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.

Keywords: Process-aware Information Systems, Object-aware Process Management, 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 by these PrMS. 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 tight 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

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

K. Aberer, E. Damiani, and T. Dillon (Eds.): SIMPDA 2011, LNBIP 116, pp. 169–188, 2012.

data elements, which are needed for control flow routing and for supplying input 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 of this evaluation. Finally, Section 5 introduces a comprehensive framework for object-aware process management. We close with a summary and outlook in Section 6.

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-world processes of high practical relevance, we particularly considered 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. With the latter we want to ensure that we have not excluded important properties already identified by other researchers. However, in this study we did not consider properties in respect to process change and process evolution. Instead, our focus was on process modeling, execution and monitoring. In order to emphasize the relevance of the identified characreristics and their inter-relations we contrast them with the different application examples considered by existing approaches (cf. 4) As illustrated in Fig. 1, we discuss the characteristics along a (simplified) real-world 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.

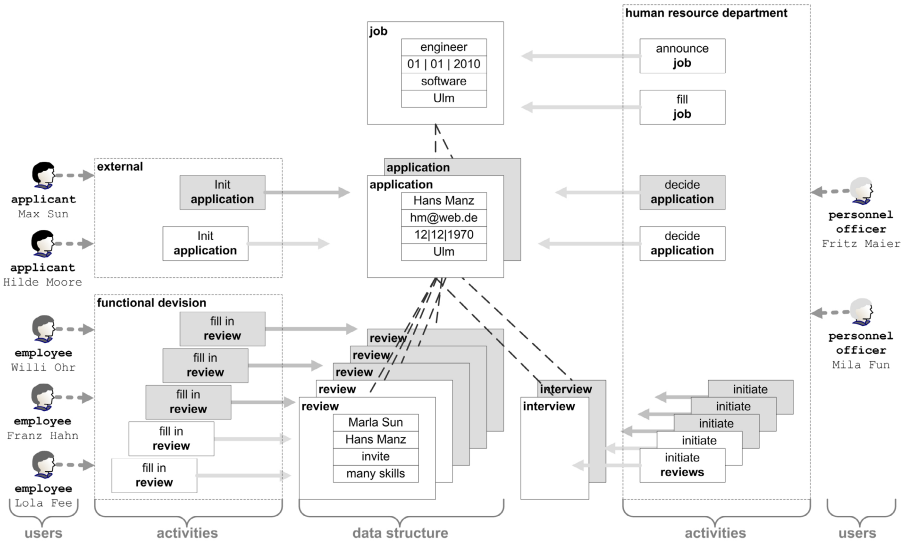

Fig. 1. Example of a recruitment process from the human resource domain

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). 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).

