Process and Data: Two Sides of the Same Coin?

Manfred Reichert

Institute of Databases and Information Systems, Ulm University, Germany
manfred.reichert@uni-ulm.de

Abstract. Companies increasingly adopt process management technology which offers promising perspectives for realizing flexible information systems. However, there still exist numerous process scenarios not adequately covered by contemporary information systems. One major reason for this deficiency is the insufficient understanding of the inherent relationships existing between business processes on the one side and business data on the other. Consequently, these two perspectives are not well integrated in many existing process management systems. This paper emphasizes the need for both object- and process-awareness in future information systems, and illustrates it along several examples. Especially, the relation between these two fundamental perspectives will be discussed, and the role of business objects and data as drivers for both process modeling and process enactment be emphasized. In general, any business process support should consider object behavior as well as object interactions, and therefore be based on two levels of granularity. In addition, data-driven process execution and integrated user access to processes and data are needed. Besides giving insights into these fundamental characteristics, an advanced framework supporting them in an integrated manner will be presented and its application to real-world process scenarios be shown. Overall, a holistic and generic framework integrating processes, data, and users will contribute to overcome many of the limitations of existing process management technology.

1 Introduction

Despite the widespread adoption of process management systems [1] there exist many business processes not adequately supported by these systems. In this context, different authors state that many deficiencies of contemporary process management systems (PrMS) can be traced back to the missing integration of processes and data [2–11]. Although processes and data seem to be closely related, a unified understanding of the inherent relationships existing between them is still missing.

In the PHILharmonicFlows project, we analyzed numerous business processes from different domains which require a tight data integration [12–15]. We learned that many of these processes are *data-driven* and that their support requires *object-awareness*; i.e., the progress of these processes depends on the processing of certain business data represented through *business objects*. Objects comprise a set of *object attributes* and are *inter-related*. In this context, business processes

R. Meersman et al. (Eds.): OTM 2012, Part I, LNCS 7565, pp. 2–19, 2012.

coordinate the processing of business objects among different users enabling them to cooperate and communicate with each other. Most existing PrMS, however, mainly focus on business functions and their flow of control, whereas business objects are "unknown" to them. As a consequence, most PrMS only cover simple data elements needed for control flow routing and for supplying input parameters of activities with values. In turn, business objects are usually stored in external databases and are outside the control of the PrMS. Hence, existing PrMS are unable to adequately support *object-aware processes* [16].

This paper shows that process and data are actually two sides of the same coin. Section 2 introduces the main characteristics of data-driven and object-aware processes, which we gathered in a number of case studies [12, 13, 17] (see [18] for details about the research methodology applied). Following this, Section 3 sketches core components of our PHILharmonicFlows framework, which enables comprehensive support of data-driven and object-aware processes. Section 4 discusses related work and Section 5 concludes the paper with a summary.

2 Data-Driven and Object-Aware Processes

We first discuss fundamental characteristics of data-driven and object-aware processes, we derived from a more detailed property list related to process modeling, execution, and monitoring. The respective properties were discovered in an extensive analysis of processes currently not adequately supported by process management technology [12–14, 16]. To ensure that the processes we considered are not "self-made" examples, but constitute real-world processes of high practical relevance, we further analyzed processes implemented in existing business applications. Further, we have deep insights into the code and process logic of these applications. To justify our findings, we complemented our process analyses by an extensive literature study ensuring relevance and completeness.

2.1 Application Example

We discuss the characteristics of data-driven and object-aware processes along a simple job recruitment scenario (cf. Fig. 1).

Example 1 (Recruitment). *In recruitment,* `applicants` *may apply for* `job vacancies` *via an Internet online form. Once an* `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 get the* `job`*. If an* `application` *is ineligible the* `applicant` *is immediately rejected. Otherwise,* `personnel officers` *may request internal* `reviews` *for each* `applicant`*. In this context, the concrete number of* `reviews` *may differ from* `application` *to* `application`*. Corresponding* `review` *forms have to be filled by* `employees` *from functional divisions. They make a* `proposal` *on how to 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 back to the* `personnel officer`*. In the meanwhile, additional* `applications` *might have arrived; i.e.,* `reviews` *relating to the same or to different* `applications` *may be requested or submitted at different points in time. The processing of the* `application`*, however, proceeds while corresponding* `reviews` *are created; e.g., the* `personnel officer` *may check the* `CV` *and study the* `cover letter` *of the* `application`*. Based on the incoming* `reviews` *he makes his* `decision` *on the* `application` *or initiates further steps (e.g.,* `interviews` *or additional* `reviews`*). Finally, he does not have to wait for the arrival of all* `reviews`*; e.g., if a particular* `employee` *suggests hiring the* `applicant` *he can immediately follow this recommendation.*

