Schema Evolution in Object and Process-Aware Information Systems: Issues and Challenges

Carolina Ming Chiao, Vera Künzle, and Manfred Reichert

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

Abstract. Enabling process flexibility is crucial for any process-aware information system (PAIS). In particular, implemented processes may have to be frequently adapted to accommodate to changing environments and evolving needs. When evolving a PAIS, corresponding process schemas have to be changed in a controlled manner. In the context of object-aware processes, which are characterized by a tight integration of process and data, PAIS evolution not only requires process schema evolution, but the evolution of data and user authorization schemas as well. Since the different schemas of an object-aware PAIS are tightly integrated, modifying one of them usually requires concomitant changes of the other schemas. This paper presents a framework for object-aware process support and discusses major requirements and challenges for enabling schema evolution in object-aware PAIS.

1 Introduction

Contemporary PAISs are usually *activity-driven*; i.e., processes are modeled in terms of "black-box" activities and their control flow, defining the order and constraints for executing these activities. Business data, in turn, is treated as a second-class citizen [4, 1] and is usually stored in external databases. Hence, activity-centric PAISs are unable to provide immediate access to process-related information at any point of time. Moreover, many PAIS limitations (e.g., application data only being accessible in the context of an activity) can be traced back to the missing integration of process and data [5, 6, 7, 8]. To address these drawbacks, we have developed the PHILharmonicFlows framework, which allows for the operational support of *object-aware* processes at two levels of granularity: *object behavior* and *object interaction* [7, 9]. In addition, data-driven process execution as well as integrated access to process and application data become possible. One aspect neglected so far PHILharmonicFlows concerns the evolution of object-aware processes and their components (i.e., the schemas defining object behavior and interactions, data structures, and user authorization). In this context, one does not only have to deal with changes of process schemas (including their propagation to running instances), but of other components of the object-aware PAIS as well (e.g., changes of the data model might affect object behavior and object interactions). Generally, when evolving one particular component of an object-aware PAIS, this might necessitate changes of dependent

components as well (with potentially cascading effects). This paper discusses some of the fundamental challenges to be tackled when targeting at schema evolution in object-aware PAISs. These challenges were derived from case studies as well as a comprehensive literature study.

Section 2 gives an overview of the PHILharmonicFlows framework. The challenges emerging in the context of schema evolution in an object-aware PAIS are presented in Section 3. Section 4 discusses related work, and Section 5 concludes the paper.

2 Object-aware Process Support

We first present a simple process scenario along which we introduce basic concepts related to object-aware processes. This scenario deals with proposing extension courses at a university; i.e., courses for professionals that aim at refreshing and updating their knowledge in a certain area of expertise. To propose an extension course, the course coordinator must create a project describing it. The latter must be approved by the faculty coordinator as well as the extension course committee.

Example 1 (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`, `duration`, and `description`. Following this, `professors` may start creating the `lectures` for the extension course. Each `lecture`, in turn, must have detailed `study plan items`, which describe the activities of the lecture. To each lecture, (external) `invited speakers` may be assigned. The latter either may `accept` or `reject` the `invitation`.

After receiving the responses for these `invitations` and creating the `lectures`, the `coordinator` may request an approval for the `extension course project`. First, it must be approved by the `faculty director`. If he wants to `reject` it, he must provide a `reason` for his decision and the 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.

Our PHILharmonicFlows framework allows for the comprehensive support of such scenarios. In particular, it overcomes many limitations of existing PAISs by enabling a tight integration of process and data [7, 8]. The framework supports object-aware processes focusing on the processing of business data and business objects respectively. In this context, *object-awareness* means that the overall process model is structured and divided according to the *object types* involved. These object types are organized in a data model and may refer to other object types or be referenced by them. Moreover, for each object type, a separate process type, defining the corresponding *object behavior*, exists. At runtime, each object type then may comprise a varying number of object instances.

Since the creation of an object instance is directly coupled with the creation of a corresponding process instance, a complex *process structure* emerges. Thereby, process instances referring to object instances of the same type are executed asynchronously to each other as well as asynchronously to process instances related to object instances of different types. However, their execution may have to be synchronized at certain points in time. Overall, we differentiate between *micro* and *macro processes* to capture *object behavior* as well as *object interactions*.

