

Enhancing the Case Handling Paradigm to Support Object-aware Processes

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. Despite the widespread adoption of process management systems (PrMS) by industry, there exist numerous processes that cannot be adequately supported by PrMS so far. A common characteristic of these processes, which is usually neglected by traditional activity-centric PrMS, is the role of data as driver for process modeling and enactment. To overcome the limitations caused by missing integration of data and process, several data-centric process management approaches have emerged. A popular one is the Case Handling (CH) paradigm. However, previous case studies pointed out that, although it targets some of the limitations from activity-centric PrMS, the integration of processes and data supported by CH is still unsatisfactory. In this paper, we present the lessons learned from previous case studies and discuss the limitations of CH. We then present the PHILharmonicFlows framework, which enhances the power of data-centric approaches such as CH by enforcing a well-defined modeling methodology governing the object-centric specification and execution of processes and based on a formal operational semantics.

Key words: Case Handling, Data-centric Processes, Object-aware Process Management

1 Introduction

Business process management provides generic methods, tools and techniques for designing, configuring, enacting, and monitoring *business processes* [1]. Existing process management systems (PrMS) are usually *activity-centric*; i.e., processes are defined as a set of “black-box” activities and control flow elements, expressing the order and constraints for executing these activities. However, in these PrMS, business data is typically treated as second-class citizen [2, 3]. Most PrMS only cover atomic data elements, which are needed for control flow routing and as input parameters of process activities. In turn, business objects are usually stored in external databases; i.e., they are outside the control of the activity-centric PrMS. Traditional activity-centric PrMS have been primarily designed for *highly structured, repetitive processes*. By contrast, *knowledge-intensive* processes are often *unstructured* or *semi-structured* [4]; i.e., these processes are driven by user decisions and cannot be *straight-jacketed into activities* [2]. Moreover, such

processes require integrated access to data; i.e., users shall be able to immediately access important information at arbitrary points in time during process execution. Additionally, the execution of knowledge-intensive processes depends on the availability of certain information, but not on the completion of a certain activity (as in activity-centric PrMS). Consequently, they are *data-driven*; i.e., instead of depending on activity completion, the progress of process execution depends on changes of correspondent business objects. Besides, these processes normally depend on data from other process instances from the same or different type. Therefore, a PrMS must provide a mechanism for coordinating the interactions between such interdependent processes.

In several case studies in different domains [5, 6, 7, 8, 9], we learned that the described limitations can be traced back to the missing integration of business data and processes. To overcome at least some of the more severe limitations, there exist several approaches that support a tight integration of process and data [2, 3, 7, 10, 11, 12, 13]. One prominent *data-centric* approach is provided by the *Case Handling paradigm* (CH) [2, 14, 15]. In CH, the central concept is the *case* (e.g., an insurance claim or a job offer), which comprises tasks (i.e., activities), data elements, and relations between the tasks making up the process. Although CH overcomes some of the limitations known from activity-centric PrMS, the paradigm is still not broadly used in practice. To better understand the reasons for this, in several case studies [5, 6, 8] we applied the CH paradigm to existing processes. Thereby, we have observed that CH is limited in respect to *object-awareness*. Although CH permits to associate different types of data elements to a case, which may be considered in tight accordance with an *object*, it neither provides explicit support for complex objects nor the relations between them. More precisely, CH supports *object behavior*; i.e., it allows specifying in which order and by whom the data elements (i.e., *object attributes*) shall be written at runtime. However, CH does not properly take into account the *interaction* among different cases or different instances of the same case type. In this paper, we present the lessons learned in these case studies and discuss some of the fundamental limitations of CH. We further discuss the challenges to be tackled to improve the paradigm in order to provide adequate support for object-aware processes. We then give insights into the PHILharmonicFlows framework, which enhance the CH paradigm by giving adequate support to object-aware processes.

Section 2 provides more details on object-aware processes and their characteristics. In Section 3, we present a job application process as example. Along with this example, we present a set of requirements to be met by a PrMS in order to provide an adequate support. In Section 4, CH is introduced followed by a discussion of how CH meets the requirements. Finally, we sketch how to enhance the CH paradigm (and other data-centric approaches as well) by introducing the PHILharmonicFlows framework in Section 5. Section 6 closes with a summary and an outlook.