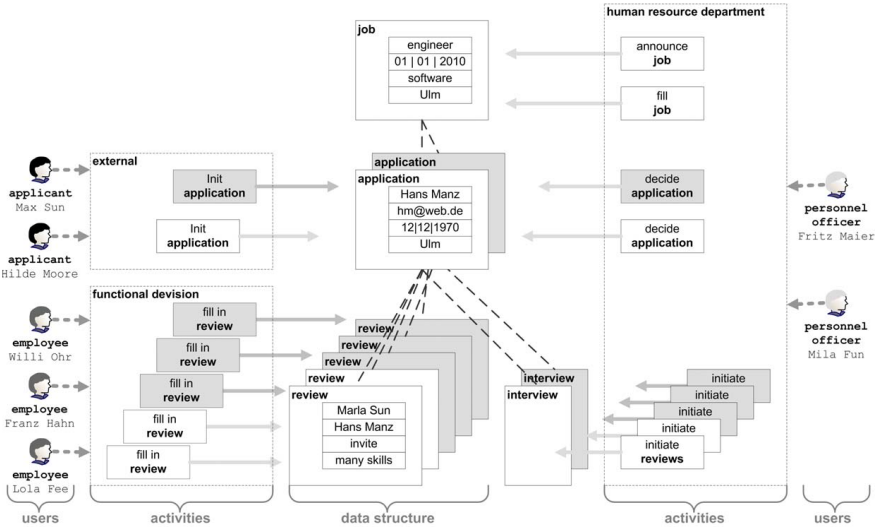

Fig. 1. Example of a recruitment process from the human resource domain

2.2 Basic Characteristics

Basically, data must be manageable in terms of *object types* comprising *object attributes* and *relations* to other object types (cf. Fig. 2a). At run-time, the different object types comprise a varying number of inter-related object instances, whereby the concrete instance number should be restrictable by lower and upper cardinality bounds (cf. Fig. 2b). For each application, for example, at least one and at most five reviews must be initiated. While for one application two reviews are are available, another one may comprise three reviews (cf. Fig. 1).

In accordance to data modeling, the modeling and execution of processes can be based on two levels of granularity: *object behavior* and *object interactions*.

Object Behavior. To cover the processing of individual object instances, the first level of process granularity concerns *object behavior*. More precisely, for

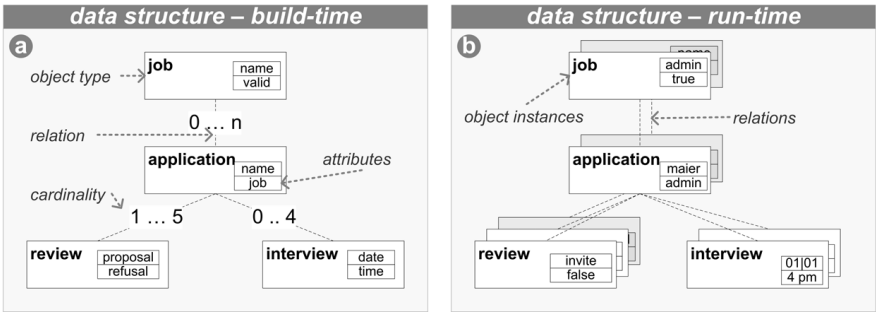

Fig. 2. Data structure at build-time and at run-time

each object type a separate process definition should be provided (cf. Fig. 3a), which can be used for coordinating the processing of an individual object instance among different users. In addition, it should be possible to determine in which order and by whom the attributes of a particular object instance have to be (mandatorily) written, and what valid attribute values are. At run-time, the creation of an object instance is directly coupled with the creation of its corresponding process instance. In this context, it is important to ensure that mandatorily required data is provided during process execution. For this reason, object behavior should be defined in terms of *data conditions* rather than based on black-box activities.

Example 2 (Object behavior). *For requesting a* `review` *the responsible* `personnel officer` *has to mandatorily provide values for object attributes* `return date` *and* `questionnaire`*. Following this, the* `employee` *being responsible for the* `review` *has to mandatorily assign a value to object attribute* `proposal`*.*

Object Interactions. Since related object instances may be created or deleted at arbitrary points in time, a complex data structure emerges, which dynamically evolves depending on the types and number of created object instances. In addition, individual object instances (of the same type) may be in different processing states at a certain point in time.

Taking the behavior of individual object instances into account, we obtain a *complex process structure* in correspondence to the given data structure (cf. Fig. 3a). In this context, the second level of process granularity comprises the *interactions* that take place between different object instances. More precisely, it must be possible to execute individual process instances (of which each corresponds to a particular object instance) in a loosely coupled manner, i.e., concurrently to each other and synchronizing their execution where needed. First, it should be possible to make the creation of a particular object instance dependent on the progress of related object instances (*creation dependency*). Second, several object instances of the same object type may be related to one and the same object instance. Hence, it should be possible to aggregate information; amongst

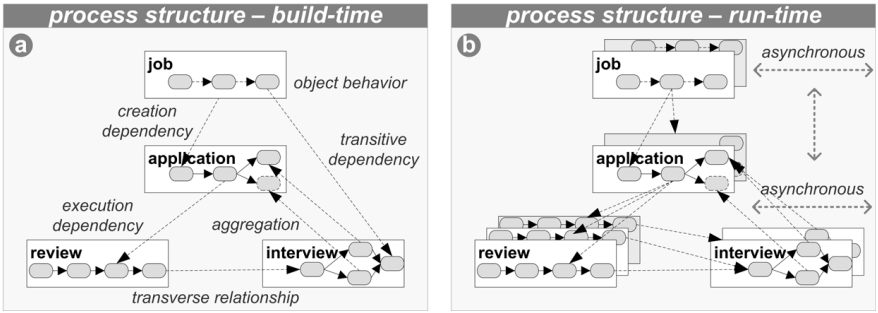

Fig. 3. Process structure at build-time and at run-time

others, this requires the aggregation of attribute values from related object instances (*aggregation*). Third, the executions of different process instances may be mutually dependent; i.e., whether or not an object instance can be further processed may depend on the processing progress of other object instances (*execution dependency*). In this context, interactions must also consider *transitive dependencies* (e.g., reviews depend on the respective job offer) as well as *transverse* ones (e.g., the creation of an interview may depend on the proposal made in a review) between object instances (cf. Fig. 3).

