figure 3 several examples of change patterns are illustrated. The atomic change operations involved are symbolized by a $(+)$ for $add(\dot{m})$ and $(+)$ for $delete(\dot{m})$ operations.

Each of these change patterns is described in a structured way. We used an approach similar to [19] to specify patterns by means of a scheme which requires a unique identifier for each pattern, a name that reflects the rationale of a change pattern in short, a description that reflects the rationale in detail, an illustration, the set of atomic change operations involved, and a category that allows for a "on-the-fly" classification of change patterns. In table I an exemplary change pattern for a *serial insert* is described. Accordingly, the *serial insert* pattern can be characterized by a sequence of atomic change operations $\langle deleteSeqFlow(\overrightarrow{AB}), addTask(X), addSeqFlow(\overrightarrow{AX}), addSeqFlow(\overrightarrow{XB})\rangle$. In fact X is a placeholder for a sequence of arbitrary activities. In fact each change pattern has an inverse function that itself is regarded a change pattern.

Given a set of unordered change operations (e.g. from a model revision comparison) recognition of a change pattern requires a clear definition of the semantics of change patterns and an algorithm that is able to detect such patterns. Moreover, recognition of change patterns requires that each change pattern can be distinguished from others by a uniqueness criterion. In the above example, a serial insert would be recognizable by a sequence of change operations like $\langle deleteSeqFlow(\overrightarrow{AB}),\ addTask(X),\ addSeqFlow(\overrightarrow{AX}),\ addSeqFlow(\overrightarrow{XB})\rangle$ and the fact that these operations refer to a special kind of target object within the original revision

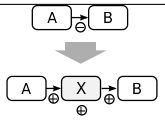
ID	P.SerIns.01
Name	Serial Insert
Description	Between two subsequent activities an arbitrary number of additional activities is inserted to form again a sequence of activities.
Category	Extension patterns
Signature	$\langle deleteSeqFlow(\overrightarrow{AB}), addTask(X), addSeqFlow(\overrightarrow{AX}), addSeqFlow(\overrightarrow{XB})\rangle$
Target objects	$TaskA$, $TaskB$
Illustration	
Example	$\langle deleteSeqFlow(\overrightarrow{AB}), addTask(C), addTask(D), addSeqFlow(\overrightarrow{AC}), addSeqFlow(\overrightarrow{CD}), addSeqFlow(\overrightarrow{DB})\rangle$

Table I
EXEMPLARY SPECIFICATION OF A CHANGE PATTERN

r_i – in this case a pair of `Task` elements. We call this specification of a minimum set of involved change operation types in combination with a specification of the target object types a *signature* that we use to identify instances of change patterns.

III. HANDLING CONCURRENT CHANGES BASED ON CHANGE PATTERNS

Identification of change patterns and their instantiations is the prerequisite for a proper handling of concurrent changes - that is avoiding the violation of constraints. As mentioned above basically two categories of constraints exist. Modeling process-related constraints and modeling environment related constraints. By process-related we mean those constraints that are originate from rules and conventions defined for the purpose of guiding the modeling process. For example, a rule can be defined that prohibits a naive (unreflected) sequential application of concurrent changes to ensure an incremental development of a process model. By environment-related we mean those constraints that are related to either the modeling language or the modeling tool. For example, a syntactical rule may exist that prohibits an `Activity` element to have more than one outgoing sequence flow, or a `SequenceFlow` requires an activity element to be existent to which it refers to. Another type of modeling environment-related constraint may be defined that does not allow for overlapping shapes in a process diagram.

Detection of constraint violations due to concurrent changes requires that these are known and formulated in advance. For example, the detection of a naming conflict of a model element presumes that a model element may not have two names or labels. The formulation of constraints in advance is not trivial as constraints often exist implicitly and are not explicitly described. For our approach we have derived syntactic and attribute constraints from the BPMN meta-model and have established as well some aesthetic constraints (which will not further discussed here). Regarding syntactic constraints we have only included a limited set of rules to allow as well for incomplete models. For example, a process model revision must not include an `end event` whereas a `SequenceFlow` must always have a reference to a source element (`SourceRef`) and a target element (`TargetRef`).