2 Object-aware Processes

This section describes the fundamental characteristics of *object-aware processes*. Several require a full integration of process and data. As we learned in case studies in a variety of domains [5, 6, 7, 8, 9], object-aware processes present the following major characteristics:

Object behavior. This characteristic deals with the processing of individual object instances. More precisely, for each object type a separate process definition must be provided. At runtime, the latter is then used for coordinating the processing of individual object instances among different users. In addition, it must be specified in which order and by whom the attributes of a particular object instance shall be (mandatorily) written, and what valid attribute settings (i.e., attribute values) are. Furthermore, when executing activities, the involved object instances need to be in certain *states*. Consequently, for each object type, its behavior should be definable in terms of states and transitions. At runtime, the creation of an *object instance* shall be directly coupled with the creation of its corresponding *process instance*. In this context, it is important to ensure that mandatory data is provided during process execution; i.e., during the processing of object instances. For this reason, object behavior should be defined in terms of *data conditions* rather than based on black-box activities.

Object interactions. The behavior of a particular object must be coordinated with the one of other related objects. The related object instances may be created or deleted at arbitrary point in time, resulting in a *complex data structure*. The latter dynamically evolves during runtime, depending on the types and numbers of created object instances. Further, individual object instances of the same type may be in different processing states at a certain point in time. More precisely, it must be possible to execute individual process instances (of which each corresponds to the processing of a particular object instance) in a loosely coupled manner; i.e., concurrently to each other and synchronizing their execution where needed by taking semantic object relations and cardinality constraints into account.

Data-driven execution. To proceed with the processing of a particular object instance, in a given state, certain attribute values are mandatorily required. Hence, object attribute values reflect the progress of the corresponding process instance. More precisely, the setting of certain object attribute values is enforced in order to progress with the process through the use of *mandatory activities*. However, if required data is already available (e.g., it may be optionally provided by authorized users before the respective mandatory activity becomes enabled), these activities will be automatically skipped when being activated. Furthermore, users shall be able to *re-execute* a particular activity, even if all mandatory object attributes have been already set. For this purpose, data-driven execution must be combined with *explicit user commitments*. Finally, the execution of a mandatory activity may depend on attribute values of related object instances. Thus, the coordination of multiple process instances should be supported in a data-driven way as well.

Flexible activity execution. For creating object instances and changing object attribute values, *form-based activities* can be used. Respective user forms should comprise *input fields* (e.g., text fields or check-boxes) for writing selected attributes and *data fields* for reading attributes of object instances. However, different users might prefer different work practices. Activities should therefore be executable at different levels of granularity; e.g., it should be possible that an activity may relate to one or multiple object process instances.

Integrated access. Authorized users should be able to access and manage process-related data objects at any point of time. More precisely, *permissions* for creating and deleting object instances, as well as for reading and writing their attributes need to be defined. Attribute changes contradicting specified object behavior must be prevented. Which attributes may be written or read by a particular (form-based) activity not only depends on the user invoking this activity, but also on the progress of the corresponding process instance. While certain users must execute an activity mandatorily in the context of a particular object instance, others might be authorized to optionally execute this activity; i.e., a distinction is made between mandatory and optional permissions. Furthermore, for object-aware processes, the selection of actors usually not only depends on the activity to be performed, but also on the object instances processed by this activity. In this context, the relationships between users and object instances must be taken into account.

3 Illustrating Example and Requirements

This section presents an example of an *object-aware process* showing the characteristics sketched in Section 2. Following this, we discuss some of the requirements to be met by a PrMS in order to give adequate support to this process.

3.1 Illustrating Scenario: Recruitment Process

As example we consider a (simplified) scenario for recruiting people as known from human resource management (cf. Fig. 1).

Example 1 (Recruitment Process). In the context of recruitment, **applicants** may apply for **job vacancies** via an Internet online form. Before an **applicant** may send her **application** to the respective company, specific information (e.g., **name**, **e-mail address**, **birthday**, **residence**) must be provided. Once the **application** has been submitted, the responsible **personnel officer** in the human resource department is notified. The overall process goal is to decide which **applicant** shall be **invited** for the interview.

