Object-aware Business Processes: Fundamental Requirements and their Support in Existing Approaches

Vera Künzle[1,*], Barbara Weber[2], and Manfred Reichert[1]

[1]Institute of Databases and Information Systems, Ulm University, Germany
{vera.kuenzle,manfred.reichert}@uni-ulm.de
[2]Institute of Computer Science, University of Innsbruck, Austria
{barbara.weber}@uibk.ac.at

Abstract. Despite the increasing maturity of process management technology not all business processes are adequately supported by it. In particular, support for *unstructured* and *knowledge-intensive* processes is missing, especially since they cannot be *straight-jacketed* into predefined activities. A common characteristic of these processes is the role of business objects and data as drivers for process modeling and enactment. This paper elicits fundamental requirements for effectively supporting such *object-aware processes*; i.e., their modeling, execution and monitoring. Based on these requirements, we evaluate imperative, declarative, and data-driven process support approaches and investigate how well they support object-aware processes. We consider a tight integration of process and data as major step towards further maturation of process management technology.

Keywords: Process-aware Information Systems, Object-aware Process Management, Data-driven Process Execution

1. Introduction

Business Process Management provides generic methods, concepts and techniques for designing, enacting, monitoring, and diagnosing *business processes* (van der Aalst & ter Hofstede & Weske, 2003). When using existing process management systems (PrMS) a business process is typically defined as set of activities representing business functions and having a specific ordering. What is done during activity execution is out of the control of the PrMS. Most PrMS consider activities as black-boxes in which application data is managed by invoked application components (except routing data and process variables). Whether an activity becomes activated during runtime depends on the state of other activities. Generally, a process requires a number of activities to be accomplished in order to terminate successfully. For end-users, PrMS provide process-oriented views (e.g., worklists.

Existing PrMS have been primarily designed for *highly structured*, *repetitive processes*. By contrast, for *unstructured* and *semi-structured processes* existing PrMS do not provide sufficient support (Silver, 2009). In particular, these processes are driven by user decisions and are *knowledge-intensive*; i.e., they cannot be expressed as a set of activities with specified order and work cannot be *straight-jacketed into activities* (van der Aalst & Weske & Grünbauer, 2005). Another limitation of PrMS is their insufficient process coordination support; i.e., process instances cannot be synchronized at a higher-level of abstraction. Consequently, all behavior relevant in a given context must be defined within one process model (van der Aalst et al., 2000; Müller & Reichert & Herbst, 2007). This, in turn, leads to a "contradiction between the way processes can be modeled and preferred work practice" (Sadiq et al., 2005, p.3). Finally, since application data is managed within black-box activities, integrated access on business processes and data cannot be provided. Due to these limitations many business applications (e.g., ERP systems) do not rely on PrMS, but are hard-coding process logic instead. Resulting applications are both complex to design and

costly to maintain, and even simple process changes require costly code adaptations and testing efforts.

To better understand which processes are handled well by existing PrMS and for which support is unsatisfactory, we conducted several case studies. Amongst others we analyzed business applications with hard-coded process logic; e.g., the processes as implemented in the human resource management system *Persis* and the reviewing system *Easychair* (Künzle & Reichert, 2009a; Künzle & Reichert, 2009b). Processes similar to the ones we evaluated can be found in many other fields like order handling, healthcare and release management (Müller & Reichert & Herbst, 2007). A major finding of all case studies was that data objects act as major *driver* for process specification and enactment. Consequently, process support requires *object-awareness*; i.e., business processes and business objects cannot be treated independently from each other. This has implications on the whole process lifecycle since PrMS should consider both object types and their inter-relations. Regarding its execution, on the one hand an *object-aware process* must be closely linked to relevant object instances; i.e., object attributes must process specific values to invoke certain activities or terminate process execution. On the other hand, an object-aware process does not only require certain data for executing a particular activity; i.e., it should be also able to dynamically react on data changes and newly emerging data. Consequently, process progress needs to be aligned with available object instances and their attribute values at runtime.

Regarding end-user functions provided by hard-coded business applications, in addition to a *process-oriented view,* there often exists a *data-oriented view* for managing and accessing data at any point in time. This includes overview tables (e.g., processed object instances) as well as activities that can be optionally executed. The latter are realized based on forms which can be invoked by authorized users to access or change object attributes regardless whether the respective activity is expected to happen during process execution. *Form-based activities* therefore constitute an important part for object management and process execution.

Our overall vision is to enable the modeling, execution and monitoring of object-aware business processes, which provide integrated access to business processes, data and application functions. We aim at the automated and model-driven generation of data-oriented views, process-oriented views and form-based activities at runtime. We also support the integration of arbitrary application components.

Based on the results of our case studies, we have already reported on fundamental challenges (Künzle & Reichert, 2009a; Künzle & Reichert, 2009b) and properties of PrMS integrating processes, data and users to provide the needed flexibility. In this paper, we elicit these properties in detail and introduce the requirements for effectively supporting *object-aware processes*. We then evaluate existing process support paradigms along these requirements and discuss which properties are well supported and in which cases additional research is needed to better capture the role of data as driver for process modeling and enactment. Overall, we believe that more profound research on object-aware processes will contribute to overcome some of the fundamental limitations known from existing PrMS.