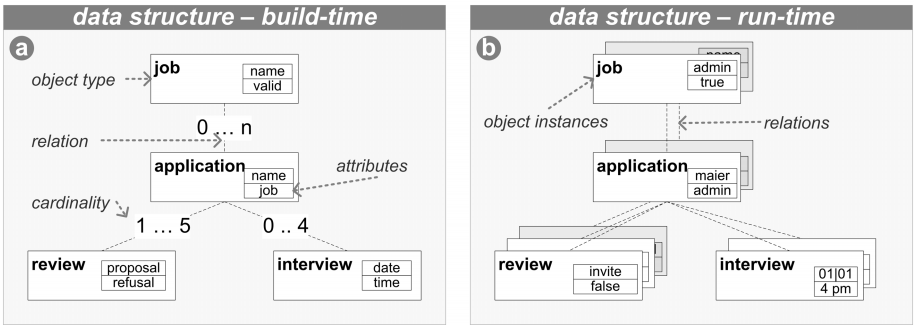

Fig. 2. Data structure at build-time and at run-time

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

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

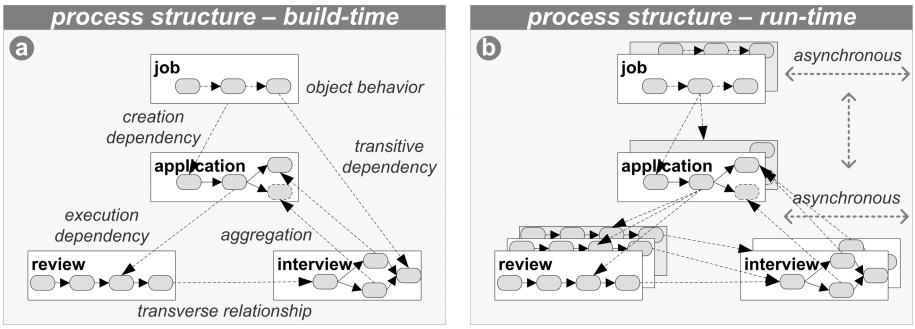

Fig. 3. Process structure at build-time and at run-time

(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 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). 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). 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. 4c). 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).

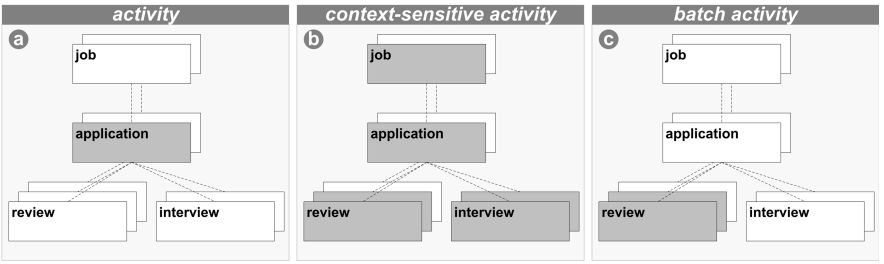

Fig. 4. Different kinds of activities

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.*

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

Many existing approaches have already confirmed the high relevance of a tighter integration of processes and data. In this section, we evaluate selected approaches along the main characteristics of object-awareness: object behavior, object interactions, data-driven process execution, variable activity granularity, and integrated access to data.

3.1 Case Handling

Case Handling [1] is a *data-driven process support paradigm* in which activities are explicitly represented through user *forms* comprising a number of input fields (i.e., text fields, combo boxes, check boxes). The latter refer to *atomic data elements* which are either defined as *mandatory*, *restricted* or *free*. An activity is considered as being completed if all mandatory data elements have an assigned

value. Beside defining who shall work on an activity, Case Handling also allows specifying who may redo or skip it; for these uses separate roles exist.

Object Behavior. Unfortunately, Case Handling does not provide explicit support for complex objects and relations between them. However, a "case" can be considered in tight accordance with an "object"; i.e., the data elements can be considered as object attributes. This enables the definition of object behavior specifying in which order and by whom object attributes shall be written. Since it is possible to assign a corresponding value constraint to each input field, valid attribute settings can be enforced. In addition, data constraints can be assigned to transitions connecting individual activities (i.e., user forms). Altogether, processes are defined in terms of data conditions.

Data-Driven Execution. An activity is considered as completed if a value for all mandatory data elements is available. Since different forms may comprise the same data element, it is possible to provide required attribute values up-front; i.e., before they are mandatory for one activity. This way, activities can be automatically skipped if mandatorily required data is already available. Despite the introduction of the redo-role, re-executing activities may be often not possible. In particular, if all data elements being mandatory for the current and for the subsequent activity are available, both activities are automatically marked as executed. In this case, a user may only re-execute the activity if he additionally owns the redo-role for both the current and the subsequent activity. In particular, it is not supported that users re-execute a certain activity as long as they have not explicitly confirmed its completion.