When the **personnel officer** receives a **job application**, he may request **internal reviews** for each **applicant**. The concrete number of **reviews** may differ from **application** to **application**. Corresponding **review** forms have to be filled by **employees** from **functional divisions**. **Employees** make a **proposal** on how to

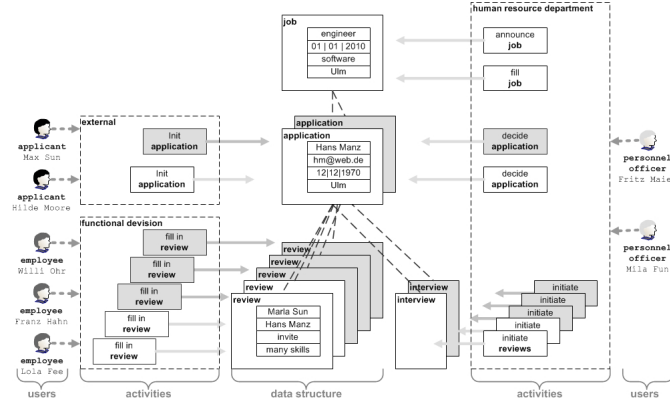


Fig. 1. Example of a recruitment process from the human resource domain

proceed; i.e., they indicate whether the applicant shall be **invited** for an interview or be **rejected**. In the former case an additional **appraisal** is needed. After the **employee** has filled the **review** form, she submits it to the **personnel officer**. Based on the incoming **reviews**, he makes his decision on the **application**; i.e., if there are **reviews** indicating the applicant's interview, the **personnel officer** shall invite the **applicant** for an interview. Otherwise, the application is **rejected**.

3.2 Scenario Requirements

To adequately support such a scenario, any PrMS must meet a set of requirements, which we describe in the following. This will later be followed by a discussion, where we point out which of these requirements are met by CH and which are not.

R1 (Data integration): According to our scenario, the data should be managed in terms of object types comprising object attributes and relations to other object types.

Example R1 (Data integration): For each job, a set of applications may be created. In turn, for each application, several reviews may exist. Thereby, a review comprises attributes like application, employee, remark, proposal and appraisal.

R2 (Flexible access to data): Authorized data access should be enabled at any point in time during process execution; i.e., not only during the execution of a particular activity.

Example R2 (Flexible access to data): The **personnel officer** should be allowed to access an **application** even if no activity is currently contained

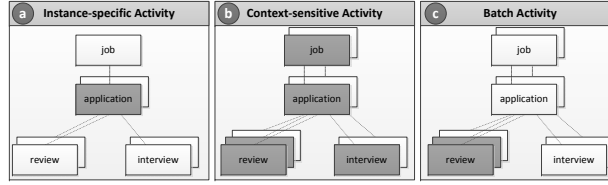


Fig. 2. Types of form-based activities

in his worklist. Furthermore, he should be allowed to update selected attributes from an `application` whenever needed.

R3 (Support of form-based activities and control flow within user forms): A form-based activity comprises a set of *atomic actions*. Each of them corresponds to either an *input field* for writing or a *data field* for reading the value of an object attribute. Which attributes may be written or read in a particular form-based activity may depend on the user invoking this activity and the state of the object instance. In addition, since the writing of particular attributes are mandatory, these forms must signalize which are the corresponding mandatory input fields. However, whether a certain object attribute is mandatory in an activity might depend on the value of other related attributes; i.e., when filling a form, certain attributes might become mandatory on the fly.

Example R3a (Form-based activities): An employee requires a form-based activity to write a review for an `application`. To complete this activity, she must assign values to attributes `remark`, `proposal`, and `appraisal`. In addition, she might access the attributes values of the corresponding `application`.

Example R3b (Control flow within user forms): If an employee chooses to reject a job application, she must provide a `reason` for this; i.e., attribute `rejection reason` becomes mandatory and a value must be set for it.

