Striving for Object-aware Process Support: How Existing Approaches Fit Together

Vera Künzle and Manfred Reichert

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

Abstract. Many limitations of contemporary process management systems (PrMS) can be traced back to the missing integration of processes and data. A unified understanding of the inherent relationships existing between processes and data, however, is still missing. In the PHILharmonicFlows project we figured out that process support often requires *object-awareness*. This means, data must be manageable in terms of object types comprising object attributes and relations to other object types. In this paper, we systematically introduce the fundamental characteristics of object-aware processes. Further, we elaborate existing approaches recognizing the need for a tighter integration of processes and data along these characteristics. This way, we show the high relevance of the identified characteristics and confirm that their support is needed in many application domains.

Key words: Object-aware Processes, Data-driven Process Execution

1 Introduction

Despite the widespread adoption of existing process management systems (PrMS) there exist numerous processes which are currently not adequately supported. In this context, it has been confirmed by different authors that many limitations of contemporary PrMS can be traced back to the missing integration of processes and data [1, 2, 3, 4, 5, 6, 7, 8]. However, a unified understanding of the inherent relationships existing between processes and data is still missing.
In the PHILharmonicFlows[1] project, we analyzed various processes from different domains which require a tighter data integration [9, 10, 11]. We figured out that the support of many of these processes requires *object-awareness*; i.e., these processes focus on the processing of business data represented through *business objects*. The latter comprise a set of *object attributes* and are *inter-related*. In this context, business processes coordinate the processing of business objects among different users enabling them to cooperate and communicate with each other.
Existing PrMS, however, focus on business functions and their control flow, whereas business objects are "unknown" to them. Most PrMS only cover simple data elements, which are needed for control flow routing and for supplying input

[1] Process, Humans and Information Linkage for harmonic Business Flows

parameters of activities. Business objects, in turn, are usually stored in external databases and are outside the control of the PrMS. For this reason, existing PrMS are unable to adequately support object-aware processes [12].
In this paper, we introduce the main characteristics of object-aware processes which we gathered in several case studies [9, 10] (see [13] for details about the research methodology we applied). Following this, we evaluate to what extent existing data-aware or data-driven process support paradigms support these characteristics. Overall, this evaluation reveals three major results: First, the characteristics we identified for object-aware processes are of high relevance. Second, object-aware process support is needed in many domains. Third, a comprehensive framework for object-aware process management is still missing.
The paper is structured as follows. In Section 2 we elaborate the role of business objects in the context of process management in detail and introduce fundamental characteristics of object-aware processes along a running example. Following this, we evaluate existing approaches against these characteristics in Section 3. Section 4 discusses the outcomes we obtained during our evaluation. We close with a summary and outlook in Section 5.

2 Object-aware Process Support

We first discuss fundamental characteristics of object-aware processes which constitute aggregations of more detailed property lists. The latter rely on an extensive analysis of processes currently not adequately supported by PrMS [9, 10, 12, 11]. To ensure that the processes we analyzed are not "self-made" examples, but constitute real-worl processes of high practical relevance, we particularly analyzed processes as implemented in existing business applications. In addition, we rely on extensive practical experiences gathered when developing contemporary business applications; i.e., we have deep insights into their application code and process logic. In order to justify our findings, we complemented our process analyses by an extensive literature study to ensure both importance and completeness. Regarding the latter, we ensure that we do not have excluded important properties already identified by other researchers. However, we excluded properties in respect to process change and process evolution. Instead, our focus was on process modeling, execution and monitoring. As illustrated in Fig. 1, we discuss the characteristics along a (simplified) scenario for recruiting people as known from human resource management.