Object Interactions. In Case Handling, it is possible to involve related cases by using *sub-plans*. The latter must be instantiated at specific points during the execution of the higher-level case (i.e., creation dependency). Further, these sub-plans can be categorized as dynamic. This enables the instantiation of a variable number of instances at run-time. In this context, cardinality constraints are considered by defining a minimal and maximal number of instances which should be completed at run-time. However, it is not possible to execute sub-plans asynchronously to the higher-level one. Thus, execution dependencies cannot be defined. Aggregations, in turn, are possible. For this purpose, the higher-level case may include an array whose elements are mapped to data elements of the respective sub-plan. Finally, since sub-plans require a strong hierarchical collocation of related cases, the consideration of arbitrary relationships between cases is not supported. It is not possible, for example, to define a dependency between reviews and interviews (cf. Fig. 1); i.e., it is only allowed to initiate an interview if at least one review proposes to invite the applicant.

Variable Activity Granularity. Using Case Handling, all forms must be predefined at build-time. This means, it is not possible to automatically generate user forms based on the current processing state und the user executing the

activity. However, when creating forms, it is possible to invoke 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 support. Context-specific activities contain input fields corresponding to selected object instances. In this context, it is a cumbersome and desperate task to provide all conceivable granularities of forms.

Integrated Access. In principle, each involved user may execute the currently activated activity. In this context, he can read all data elements and additionally write all data elements which are not categorized as mandatory or restricted. However, it is not possible to assign different permissions to optionally read and write data elements for different user (roles). Moreover, it is not possible to make these permissions dependable on the process state. Opposed to free data elements, mandatory and restricted ones can only be written by users owning the execute-role. This prevents changes of data element values contradicting process execution. However, it is not possible to define one and the same activity as optional for one user while being mandatory for another one.

3.2 Proclets

A proclet [2,14] is an *object-specific process* which is modeled using a Petri net. The latter consists of nodes which comprise places (e.g. circles) and transitions (e.g., rectangles). Places and transitions are connected with arcs. In this context, it is only allowed to connect different node types; i.e., places with transitions or transitions with places. Transitions represent activities. Places, in turn, have assigned tokens representing the current state of the Petri net. In particular, a transition (i.e., an activity) can be activated if each input place contains at least one token. During its activation, one token from each input place is removed and to each output place, in turn, one token is assigned. The proclet framework enables the communication between different proclets based on *messages*. The latter are exchanged through *ports* which are connected by transitions. Each sent or received message is stored in the *knowledge base* of the respective proclet. To discover related proclets for communicating with them, a naming service is called which manages all existing proclets based on their unique proclet ID. Activities have assigned pre- and post-conditions using information from the knowledge base. At run-time, an activity becomes enabled if its corresponding transition is activated (i.e., each input place contains at least one token), the pre-condition evaluates to true, and all input ports contain a message.

Object Behavior. Although proclets are object-specific processes, they do not support the definition of object behavior as required for the support of object-aware processes. In particular, these processes are defined in an activity-centred way and not in terms of data conditions. This way, it is neither possible to determine the order in which object attributes should 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. In addition, it is possible to send messages to multiple proclets; e.g., to all instances of a certain proclet type. This way, a complex process structure evolves in which the individual proclet instances can be executed asynchronously to each other. In this context, for each port a cardinality constraint can be defined which determines the number of recipients. Creation and execution dependencies as well as aggregations can be defined based on pre-conditions for activities. In this context, however, it is not possible to handle transitive or transverse relationships between proclet instances. This means, it is not possible to synchronize a job offer instance with the set of corresponding reviews directly. Such dependencies must be specified using (several) intermediate dependencies. More precisely, reviews must be synchronized with their corresponding applications and applications with the job offer to which they refer. Otherwise, it is not possible to ensure that only the reviews required for the applications belonging to the respective job offer are considered; i.e., reviews for applications referring to other job offers are then invoked as well.