R4 (Support of variable activity granularity): Due to the tight integration with data, the behavior of the form-based activities might be related to more than one object instance; i.e., some activities might read/write data in more than one object instance. Accordingly, they may be classified as instance-specific, context-specific, and batch activities. *Instance-specific* activities correspond to exactly one object instance (cf. Fig. 2a). When executing it, attributes of that object instance may be read, written or updated using a form. In turn, a *context-sensitive activity* additionally includes form fields corresponding to higher- or lower-level object instances (cf. Fig. 2b). Finally, *batch activities* allow users to change a collection of object instances in one go; i.e., attribute values are assigned to all selected object instances using one single form (cf. Fig. 2c).

Example R4 (Support of variable activity granularity): An `employee` may choose a *context-sensitive activity* to edit a `review`; i.e., to write attributes `proposal` and `appraisal` and to read attributes referring to the respective application. In turn, `personnel officer` may choose a *batch activity* to mark all the other applications as “rejected” when an `applicant` is hired for the job.

R5 (Support of mandatory as well as optional activities): In order to reach process objectives, certain activities must be mandatorily executed for progressing with the control-flow. At the same time, users should be allowed to optionally execute additional activities; e.g., to write certain attributes even if they are not required at the moment.

Example R5a (Mandatory activity): When performing a `review` of a job application, the `employee` must provide a recommendation on whether to invite the job applicant for an interview or reject the job application; i.e., a form-based activity needs to be mandatorily executed.

Example R5b (Optional activity): After a `review` has been requested and performed by an `employee`, the `personnel officer` might want to update the `review` request; e.g., attribute `remark` may be optionally updated even if it has been already set.

R6 (Alignment of process execution with object behavior): It should be possible to determine in which order and by whom object attributes must be (mandatorily) written and what valid attribute value settings are. Consequently, for each object type, its behavior should be definable in terms of states and transitions. In particular, it should be possible to drive process execution based on data and to dynamically react on attribute value changes. Hence, it is crucial to map states to attribute values.

Example R6 (Object behavior): The object `review` can be defined by different states: `initiated` (when the `personnel officer` is setting which job application is going to be reviewed and which `employee` shall perform the review), `under review` (when the `employee` decides whether `reject` or `invite` the job applicant), `rejected` (if the job application is `rejected`), and `invited` (if the `employee` decides to invite the job applicant for an interview). An `employee` may only provide a `review` for a particular job application if the process is currently at state `under review`. The latter is automatically activated as soon as values for attributes `employee` and `application` have been assigned. If he `rejects` the job application (i.e., attribute `proposal` is set as `rejected`), then the attribute `remark` shall instantly become mandatory.

R7 (Support of flexible process execution): The value setting of certain object attributes are mandatory for process execution; i.e., the mandatory activities enforce the value setting of these object attributes as required for progressing with the process. In principle, respective attributes might be written

by executing optional activities as well. If an optional activity is executed before activating a mandatory activity, the latter may be automatically skipped (if other attribute mandatorily set by it have been written before as well).

Example R7 (Flexible process execution): An employee might reject a job application, through an optional activity, while the personnel officer is still setting values for other attributes regarding the review; i.e., the review activity is skipped and will not be shown in the employee's worklist.

R8 (Support of activity re-execution): Users should be allowed to re-execute a particular activity (i.e., to update the referring attributes), even if all mandatory attributes have been already set. To reflect this behavior, users must explicitly commit the completion of the respective activity.

Example R8 (Activity re-execution): An employee may change his proposal (i.e., invite or reject the job applicant) in the context of a review arbitrarily often, as long as he has not explicitly confirmed his decision.

R9 (Enable proper data authorization): To enable access to data at any point in time, permissions for creating and deleting object instances as well as for reading/writing their attributes must be defined. However, attribute changes contradicting to object behavior must be prevented. To achieve this, the progress of the process must be taken into account when granting permissions to change object attributes.

Example R9 (Proper data authorization): An employee must not see the review proposal of other employees. At the same time, she must not update her own proposal after submitting it to the personnel officer.

4 The Case Handling Paradigm

Case Handling (CH) [2, 14, 15] is a paradigm for supporting flexible and knowledge-intensive processes by strongly integrating them with data. This section first summarizes basic CH concepts. This is followed by a discussion on whether or not CH meets the requirements introduced in Section 3.2.

