Towards Schema Evolution in Object-aware Process Management Systems

Carolina Ming Chiao, Vera Künzle, Manfred Reichert

Institute of Databases and Information Systems
University of Ulm, Germany
{carolina.chiao, vera.kuenzle, manfred.reichert}@uni-ulm.de

Abstract: Enterprises want to improve the lifecycle support for their businesses processes by modeling, enacting and monitoring them based on process management systems (PrMS). Since business processes tend to change over time, process evolution support is needed. While process evolution is well understood in traditional activity-centric PrMS, it has been neglected in *object-aware* PrMS so far. Due to the tight integration of processes and data, in particular, changes of the data and process schemes must be handled in an integrated way; i.e., the evolution of the data schema might affect the process schema and vice versa. This paper presents our overall vision on the controlled evolution of object-aware processes. Further, it discusses fundamental requirements for enabling the evolution of object-aware process schemas in PHILharmonicFlows, a framework targeting at comprehensive support of object-aware processes.

1 Introduction

Aiming at improved process lifecycle support, a decade ago, many researchers started working on process schema evolution. In general, business processes may evolve for several reasons; e.g, due to changes in the business, technological environment, or legal context [RW12]. Consequently, business process changes need to be rapidly mapped to the process-aware information system (PAIS) implementing these processes.

Activity-centric process management systems (PrMS) like YAWL [vdAtH05] and ADEPT [RD98, RRD04, RW12] already provide comprehensive process lifecycle support, including the controlled evolution of business processes. Regarding object-aware processes [Kün13], however, this does not apply yet. Due to the tighter integration of process and data, changes of the data and process schemes must be handled in an integrated way. In other words, changing the data schema may affect the schema of an object-aware process and vice versa. Note that respective dependencies might become complex when taking different levels of process granularity as well as authorization constraints into account as well.

The example below is based on a real educational scenario. It comprises a process for managing extension course projects. Extension courses target at professionals that want to refresh and update their knowledge in a certain area. In order to propose a new extension course, the course coordinator must create a corresponding project description. The latter must then be approved by the faculty coordinator and the extension course committee.

Example 1 (Object-aware Process: Extension course proposal). The `course coordinator` creates an `extension course project` using a form. In this context, he must provide details about the course, like `name`, `start date` and `description`. Following this, `professors` may start creating the `lectures` of the extension course. In turn, each `lecture` comprises `study plan items`, which describe the topics to be covered by the lecture. After creating the lectures, the `coordinator` may request an approval of the `extension course project`. First, an approval must be provided by the `faculty director`. If he wants to `reject` the proposal, the extension course must not take place. Otherwise, the project is sent to the `extension course committee`, which will evaluate it. If there are more `rejections` than `approvals`, the `extension course project` is `rejected`. Otherwise, it is `approved` and hence may take place in future.

The process from Example 1 can be characterized by its need for *object-awareness*; i.e., business processes and business objects must not be treated independently from each other. In general, *object-aware processes* show three major characteristics. First, they are based on *two levels of granularity*. On the one hand, the *behavior* of individual object instances needs to be considered during process execution; on the other, the *interactions* among different object instances must be taken into account. Second, process execution is *data-driven*; i.e., the progress of a process depends on available object instances as well as the values of their attributes. Third, *flexible activity execution* is crucial. In particular, activities need not always coincide with process steps.

The PHILharmonicFlows framework we are developing targets at a comprehensive support of object-aware processes [KR09b, KR09a, KR11, Kün13]. It comprises modules for the modeling, execution and monitoring of object-aware processes. In this framework, *object behavior* is captured through *micro processes*. In turn, *object interactions* are captured by a *macro process*. Furthermore, data is modeled separately from micro and macro processes. Note that each of these models comprises different *components*. For example, a data model comprises object types as well as their attributes and relations to other object types.

Schema evolution has neither been considered by PHILharmonicFlows nor other frameworks for artifact-based or object-aware processes yet. As a major challenge, one must cope with the complex interdependencies that exist between the models and components (e.g., data model, object types, attributes, or micro process types) of the framework; i.e., changing one component (e.g., deleting an object attribute) may require concomitant changes of other components (e.g., changing the behavior of the object type). Moreover, changes must be handled at both the static and dynamic (i.e., instance) level. Changing an object-aware process without any user assistance will be error-prone and time-consuming. Therefore, user interactions should be properly supported in order to guide the modeler when changing an object-aware process. In particular, any guidance must hide complexity from users, taking correctness constraints and component dependencies into account.

This paper presents requirements necessary to enable schema evolution for object-aware processes. To illustrate how these requirements were derived, we sketch our vision on how the user should interact with the PHILharmonicFlows tool when changing an object-aware process. Sect. 2 provides an overview of the PHILharmonicFlows framework. Sect. 3 presents research questions to emphasize the scope of our work. In Sect. 4, we introduce

our vision on how the user (i.e., modeler) should be supported when evolving object-aware processes. Sect. 5 presents major requirements emerging in this context. Sect. 6 discusses the related work and Sect. 7 gives a summary and outlook.