Our approach in handling concurrent changes to a process model is mainly based on the assumption that in a concurrent scenario both users are unaware of each other's changes and that these changes to a common antecedent revision have to be treated equivalently to enable a smooth evolution of the model artifact. We call this approach "in-favor-of-none" as it does not prefer one modeler's changes over the other's [18]. This implies that individual change operations of both users are checked against each other and are evaluated whether and how they can be integrated. Depending on the arguments of the two concurrent changes different situations may occur:

- **Identical** concurrent changes have an identical impact on the model and therefore one of the two edit operations can be ignored.
- **Inclusive** concurrent changes include each other. Therefore both changes can be replaced by the one that includes the other. E.g. a label suggested by one user may include the label string of another user. Or a sequence of activities suggested by one modeler might contain a sequence suggested by the other modeler.
- **Exclusive** concurrent changes are mutually exclusive. That is either edit operation from user A or edit operation form user B can be applied only. Edit operations are not identical and inclusive. They cannot sequentially be applied as this would disregard/overwrite one user's input or lead to an invalid state of the model or model element. E.g. a value suggestion for an integer value. In this case a resolution mechanism has to be applied to decide among the two edit operations or to compute an alternative.
- **Independent** concurrent changes do not interfere with each other. Sequential application of both edit operations would not lead to a conflict. As a consequence both edit operations can be applied without violating a constraint or overwriting one edit operations.

To identify and classify combinations of change patterns according to the above categories we make use of a merge matrix [20]–[22]. The merge matrix consists of columns representing change patterns of one modeler and rows representing change patterns from the other modeler. The items of the matrix represent all possible combinations of concurrent change patterns where for each combination conditions can be defined under which a combination is assigned to a particular category and which strategy needs to be applied to handle these concurrent changes to comply with existing constraints. For example, identical and inclusive conflicts

will be resolved by simply skipping or ignoring one of the two concurrent changes. Exclusive changes (true conflicts) need to be delegated to the user or a decision heuristic can be applied (see for example the recommendation based heuristic from [23]).

	$P.SerIns.01$	$P.SerExtR.01$	$ParIns.01$
$P.SerIns.01$	→C.1	...	...
$P.SerExtR.01$	...	...	...
$P.ParIns.01$	...	...	...

Table II
EXCERPT OF MERGE MATRIX USED FOR CRITICAL PAIRS ANALYSIS OF CHANGE PATTERNS. C.1 POINTS TO A CONFLICT TYPE THAT IS DESCRIBED IN DETAIL IN TABLE III

Table II shows an exemplary merge matrix that refers to selected change patterns and corresponding conflict types. A detailed specification of an exemplary conflict type along with a merge strategy is provided in table III. For this purpose a structure is used that describes the conditions under which a conflict can be detected, a merge strategy and an illustrative example. We used the concept of a merge matrix on a conceptual level to systematically identify the various types of conflicts and as a basis to formalize conflicts for a later implementation.

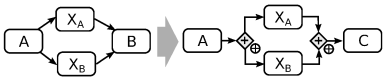
ID	C.1
Name	Concurrent Serial Inserts
Change patterns involved	$P.SerIns.01$, $P.SerIns.01$
Conditions for conflict	Serial inserts address identical subsequent flow elements of type activity A_1 and A_2
Sequential application would lead to (constraints violated)	two parallel sequences of X activities in between the two target activities A_1 and A_2. constraints violated: (1) a flow element of type activity must have not more than one outgoing sequence flow, (2) a flow element of type activity must not have more than one incoming sequence flows
Resolution strategy	compute similarity and in case (1) activity sequence $X_A = X_B \rightarrow$ decide for the first, (2) activity sequence $X_A \subset X_B \rightarrow$ decide for X_B and vice versa, (3) activity sequence $X_A \neq X_B \rightarrow$ insert both X_A and X_B enclosed in parallel gateways,
Example	The example shows case (3) of res. strategy where both serial inserts are extended with parallel gateways to comply with syntactic constraints. 

Table III
EXEMPLARY SPECIFICATION OF A CONFLICT TYPE

IV. IMPLEMENTATION IN A WIKI-BASED COLLABORATIVE PROCESS MODELING ENVIRONMENT

As a proof of concept we implemented the above outlined mechanism of concurrent change handling into a highly dynamic collaboration environment. Namely, we incorporated the concept into a special wiki application engine that was previously extended towards comprehensive process modeling support. *xoWiki* is an open-source wiki engine which was developed at the Institute of Information Systems and New Media. It is implemented in *xoTcl* – an object-oriented extension of Tcl [24], [25] – and rests upon a highly scalable community platform development framework [26]. XoWiki combines aspects of wikis (ease of page-creation) with typical features of a content management system (revisions, reusable content, multiple languages, page templates). For the purpose of process modeling we extended the user interface to support diagrammatic creation of process models and their syntactical validation. *xoWiki* also has a built-in workflow engine that allows for flexible enactment of form-based workflows.