Example 3 (Object interactions). *A* `personnel officer` *must not initiate any* `review` *as long as the corresponding* `application` *has not been finally submitted by the* `applicant` *(creation dependency). Furthermore, individual* `review` *process instances are executed concurrently to each other as well as concurrently to the* `application` *process instances; e.g., the* `personnel officer` *may read and change the* `application`*, while the corresponding* `reviews` *are processed. Further,* `reviews` *belonging to a particular* `application` *can be initiated and submitted at different points in time. Besides this, a* `personnel officer` *should be able to access information about submitted* `reviews` *(aggregative information); i.e., if an* `employee` *submits her* `review` *recommending to invite the* `applicant` *for an* `interview`*, the* `personnel officer` *needs this information immediately. Opposed to this, when proposing rejection of the* `applicant`*, the* `personnel officer` *should only be informed after all initiated* `reviews` *have been submitted. Finally, if the* `personnel officer` *decides to hire one of the* `applicants`*, all others must be rejected (execution dependency). These dependencies do not necessarily coincide with the object relations. As example consider* `reviews` *and* `interviews` *corresponding to the same* `application`*; i.e., an* `interview` *may only be conducted if an* `employee` *proposes to invite the* `applicant` *during the execution of a* `review` *process instance.*

Data-Driven Execution. In order to proceed with the processing of a particular object instance, usually, in a given state certain *attribute values are mandatorily required*. Thus, object attribute values reflect the progress of the corresponding process instance. In particular, the activation of an activity does

not directly depend on the completion of other activities, but on the values set for object attributes. More precisely, *mandatory activities* enforce the setting of certain object attribute values in order to progress with the process. If required data is already available, however, mandatory activities can be *automatically skipped* when being activated. In principle, it should be possible to *set respective attributes also up front*; i.e., before the mandatory activity normally writing this attribute becomes activated. However, users should be allowed 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* (i.e., activity-centred aspects). Finally, the execution of a mandatory activity may also depend on available attribute values of related object instances. Thus, coordination of process instances must be supported in a data-driven way as well.

Example 4 (Data-driven execution). *During a* `review` *request the* `personnel officer` *must mandatorily set a* `return date`*. If a value for the latter is available, a mandatory activity for filling in the* `review` *form is assigned to the responsible* `employee`*. Here, in turn, a value for attribute* `proposal` *is mandatorily required. However, even if the* `personnel officer` *has not completed his* `review` *request yet (i.e., no value for attribute* `return data` *is available), the* `employee` *may optionally edit certain attributes of the* `review` *(e.g., the* `proposal`*). If a value of attribute* `proposal` *is already available when the* `personnel officer` *finishes the request, the mandatory activity for providing the* `review` *is automatically skipped. Opposed to this, an* `employee` *may change his* `proposal` *arbitrarily often until he explicitly agrees to submit the* `review` *to the* `personnel officer`*. Finally, the* `personnel officer` *makes his decision (e.g., whether to reject or to accept the* `applicant`*) based on the incoming* `reviews`*.*

Variable Activity Granularity. For creating object instances and changing object attribute values, *form-based activities* are required. Respective user forms comprise *input fields* (e.g., text-fields or checkboxes) for writing and *data fields* for reading selected attributes of object instances. In this context, however, different users may prefer different work practices. In particular, using *instance-specific activities* (cf. Fig. 5a), all input fields and data fields refer to attributes of one particular object instance, whereas *context-sensitive activities* (cf. Fig. 5b) comprise fields referring to different, but related object instances (of potentially different type). When initiating a review, for example, it is additionally possible to edit the attribute values of the corresponding application. Finally, *batch activities* involve several object instances of the same type (cf. Fig. 5c). Here, the values of the different input fields are assigned to all involved object instances in one go. This enables a personnel officer, for example, to reject a number of application in one go. Depending on their preference, users should be able to freely choose the most suitable activity type for achieving a particular goal. In addition to form-based activities, it must be possible to integrate *black-box activities*. The latter enable complex computations as well as the integration of advanced functionalities (e.g., provided by web services).

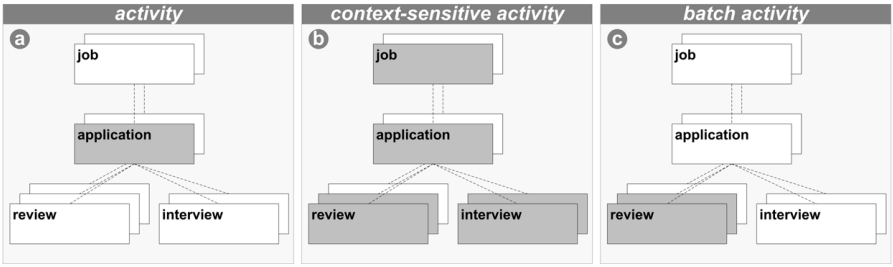

Fig. 4. Different kinds of activities

Moreover, whether certain object attributes are mandatory when processing a particular activity might depend on other object attribute values as well; i.e., when filling a form certain attributes might become mandatory on-the-fly. Such *control flows being specific to a particular form* should be also considered.

Example 5 (Activity Execution). *When an* `employee` *fills in a* `review`*, additional information about the corresponding* `application` *should be provided; i.e., attributes belonging to the* `application` *for which the* `review` *is requested. For filling in the* `review` *form, a value for attribute* `proposal` *has to be assigned. If the* `employee` *proposes to invite the* `applicant`*, additional object attributes will become mandatory; e.g., then he has to set attribute* `appraisal` *as well. This is not required if he assigns value reject to attribute* `proposal`*. Further, when a* `personnel officer` *edits an* `application`*, all corresponding* `reviews` *should be visible. Finally, as soon as an* `applicant` *is hired for a* `job`*, for all other* `applications` *value reject should be assignable to attribute* `decision` *by filling one form.*

Integrated Access. To proceed with the control flow, *mandatory activities* must be executed by responsible users in order to provide required attribute values. Other attribute values, however, may be optionally set. Moreover, users who are usually not involved in process execution should be allowed to optionally execute selected activities. In addition to a *process-oriented view* (e.g. work lists), a *data-oriented view* should be provided enabling users to access and manage data at any point in time. For this purpose, we need to define permissions for creating and deleting object instances as well as for reading/writing their attributes. However, attribute changes contradicting to specified object behavior should be prevented. Which attributes may be (mandatorily or optionally) 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 instances. 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., *mandatory and optional permissions* should be distinguishable. Moreover, for object-aware processes, the selection of potential actors should not only depend on the activity itself, but also on the object instances processed by this

activity. In this context, it is important to take the relationships between users and object instances into account.