Example 1 (Recruitment Example) *In the context of 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.*

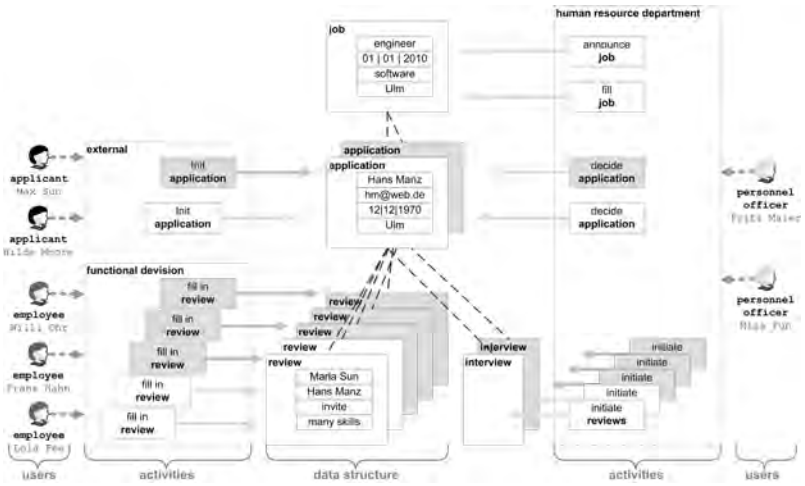

Fig. 1. Example of a recruitment process from the human resource domain

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

2.1 Object Behavior

To capture the processing of individual object instances, the first level of process granularity concerns *object behavior*. More precisely, for each object type a

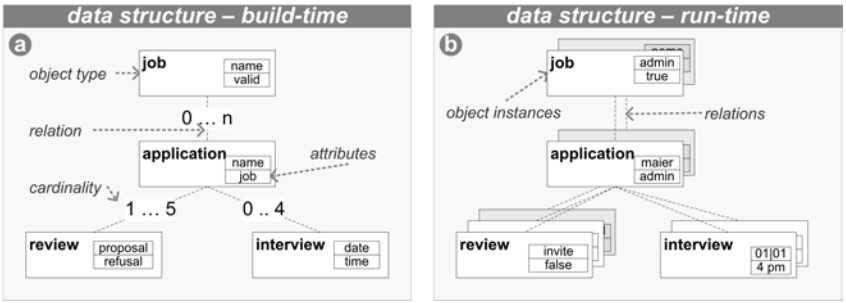

Fig. 2. Data structure at build-time and at run-time

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`*.*

2.2 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

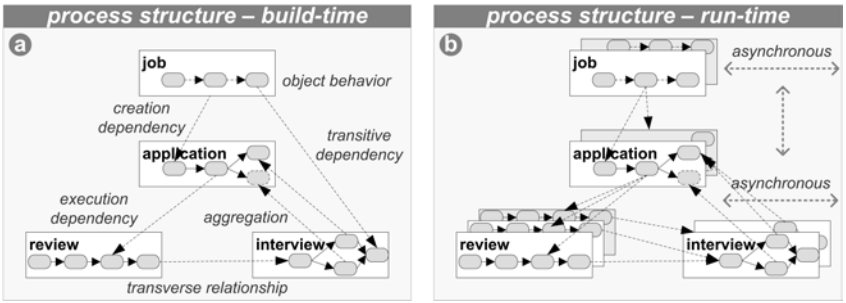

Fig. 3. Process structure at build-time and at run-time

object instance. Hence, it should be possible to aggregate information; amongst 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* (e.g., reviews depend on the respective job offer) as well as *transverse dependencies* (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). Further, individual* `review` *process instances are executed concurrently to each other as well as to the* `application` *process instances; e.g., the* `personnel officer` *may read and change the* `application` *while the* `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.*

2.3 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`*.*

2.4 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. 4a), all input fields and data fields refer to attributes of one particular object instance, whereas *context-sensitive activities* (cf. Fig. 4b) comprise fields referring to different, but related object instances (of potentially different type). Finally, *batch activities* involve several object instances of the same type (cf. Fig. 4c). Here, the values of the different input fields are assigned to all involved object instances 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).