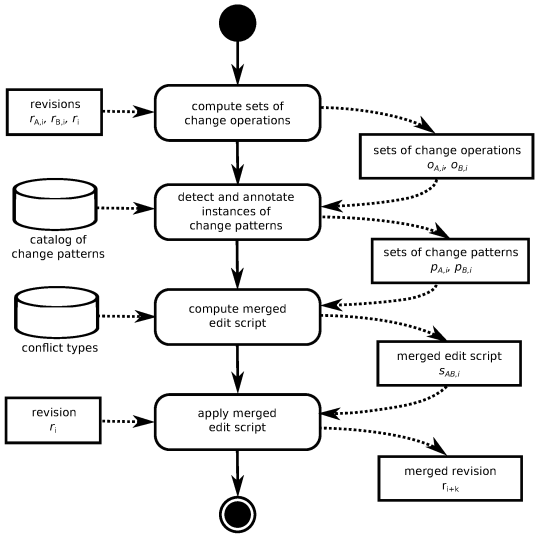


Figure 4. Implemented workflow for handling concurrent changes

In a first step we focused on the implementation of a limited set of change patterns and a respective merge matrix. Additionally we aimed to provide a user-interface that efficiently supports users in case manual interventions are required. The basic workflow of our implementation can be seen in figure 4. Accordingly we implemented several methods to accomplish the tasks outlined in the workflow. First, the concurrent revisions are compared with their antecedent revision of the current modeler and two sets of atomic change operations constituting the revisions are computed. Second, the two unordered sets of change operations are matched against the catalog of change patterns to compute two sets containing semantically annotated instances of retrieved change patterns. We accomplished this mainly by recursively searching for related atomic change operations. As a result we obtain instances of change patterns that are classified according to our catalog of change patterns. Third, each combination of change patterns are

checked whether the conditions for a conflict hold and in case changes are not independently applicable a respective resolution strategy is chosen. As a result we produce a set of ordered atomic change operations that constitutes an edit script $s_{AB,i}$ which is subsequently applied to the antecedent revision to obtain a merged revision. All those pairs of concurrent change operations where automated resolution is not possible are transformed into a change operation that creates an annotation element (e.g. for BPMN a documentation element) which contains all relevant data to reconstruct the conflicting changes though the change operations themselves are omitted.

As explicated above "true" conflicts need – in case no resolution heuristic is available – to be delegated to the user. For this purpose the different change patterns involved need to be visually presented to the user. For example, in case two modelers suggest different values for the `name` property than those values must be presented in a way that the user can decide which value to use. In addition a user inspecting a conflict and striving for a resolution needs a possibility to simulate the impact of the two suggestions and must be able to switch back and forth between the suggestions. In case a modeler is not able to take a decision immediately (at the time of the recognition of the concurrent situation) it should be possible to procrastinate the conflict resolution to a later point in time or to another modeler (see also figure 5).

The result of revision merging is presented to the user through the model editor component. Additionally, the antecedent revision, the concurrent revision and the user's revision are presented on multiple transparent layers (instantiations of the canvas). The user confronted with the merged revision may inspect all revisions and may hide or show revisions and overlay revisions as needed. Also revisions can be selected to serve as a basis for manually resolving conflicts in case the computed revision is not satisfying.

Conclusion

In this paper we have summarized findings from a conceptual investigation of concurrent situations in process modeling. We argue that given a concurrent situation as defined in section II changes should be integrated in a way that ideally reflects intentions of all modelers. Moreover we suggested merging of concurrent changes to be based on so called change patterns that refer to the rationale of a change which in turn will facilitate the understanding of changes and their resolution in case conflicts need to resolved manually.

The contributions of the paper are threefold. First, we introduced the notion of change patterns and show how we semi-formally described them through a structured pattern language, and collected them in a pattern catalog. Second, we showed how we use these patterns together with a merge matrix and corresponding conflict types to capture the complexity of process model merging. Finally, we illustrated in short how we implemented a preliminary version of our concurrency handling approach in a highly dynamic wiki-based process modeling environment.