Example 6 (Integrated Access). *A* `personnel officer` *may only decide on* `applications` *for which the name of the* `applicants` *starts with a letter between 'A' and 'L', while another* `officer` *may decide on* `applicants` *whose name starts with a letter between 'M' und 'Z'. An* `employee` *must mandatorily write attribute* `proposal` *when filling in a* `review`*. However, her* `manager` *may optionally set this attribute as well. The mandatory activity for filling the* `review` *form, in turn, should be only assigned to the* `employee`*. After submitting her* `review`*, the* `employee` *still may change her* `comment`*. In this context, it must be ensured that the* `employee` *can only access* `reviews` *she submitted before. However, attribute* `proposal`*, in turn, must not be changed anymore. The* `personnel officer` *might have already performed the proposed action.*

3 A Framework Enabling Data-Driven and Object-Aware Processes

In the PHILharmonicFlows project, we developed a framework that enables the characteristic properties of data-driven and object-aware processes and hence contributes to overcome many of the limitations of existing process management technology. The PHILharmonicFlows framework will be described in this section using another illustrating example. In particular, the framework provides advanced concepts and components enabling comprehensive support for data-driven and object-aware processes.

3.1 Illustrating Example

We first present another example of an object-aware process along which we will illustrate basic concepts of the PHILharmonicFlows framework.

The selected 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. In order to propose a new 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 7 (Extension course proposal). *The* `course coordinator` *creates a new* `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. For each lecture, there may be some (external)* `invited speakers`*. The latter either may* `accept` *or* `reject` *the* `invitation`*. After receiving the responses for the* `invitations` *and creating the* `lectures`*, the* `coordinator` *may request the approval of 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` *for evaluation. If there are more* `rejections` *than* `approvals`*, the* `extension course project` *is* `rejected`*. Otherwise, it is* `approved` *and hence may take place.*

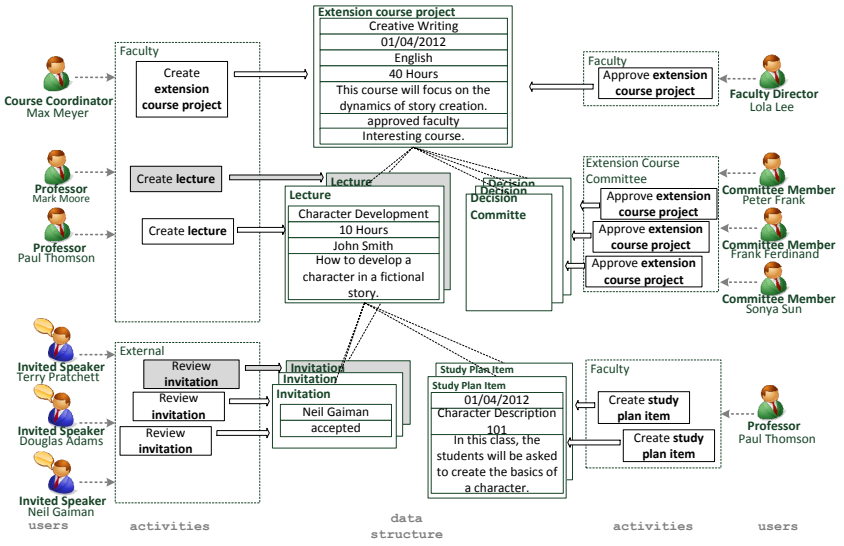

Fig. 5. Extensions course proposal

3.2 Selected Components of the PHILharmonicFlows Framework

The PHILharmonicFlows framework supports object-aware processes focusing on the processing of business data and business objects respectively. More precisely, *object-awareness* means that the overall process model is structured and divided according to the *object types* involved. In turn, these object types are organized in a data model and may be related to other object types. Moreover, for each object type, a separate process type defining its *object behavior* exists. At run-time, 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 objects of different types. However, their execution may have to be synchronized at certain points in time. Overall, PHILharmonicFlows differentiates between *micro* and *macro processes* which allow capturing both *object behavior* and *object interactions*. Furthermore, the execution of micro and macro processes is data-driven and integrated access to processes and data objects is enabled. Finally, different kinds of activity granularities are supported.