Variable Activitiy Granularity. All activities are defined in the context of one proclet type: This way, instance-specific activities are supported 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). Regarding the latter, information from other proclet instances can be transferred using messages. However, there is no interrelationship between activity execution and the content of messages. Batch-oriented activities, in turn, are partially supported by enabling multiple instantiation of related proclet instances. The execution of a set of instance-specific activities (which are represent as transitions of individual proclet instances) in one go, however, is not possible. 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

The business artifacts framework [15,3] is a process design methodology which focuses on data objects (i.e., business artifatcs) rather than on activities. A business artifact holds all business information relevant about itself. This includes *atomic and structured attributes* as well as all *related artifacts*. In addition, a *life-cycle* is defined for each business artifact which is specified using a *finite-state*

machine capturing the main *processing stages* and the *transitions* between them. Transitions can be associated with *conditions*. The latter can be defined in terms of attribute values or relationships to other business artifacts. Attribute values are assigned during the execution of services. In particular, *services* are executed to move business artifacts through their lifecycles. Services as associated with *pre- and post-conditions* for their execution (i.e., available attribute values, stages). In addition, *associations* specify how services are associated with artifacts. They are defined using ECA-rules (i.e., event, condition, action). *Events* may comprise currently assigned attribute values, a reached state, a launched or completed service, an incoming message, or a performer request. Conditions, in turn, are defined using first-order logic (e.g., SQL statements). Finally, *actions* represent service invocations or the activation of a subsequent artifact stage. 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 exists *no well-defined operational semantics* for the direct execution of the defined models. Instead, the definitions are mapped to an activity-centred flow diagram (i.e. for optimization) and are then (manually) implemented. This results in hard-coded process logic.

Object Behavior. Since transitions are associated with (data-) conditions, it is possible to synchronize process state and data state. However, it is a cumbersome task to ensure that pre- and post-conditions assigned to services are consistent with ECA-rules and the conditions defined for the transitions between the stages an artifact moves through.

Object Interactions. Within the data structure of an artifact, related artifacts are defined as well. Their corresponding lifecycles, however, are treated independently (i.e., within their own artifact model). Consequently, the arising data structure is distributed among several data models and overlapping; i.e., some artifacts are defined in the context of several other artifacts. This makes it a hard job to comprehend the emerging process structure and to keep it consistent. In addition to object behavior support, ECA-rules can be used for coordinating artifact lifecycles as well (i.e., by using quantifiers). Consequently, there is no clear separation between object behavior and object interactions. Moreover, services may change attribute values of related business artifacts. In this context, it is a cumbersome task to take care of the lifecycles of related artifacts. Finally, aggregations are not taken into account and transitive and transverse relationships between business artifacts are not considered.

Data-Driven Execution. It is possible to activate a service based on data conditions. However, since the invocation of a service depends on pre-conditions, it is not possible to dynamically skip services if their post-condition has already been fulfilled. Moreover, if certain pre-conditions cannot be met during run-time, process execution will be blocked.

Variable Activitiy Granularity. Each service requires its own implementation. For that reason, the granularity of each service is fixed at build-time; i.e. form-based activities are not explicitly supported. In addition, control flows specific to a particular form (i.e., some attributes values become mandatory on-the-fly) are not considered.

Integrated Access. Optional activities can be enforced by using ECA-rules as well; e.g., without specifying an event or a condition. However, the framework focuses on process-relevant activities. For this reason, it is not possible for users to distinguish optional and mandatory activities in their worklist. For process authorization (i.e,, service execution) users are directly assigned to the service they should execute. In this context, it is neither possible to consider the business artifact properties nor the relation of a user to the respective artifacts. Data authorization (i.e., permissions for creating, deleting and changing of business artifacts), in turn, is not explicitly considered. However, services are annotated with conditional effects (i.e., for ensuring specific attribute values to be set). This way, mandatorily required attribute values can be distinguished from optional ones. However, it depends on the respective service implementation how these assumptions are ensured at run-time. Attribute values violating a constraint are marked as invalidated. These attribute values should then be changed in a subsequent service invocation.

3.4 Data-Driven Coordination

The data-driven coordination framework [4,16] 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 are assigned with processes which must be executed to reach the subsequent state. According to the relations between objects, so called *external transitions* connect states belonging to different objects with each other. This enables the coordination of the individual object lifecycles. More precisely, whether or not a certain state of an object can be activated may depend on the currently activated states of other objects. In addition to correctness constraints (i.e., for ensuring the proper termination of the arising process structure), the approach also comprises well-defined operational semantics for the automatic enactment of object lifecycles and process structures.