2 The PHILharmonicFlows Framework

The PHILharmonicFlows framework enforces a modeling methodology governing the object-centric specification of business processes based on a well-defined formal semantics [KR11, Kün13]. In general, an *object-aware process schema* comprises the following sub-schemas: data model (cf. Fig. 1a), micro process types (cf. Fig. 1b), macro process types (cf. Fig. 1d), and authorization settings (cf. Fig. 1c). In turn, each sub-schema comprises a set of components (e.g., a data model comprises object types, object type attributes and relations to other object types), which may be related to components of other sub-schemas (e.g., a micro step type depends on an attribute of an object type). When changing components, hence, concomitant changes of dependent components become necessary as well.

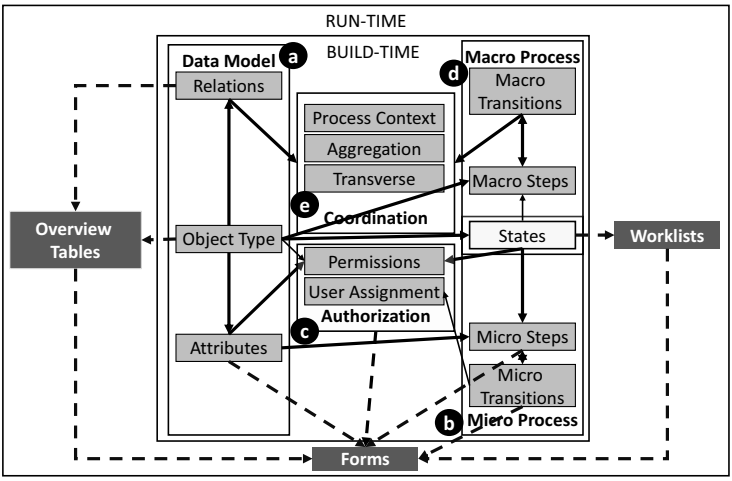

Figure 1: The PHILharmonicFlows framework

As a fundamental prerequisite, object types and their relations need to be captured in a *data model* (cf. Fig. 1a). Furthermore, for each *object type*, a corresponding *micro process type* needs to be specified (cf. Fig. 1b). The latter defines the behavior of related object instances, and consists of a set of *micro steps* as well as the *transitions* between them. In turn, each micro step is associated with an *object type attribute*. Further, micro steps are grouped in object *states*. At run-time, for each *object instance*, a corresponding *micro process instance* is created. A micro process instance being in a particular state may only proceed if specific values are assigned to the object instance attributes associated with this state; i.e., *data-driven process execution* is enabled. In addition, *optional data access* is accomplished asynchronously to micro process execution based on the permissions granted for reading or writing object attributes. In this context, access rights for an object instance depend on the progress of the corresponding micro process instance as well. Altogether,

the framework maintains an *authorization table* assigning data permissions to user roles which may also depend on the respective state of the micro process type (cf. Fig. 1c).

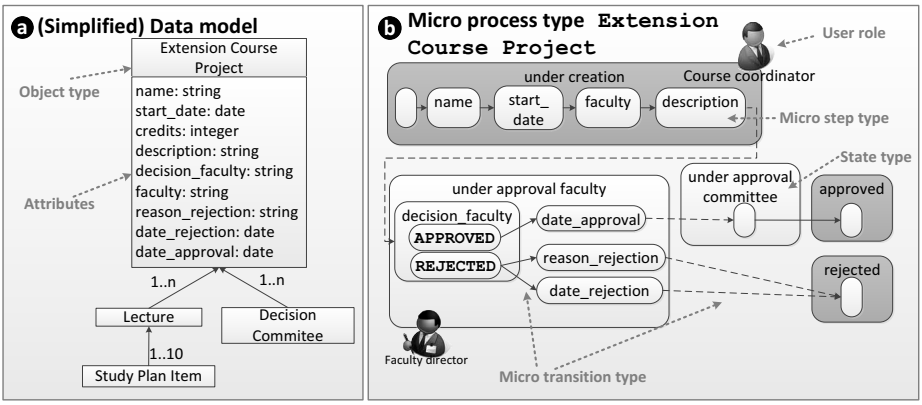

Figure 2: Data model and micro process type modeled with PHILharmonicFlows

Taking the relations between the object instances of the overall *data structure* (i.e., the instance of a data model) into account, the corresponding micro process instances form a complex *process structure*; i.e., their execution needs to be coordinated according to the given data structure. In PHILharmonicFlows, this is accomplished by means of macro processes. A *macro process type* consists of *macro steps* linked by *macro transitions* (cf. Fig. 1d). Opposed to micro steps, which refer to single attributes of a particular object type, a macro step refers to a particular state of an object type. In addition, for each macro transition, a coordination component must be specified (cf. Fig. 1e). The latter hides the complexity of large process structures from modelers as well as end-users. More precisely, such a coordination component coordinates the interactions among the object instances of the same type as well as different types. Opposed to existing approaches, the semantic relations between the object instances and their cardinalities are also taken into account.

Figs. 2 and 3 show how Example 1 can be modeled based on PHILharmonicFlows. Micro process type `Extension course project` is derived from the object type having same name (cf. Figs. 2a+b). In this micro process type, the `course coordinator` must write attributes `name`, `start_date`, `faculty`, and `description`. Note that this configuration needs to be reflected in the authorization settings (cf. Fig. 3a). Furthermore, attribute `credits` may be *optionally* written. In turn, a macro process type (cf. Fig. 3b) is composed of macro step types. The latter reference state types from the micro process types. For example, macro step type `Extension Course Project - under creation` refers to state type `under creation` of micro process type `Extension Course Project` (cf. Fig. 2b).