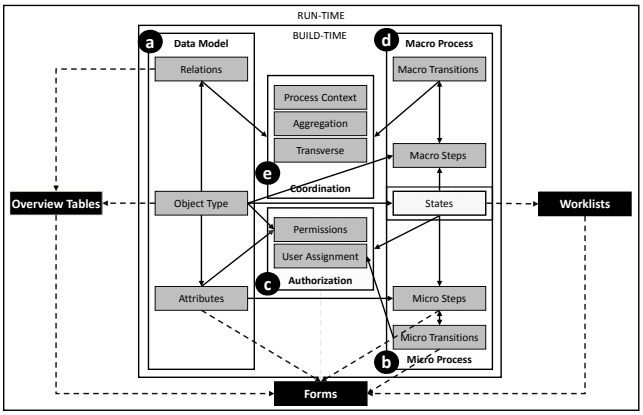

Fig. 1. Overview of the PHILharmonicFlows Framework

Data model (cf. Fig. 1a): A *data model* defines the object types as well as their attributes and relations (including cardinalities).

Example 2 (Data structure): Fig. 2a illustrates the data model relating to Example 1. Object types `lecture` and `decision committee` refer to object type `extension course project`. In turn, object types `invitation` and `study plan item` refer to `lecture`. At run-time, these relations allow for a varying number of interrelated object instances whose processing must be coordinated. Additionally, cardinality constraints restrict the minimum and maximum number of instances of an object type that may reference the same higher-level object instance. Fig. 2b shows a corresponding run-time data structure.

Micro process level (cf. Fig. 1b): To express object behavior, for each object type of a data model, a *micro process type* must be defined. At run-time, the creation of an object instance is directly coupled with the creation of a corresponding *micro process instance*. The latter coordinates the processing of the object instance among different users and specifies the order in which object attributes may be written. For this purpose, a micro process type comprises a number of *micro step types* (cf. Fig 1b), of which each refers to one specific object attribute and describes an atomic action for writing it. At run-time, a

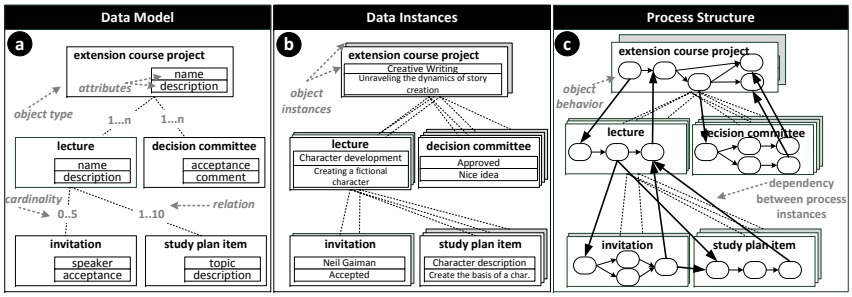

Fig. 2. Data Structure (Data Model and Instances) and Process Structure

micro step is reached if a value is set for the corresponding attribute; i.e., a *data-driven execution* is enabled. Further, micro step types may be inter-connected using *micro transition types* in order to express their default execution order. When using form-based activities, these transition types define the order in which input fields shall be filled (i.e., the internal logic of forms).

To coordinate the processing of individual object instances among different users, micro step types can be aggregated to *state types*. At the instance level, a state may only be left if the values for all attributes associated with the micro steps of the respective state type are set.

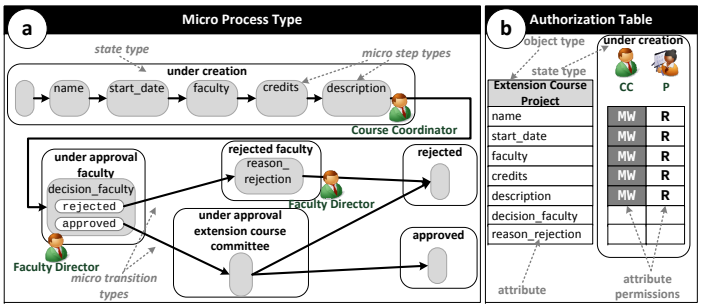

Fig. 3. Micro Process Type and Authorization Table for State "under creation"