4.1 Basic Case Handling Concepts

The core concept of the CH approach is the *case type*. The latter comprises tasks (i.e., activities), atomic data elements, and a set of precedence relations between the tasks making up a process. Opposed to traditional activity-centric PrMS, the primary driver for progressing with a case (i.e., a process instance) is not the event related to task completion, but the availability of values for the data elements of the case. While an activity-centric process model clearly separates the process from its associated data, CH integrates both in a tighter manner,

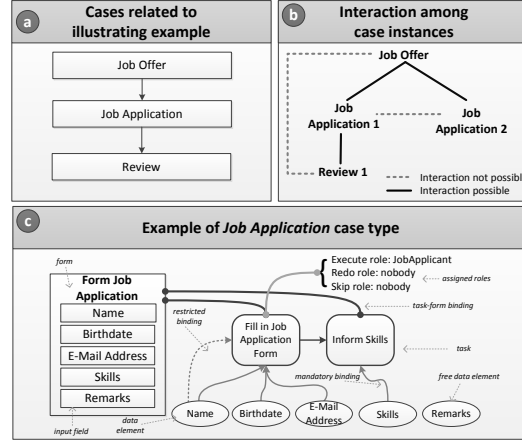


Fig. 3. CH paradigm applied to our example

using produced data not only for routing decisions, but also for determining which parts of the process have already been accomplished. With CH, each *task* may be associated with three sets of *data elements*, each serving a distinct purpose: the first association is between a task and all *data elements* that must be accessible while performing this task. Further, all data elements *mandatory* for a task must be set (i.e., bound to a value) before the task itself is considered to be completed by the CH system. Finally, a data object may have a random number of tasks to which it is *restricted*, meaning that it can only be updated while performing one of these tasks. Interactive tasks are connected to *forms*, each providing access to a selection of data objects. Note that a particular form may be associated with multiple tasks. Finally, it is possible to associate a form to the case itself; i.e., the case and its data elements may be accessed at any point in time using this form.

If a user closes a form after filling out only parts of the mandatory data fields of a task, despite the task not considered as finished, data already entered will still be available to the person who continues working on that task. Such a closely intertwined relation between data and process, however, abandons their often unnatural separation as pursued in traditional PrMS. With the status of data elements being the primary determinant of the case status, this concept overcomes some of the limitations of traditional PrMS:

- Work can now be organized by those performing it with a far higher degree of freedom. Activities may either be performed only partially, without losing intermediary results, or multiple related activities may be handled in one go, surpassing the weakened border between tasks.
- Routing is no longer solely determined by the pre-specified process model. Case types may be designed in such a manner that multiple activities are enabled concurrently, providing different ways of achieving the same goal.

In addition to the *execute role*, specifying the subset of resources allowed to handle a specific task, CH introduces two other roles being crucial for any operational support: the *skip* and *redo* roles. The *skip role* allows users to bypass a selected task. To skip a task, however, all preceding tasks, which have not been completed yet must be skipped (or completed) beforehand. This becomes necessary to ensure an unambiguous state of the process. In turn, the *redo role* enables the user to deliberately roll back the state's case by undoing tasks. In the latter context, the values provided for data objects during the execution of tasks, which are now undone, are not discarded, but merely marked as *unconfirmed*. Hence, they may serve as a default value when re-executing the respective tasks later on. Before a task may be redone, all subsequent tasks that have already been completed must be rolled back as well.

Fig. 3a illustrates an example of how cases and sub-cases can be related. In this example, the case of type *job offer* is related to a varying number of sub-cases of type *job application*. In turn, this sub-case may be related to different *reviews*. Details of a case type are shown in Fig. 3c. Thereby, data elements are associated with forms representing corresponding input fields. In turn, forms are associated with tasks. In our example, the depicted form is associated to both tasks of the case. More precisely, data elements *name*, *birthdate* and *e-mail address* are mandatory for completing task *Fill in job application form*, while data element *skills* is mandatory for task *Inform skills*. In particular, data element *name* may be only filled in the context of task *Fill in job application form*. Data element *remarks*, in turn, is a so-called *free data element*; i.e., it may be accessed at any point in time during case execution by any involved user role.