3 Research Questions

To emphasize the scope of our problem, we consider the following research questions:

Research Question 1 (RQ1): How to change an object-aware process schema without violating correctness neither of the modified component itself nor any dependent compo-

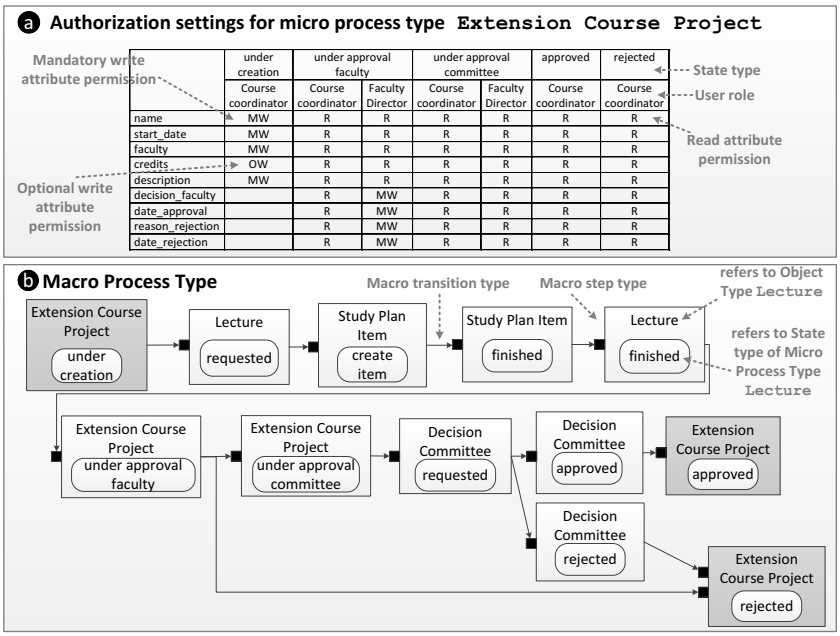

Figure 3: Authorization settings and macro process type modeled with PHILharmonicFlows

nents?

Research Question 2 (RQ2): How to handle active instances (i.e., object and micro process instances) when evolving the object-aware process schema?

Research Question 3 (RQ3): How to assist users in evolving an object-aware process schema?

RQ1 refers to changes at the static level. It deals with structural changes of an object-aware process schema; i.e., its sub-schemas and their components. In turn, RQ2 addresses issues related to dynamic changes; i.e., managing different schema versions and adopting the best policy to migrate active instances to the new schema version. Finally, RQ3 deals with user issues, such as providing user guidance while hiding the complexity of schema changes from them. These research questions guide our vision discussed in Sect. 4. Further, they serve as starting point for eliciting requirements related to the evolution of object-aware processes.

4 Overall Vision

To illustrate the scope of our research, we define a number of scenarios (i.e., user stories) dealing with schema changes of our sample process. While some scenarios are rather simple, not requiring any concomitant change, others are more complex involving several schemas of the object-aware process. Thereby, a major challenge concerns user interaction as changing an object-aware process schema constitutes an error-prone and complex task. Hence, user guidance is required to assist users when changing the schema of an object-aware process, e.g., by indicating the components affected by an intended change. For