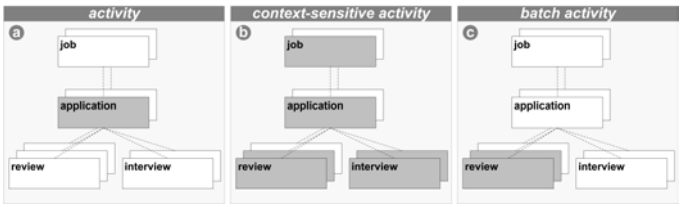

Fig. 4. Different kinds of activities

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.*

2.5 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. worklists), 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 Existing Approaches

In this section, we evaluate existing data-aware approaches along the main characteristics of object-awareness.

3.1 Case Handling

Case Handling (CH) [1] is a *data-driven process support paradigm* in which activities are explicitly represented through user *forms* comprising a number of input fields. The latter refer to *atomic data elements* which are either defined as *mandatory*, *restricted* or *free*.

Object Behavior. CH does not provide explicit support for complex objects and relations between them. However, a "case" can be considered in tight accordance with an "object". This enables the definition of object behavior specifying in which order and by whom object attributes shall be written. In addition, input fields and transitions can be associated with constraints on attribute values. This way, processes are defined in terms of data conditions rather than based on black-box activities.

Data-driven Execution. An activity is considered as being completed if all mandatory data elements have an assigned value. Since different forms may comprise the same data element, it is possible to provide required attribute values before they become mandatory for a particular activity. Hence, activities are automatically skipped if all mandatorily required data usually provided by them is already available. Besides defining who shall work on an activity, CH allows defining who may redo an activitiy or (manually) skip it.

Object Interactions. Related cases can be invoked using *sub-plans*. The latter must be instantiated at specific points during the execution of the higher-level case (i.e., creation dependency). In addition, dynamic sub-plans enable the creation of a varying number of instances at run-time (including cardinality constraints). Since higher-level cases may include data arrays containing data elements from sub-plans, certain kinds of aggregations can be supported. However,

it is not possible to execute sub-plans asynchronously to the higher-level sub-plan and to define execution dependencies. Finally, sub-plans require a strong hierarchical collocation of related cases. For this reason, it is not possible to consider arbitrary relationships between cases and sub-plans respectively.

Variable Activity Granularity. When creating forms, it is possible to use data elements corresponding to the higher- and lower-level plans as well. Thus, in addition to instance-specific forms, context-specific ones can be provided. Batch activities, in turn, are not supported. Finally, each form (incuding their input fields and data fields) must be pre-defined at build-time. For that reason, providing all conceivable forms taking different roles as well as the progress of the process into account is a very cumbersome and expensive task.

Integrated Access. In principle, each user may invoke the activity currently activated. He then can read all data elements of the case and additionally write all data elements not categorized as mandatory or restricted. It is not possible to assign different permissions for optionally reading and writing data elements to different user (roles). Moreover, permissions for reading and writing free data elements are treated independently from the process state. Finally, since mandatory and restricted data elements can be only written by users owning the execute-role, it is not possible to define one and the same activity as optional for a particular user while being mandatory for another one.

3.2 Proclets

Proclets [2] are *object-specific processes* which communicate with each other based on *messages*. The latter are exchanged through *ports* connected with transitions. Each message sent or received is stored in the *knowledge base* of the respective Proclet.

Object Behavior. Proclets are defined based on black-box activities. For this reason, it is neither possible to determine the order in which object attributes shall be written nor to define what valid attribute settings are.

Data-driven execution. Since the Proclet framework is based on an activity-centered paradigm, data-driven activation of activities is not possible. However, each activity can be associated with a *pre- and post-condition* defined on basis of the information from the knowledge base (i.e., exchanged messages). At run-time, an activity becomes enabled if its incoming transition is activated, the pre-condition evaluates to true, and required messages are available. This way, data-driven coordination of Proclet instances becomes possible.