Object Behavior. Although the behavior of objects is explicitly defined, object attributes and their respective values are not taken into account. For this reason, it is not possible to determine mandatorily required data and to define what valid attribute settings are. Consequently, processes are further defined in terms of black-box activities rather than based on data conditions. Thus, it is not possible to ensure that the actual processing state is in accordance with corresponding attribute values.

Object Interactions. It is possible to asynchronously execute individual object-related processes and to synchronize them. 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 which are directly represented within the corresponding data structure. Transitive or transverse relations, in turn, are not supported. 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 incoming state transition. Thus, it is not possible to execute processes or respective activities up front or to dynamically skip them. 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

The product-based workflow design or support approach [5,17,6] uses a so-called *product data model* which specifies required data elements to assemble a particular product. This product data model is described by a tree [5] and consists of *atomic data elements* and *operations*. Data elements are depicted as circles. Operations, in turn, are represented by arcs with small cycles having zero or more input data elements and exactly one output data element. An operation is executable if values for all input data elements are available. In addition, conditions on the value of the input data elements may restrict the execution of the operation. If the condition is not satisfied, the operation is not executable, even if values for all of its input data elements are available. The product is considered as being fully processed as soon as a value for the top element of the product data model (i.e., the root element) is available. Several operations can have the same output element while having a different set of input elements. Such a situation represents alternative ways to produce a value for that output element. For selecting the best execution alternative, operations are associated with different attributes describing characteristics of the operations like execution cost, processing time, and failure probability. This enables the consideration of different execution priorities during process enactment. To achieve a corresponding process model, the product data model can be manually translated [5]. Since this is time-consuming and error-prone, different algorithms are provided to automatically generate a process model on the basis of a given product data model [17]. Opposed to the consideration of an explicit process model, however, the approach also provides the direct execution of the product data model [17,6]. For this purpose, end users are supported with recommendations about which operation should be executed next. Such recommendations take care of the required performance criterion and calculate how the various alternative executions differ

from each other with respect to that performance criterion. This way, the need to translate a product data model into a process model disappears.

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, and to specify what valid attribute settings are.

Object Interactions. Since each instance is executed in isolation, it is not possible to coordinate them.

Integrated Access. Access to data is only possible during the execution of operations specified within the product data model; i.e., users are not allowed to access and manage data at any point in time. Different ways for reaching a process goal are definable using alternative execution paths. If one path has been selected, however, it is no longer possible to additionally execute the other one. Thus, it is not possible to differentiate between optional and mandatory operations.

Data-Driven Execution. The direct execution of the product data model enables data-driven process execution. In particular, a subsequent operation can be executed if for all required input data elements a corresponding value is available and the eventually associated data condition evaluates to true. Since activity activation depends on pre-conditions, however, it is not possible to automatically skip an activity if the required output data element has been 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. As a consequence, all operations have a fixed granularity and control-flow support during the execution of activities is not provided.

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. For coordination, however, cardinality constraints as well as creation and execution dependencies are taken into account. Finally, [18] proposes to group and ungroup related activities within user worklists. This way, batch activities can be supported.

4 Comparing the Different Approaches

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).