Example 3 (Micro process type): Fig. 3a shows the micro process type related to object type `extension course project`. While the extension course project is in state `under creation`, the `course coordinator` may set the attributes to which the corresponding micro step types refer (e.g., `name`, `start_date`, or `description`). Following this, a user decision is made in state `under approval faculty`; i.e., the `faculty director` either approves or rejects the extension course project. If the value of attribute `decision_faculty` corresponds to `rejected`, a value for attribute `reason_rejection` is required.

User authorization (Fig. 1c): User roles are associated with the different states of a micro process type. At run-time, corresponding users must assign required attribute values, as indicated by the micro steps related to the respective state; i.e., a mandatory activity (i.e., a user form) is created and assigned to the user's worklist. To allow for optional activities, for each object type, PHILharmonicFlows additionally generates an *authorization table*. More precisely, it allows granting different permissions to user roles for reading and writing attribute values as well as for creating and deleting object instances (cf. Fig. 1d). Furthermore, permissions may vary depending on the state of an object instance. The framework ensures that each user, who must execute a mandatory activity, owns corresponding write permissions; i.e., data and process authorization are compliant with each other. The initially generated authorization table may be further adjusted by assigning optional permissions to other users. In this context, we differentiate between *mandatory* and *optional write* permissions. Attributes, permissions, and the described micro process logic together provide the foundation for automatically generating *user forms* at run-time. In particular, when taking the currently activated state of the micro process instance into account, the authorization table specifies which input fields can be read or written by the respective user in this state. Opposed to existing PAISs, any alteration directly affecting the forms becomes transparent to the end-user; i.e., the forms do not need to be manually updated.

Example 4 (Authorization Table): In Fig. 3b, in state `under creation` of micro process type `extension course project`, the `course coordinator` (`CC`) has mandatory write (`MW`) permission for attributes `name`, `start_date`, `faculty`, `credits`, and `description`. A `professor` (`P`), in turn, has read permission (`R`) to these attributes in the respective state.

Macro process level (cf. Fig. 1d): At run-time, object instances of the same or different types may be created or deleted at arbitrary points in time; i.e., the data structure dynamically evolves depending on the number of created object instances and the types. In particular, whether subsequent states of micro process instances can be reached may depend on other micro process instances as well; i.e., the processing of an object instance may depend on the processing of a varying number of instances of a related object type. Taking these dependencies among objects into account, a complex *process structure* results (Fig. 2c). To enable proper interaction among the micro process instances, a *coordination* mechanism is required to specify the interaction points of the processes involved. For this purpose, PHILharmonicFlows automatically derives a state-based view for each micro process type. This view is then used for modeling *macro process types* defining the respective *object interactions*. The latter hides most of the complexity of the emerging process structure from users. Each *macro process type* (Fig. 4) consists of *macro step types* and *macro transitions types* connecting them. As opposed to traditional process modeling approaches, where process steps are defined in terms of black-box activities, a macro step type always

refers to an object type together with a corresponding state type; i.e., the latter serve as interface between micro and macro process types.

The activation of a particular macro state might depend on instances of different micro process types. To express this, for each macro step type, a respective *macro input type* has to be defined. The latter can be connected to several incoming macro transitions. At run-time, a macro step is enabled if at least one of its macro inputs becomes activated.

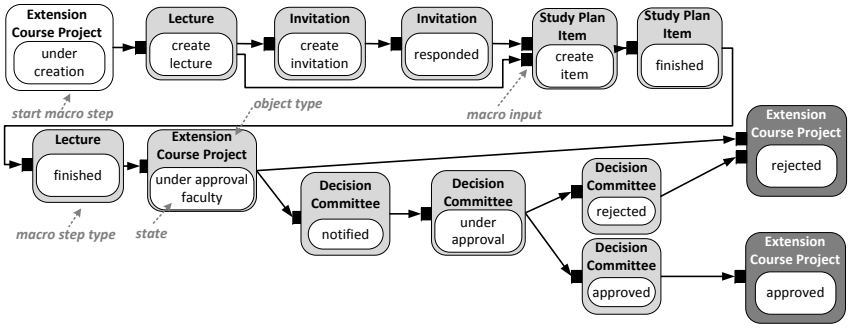

Fig. 4. Example of a Macro Process Type

Coordination (cf. Fig. 1e): To take the dynamically evolving number of object instances as well as the asynchronous execution of corresponding micro process instances into account, for each macro transition, a corresponding *coordination component* needs to be defined. For this purpose, PHILharmonicFlows utilizes object relations from the data model. More details about the coordination components can be found in [7].