4.2 How Does Case Handling Meet the Requirements of Object-aware Processes

We now analyze and discuss whether CH meets the requirements of object-aware processes presented in Section 3.2. This analysis is based on previous case studies [5, 6, 8] as well as on hands-on experience with the CH tool BPMone¹. The latter is a commercial tool that implements the concepts of the CH paradigm.

R1 (Data integration): In CH, a *case* may be considered in tight accordance with an *object*; i.e., the data elements related to a case may logically be considered as data attributes of an object. Further, a case may be hierarchically related with *sub-cases*, representing object relationships. However, only direct relations (i.e., case and correspondent sub-cases) are supported. Neither interactions between two instances of the same case nor access to a case's data elements by a corresponding sub-case are allowed (cf. Fig. 3b).

R2 (Flexible access to data): One of the main characteristics of the CH paradigm is its focus on the *entire case*; i.e., all users get full reading access to the whole case when they are executing any activity. In particular, *context*

¹ <http://www.perceptivesoftware.com/products/perceptive-process/business-process-management>

tunneling (i.e., limitation of the user's view to single work items) is avoided, which increases process flexibility. The users involved in one case instance may read all related data elements of the case at any point in time. Additionally, values of *free data elements* may be edited by all users at any point in time during case execution.

R3 (Support of form-based activities and control flow within user forms): Like activity-based modeling approaches, in CH, process steps correspond to activities. Such activities can be implemented in terms of *forms*. Each form is linked with a collection of atomic data elements, which are either mandatory or restricted. An activity will be considered as completed if all mandatory data elements have an assigned value. However, CH does not allow a field (and the corresponding data element) to become mandatory dynamically; i.e., there are no mechanisms that allow defining the internal flow logic within a form. Hence, this requirement is not completely met.

R4 (Support of variable activity granularity): Similar to activity-centric approaches, in the CH paradigm, each task instance is related to exactly one process instance (i.e., instance-specific activities). However, using sub-cases as workaround, context-sensitive activities are partially; i.e., if a case has sub-cases, its tasks may access data elements from its sub-cases. Since CH does not support interactions among different instances of the same case type, batch activities are not directly supported.

R5 (Support of mandatory as well as optional activities): In CH, a task may have associated data elements that must be set before completing the case. Setting these *mandatory* data elements is considered as a mandatory activity. In turn, the definition of optional activities in CH is enabled by the use of free data elements. Since the latter are not relevant for process control, they may be set at any point in time during case execution by any user involved in the case.

R6 (Alignment of process execution with object behavior): In particular, the data elements of a case may be associated with tasks (i.e., activities) and be declared as *mandatory* or *restricted*. *Free* data elements, in turn, may be changed at any point in time by any user involved in the case. Thereby, CH allows capturing of the *object behavior*. In other words, by associating mandatory data elements with tasks from a case, it becomes possible to specify in which order and by whom these data elements (or object attributes) shall be written. Additionally, input fields and transitions may be associated with constraints on attribute values. Finally, it is possible to initialize and terminate a case instance at any point in time.

R7 (Support of flexible process execution): Enabling of a task is driven by data; i.e., tasks are enabled when data becomes available. When all mandatory data elements, related to a particular activity, are set, the subsequent activity becomes enabled. Additionally, tasks may be automatically skipped at runtime if their mandatory data elements have been provided by other tasks before.

R8 (Support of activity re-execution): For each task in CH, separate roles can be defined. More precisely, it is possible to define who shall work on

an activity and who may redo or skip it. The redo role allows actors to execute activities several times. However, the user may re-execute the task only until the completion of the respective case. If the execution of the case is completed, the user must not re-execute the task anymore. Hence, *user commitments* are not considered; i.e., the user cannot indicate to the system that he agrees with the values, set for the data elements, so that the task is then finished in a user-controlled way.

R9 (Enable proper data authorization): In CH, data elements are associated with the tasks in order to define the order in which they shall be set. The user role assigned as *execute* role to a particular activity is the one allowed to write the related data elements. However, the direct association of roles with tasks does not allow declaring such tasks as *optional* for other user roles. Moreover, to avoid *context-tunneling*, any user involved in the case may read information of the *entire case* at any point in time; i.e., data privacy becomes a severe issue.