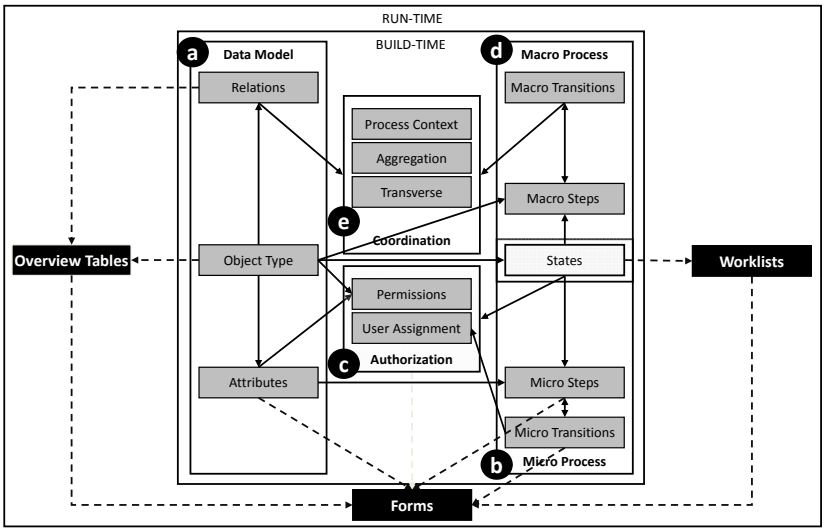

Fig. 6. Overview of the PHILharmonicFlows framework

Data Model. A *data model* defines the object types as well as their corresponding attributes and relations with cardinalities (cf. Fig. 6a).

Example 8 (Data structure). *Fig. 7a illustrates the data model relating to our example from Section 3.1. 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 inter-related object instances whose processing must be coordinated. Further, cardinality constraints restrict the minimum and maximum number of instances of an object type that may reference the same higher-level object instance. Fig. 7b shows a corresponding data structure at run-time.*

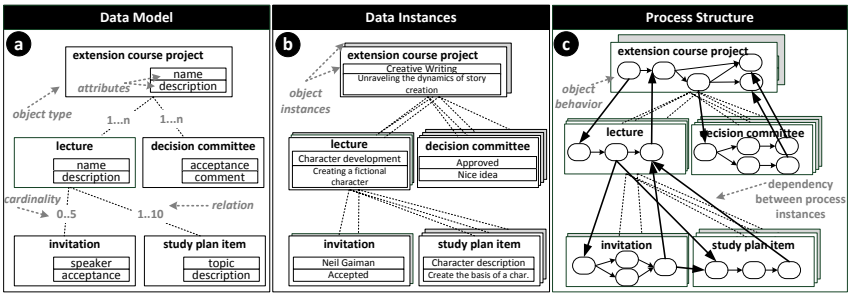

Fig. 7. Data structure (data model and instances) and process structure

Micro Processes. To express *object behavior*, for each object type of a data model, a *micro process type* must be defined (cf. Fig. 6b). At run-time, the creation of object instances is then directly coupled with the creation of a corresponding *micro process instances*. 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 6b), of which each refers to one specific object attribute and describes an atomic action for writing it. At run-time, a micro step is reached if a value is set for the corresponding attribute; i.e., a *data-driven execution* is enabled. Micro step types may be inter-connected using *micro transition types* in order to express their default execution order. When using form-based activities, micro transitions define the order in which the input fields of the respective form shall be filled (i.e., the internal processing logic of the form). Finally, to coordinate the processing of individual object instances among different users, several micro step types can be grouped into *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.

Example 9 (Micro process type). *Fig. 8a shows the micro process type of 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` *is* `rejected`*, a value for attribute* `reason_rejection` *is required.*

User Authorization. PHILharmonicFlows provides advanced support for user authorization while enabling an integrated access to process and data (cf. Fig. 6c). User roles are associated with the different states of a micro process type.

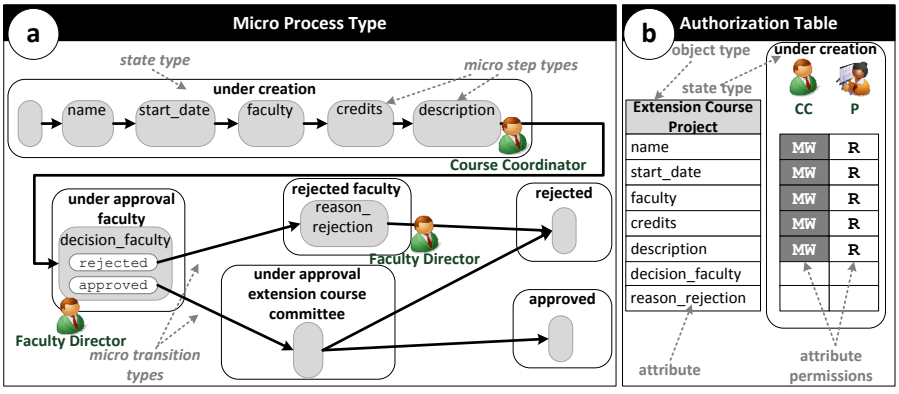

Fig. 8. Micro process type and authorization table for state "under creation"

At run-time, users owning the respective role then must set required attribute values as indicated by the micro steps corresponding to the respective state; i.e., a mandatory activity (i.e., a user form) is created and assigned to the user's work list. To enable optional activities, in addition, PHILharmonicFlows generates an *authorization table* for each object type. More precisely, the framework allows granting different permissions for reading and writing attribute values as well as for creating and deleting object instances to different user roles (cf. Fig. 6d). Furthermore, permissions may vary depending on the state of an object instance. The framework ensures that each user who must execute a mandatory activity also 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 also provide the basis 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. Hence, any change directly affecting directly the forms will be transparent to the end-user; i.e., the forms do not need to be manually updated as in existing process-aware information systems.

Example 10 (Authorization table). *In Fig. 8b, 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`*. In turn, a* `professor` *(P) is authorized to read (R) these attributes in the respective state.*

Macro Process Level. At run-time, object instances of the same and different types may be created or deleted at arbitrary points in time; i.e., the data structure dynamically evolves depending on the types and number of created object instances. 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 variable number of instances of a related object type. Taking these dependencies among objects into account, a complex *process structure* results (cf. Fig. 7c). To enable proper interaction among the micro process instances, a *coordination* mechanism is required to specify the interaction points of the involved processes. For this purpose, PHILharmonicFlows automatically derives a state-based view for each micro process type. This view is then used for modeling so-called *macro process types* defining the respective *object interactions* (cf. Fig. 6d). The latter hides the complexity of emerging process structure from users. Each *macro process type* (cf. Fig. 9) 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, a respective *macro input type* has to be defined for each macro step types. 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.