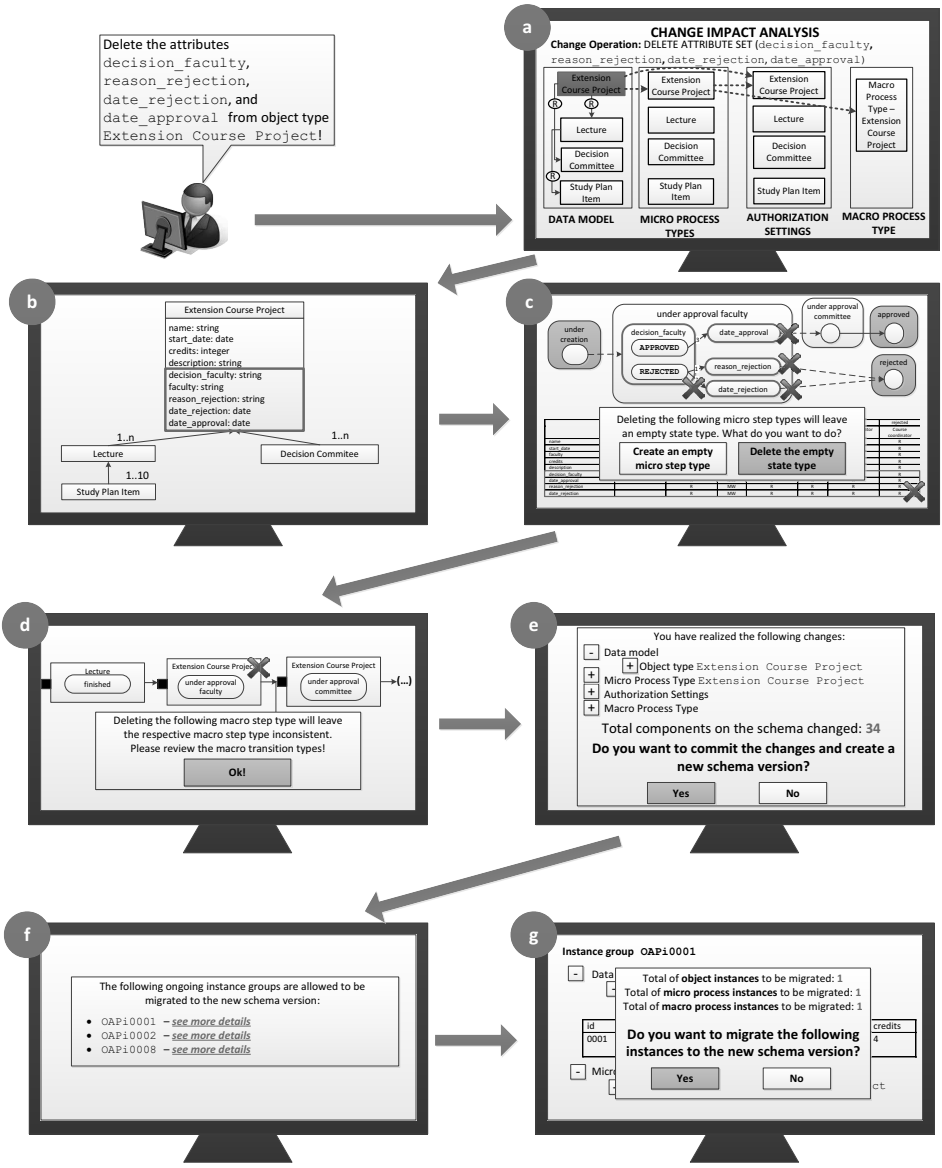

Figure 4: Sketch of end-user guidance for object-aware process schema changing

illustration purpose, we provide a mockup showing how to guide a user in the context of a concrete change scenario.

Change scenario (SC): The `extension course project` needs not be approved by the `faculty director` anymore, and the overall schema shall be adapted accordingly.

Since for an extension course project the approval of the faculty director is no longer needed, the user wants to delete the attributes referring to it (e.g., `decision_faculty`, `reason_rejection`, `date_rejection`, and `date_approval`); i.e., he wants to change the data model. In turn, this requires concomitant schema adaptations. First of all, the user should be notified about the effects of the change; i.e., a *change impact analysis* is required. Based on such an analysis, it can be visualized which sub-schemas and components are affected by the change and how they are related. Regarding our mockup (cf. Fig. 4a), the object-aware process schema is divided into four levels corresponding to the sub-schemas data model, micro process types, authorization settings, and macro process type. The dotted arrows represent potential effects caused by the change. In the given scenario, changing object type `Extension Course Project` affects the corresponding micro process type as well as the authorization settings. Changing the micro process type `Extension Course Project`, in turn, may further affect the macro process type.

Being aware of the possible effects of the intended change, the user may then be guided in performing required concomitant changes (cf. Fig. 4b). According to PHILharmonicFlows (cf. Sect. 2), micro steps of a particular micro process are directly related to the attributes of the corresponding object type. Therefore, when an attribute is deleted, the corresponding micro step type needs to be deleted as well. In Fig. 4c, in turn, the deletion of the respective micro step types will result in an "empty" state type. According to the correctness constraints of the framework, this is not possible, and would leave the micro process type in an inconsistent state. Therefore, the user should be notified about this problem and guided in resolving it. In Fig. 4d, the user decides to delete the entire state type, which, in turn, causes another inconsistency in the macro process type. Again, the user should be notified about this, enabling him to redefine the macro transition types that link the respective macro step type. In general, every time the user changes a component, respective correctness checks should be performed automatically in order to be able to guide the user in resolving potential inconsistencies.

Since changes of one component might trigger changes of others, the user will not always be aware of the number of components actually changed. Hence, after guiding him through required adaptations of the object-aware process, an overview of all components to be changed should be provided; e.g., such overview could present information about the components to be changed as well as quantitative metrics (e.g., number of components and models to be changed) (cf. Fig. 4e). Finally, the user should explicitly commit the changes, resulting in a new version of the object-aware process schema.

In addition to structural adaptations and structural consistency, active instances must be taken into account; i.e., it should be possible to adapt the running instances according to the changed object-aware process schema. Fig. 4f presents the active instances that may be migrated to the new schema version without causing any run-time error. To foster

visualization, instances are grouped according to the underlying object type; i.e., each instance group refers to one particular object type. In our example, the instances refer to an `extension course project`. Then, the user may choose which group of instances he wants to migrate. Further, he may retrieve more detailed information about the respective instances (cf. Fig. 4g). Finally, like in the context of model changes, the user will get an overview of the instances to be migrated.

5 Requirements

Based on our research questions, the sketched vision, and an extensive literature study, we derived major requirements. The requirements of Sect. 5.1 are related to RQ1, while the ones of Sect. 5.2 are related to RQ2. Finally, the requirements of Sect. 5.3 are related to user guidance issues (i.e., RQ3).