Object Interactions. At run-time, for each Proclet type a dynamic number of instances can be handled. This way, a complex process structure evolves in which the individual Proclet instances can be executed asynchronously to each other and be synchronized where needed. It is possible to send a message to multiple Proclets or to restrict the number of recipients by using cardinality constraints. In addition, the pre- and post-conditions of activities can be used to define creation and execution dependencies as well as aggregations. However, it is not possible to handle transitive or transverse relationships between Proclet instances.

Variable Activitiy Granularity. Activities do not comprise several Proclet types. Thus, instance-specific activities are enabled, while context-specific ones are not explicitly considered; i.e., it is not possible to access data elements from lower- or higher-level Proclet instances. Batch-oriented activities, in turn, are partially supported by enabling multiple instantiation of corresponding Proclet instances. The execution of a set of instance-specific activities in one go, however, is not possible. Finally, since activities are treated as black-boxes, internal process logic is not supported.
Integrated Access. Integrated access to application data is taken into account.

3.3 Business Artifacts

A business artifact comprises *atomic and structured attributes*, *related business artifacts*, and a *lifecycle* [3]. The latter is defined using a *finite-state machine* capturing the main *processing stages* and the *transitions* between them. Transitions can be associated with *conditions* defined in terms of attribute values or relationships to other business artifacts. *Services*, in turn, are executed to evolve business artifacts through their entire lifecycle. For this purpose, *associations* (i.e., ECA-rules) specify how services are linked with artifacts. Note that artifacts (including their informational structure and lifecycles), services, and ECA-rules only constitute a logical representation of business processes and business data. In particular, there is *no well-defined operational semantics* for directly executing the defined models. Instead, definitions are mapped to an activity-centred flow diagram (i.e. for optimization) and are then (manually) implemented resulting in hard-coded process logic (at the end).
Object Behavior. Services are defined in terms of black-box activities. Thus, it is neither possible to determine the order in which object attributes shall be written nor to define what valid attribute settings are. However, for each transition of an artifact lifecycle, a (data-) condition can be specified. This way, it becomes possible to synchronize data state and process state. However, it is a tedious task to ensure that these conditions are consistent with the ECA-rules specified for service invocations.
Object Interactions. Related artifacts can be defined within the informational structure of a business artifact as well. Their corresponding lifecycles, however, are treated independently (i.e., within their own artifact model). Consequently, the emerging data structure is redundantly distributed among several data models. This makes the corresponding process structure hard to comprehend and difficult to maintain. In addition to object behavior support, ECA-rules can be used for coordinating artifact lifecycles (i.e., by using quantifiers). Consequently, there is no clear separation between object behavior and object interactions. Finally, aggregations are not taken into account and transitive and transverse relationships between business artifacts are not considered.
Data-driven Execution. Associations (i.e., ECA-rules) enable the activation of services based on data conditions. Since these rules constitute pre-conditions rather than post-conditions, it is not possible to dynamically skip services.

Variable Activitiy Granularity. Each service requires its own implementation. Hence, its granularity is fixed at build-time; i.e., form-based activities are not explicitly supported. In addition, the internal control flow of a particular form is not considered.
Integrated Access. Although optional activities can be realized using ECA-rules, the business artifacts framework does not target at an integrated access to application data. For example, for users it is not possible to distinguish optional and mandatory activities contained in their worklist.