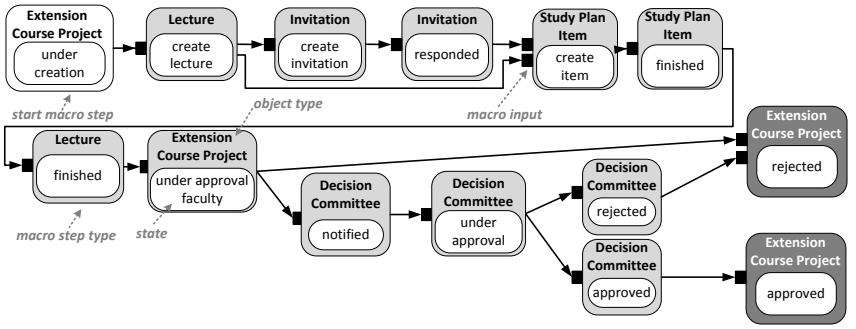

Fig. 9. Example of a Macro Process Type

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 (cf. Fig. 6e). For this purpose, PHILharmonicFlows utilizes object relations from the data model; i.e., takes the relations between the object type of a source macro step type and the one of a target macro step type into account. For this purpose, the framework organizes the data model into different *data levels*. All object types not referring to any other object type are placed on the top level. Generally, any other object type is always assigned to a lower data level as the object type it references. Based on this, PHILharmonicFlows can automatically classify the macro transitions either as *top-down* or as *bottom-up* (cf. Fig. 10a). If the object types of the source and sink macro step types refer to a common higher-level object type, the macro transition is categorized as *transverse*.

For each of these categories of macro transition type, a particular coordination component is required. A *top-down transition* characterizes the interaction from an upper-level object type to a lower-level one. Here, the execution of a varying number of micro process instances depends on one higher-level micro process instance. In this context, a so-called *process context type* must be assigned to the respective macro transition type. Due to lack of space, we do not go into details. We also do not discuss transverse macro transition types here. In turn, a *bottom-up transition* characterizes an interaction from a lower-level object type to an upper-level one. In this case, the execution of one higher-level micro process instance depends on the execution of several lower-level micro process instances of the same type. For this reason, each bottom-up transition requires an *aggregation component* for coordination. To manage the total number of lower-level micro process instances related to the dependent upper-level micro process

instance, PHILharmonicFlows provides *counters*; i.e., the latter permit to control the number of micro process instances that have reached the respective state as well as the ones that have not yet reached the state and the ones that have skipped the same state. These counters can be used for defining aggregation conditions enabling the higher-level micro process instance to activate the state.

Example 11 (Aggregation component). *Fig. 10b shows an aggregation condition (`#IN < #ALL/2`) expressing that the `extension course project` will be `approved` if more than half of the `committee members approve` the project. In this example, there are three micro process instances of `decision committee` related to one instance of `extension course project`. The counter of this example indicates that two of the running instances of `decision committee` have already reached state `approved` (`#IN = 2`), while one instance has not yet reached this state (`#BEFORE=1`). In this case, the condition is already fulfilled and the upper-level micro process instance may continue its execution.*

4 Related Work

In [16], we have shown why existing imperative, declarative, and data-driven (i.e., Case Handling [2, 19, 20]) process support paradigms are unable to adequately support object-aware processes. However, to enable consistency between process and object states, extensions of these approaches based on object life cycles have been proposed. These extensions include object life cycle compliance [21], object-centric process models [8, 9], business artifacts [4, 22], data-driven process coordination [5, 23], and product-based workflows [6, 24]. However, none of these approaches explicitly maps states to object attribute values. Consequently, if certain pre-conditions cannot be met during run-time, it is not possible to dynamically react to this. In addition, generic form logic is not provided in a

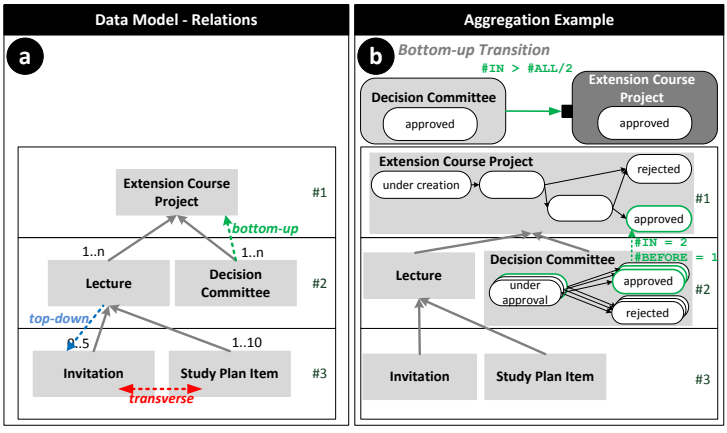

Fig. 10. a) Kinds of relations between object Types; b) Aggregation example

flexible way; i.e., there is no automatic generation of forms taking the individual attribute permissions of a user as well as the progress of the corresponding process into account. Finally, opposed to these approaches, PHILharmonicFlows captures the internal logic of an activity as well.