3 Challenges

As shown, data structures as well as fine-grained authorization mechanisms are incorporated into the PAIS. Regarding PHILharmonicFlows, not only the schemas of micro and macro process are required to evolve, but the data schemas and authorization settings as well. Another challenge stems from the interdependences among the different components of an object-aware PAIS (i.e., object types, micro process types, macro process type, authorization table, etc.) (cf. Fig. 1). More precisely, changing one of these components might necessitate changes of dependent components. In turn, the latter might trigger cascading changes. This section discusses challenges related to schema evolution in object-aware PAISs. Thereby, both type and instance levels are presented in a single as well as cross perspective. In the former perspective, we discuss which requirements are needed to evolve a particular component. The cross-perspective, in turn, focuses on the challenges regarding secondary changes due to the dependencies among components.

3.1 Schema Evolution at Type Level

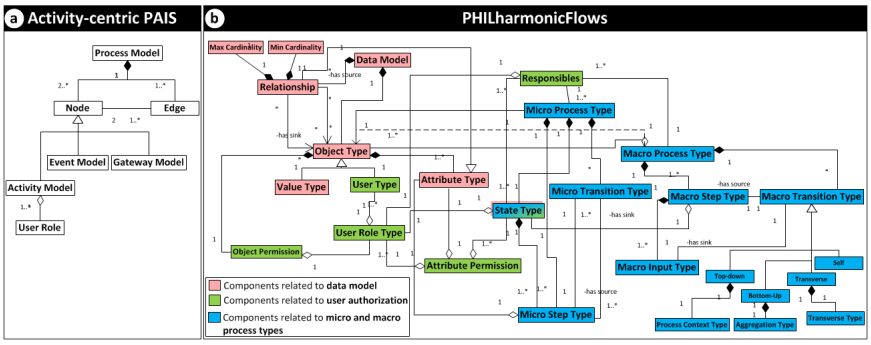

Fig. 5. Metamodel of a) Activity-centric PAIS and b) PHILharmonicFlows

Evolution in single perspective. First, a set of primitive change operations with precise semantics is required for accomplishing changes of each perspective. Such a set must be *complete*, i.e., the change operations comprising this set must allow transforming a valid schema S to any other valid schema S' [15]. Compared to activity-centric PAISs (Fig. 5a), in which a process model comprises activities, events, gateways, and connectors (edges), defining a complete set of change operations for object-aware PAISs causes more efforts due to the vast number of components (Fig. 5b). In particular, we must define change operations for each perspective (e.g., data schema, micro process schema, macro process schema, and user authorization) (cf. Fig. 6). Particularly, for each component, at least one operation for *adding* and *deleting* it must be defined. Note that similar concerns hold for other proposals related to data-centric processes (e.g., artifact-centric [4, 13] or product-based processes [16]).

Data Model	Micro Process Type	Macro Process Type	User Authorization
add object type delete object type add attribute delete attribute add relation delete relation	add micro process type delete micro process type add micro step type delete micro step type add micro transition type (…)	add macro step type delete macro step type add macro transition type delete macro transition type (…)	add attribute permission delete attribute permission add object permission delete object permission (…)

Fig. 6. Set of Primitive Change Operations

Moreover, when applying a change to a schema, schema *correctness* must not be affected; i.e., the changed schema must confirm with a set of correctness constraints. For example, when adding a micro step type in a state type, the latter must not refer to an attribute, if this attribute is already referred by another micro step type having same state type [9].

Example 5 (Change scenario I: add attribute): When the `faculty director approves` the `extension course project`, he may add comments on the extension course in question.