3.4 Data-driven Coordination

The data-driven coordination framework Corepro [4] enables the coordination of individual processes based on *objects* and *object relations*. Objects are defined in terms of *states* and *(internal) transitions* between them. The latter can be associated with processes which must be completed in order to reach the subsequent state. In correspondance to the relations between objects, *external transitions* connect states belonging to different objects with each other.
Object Behavior. Although the behavior of objects is explicitly defined, object attributes and their values are not taken into account. Thus, it is impossible to determine mandatorily required data or to define what valid attribute settings are. Consequently, processes are defined in terms of black-box activities rather than based on data conditions.
Object Interactions. It is possible to asynchronously execute object-related process instances and to synchronize their execution. In particular, creation as well as execution dependencies can be defined by using external transitions. The latter, however, can only be specified along relations between objects directly defined by the corresponding data structure. Transitive or transverse relations, in turn, are not considered. Although it is possible to create a variable number of instances at run-time, aggregations are not supported.
Data-driven Execution. Processes themselves are still activity-driven; i.e., the activation of a subsequent state depends on the completion of a process associated with the corresponding state transition. Opposed to this, process synchronization follows a data-driven approach.
Finally, since object attributes are out of scope and process execution is based on black-box activities, neither a **variable granularity of activities** nor **integrated access** to application data is provided.

3.5 Product-based Workflow Support

Product-based workflows [5, 14, 6] use a so-called *product data model* which is described by a tree-like structure [5] comprising *atomic data elements* and *operations*. The latter have zero or more input data elements and exactly one output data element. An operation is executable if (specific) values are available for all input data elements. The product is fully processed as soon as a value for the top element of the product data model (i.e., the root element) is available. Process

models can be manually derived [5] or automatically generated [14] based on the product data model. In addition, it is possible to directly execute the product data model [14, 6].
Object Behavior. Using atomic data elements, it is possible to specify which data is mandatorily required during process execution. In addition, it is possible to determine the order in which data elements have to be written as well as to restrict valid attribute settings.
Object Interactions. Since each process instance is executed in isolation, it is not possible to coordinate their execution.
Integrated Access. Access to data is only possible during the execution of operations specified within the product data model; i.e., users cannot access data at arbitrary point in time.
Data-driven Execution. The direct execution of the product data model enables data-driven process execution. Since activity activation depends on preconditions, however, it is not possible to automatically skip an activity if the required output data element is already available. In addition, re-execution of activities is not possible.
Variable Activity Granularity. Each operation requires a specific implementation and therefore constitutes a black-box activity. Consequently, all operations have fixed granularity and the internal control-flow of a particular form cannot be considered.

3.6 Further Approaches

Similar to the Proclets [2] framework, the Object-centric Business Process Modeling framework [7, 8] enables the coordination of object-specific processes along their corresponding object relations. However, processes are defined in an activity-centred way; i.e., in terms of black-box activities. Regarding coordination, cardinality constraints as well as creation and execution dependencies are taken into account. Finally, [15] proposes to group and ungroup related activities within user worklists. This way, batch activities can be supported.

4 Discussion

As illustrated in Fig. 5, each characteristic is addressed by at least one existing approach. Although the mentioned approaches have limitations (see footnotes in Fig. 5), they can be considered as pioneer work towards object-aware process support. However, none of them covers all characteristics in a comprehensive and integrated way. Also note that Fig. 5 does not make a difference between process modeling and process execution. Though some approaches (e.g., the business artifacts framework [3]) provide rich capabilities for process modeling, they do not cover run-time issues (or at least do not treat them explicitly).
In order to underline the high practical impact of object-aware process support, we contrast the characteristics with the different application examples considered