As illustrated in Fig. 11, each characteristic from Section 2 is addressed by at least one existing approach. Although the mentioned approaches have deficiencies (see footnotes in Fig. 11), they can be considered as pioneering work towards data-driven and object-aware process support. As opposed to PHILharmonicFlows, however, none of them covers all characteristics in a comprehensive and integrated way. Also note that Fig. 11 does not make a difference between process modeling and process execution. Though some approaches (e.g., the business artifacts framework [4]) provide rich capabilities for process modeling, they do not cover run-time issues (or at least do not treat them explicitly).

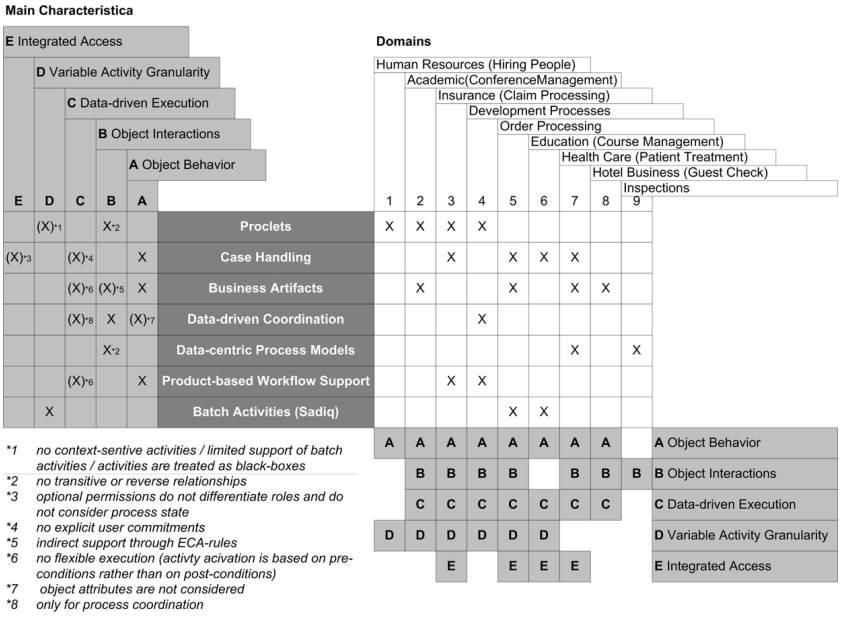

Fig. 11. Evaluation of existing approaches

Interestingly, existing approaches partially consider similar scenarios, while addressing different characteristics (see the grey boxes on the bottom of Fig. 11). For example, *order processing* was taken as illustrating scenario by Case Handling [2], Batch Activities [25], and Business Artifacts [4]. Case Handling addresses the need for enabling object behavior, data-driven execution, and integrated access. In turn, Business Artifacts consider data-driven execution, object behavior and object interactions. Finally, [25] describes the need for executing several activities in one go (i.e., the execution of batch-activities). Hence, the integrated support of all characteristics described is urgently needed to adequately cope with *order processes*.

5 Summary

Through elaborating the main characteristics of object-aware processes, this paper has shown that process and data more or less constitute two sides of the same coin, which must be tightly integrated. Hence, data-driven and object-aware process support will provide an important contribution towards the realization of a more flexible process management for which daily work can be accomplished in a more natural way than in traditional process-aware information systems. As illustrated in Fig. 12, a comprehensive integration of processes and data entails three major benefits:

1. Flexible execution of unstructured, knowledge-intensive processes.
2. Integrated view on processes, data, and functions to users.
3. Generic business functions, e.g., automatically generated form-based activities.

PHILharmonicFlows offers a comprehensive solution framework to adequately support data-driven and object-aware processes. In particular, this framework supports the definition of data and process in separate, but well integrated models. So far, PHILharmonicFlows has addressed process modeling, execution and monitoring, and it provides generic functions for the model-driven generation of end user components (e.g., form-based activities). Furthermore, PHILharmonicFlows considers all components of the underlying data structure; i.e., objects, relations and attributes. For this purpose, it enables the modeling of processes at different levels of granularity. In particular, it combines object behavior based on states with data-driven process execution. Further, it provides advanced support for process coordination as well as for the integrated access to business processes, business functions and business data. We believe that the described framework offers promising perspectives for overcoming many of the limitations of contemporary PrMS.