Example 5 refers to object type `extension course project` (cf. Fig. 3b). To be more precise, a new attribute is needed; i.e., the change operation `add attribute` should be applied. However, this change must be in accordance with the correctness constraints set out by PHILharmonicFlows. For example, the attribute must have a unique name (e.g., `approval_remarks`). Besides, only adding a new attribute to the object type is not sufficient. To make the attribute accessible at run-time, a respective attribute permission must be created as well; e.g., the `faculty director` must obtain an *optional write* permission for this attribute.

Evolution in cross-perspective. As shown in Example 5, focusing only on the component, which is primarily changed, is insufficient. Instead, the change of one component may trigger secondary (i.e., cascading) changes in other components.

Example 6 (Change scenario II: delete attribute): The `start date` of the course will not be specified in the `extension course project` anymore.

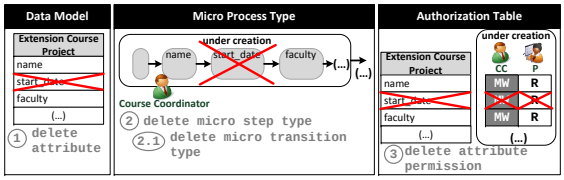

Fig. 7. Change Scenario II - Delete Attribute

In Example 6, operation `delete attribute` is applied to attribute `start_date`. However, to maintain correctness of the data schema, micro and macro process schemas, and user authorization settings, cascading changes are required as well. Fig. 7 illustrates the effects of this attribute deletion. At the micro process type, the micro step type associated with the deleted attribute must be deleted as well. Again, only deleting the micro step type might not be sufficient. In the given example, for instance, the user must be informed about the inconsistency of the micro process type due to the "gap" left between micro step types `name` and `faculty`. In addition, attribute permissions in the authorization table must be deleted as well.

Example 7 (Change scenario III - add object type): When approving the `extension course project` by the `extension course committee`, the members

of the committee may ask `questions` to the `course coordinator`. These `questions` must be answered, before committee members make their decision.

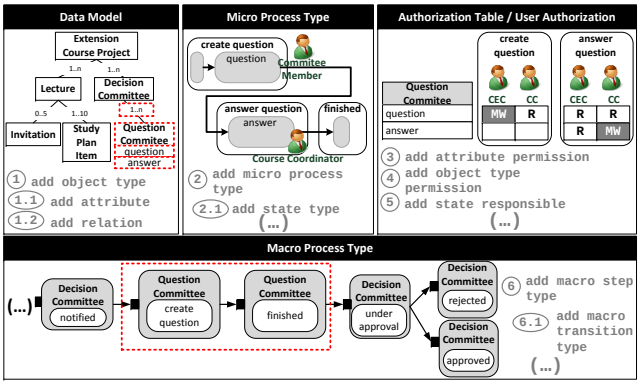

Fig. 8. Change Scenario III - Add Object Type

In Example 7, a new object type `question committee` is added to the data model. Accompanying to this, a micro process type needs to be added as well as respective attributes and user permissions (cf. Fig. 8). In this particular case, new instances of object type `question committee` may only be created when an instance of object type decision committee is initialized; i.e., when the `extension course project` is approved by the `committee` members. Additionally, the micro process instance related to object instance `decision committee` might continue its execution only after finishing all micro process instances of object type `question committee` (i.e., all questions of a committee member are answered). Therefore, new synchronization points must be added to enable the interaction between the two object types; i.e., new macro step types must be created in the macro process type as well. However, the addition of new macro step types is a design choice in the given context; i.e., the engineer may decide to change the macro process type, but this is not essential to maintain correctness of all schemas in the given scenario.

As shown, there are two categories of secondary changes: *mandatory* and *optional* ones. A mandatory secondary change must be applied to maintain correctness of all related schemas. For example, when deleting an attribute, the micro step types referring to it as well as corresponding attribute permissions must be deleted as well. In turn, optional secondary changes refer to design choices made by the user when changing a schema. Hence, mechanisms identifying the impact caused by any schema change become necessary. In particular, these must identify and inform the user about required (i.e., mandatory) secondary changes. To modify the schemas in a controlled manner, an input from the user confirming the schema modification is needed. Therefore, an interface

assisting the user with decision making is required. In addition, such an interface must assist users by displaying optional secondary changes to them. In Example 7, adding new macro step types contribute on examples of optional secondary changes. In turn, the addition of new macro transition types to connect the macro step types are mandatory secondary changes, once they are necessary to avoid inconsistencies at the macro process type.