5.1 Structural Changes at the Static Level

Requirement 1 (Change primitives). To accomplish structural adaptations of an object-aware process schema, change primitives are required to directly operate on single schema elements. In our context, such primitives denote atomic operations like `add attribute type`, `delete micro step type`, and `add state type`. In general, the set of available change primitives should be *complete* and *minimal* [CCPP98, RD98, RW12]. Completeness means that the available set of change primitives shall allow transforming any object-aware process schema `S` into any other object-aware process schema `S'`. In addition, the core set of provided change primitives should be minimal; i.e., it should not contain any primitive that can be simulated through the combination of other primitives. Finally, for each change primitive, a precise definition of parameters, pre-conditions, and post-conditions (i.e., effects) is required [CCPP98].

Example 2 (Requirement 1: Change primitives). For removing micro step type `date_rejection` of state type `under approval faculty`, change primitives for deleting the micro step type and its related micro transition types are required (cf. Fig. 5).

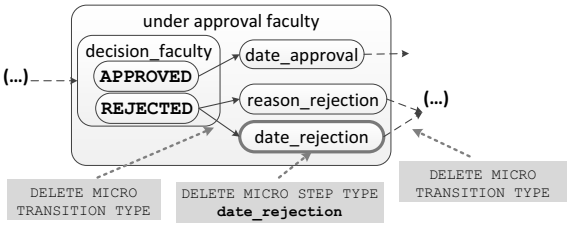

Figure 5: Example of change primitives

Requirement 2 (Cascading effects). Fig. 6 presents the meta model of PHIlharmonicFlows, expressed in terms of an UML class diagram. The dependencies between the different components of the framework are represented as bidirectional associations, compositions and aggregations. Regarding the class diagram, the *bidirectional association*

represents components linked with each other in the context of a particular model (e.g., micro step types are linked with micro transition types within a micro process type). A *composition dependency* indicates a "strong" association between components, making one component (i.e., parent component) responsible for the creation and destruction of other components (i.e., child components). For example, an object type is strongly associated with attribute types. If the object type is deleted, all related attributes must be deleted as well. In turn, the *aggregation dependency* constitutes a "weaker" relationship between components: even when deleting the parent component, the child components will not be removed. An example of an aggregation dependency is provided by the relationship between the micro step types and attributes. If a micro step type is deleted, the associated attribute is preserved. Due to these dependencies and associations among different components of the framework, changing one of them might require changes of dependent components as well. In turn, such concomitant changes might again trigger additional changes on other components (i.e., a *cascading* change). In general, mechanisms are required to detect necessary concomitant changes and to guide the user in applying them.

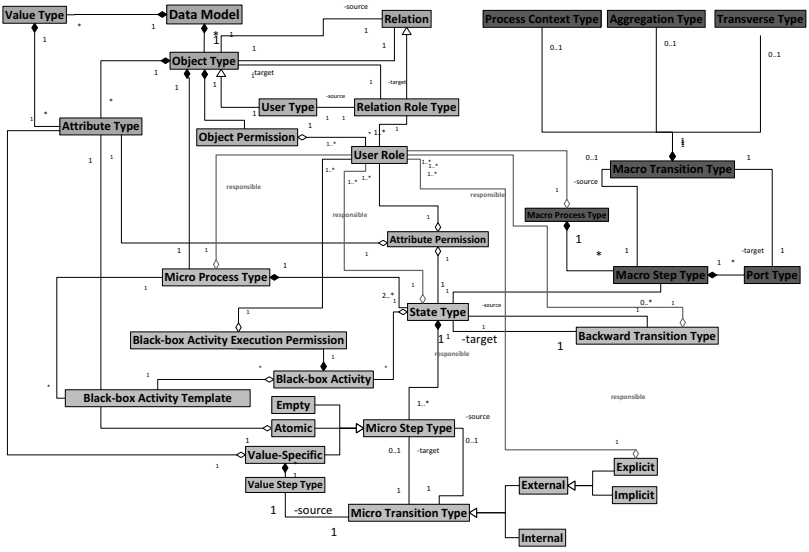

Figure 6: PHILharmonicFlows meta-model

Requirement 3 (Change operations and change patterns). Realizing structural adaptations based on change primitives might introduce errors and inconsistencies. Usually, at such a low level of abstraction, the combined application of several change primitives is required to ensure schema correctness. As an alternative, high-level change operations may be used; e.g., it should be possible to move an entire state type within a micro process type based on a single change operation. Like change primitives, high-level change operations should have pre-conditions. Generally, empirically-grounded change patterns should be defined, which capture the semantics of frequent changes, thus raising the level

of abstraction [WRRM08, RW12].

Example 3 (Requirement 3: Change operations). Deleting state type `under approval faculty` (cf. Fig. 5) requires deleting all micro step types associated with this state type as well (i.e., `decision_faculty`, `date_approval`, `reason_rejection`, and `date_rejection`). Further, this deletion includes the micro transition types linking the micro steps and the authorization settings of state type `under approval faculty`. In this context, a change operation allowing for the deletion of the entire state type together with its components would facilitate change definition significantly, and hence reduce errors and inconsistencies of the object-aware process schema.

Requirement 4 (Complex changes). When adapting an object-aware process schema, the integrity and consistency of the various sub-schemas must be preserved; i.e., the changes applied to the sub-schemas will only be applied if this does not result in any inconsistency. Like for database transactions, the changes applied jointly to an object-aware process schema must be treated atomically (i.e., as transaction). Accordingly, modelers must explicitly *commit* complex changes. Finally, multiple users may want to change the same schema version at the same time, requiring proper *concurrency control*.