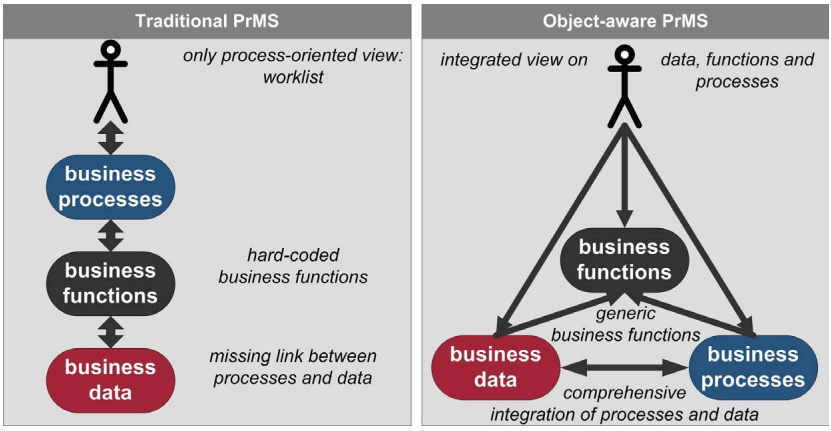

Fig. 12. Object-aware Process Management

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. DKE 53, 129–162 (2005)
3. van der Aalst, W.M.P., Barthelmess, P., Ellis, C.A., Wainer, J.: Workflow Modeling using Proclets. In: Scheuermann, P., Etzion, O. (eds.) CoopIS 2000. LNCS, vol. 1901, pp. 198–209. Springer, Heidelberg (2000)
4. Bhattacharya, K., Hull, R., Su, J.: A Data-Centric Design Methodology for Business Processes, pp. 503–531. IGI Global (2009)
5. Müller, D., Reichert, M., Herbst, J.: Data-Driven Modeling and Coordination of Large Process Structures. In: Meersman, R., Tari, Z. (eds.) OTM 2007, Part I. LNCS, vol. 4803, pp. 131–149. Springer, Heidelberg (2007)
6. Reijers, H.A., Liman, S., van der Aalst, W.M.P.: Product-Based Workflow Design. Management Information Systems 20, 229–262 (2003)
7. Vanderfeesten, I., Reijers, H.A., van der Aalst, W.M.P.: Product-based Workflow Support. Information Systems 36, 517–535 (2011)
8. Redding, G., Dumas, M., ter Hofstede, A.H.M., Iordachescu, A.: Transforming Object-Oriented Models to Process-Oriented Models. In: ter Hofstede, A.H.M., Benatallah, B., Paik, H.-Y. (eds.) BPM Workshops 2007. LNCS, vol. 4928, pp. 132–143. Springer, Heidelberg (2008)
9. Redding, G.M., Dumas, M., ter Hofstede, A.H.M., Iordachescu, A.: A flexible, object-centric approach for business process modelling. Service Oriented Computing and Applications, 1–11 (2009)
10. Künzle, V., Reichert, M.: Striving for object-aware process support: How existing approaches fit together. In: 1st Int'l Symposium on Data-driven Process Discovery and Analysis, SIMPDA 2011 (2011)
11. Rinderle, S., Reichert, M.: Data–Driven Process Control and Exception Handling in Process Management Systems. In: Martinez, F.H., Pohl, K. (eds.) CAiSE 2006. LNCS, vol. 4001, pp. 273–287. Springer, Heidelberg (2006)
12. Künzle, V., Reichert, M.: Towards Object-Aware Process Management Systems: Issues, Challenges, Benefits. In: Halpin, T., Krogstie, J., Nurcan, S., Proper, E., Schmidt, R., Soffer, P., Ukor, R. (eds.) BPMDS 2009 and EMMSAD 2009. LNBIP, vol. 29, pp. 197–210. Springer, Heidelberg (2009)
13. Künzle, V., Reichert, M.: Integrating Users in Object-Aware Process Management Systems: Issues and Challenges. In: Rinderle-Ma, S., Sadiq, S., Leymann, F. (eds.) BPM 2009 Workshops. LNBIP, vol. 43, pp. 29–41. Springer, Heidelberg (2010)
14. Künzle, V., Reichert, M.: PHILharmonicFlows: Towards a Framework for Object-aware Process Management. Journal of Software Maintenance and Evolution: Research and Practice 23, 205–244 (2011)
15. Künzle, V., Reichert, M.: A Modeling Paradigm for Integrating Processes and Data at the Micro Level. In: Halpin, T., Nurcan, S., Krogstie, J., Soffer, P., Proper, E., Schmidt, R., Bider, I. (eds.) BPMDS 2011 and EMMSAD 2011. LNBIP, vol. 81, pp. 201–215. Springer, Heidelberg (2011)
16. Künzle, V., Weber, B., Reichert, M.: Object-aware Business Processes: Fundamental Requirements and their Support in Existing Approaches. International Journal of Information System Modeling and Design (IJISMD) 2, 19–46 (2011)
17. Chiao, C.M., Künzle, V., Reichert, M.: Towards object-aware process support in healthcare information systems. In: 4th International Conference on eHealth, Telemedicine, and Social Medicine, eTELEMED 2012 (2012)

18. Künzle, V., Reichert, M.: PHILharmonicFlows: Research and Design Methodology. Technical Report UIB-2011-05, University of Ulm, Ulm, Germany (2011)
19. Weber, B., Mutschler, B., Reichert, M.: Investigating the effort of using business process management technology: Results from a controlled experiment. Science of Computer Programming 75, 292–310 (2010)
20. Guenther, C.W., Reichert, M., van der Aalst, W.M.: Supporting flexible processes with adaptive workflow and case handling. In: Proc. WETICE 2008, 3rd IEEE Workshop on Agile Cooperative Process-aware Information Systems (ProGility 2008), pp. 229–234. IEEE Computer Society Press (2008)
21. Küster, J.M., Ryndina, K., Gall, H.: Generation of Business Process Models for Object Life Cycle Compliance. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 165–181. Springer, Heidelberg (2007)
22. Gerede, C.E., Su, J.: Specification and Verification of Artifact Behaviors in Business Process Models. In: Krämer, B.J., Lin, K.-J., Narasimhan, P. (eds.) ICSOC 2007. LNCS, vol. 4749, pp. 181–192. Springer, Heidelberg (2007)
23. Müller, D., Reichert, M., Herbst, J.: A New Paradigm for the Enactment and Dynamic Adaptation of Data-Driven Process Structures. In: Bellahsène, Z., Léonard, M. (eds.) CAiSE 2008. LNCS, vol. 5074, pp. 48–63. Springer, Heidelberg (2008)
24. Vanderfeesten, I., Reijers, H.A., van der Aalst, W.M.P.: Product Based Workflow Support: Dynamic Workflow Execution. In: Bellahsène, Z., Léonard, M. (eds.) CAiSE 2008. LNCS, vol. 5074, pp. 571–574. Springer, Heidelberg (2008)
25. Sadiq, S.W., Orlowska, M.E., Sadiq, W., Schulz, K.: When workflows will not deliver: The case of contradicting work practice. In: Proc. BIS 2005 (2005)