3.2 Schema Evolution at Instance Level

Evolution in single perspective. When evolving schemas in an object-aware PAIS, we must ensure that no error occurs concerning object instances and corresponding micro and macro process instances. Hence, for each schema (e.g., data schema, micro and macro process schemas, and user authorization settings), different issues arise. For example, when modifying a data schema, the risk of data loss must be taken into account, since missing data might cause several inconsistencies for running processes. For example, when deleting an attribute from an object type, some object instances or micro process instances related to the object type in question may still depend on data related to this attribute, causing inconsistencies at run-time. To avoid respective problems, data relating to the deleted attribute must still be accessible for reading or writing; i.e., even if in the new schema this attribute does not exist, it must be possible that old object and micro process instances refer to a schema where this attribute (and its respective data) still exists. For this, mechanisms for *data schema versioning* must be provided. With them, different versions of the same data schema may co-exist, letting instances that were created before modifying the data schema refer to older schema versions.

A similar problem arises when evolving a micro process schema. If a micro step type or a state type is deleted, there might be inconsistencies in running micro process instances, since they now refer to inexistent micro steps or states. To avoid such inconsistencies, these instances must be able to reach such states or micro steps by referring to the old schema version, but not the new one. However, only maintaining different versions of the schema is not sufficient. The engineer may decide that running micro process instances should be executed according the new schema, if possible. Therefore, a mechanism permitting the *migration* of micro process instances to the a new schema version is needed. However, not all micro process instances can be migrated to the new schema. For example, when deleting a state type, micro process instances for which this state is currently activated must not migrate to the new version. If micro steps relating to the deleted state were already executed, migrating these micro process instances might create inconsistencies regarding their execution. Hence, precise migration and correctness criteria must be established.

Evolution in cross-perspective. As shown in Section 3.1, changing a component triggers a set of secondary changes. These secondary changes must be also taken into account at the instance level. When managing schema versions, for each schema, it becomes necessary that all involved instances (object instances, micro process instances and macro process instances) refer to consistent

schema versions; i.e., the different schema versions must not have inconsistencies like micro process instances referring to missing attributes or macro processes referring to missing micro process states. Regarding Example 6, for instance, when object instances refer to a new data schema version, for which attribute `start_date` no longer exists, the respective micro process instances must refer to that micro process schema version, for which the respective micro step does not exist as well. Otherwise, there will be a schema inconsistency.

4 Related Work

PHILharmonicFlows provides a comprehensive framework for object-aware processes, enabling advanced support for object behavior, object interactions, and data-driven process execution. In [8], we have already shown that traditional approaches (i.e., imperative and declarative process paradigms) do not meet these properties. In literature, a number of approaches enabling data-centric processes are discussed, but they do not consider the aforementioned properties in a comprehensive and integrated way [8, 7]. Moreover, although approaches like artifact-based processes [4, 13] and product-based workflows [16] provide rich capabilities for process modeling, they do not explicitly take runtime issues into account.

Schema evolution in object-oriented databases (OODB) might trigger consistency problems in respect to external applications as well. Frameworks like ORION [2], OTGen [11], and GemStone [14] provide mechanisms for automated database reorganization. Concerning business process evolution, [17] defines a set of change patterns as well as change support features to adequately cope with business process changes. In [15], a formal framework for comprehensive support of process type and process instance changes is defined.

In the context of data-driven processes, [12] describes strategies for adapting data-driven process structures both at design- and run-time. However, changes in the definition of a single data object type (e.g., adding or deleting object attributes) are not considered. Regarding artifact-centric workflows, an approach focusing on dynamically modifiable workflow models is presented in [18]. However, this approach does not focus on artifact modifications.