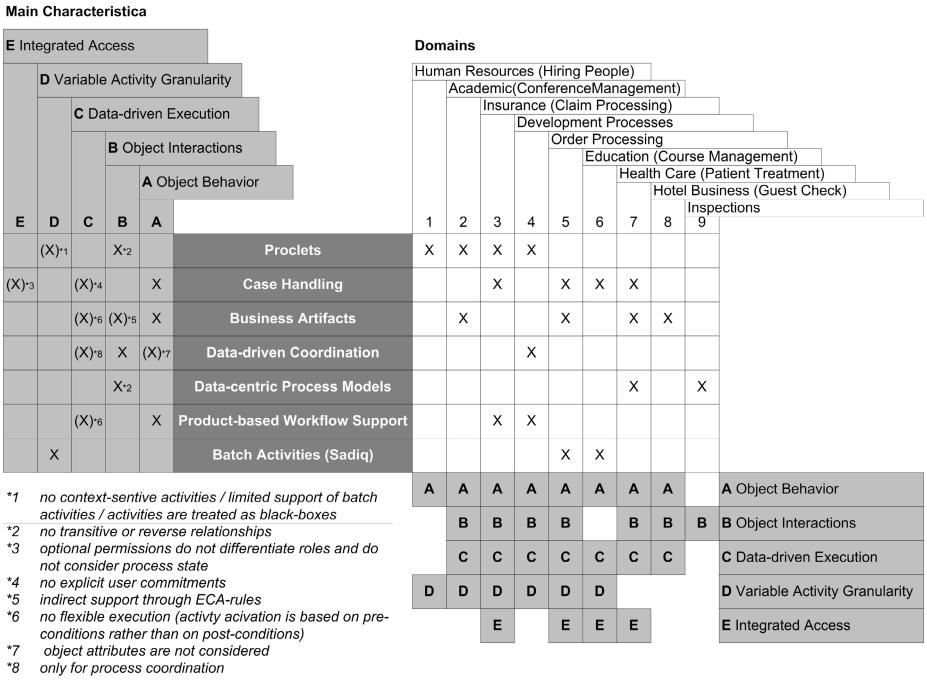

Fig. 5. Evaluation of existing approaches

In order to emphasize the high practical impact of object-aware process support, we contrast the characteristics with the different application examples considered 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 [18], 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, [18] 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 An Integrated Framework for Object-Aware Processes

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.

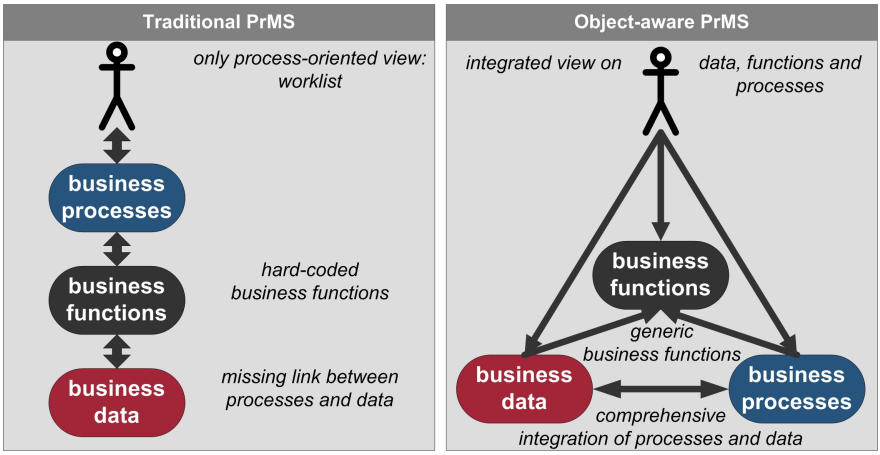

Fig. 6. Object-aware Process Management

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 (cf. Fig. 7). 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*. Due to lack of space, this paper focuses on existing approaches already addressing particular characteristics of object-aware process support. Hence, we omit details about the different modeling components and formal issues (e.g., ensuring soundness). The same applies in respect to the formal semantics process execution is based on (see [11,19] for details).

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.