Requirement 5 (Change traceability). When changing an object-aware process, information on *who* applied *which* changes, *when* and *why* shall be recorded in logs; i.e., change traceability needs to be ensured.

Requirement 6 (Correctness). Changing an object-aware process schema must not result in errors in any of the sub-schemas and not lead to soundness violations (e.g., deadlocks due to data inconsistencies or missing data at run-time). Moreover, a changed object-aware process schema must comply with the correctness criteria established in [Kün13]. Correctness checks are required at two different stages. First, when specifying the various changes of an object-aware process schema, correctness checks are "soft"; i.e., they provide basis to inform the modeler about potential inconsistencies or missing components. Second, correctness needs to be ensured when committing a change transaction; i.e., all sub-schemas forming an object-aware process schema must be correct.

5.2 Changing Active Instances at the Dynamic Level

Requirement 7 (Versioning support). Active instances whose processing started before the schema change must be properly handled. One strategy frequently applied in the context of database and process evolution, is *schema versioning* [Rod96, GdSEM05, KG99]. Every time a schema is changed, a new schema version is created; already active instances continue their processing based on the old schema version. In our context, there are various sub-schemas forming the overall object-aware process schema (i.e., data model, micro and macro process schemas, and authorization settings). Hence, for each object-aware process schema version, the versions of its sub-schemas need to be maintained (cf. Fig. 7). In particular, the instances are linked to a sub-schema version as well as the object-aware process schema version.

If a change is performed, which concerns only a part of the entire object-aware process schema, creating a new version of all sub-schemas involved (even the unchanged ones) will

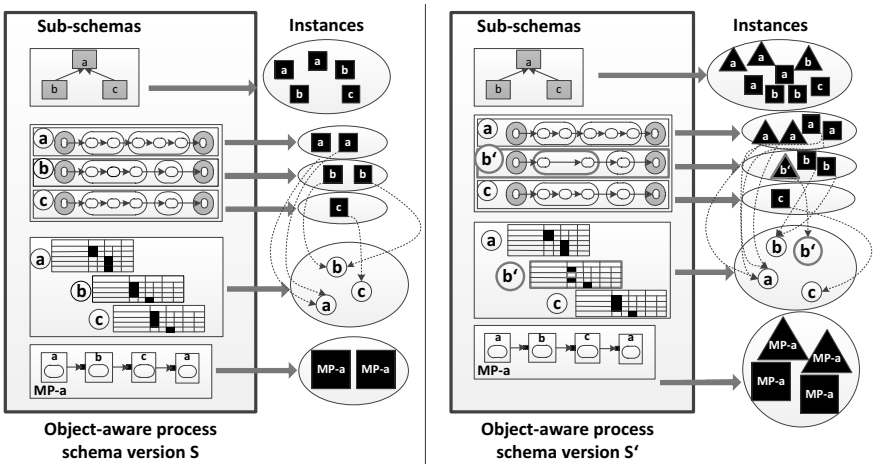

Figure 7: Object-aware process schema versions

not be optimal. Hence, a new version of an object-aware process schema shall comprise the new versions of the changed sub-schemas in combination with the versions of the unchanged sub-schemas. Fig. 7 illustrates this concept. The object-aware process schema `S` on the left side contains a data model with object types `A`, `B` and `C`. Each object type is associated with a micro process type as well as authorization settings. The latter express who may access which attributes at which stages during process execution. Finally, macro process type `MP-a` describes the interaction of the object types. The small squares represent the instances created according object-aware process schema version `S` and being active at the moment. On the right side, a new version `S'` of `S` is depicted; it resulted due to a change of micro process type `b`. The latter led to a new version `b'` of `b` and a change of related authorization settings. Instead of generating copies for all sub-schemas, `S'` comprises the new version `b'`, the new version of the respective authorization settings, and references to the versions of the unchanged sub-schemas. New instances run according to the new schema version `S'`, while the older, but still active instances continue running on the old schema version `S`; i.e., instances running on the two schema versions will co-exist. In Fig. 7, new instances are represented as triangles and old ones as squares.

Requirement 8 (Instance migration). To ensure that active instances may continue running on the old schema version is not sufficient. In addition, it shall be possible to reassign active instances to the new object-aware process schema version if desired. Like in activity-centric PrMS [CCPP98, JH98, RD98, RRMD09], such migration of active instances must be handled in a controlled manner. In general, not all instances can be migrated to the new schema version, particularly if they have progressed too much in their execution. Since there may be numerous concurrently running instances of an object-aware process (i.e., object and micro process instances), the selection of the migratable instances should not handle the instances individually. In the example from Fig. 8a, a new state is inserted in micro process type `A`. Moreover, the progress of the instances of micro process `B` now depends on the execution of micro process `A`; i.e., the instances of `B` will only reach state `s5` if all instances of `A'` reach state `s7` (cf. Fig. 8b). However, not all

active instances of micro process A can be migrated to A′. More precisely, micro process instances A1 and A2 have already completed their executions, which means that they cannot be migrated to A′. In turn, micro process instance A3 may be migrated. However, the individual migration of A3 will cause a *deadlock* at run-time, since micro process instance B1 will continue waiting for instances A1 and A2 to reach state s7. Therefore, a group of instances associated to a particular changed component must not be migrated individually.