5 Enhancing the Case Handling Paradigm

The CH paradigm overcomes many of the limitations caused by the missing integration of data and processes. However, as discussed, there still exist requirements not fully met by CH; e.g., concerning flexible activity granularity and data privacy issues. Although CH focus on data as driver for process execution, it does not take object *states* and *transitions* between them into account; i.e., CH does not enable mappings between attribute values and objects states and therefore is unable to ensure compliance between them. To deal with these drawbacks of the CH paradigm and other existing data-centric approaches, we have developed the PHILharmonicFlows framework. Fig. 4 summarizes its main components. Basically, PHILharmonicFlows comprises both *modeling* and *run-time environment* enabling full lifecycle support for object-aware processes. As a fundamental prerequisite, a *data model* describing the respective domain needs to be defined; i.e., object type and relations are defined (cf. Fig. 5a).

The *modeling environment* of PHILharmonicFlows enforces a well-defined modeling methodology governing the definition of processes at two different levels of granularity: *micro* and *macro processes*. A micro process captures the *behavior* of an object (cf. Fig. 6), while the macro process realizes the *objects interactions*. In particular, PHILharmonicFlows supports the interaction and coordination of object instances asynchronously as long this does not violate any semantic dependencies to be considered.

Process and data authorization is based on *user roles*. Data may be accessed optionally and at any point in time. In turn, process execution is based on permissions for creating and deleting object instances as well as for reading or writing their attributes. Furthermore, to enable access at the object attribute level, PHILharmonicFlows maintains a comprehensive *authorization table*. Finally, based on these authorization settings, PHILharmonicFlows automatically generates user forms. Besides the form-based activities, the framework support

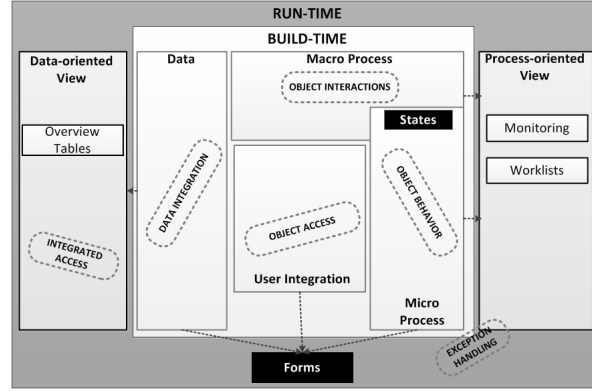


Fig. 4. PHILharmonicFlows

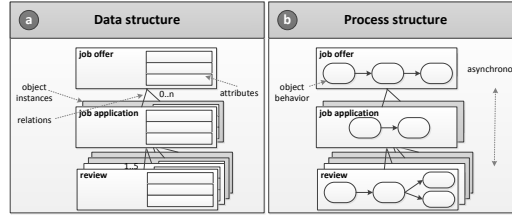


Fig. 5. Data structure and corresponding process structure

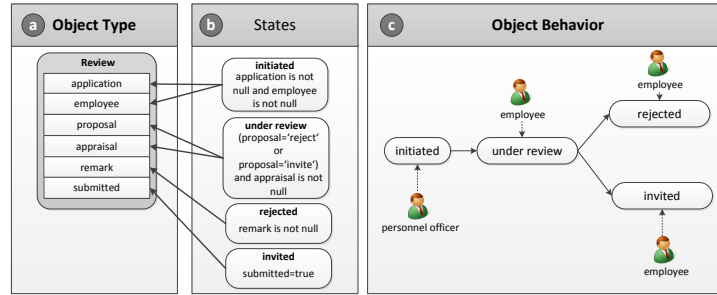


Fig. 6. Object behavior defined based on states and transitions by PHILharmonicFlows

black-box activities as well (i.e., activities that invoke external applications or functions that implement more complex computations).

The *runtime environment* provides *data-* and *process-oriented* views to end-users; i.e., authorized users may invoke activities for accessing data at any point in time as well as activities needed to proceed with the flow of the processes. Moreover, PHILharmonicFlows is based on a well-defined formal semantics, which allows for the automatic generation of end-user components corresponding to the runtime environment (e.g., user worklists and form-based activities).