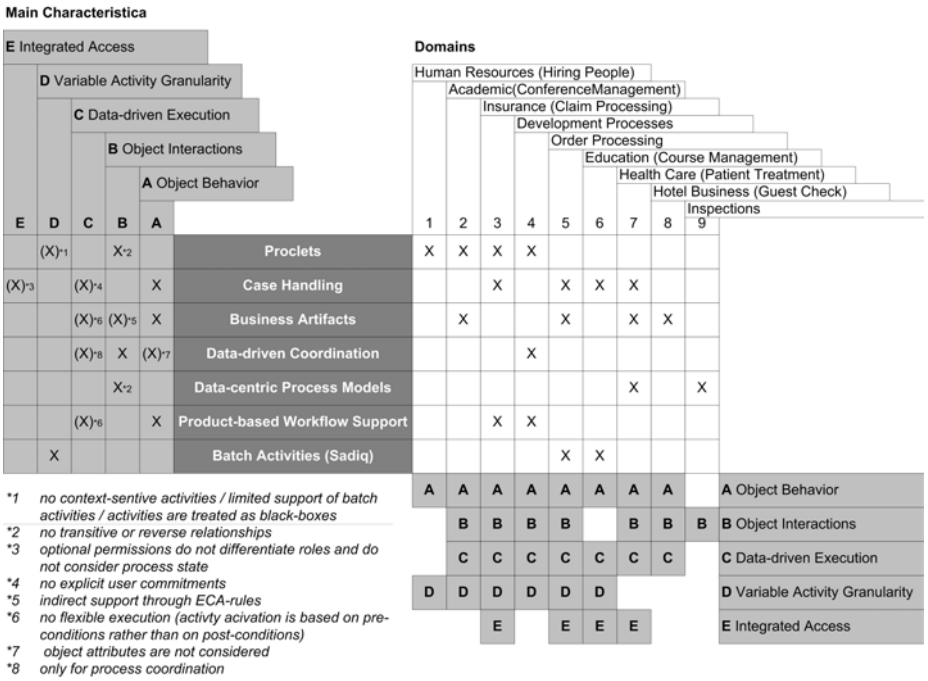

Fig. 5. Evaluation of existing approaches

by existing approaches (cf. Fig. 5). In particular, existing approaches partially consider similar scenarios, while addressing different characteristics (cf. the grey boxes on the bottom of Fig. 5). For example, order processing was taken as illustrating scenario by Case Handling [1], Batch Activities [15], and Business Artifacts [3]. Case Handling addresses the need for enabling object behavior, data-driven execution, and integrated access. Business Artifacts, in turn, consider data-driven execution, object behavior and object interactions. Finally, [15] describes the need for executing several activities in one go (i.e., the execution of batch-activities). Consequently, this indicates that integrated support of all these characteristics is urgently needed to adequately cope with *order processes*. Altogether, this comparison demonstrates two things: First, the characteristics are related to each other. Second, broad support for them is required by a variety of processes from different application domains.

5 Summary and Outlook

In this paper we made several contributions. First, we systematically introduced the main characteristics of object-aware processes. Second, we elaborated pioneering work recognizing the need for a tighter integration of process and data along these characteristics. Overall, the conducted evaluation has confirmed the

high relevance of the characteristics and that their support is needed in many application domains. However, as discussed, a comprehensive framework for object-aware process management is still missing.

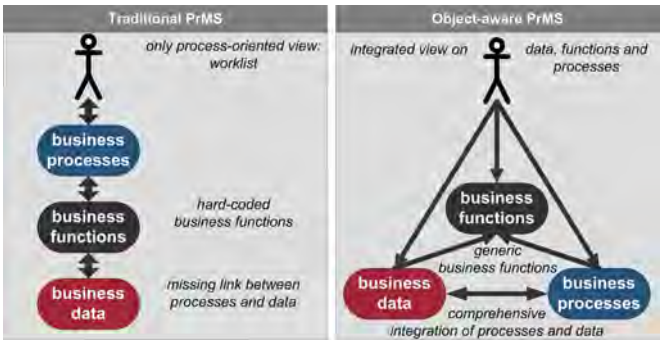

Fig. 6. Object-aware Process Management

Altogether, we believe that object-aware process management will provide an important contribution towards the realization of flexible process management technology in which daily work can be done in a more natural way. In particular, as illustrated in Fig. 6, 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: automatically generated form-based activities.