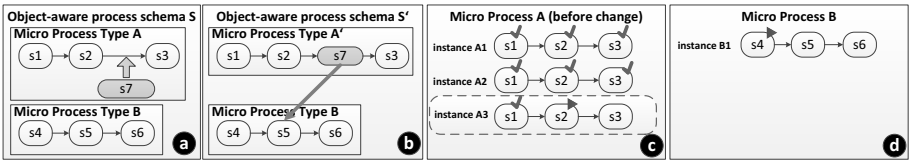

Figure 8: Example of instance migration

Requirement 9 (Data consistency). One of the biggest issues concerning database schema evolution is to prevent data loss when changing a database schema [Ra04]; i.e., the deletion of object types or attributes must not delete the data associated to them, since there may be software systems that still depend on this data. In the context of object-aware processes, data inconsistency might cause run-time errors (e.g., deadlocks). Therefore, it becomes necessary to prevent data inconsistencies (e.g., data loss or missing data important to the logic of the process) relevant for process execution.

5.3 User Requirements

Requirement 10 (User guidance). Changing an object-aware process schema is a non-trivial task from the viewpoint of the user (i.e., process modeler). Our experiences with the change scenarios have shown that user guidance is required to hide this complexity and hence to make schema changes more intuitive and less error-prone (cf. Sect. 4). Moreover, user guidance not only eases the adaptation of an existing schema, but also the modeling of new object-aware process schemas.

Requirement 11 (Change impact analysis and metrics). To better control potential costs of a change, a *change impact analysis* should be performed *before* actually applying the change. Such an analysis must consider the cascading effects; i.e., it must show to users which components are going to be affected by the change. Moreover, metrics help users to evaluate change complexity.

6 Related Work

The described requirements have been partially addressed by existing work. Fig. 9 summarizes which requirements have been addressed by which approach. We investigated data-centric approaches and traditional activity-centric ones.

Data-centric Approaches

Data-driven Process Coordination (COREPRO) [MRH07, MRH08] presents a set of change primitives and operations to change both data and process structures. Since the latter are directly related, the approach automatically adapts the process structure when chang-

+ supported ***o partially supported*** ***- not supported***	COREPRO	Artifact-centric processes	Product-based workflow	FLOWer	ADEPT	YAWL
Req. 1 (Change primitives)	+	+	o	+	+	+
Req. 2 (Cascading effect)	+	+	o	-		
Req. 3 (Change operations and change patterns)	+	+	-	+	+	-
Req. 4 (Complex changes)	+	-	-	+	+	+
Req. 5 (Change traceability)	+	-	-	o	+	+
Req. 6 (Correctness)	+	-	+	o	+	+
Req. 7 (Versioning support)	o	-	-	+	+	+
Req. 8 (Instance migration)	o	-	-	-	+	-
Req. 9 (Data consistency)	o	-	-	-		
Req. 10 (User guidance)	o	-	-	-	+	-
Req. 11 (Change impact analysis and metrics)	-	o	-	-	o	-

Figure 9: Evaluation of different approaches

ing the corresponding data structure. Moreover, it enables change traceability as well as change transactions. Correctness is ensured when changing the structures at static or instance level. In case of inconsistencies, the modeler is notified accordingly. However, even though the data objects are explicitly represented, the control of the data structures is still realized outside the scope of the PrMS. Hence, versioning support and instance migration is only available for the process structures.

Regarding artifact-centric processes, [WW14] proposes an approach for dealing with the change impact analysis of three-level artifact-centric business processes (ACBP). The authors first classify the types of changes that may be applied to an ACBP. This classification provides the basis for the change analysis. For this analysis, a graph representing the different element dependencies is created. Based on this graph, it becomes possible to calculate the direct impact of a change. The approach, however, just covers the changes at static level, without addressing the problems of complex changes and traceability. In turn, [XSY+11] allows for ad-hoc changes on the artifacts' life cycles. Such changes are based rules and declarative constructs such as *skip*, *add* and *replace* and are applied to the tasks. The artifacts, in turn, cannot be changed. Besides, the authors do not provide information on how the active instances should be handled when a change is applied.

In the context of product-based workflows, [RVV10] presents four change primitives enabling changes of the product (i.e., data) structure. These are then reflected in the corresponding process models without need for any manual adaptation. Changes at the process level, however, are not considered. Moreover, information regarding change traceability and run-time issues are neglected.

The case-handling system FLOWer [vdAWG05, MWR08] does not address cascading effects; i.e., inconsistencies in a dependent component caused by a change are not properly handled by the system. Moreover, the system does not use formal correctness criteria in

the context of schema evolution [WRRM08]. Regarding schema versioning, FLOWer allows for overwriting a process schema as well as for the co-existence of instance running on different schema versions. Instance migration is not considered.

Activity-centric Approaches

In activity-centric approaches, the processes are described on a single level (i.e., the process model). Additionally, data is managed outside the scope of the PrMS. For these reasons, requirements regarding cascading effects (Req. 2) and data consistency (Req. 9) were not addressed to the analyzed approaches. ADEPT [RD98, RRD04, RW12] is an activity-centric PrMS that enables both schema evolution and ad hoc changes of single process instances. Moreover, it focuses on the *ease of use* of its process support features. Additionally, metrics regarding instance migration are provided. In turn, YAWL supports evolutionary changes in workflows based on Worklets [vdAtH05]. The latter refer to an extensive repertoire of self-contained sub-processes and association rules, which can be inserted into the process model without any system downtime. Even though it provides primitives for changing the process model, there are no change patterns or operations to realize changes at a higher abstraction level. Regarding the user, no guidance or change impact analysis are provided.