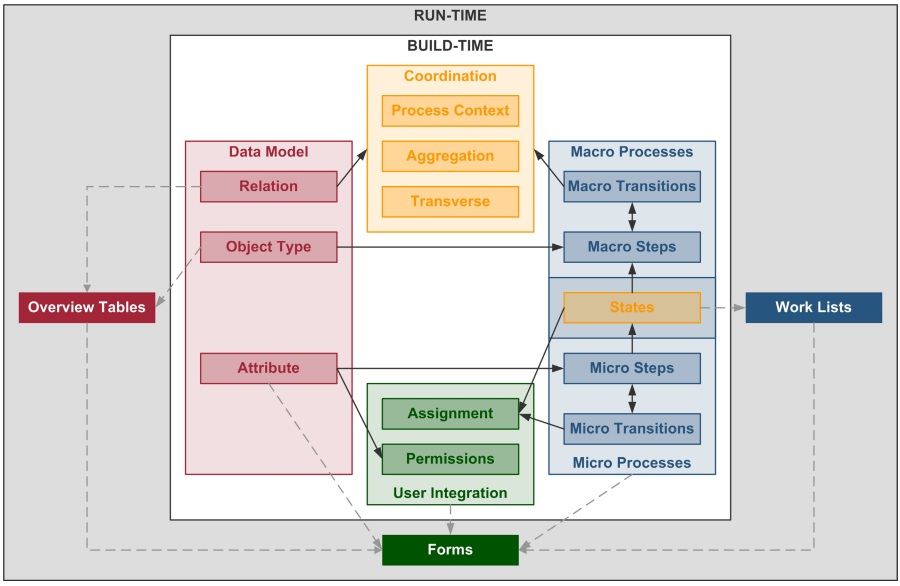

Fig. 7. Overview about the PHILharmonicFlows Framework

6 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 processes 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. PHILharmonicFlows offers a comprehensive solution framework to adequatlely support object-aware processes and to automatically generate most end-user components at run-time. 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 [19]. 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. To evaluate our framework, we are currently developing a proof-of-concept prototype for the modeling as well as run-time support of object-aware processes. In this context, additional techinques enabling improved system performance and scalability are introduced.

References

1. van der Aalst, W.M.P., Weske, M., Grünbauer, D.: Case Handling: A new Paradigm for Business Process Support. DKE 53(2), 129–162 (2005)
2. 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)
3. Bhattacharya, K., Hull, R., Su, J.: In: A Data-Centric Design Methodology for Business Processes, pp. 503–531. IGI Global (2009)
4. 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)
5. Reijers, H.A., Liman, S., van der Aalst, W.M.P.: Product-Based Workflow Design. Management Information Systems 20(1), 229–262 (2003)
6. Vanderfeesten, I., Reijers, H.A., van der Aalst, W.M.P.: Product-based Workflow Support. Information Systems 36(2), 517–535 (2011)
7. Redding, G.M., Dumas, M., ter Hofstede, A.H.M.: 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)
8. Redding, G.M., Dumas, M., ter Hofstede, A.H.M., Iordachescu, A.: A flexible, object-centric approach for business process modelling. In: Service Oriented Computing and Applications, pp. 1–11 (2009)
9. 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. LNBIP, vol. 29, pp. 197–210. Springer, Heidelberg (2009)
10. 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. LNBIP, vol. 43, pp. 29–41. Springer, Heidelberg (2010)
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), 205–244 (2011)
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), 19–46 (2011)
13. Künzle, V., Reichert, M.: PHILharmonicFlows: Research and Design Methodology. Technical Report UIB-2011-05. University of Ulm, Ulm, Germany (May 2011)

14. van der Aalst, W.M.P., Mans, R.S., Russell, N.C.: Workflow Support Using Proclets: Divide, Interact and Conquer. Bulletin of the IEEE Computer Society Technical Committee on Data Engineering 32(3), 16–22 (2009)
15. Liu, R., Bhattacharya, K., Wu, F.Y.: Modeling Business Contexture and Behavior Using Business Artifacts. In: Krogstie, J., Opdahl, A.L., Sindre, G. (eds.) CAiSE 2007 and WES 2007. LNCS, vol. 4495, pp. 324–339. Springer, Heidelberg (2007)
16. 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)
17. Vanderfeesten, I.: Product-Based Design and Support of Workflow Processes. Phd thesis. Eindhoven University of Technology (2009)
18. 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)
19. 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)