5 Outlook

Our overall vision is to develop a mechanism enabling schema evolution in object-aware PAIS; i.e., a generic component enabling evolutionary changes in object-aware processes. However, this is a non-trivial task, since object-aware PAISs not only comprise process schemas, but also data and user authorization schemas. These different schemas are tightly integrated, and modifying one of them might require concomitant changes of other schemas. In this paper, we discussed some of the major challenges to be tackled in order to enable schema evolution in object-aware PAISs at both type and instance level. The main challenge is to

cope with concomitant changes of the different schemas; i.e., a schema change of any component might require secondary changes of related schemas to preserve consistency. In future work, we will provide comprehensive solutios to cope with these challenges.

References

1. van der Aalst, W.M.P., Weske, M., Grünbauer, D.: Case Handling: A New Paradigm for Business Process Support. Data & Know. Eng., 53(2), 129–162 (2005)
2. Banerjee, J., Kim, W., Kim, H., Korth, H.F.: Semantics and Implementation of Schema Evolution in Object-oriented Databases. In: Proc. SIGMOD'87, 331–322 (1987)
3. Casati, F., Ceri, S., Pernici, B., Pozzi, G.: Workflow Evolution. In: Proc. ER96, 438–455 (1996)
4. Cohn, D., Hull, R.: Business Artifacts: A Data-centric Approach to Modeling Business Operations and Processes. IEEE Data Engineering Bull., 32(3), 3–9 (2009)
5. Künzle, V., Reichert, M.: Towards Object-aware Process Management Systems: Issues, Challenges, Benefits. In: Proc. BPMDS'09, LNBIP 29, 197–210 (2009)
6. Künzle, V., Reichert, M.: Integrating Users in Object-aware Process Management Systems: Issues and Challenges. In: Proc. BDP'09, LNBIP 43, 29–41 (2009)
7. Künzle, V., Reichert, M.: PHILharmonicFlows: Towards a Framework for Object-aware Process Management. Journal of Software Maintenance and Evolution: Research and Practice, 23(4), 205–244 (2011)
8. Künzle, V., Weber, B., Reichert, M.: Object-aware Business Processes: Fundamental Requirements and their Support in Existing Approaches. Int'l Journal of Information System Modeling and Design, 2(2), 19–46 (2011)
9. Künzle, V., Reichert, M.: A Modeling Paradigm for Integrating Processes and Data at the Micro Level. In: Proc. BPMDS'11, LNBIP 81, 201–215 (2011)
10. Künzle, V., Reichert, M.: Striving for Object-aware Process Support: How Existing Approaches Fit Together. In: Proc. SIMPDA'11 (2011)
11. Lerner, B.S., Habbermann, A.N.: Beyond Schema Evolution to Database Reorganization. In: Proc. OOPSLA/ECOOP'90, 67–76 (1990)
12. Müller, D., Reichert, M., Herbst, J.: A New Paradigm for the Enactment and Dynamic Adaptation of Data-driven Process Structures. In: Proc. CAiSE'08, LNCS 5074, 48–63 (2008)
13. Nigam, A., Caswell, N.S.: Business artifacts: An Approach to Operational Specification. IBM Systems Journal, 42(3), 428–445 (2003)
14. Penney, D.J., Stein, J.: Class Modification in the GemStone Object-oriented DBMS. In: Proc. OOPSLA'87, 111–117 (1987)
15. Rinderle, S., Reichert, M., Dadam, P.: Flexible Support of Team Processes by Adaptive Workflow Systems. Distr. and Par. Databases, 16(1), 91–116 (2004)
16. Vanderfeesten, I., Reijers, H.A., van der Aalst, W.M.P.: Product-Based Workflow Support: Dynamic Workflow Execution. In: Proc. CAiSE'08, LNCS 5074, 571–574 (2008)
17. Weber, B., Reichert, M., Rinderle-Ma, S.: Change Patterns and Change Support Features - Enhancing Flexibility in Process-Aware Information Systems. Data & Know. Eng., 66(3), 438–466 (2008)
18. Xu, W., Su, J., Yan, Z., Yang, J., Zhang, L.: An Artifact-centric Approach to Dynamic Modification of Workflow Execution. In: Proc. OTM'11, 256–273 (2011)