Our approach reflects a preliminary state of research where we aim to provide mechanisms that support modelers in highly dynamic scenarios of collaborative modeling. In future we aim to put the above conceptual framework on a formal basis that is largely independent from a particular implementation and modeling language. Furthermore, we will aim to develop a comprehensive, open and retrievable catalog of change patterns and associated conflict types in process modeling. Doing so we expect that we can classify and generalize change patterns recurrent in behavioral modeling. Thus, providing a general basis to capture the semantics of model changes and conflicts.

References

[1] I. Davies, P. Green, M. Rosemann, M. Indulska, and S. Gallo, "How do practitioners use conceptual modeling in practice?" *Data and Knowledge Engineering*, vol. 58, no. 58, pp. 358–380, 2006.

[2] J. Recker and J. Mendling, "Adequacy in process modeling: A review of measures and a proposed research agenda-position paper," in *CAiSE 2007 Workshop Proceedings Vol. 1*. Tapir Academic Press, 2007, pp. 235–244.

[3] A. W. Scheer, *ARIS – vom Geschäftsprozess zum Anwendunssystem*, A. W. Scheer, Ed. Springer, 1998.

[4] J. Mendling, G. Neumann, and M. Nüttgens, "Yet another event-driven process chain," in *Business Process Management*. Springer, 2005, pp. 428–433.

[5] M. Weske, *Business Process Management: Concepts, Languages, Architectures*, M. Weske, Ed. Springer, 2007.

[6] P. Harmon and C. Wolf, "Business Process Modeling Survey 2011," BPTrends, Tech. Rep., Dec 2011.

[7] T. Mens, "A state-of-the-art survey on software merging," *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING*, vol. 28, pp. 449–461, 2002.

[8] C. Ignat, G. Oster, P. Molli, M. Cart, J. Ferrié, A. Kermarrec, P. Sutra, M. Shapiro, L. Benmouffok, R. Guerraoui *et al.*, "A comparison of optimistic approaches to collaborative editing of wiki pages," in *Int. Conf. on Collaborative Computing. CollaborateCom.* IEEE, 2007, pp. 474–483.

[9] K. Altmanninger, M. Seidl, and M. Wimmer, "A survey on model versioning approaches," *International Journal of Web Information Systems*, vol. 5, no. 3, pp. 271–304, 2009.

[10] B. Westfechtel, "A formal approach to three-way merging of EMF models," in *Proceedings of the 1st International Workshop on Model Comparison in Practice*. ACM, 2010, pp. 31–41.

[11] J. Küster, C. Gerth, A. Forster, and G. Engels, "Detecting and resolving process model differences in the absence of a change log," in *Proc. 6th Int. conf. on Business Process Management. BPM.*, vol. 5240. Springer-Verlag New York Inc, 2008, p. 244.