7 Summary and Outlook

Our overall vision is to enable schema evolution in object-aware processes. The major challenge lies on the very tight integration of the components of the framework. Such component dependencies might not only affect the object-aware process schema at static level, but active instances as well; i.e., new versions of a particular sub-schema must coexist with unchanged versions of other sub-schemas. Moreover, we observed that user guidance is crucial to hide the complexity from the modeler and to avoid schema errors. In future work, we will provide detailed insights into our solution tackling the discussed requirements.

References

[CCPP98] F. Casati, S. Ceri, B. Pernici, and G. Pozzi. Workflow Evolution. *Data & Know Eng*, 24(3):211–238, 1998.

[GdSEM05] R. Galante, C. Saraiva dos Santos, N. Edelweiss, and A. F. Moreira. Temporal and Versioning Modeling for Schema Evolution in Object-oriented Databases. *Data & Know Eng*, 53(2):99–128, 2005.

[JH98] G. Joeris and O. Herzog. Managing Evolving Workflow Specifications. In *Proc.CoopIS'98*, pages 310–319, 1998.

[KG99] M. Kradolfer and A. Geppert. Dynamic Workflow Schema Evolution Based on Workflow Type Versioning and Workflow Migration. In *Proc. CoopIS'99*, pages 104–114, 1999.

[KR09a] V. Künzle and M. Reichert. Integrating Users in Object-aware Process Management Systems: Issues and Challenges. In *Proc. BPM'09 Workshops*, pages 29–41, 2009.

[KR09b] V. Künzle and M. Reichert. Towards Object-aware Process Management Systems: Issues, Challenges, Benefits. In *Proc. BPMDS'09*, pages 197–210, 2009.

[KR11] V. Künzle and M. Reichert. PHILharmonicFlows: Towards a Framework for Object-aware Process Management. *Journal of Software Mainteinance and Evolution: Research and Practice*, 23(4):205–244, 2011.

[Kün13] V. Künzle. *Object-aware Process Management*. PhD thesis, Ulm University, 2013.

[MRH07] D. Müller, M. Reichert, and J. Herbst. Data-driven Modeling and Coordination of Large Process Structure. In *Proc. CoopIS'07*, pages 131–149, 2007.

[MRH08] D. Müller, M. Reichert, and J. Herbst. A New Paradigm for the Enactment and Dynamic Adaptation of Data-driven Process Structures. In *Proc. CAiSE'08*, pages 48–63, 2008.

[MWR08] B. Mutschler, B. Weber, and M. Reichert. Workflow Management versus Case Handling: results from a Controlled Software Experiment. In *Proc. SAC'08*, pages 82–89, 2008.

[Ra04] Y.-G. Ra. Relational Schema Evolution for Program Independency. In *Proc. CIT 2004*, volume 3356, pages 273–281, 2004.

[RD98] M. Reichert and P. Dadam. ADEPTflex - Supporting Dynamic Changes of Workflows Without Losing Control. *Journal of Intelligent Information Systems*, 10(2):93–129, 1998.

[Rod96] J. F. Roddick. A Model for Schema Versioning in Temporal Database Systems. *Australian Computer Science Communications*, 18:446–452, 1996.

[RRD04] S. Rinderle, M. Reichert, and P. Dadam. Flexible Support of Team Processes by Adaptive Workflow Systems. *Distr. Parallel Databases*, 16(1):91–116, 2004.

[RRMD09] M. Reichert, S. Rinderle-Ma, and P. Dadam. Flexibility in Process-Aware Information Systems. *Trans Petri Nets and other Models of Conc III*, pages 115–135, 2009.

[RVV10] H. A. Reijers, J. Vogelaar, and I. Vanderfeesten. Changing Products, Changing Processes: Dealing with Small Updates in Product-Based Design. In *Proc. eKNOW'10*, pages 56–61, 2010.

[RW12] M. Reichert and B. Weber. *Enabling Flexibility in Process-aware Information Systems: Challenges, Methods, Technologies*. Springer, 2012.

[vdAtH05] W. M. P. van der Aalst and A. H. M. ter Hofstede. YAWL: Yet Another Workflow Language. *Information Systems*, 30(4):245–275, 2005.

[vdAWG05] W. M. P. van der Aalst, M. Weske, and D. Grünbauer. Case Handling: A New Paradigm for Business Process Support. *Data & Know Eng*, 53(2):129–162, 2005.

[WRRM08] B. Weber, M. Reichert, and S. Rinderle-Ma. Change Patterns and Change Support Features - Enhancing Flexibility in Process-Aware Information Systems. *Data & Know Eng*, 66(3):438–466, 2008.

[WW14] Y. Wang and Y. Wang. Change Analysis for Artifact-Centric Business Processes. In *Proc. BIS 2014*, pages 98–109, 2014.

[XSY+11] W. Xu, J. Su, Z. Yan, J. Yang, and L. Zhang. An Artifact-centric Approach to Dynamic Modification of Workflow Execution. In *Proc. OTM 2011*, pages 256–273, 2011.