In the PHILharmonicFlows project we target at a *comprehensive framework for object-aware process management* enabling the introduced characteristics. PHILharmonicFlows enforces a *well-defined modeling methodology* governing the definition of processes at different levels of granularity and being based on a *well-defined formal semantics*. More precisely, the framework differentiates between *micro and macro processes* in order to capture both *object behavior* and *object interactions*. As a prerequisite, object types and their relations need to be captured in a data model. For each object type a corresponding *micro process type* needs to be defined. In this context, our approach applies the well established concept of modeling object behavior in terms of states and state transitions [3, 4]. Opposed to existing approaches, however, PHILharmonicFlows enables a *mapping between attribute values and objects states* and therefore ensures compliance between them. Finally, the presented execution paradigm combines *data-driven process execution* with *activity-oriented aspects*. Optional access to data, in turn, is enabled asynchronously to process execution and is based on permissions for creating and deleting object instances as well as for reading/writing their attributes. For this, PHILharmonicFlows maintains a *comprehensive authorization table* taking the current progress of the corresponding micro process instance into

account. In accordance to the relations between the invoked object instances, the corresponding micro process instances additionally form a *complex process structure*. By using *macro processes*, however, we *hide this complexity* from modelers as well as from end-users to a large degree.
In [11] we have already introduced the basic components of PHILharmonicFlows as well as their complex interdependencies. In addition, details on micro process support and the automatic generation of form-based activities can be found in [16]. Finally, a tighter integration of processes and data implicates further challenges in respect to the integration of users [10]. These issues are considered in PHILharmonicFlows as well.

References

1. van der Aalst, W.M.P., Weske, M., Grünbauer, D.: Case Handling: A new Paradigm for Business Process Support. DKE **53**(2) (2005) 129–162
2. van der Aalst, W.M.P., Barthelmess, P., Ellis, C.A., Wainer, J.: Workflow Modeling using Proclets. In: Proc. CoopIS'00. (2000) 198–209
3. Bhattacharya, K., Hull, R., Su, J. In: A Data-Centric Design Methodology for Business Processes. IGI Global (2009) 503–531
4. Müller, D., Reichert, M., Herbst, J.: Data-Driven Modeling and Coordination of Large Process Structures. In: Proc. CoopIS'07. LNCS 4803 (2007) 131–149
5. Reijers, H.A., Liman, S., van der Aalst, W.M.P.: Product-Based Workflow Design. Management Information Systems **20**(1) (2003) 229–262
6. Vanderfeesten, I., Reijers, H.A., van der Aalst, W.M.P.: Product-based Workflow Support. Information Systems **36**(2) (2011) 517–535
7. Redding, G.M., Dumas, M., ter Hofstede, A.H.M., Iordachescu, A.: Transforming Object-oriented Models to Process-oriented Models. In: Proc. BPM'07 Workshops. LNCS 4928 (2007) 132–143
8. 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 (2009) 1–11
9. Künzle, V., Reichert, M.: Towards object-aware process management systems: Issues, challenges, benefits. In: Proc. BPMDS'09. LNBIP 29 (2009) 197–210
10. Künzle, V., Reichert, M.: Integrating Users in Object-aware Process Management Systems: Issues and Challenges. In: Proc. BPD'09. LNBIP 43 (2009) 29–41
11. Künzle, V., Reichert, M.: PHILharmonicFlows: Towards a Framework for Object-aware Process Management. Journal of Software Maintenance and Evolution: Research and Practice **23**(4) (June 2011) 205–244
12. 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**(2) (April 2011) 19–46
13. Künzle, V., Reichert, M.: PHILharmonicFlows: Research and Design Methodology. Technical Report UIB-2011-05, University of Ulm, Ulm, Germany (May 2011)
14. Vanderfeesten, I.: Product-Based Design and Support of Workflow Processes. Phd thesis, Eindhoven University of Technology (2009)
15. Sadiq, S.W., Orlowska, M.E., Sadiq, W., Schulz, K.: When workflows will not deliver: The case of contradicting work practice. In: Proc. BIS'05. (2005)
16. Künzle, V., Reichert, M.: A Modeling Paradigm for Integrating Processes and Data at the Micro Level. In: Proc. BPMDS'11, Springer (2011) *(accepted for publication)*