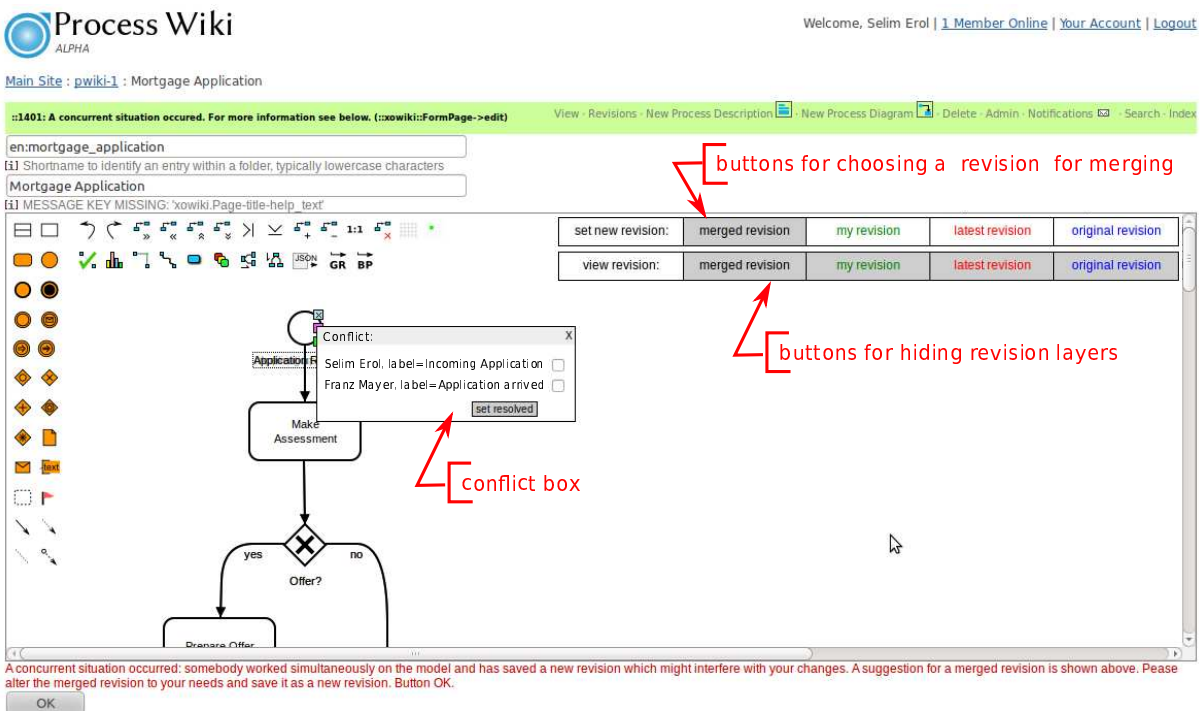


Figure 5. Screenshot of *xoWiki* model editor in conflict resolution mode: a conflict box enables a user to decide between conflicting changes or apply a manual merge. In the upper right corner several buttons enable switching between concurrent model revisions, merged model revision and the original revision.

[12] R. Dijkman, M. Dumas, L. Garcia-Banuelos, and R. Kaarik, "Aligning business process models," in *Enterprise Distributed Object Computing Conference. EDOC'09.* IEEE, 2009, pp. 45–53.

[13] C. Gerth, J. M. Küster, M. Luckey, and G. Engels, "Precise detection of conflicting change operations using process model terms," in *Proceedings of the 13th international conference on Model driven engineering languages and systems: Part II*. Springer-Verlag, 2010, pp. 93–107.

[14] M. La Rosa, M. Dumas, R. Uba, and R. Dijkman, "Business process model merging: an approach to business process consolidation," *ACM Trans. on Softw. Engineering and Methodology*, vol. -, pp. –, 2012.

[15] P. Frederiks and T. Van der Weide, "Information modeling: the process and the required competencies of its participants," *Data & Knowledge Engineering*, vol. 58, no. 1, pp. 4–20, 2006.

[16] R. Johansen, "Teams for tomorrow [groupware]," in *Proc. of the 24th Hawaii Int. Conf. on System Sciences*, vol. 3. IEEE, 1991, pp. 521–534.

[17] B. Weber, S. Rinderle, and M. Reichert, "Change patterns and change support features in process-aware information systems," in *Advanced IS Engineering*. Springer, 2007, pp. 574–588.

[18] S. Erol, "Design and evaluation of a wiki-based collaborative process modeling environment," Ph.D. dissertation, WU Vienna, 2012.

[19] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design patterns: elements of reusable object-oriented software*. Addison-Wesley, 1995.

[20] J. Munson and P. Dewan, "A flexible object merging framework," in *Proceedings of the 1994 ACM conference on Computer supported cooperative work*. ACM, 1994, pp. 231–242.

[21] R. Heckel, J. Küster, and G. Taentzer, "Confluence of typed attributed graph transformation systems," *Graph Transformation*, vol. –, pp. 161–176, 2002.

[22] T. Mens, G. Taentzer, and O. Runge, "Detecting structural refactoring conflicts using critical pair analysis," *Electronic Notes in Theoretical Computer Science*, vol. 127, no. 3, pp. 113–128, 2005.

[23] P. Brosch, M. Seidl, and M. Wimmer, "Mining of Model Repositories for Decision Support in Model Versioning," in *Proc. 2nd Europ. Workshop on Model Driven Tool and Process Integration*, 2009, pp. 25–33.

[24] G. Neumann and U. Zdun, "XOTcl, an Object-Oriented Scripting Language," in *Proceedings of 7th USENIX Tcl/Tk Conference*, 2000.

[25] G. Neumann and S. Sobernig, "An Overview of the Next Scripting Toolkit," in *Proc. 18th Annu. Tcl/Tk Conf.*, 2011.

[26] N. Demetriou, S. Koch, and G. Neumann, "The Development of the Open ACS Community," *Open Source for Knowledge and Learning Management: Strategies Beyond Tools*, vol. n.a., p. 298, 2007.