Due to the lack of space, we do not describe the components of the framework. A more detailed description of the framework as well as its components can be found in [7, 16].

6 Summary & Outlook

In this paper, we discussed the limitations of the CH paradigm regarding the support of object-aware processes. We based this discussion on several case studies, including hands-on experience we gathered when working with the CH system BPMone.

	Case Handling Paradigm	PHILharmonicFlows Framework
+ supported o partially supported - not supported		
R1 (Data integration)	o	+
R2 (Flexible access to data)	+	+
R3 (Support of form-based activities and control flow within user forms)	o	+
R4 (Support of variable activity granularity)	o	+
R5 (Support of mandatory as well as optional activities)	+	+
R6 (Alignment of process execution with object behavior)	+	+
R7 (Support of flexible process execution)	+	+
R8 (Support of activity re-execution)	o	+
R9 (Enable proper data authorization)	-	+

Fig. 7. Comparing the CH paradigm and PHILharmonicFlows

In particular, CH does not enable mappings between attribute values and objects states and therefore is unable to ensure compliance between them. Hence, problems like the limited interaction among case instances and lack of data privacy make the CH paradigm unsuitable for object-aware processes. We further presented the PHILharmonicFlows framework. Its objective is to enhance the power of CH paradigm and other data-centric approaches to enable proper support of such processes. This approach has been applied in several case studies comprising different domains (e.g., human resources management, healthcare, and automotive industry). As proof-of-concept, a prototype has been developed, enabling the modeling and enactment of object-aware processes. Finally, Fig. 7 shows a compilation of the requirements of object-aware processes and how well both approaches (i.e., CH and PHILharmonicFlows) meet them.

Acknowledgements

The authors would like to acknowledge the financial support provided by the Ernst Wilken Foundation.

References

- [1] Reichert, M., Weber, B.: Enabling Flexibility in Process-aware Information Systems: Challenges, Methods, Technologies. Springer (2012)
- [2] van der Aalst, W.M.P., Weske, M., Grünbauer, D.: Case handling: A new paradigm for business process support. *Data & Knowledge Engineering* **53**(2) (2005) 129–162
- [3] Cohn, D., Hull, R.: Business artifacts: A data-centric approach to modeling business operation and processes. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering* **32**(3) (2009) 3–9
- [4] Silver, B.: Case management: Addressing unique bpm requirements. Technical report, BPMS Watch (2009)
- [5] Künzle, V., Reichert, M.: Towards object-aware process management systems: Issues, challenges, benefits. In: *Proc. BPMDS'09*. (2009) 197–210
- [6] Künzle, V., Reichert, M.: Integrating users in object-aware process management systems: Issues and challenges. In: *Proc. BPM'09 Workshops*. (2009) 29–41
- [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) (2011) 205–244
- [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) (2011) 19–46
- [9] Chiao, C.M., Künzle, V., Reichert, M.: Towards object-aware process support in healthcare information systems. In: *Proc. eTELEMED 2012*. (2012) 227–236
- [10] Bhattacharya, K., Hull, R., Su, J.: A data-centric design methodology for business processes. In: *Handbook of Research on Business Process Management*. (2009) 503–531
- [11] Liu, R., Bhattacharya, K., Wu, F.Y.: Modeling business contexture and behavior using business artifacts. In: *Proc. CAiSE'07*. (2007) 324–339
- [12] Müller, D., Reichert, M., Herbst, J.: Data-driven modeling and coordination of large process structure. In: *Proc. CoopIS'07*. (2007) 131–149
- [13] Vanderfeesten, I., Reijers, H.A., van der Aalst, W.M.P.: Product-based workflow support: Dynamic workflow execution. In: *Proc. CAiSE'08*. (2008) 571–574
- [14] Guenther, C., Reichert, M., van der Aalst, W.M.P.: Supporting flexible processes with adaptive workflow and case handling. In: *Proc. ProGility'08*. (2008) 229–234
- [15] Mutschler, B., Weber, B., Reichert, M.: Workflow management versus case handling: results from a controlled software experiment. In: *Proc. SAC'08*. (2008) 82–89
- [16] Künzle, V.: Object-aware Process Management. PhD thesis, Ulm University, Germany (2013)