The remainder of this paper is organized as follows. In Section 2 we introduce fundamental properties of object-aware processes and elaborate on the role of data for process enactment in more detail. In Section 3 we discuss major requirements to support object-aware process management along a realistic example. Section 4 discusses the outcomes we obtained when applying imperative, declarative, and current data-driven modeling approaches to tackle the identified requirements. We close with a summary and outlook in Section 5.

2. Properties of Object-aware Business Processes

We first describe fundamental properties of object-aware business processes along the main building blocks of existing PrMS (cf. Fig. 1). In this context we discuss why objects are the driver for modeling, executing and monitoring these processes.

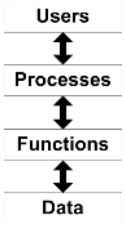

Fig. 1: Building blocks in existing PrMS

We first introduce an example of an object-aware process. As illustrated by Fig. 2, we use a (simplified) scenario from our case study in the area of human resource management.

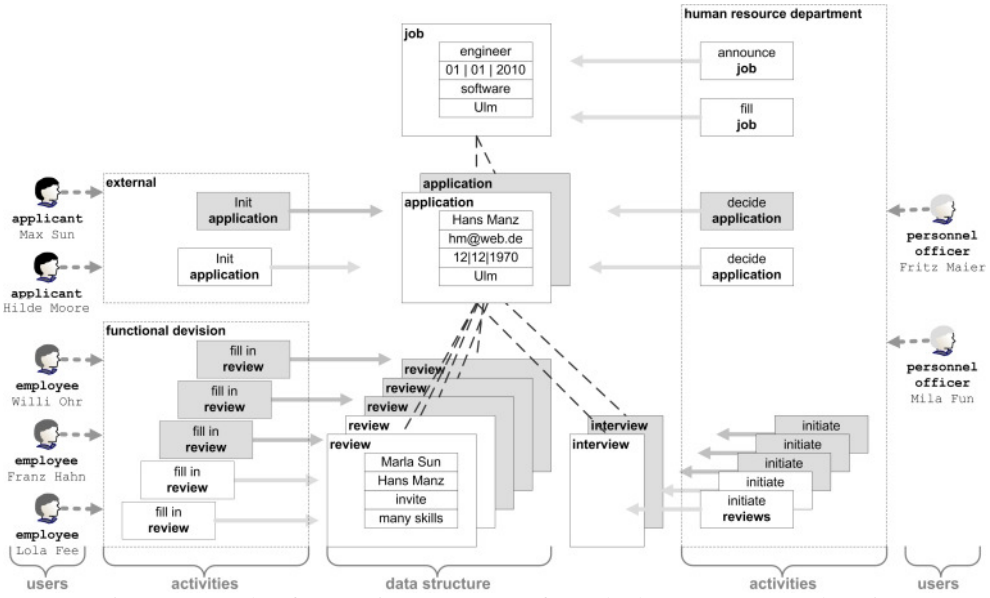

Fig. 2: Example of a recruitment process from the human resource domain

Recruitment process: In the context of recruitment `applicants` may apply for `job vacancies` via an Internet online form. Before an `applicant` can send her `application` to the respective company, specific information (e.g., `name`, `e-mail address`, `birthday`, `residence`) must be provided. Once the `application` has been submitted, the responsible `personnel officer` in the human resource department is notified. The overall process goal is to decide which `applicant` shall get the job. Since many `applicants` may apply for a `vacancy`, usually, different `personnel officers` handle the `applications`.
If an `application` is ineligible, the `applicant` is immediately rejected. Otherwise, `personnel officers` may request internal `reviews` for each `applicant`. Depending on the concerned `functional divisions`, the concrete number of `reviews` may differ from `application` to `application`. Corresponding `review` forms have to be filled by `employees` from `functional divisions` until a certain `deadline`. `Employees` may either *refuse* or *accept* the requested `review`. In the former case, they must provide a `reason`. Otherwise, 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 to the `personnel officer`. In the meanwhile, additional `applications` may have arrived; i.e., different `reviews` may be

requested or submitted at different points in time. In this context, the `personnel officer` may flag already evaluated `reviews`. The processing of the `application` 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`). Further, he does not have to wait for the arrival of all `reviews`; e.g., if a particular `employee` suggests `hiring` the `applicant`.

We analyzed additional object-aware processes as implemented in the conference reviewing system *Easychair*. Overall these processes are similar to the recruitment example: scientists may submit papers for a conference and PC chairs request reviews for them. Based on the results of these reviews, some papers are finally accepted, while others are not. In the following we first discuss fundamental properties of object-aware business processes. We illustrate them along our recruitment example. Later on we introduce a second example when discussing characteristic requirements for object-aware processes.

2.1 Data

All scenarios we analyzed in our case studies are characterized by a tight integration of process and data: i.e., besides a process-oriented view (e.g., worklists) there exists a data-oriented view that enables end-users to access data at any point in time given the required authorizations. As illustrated in Fig. 3a, data is managed based on *object types* which are *related* to each other. Each object type comprises a set of *attributes*. Object types, their attributes, and their inter-relations form a *data structure*.
At runtime the different object types comprise a varying number of inter-related *object instances*, whereby the concrete number can be restricted by lower and upper bounds (i.e., *cardinalities*). Furthermore, object instances of the same object type may differ in both their *attribute values* and *relations* to each other (cf. Fig. 3b); e.g., for one application two reviews and for another one three reviews might be requested. We denote an object instance which is directly or transitively referenced by another one as *higher-level* object instance (e.g., an application is a higher-level object instance of a set of reviews). By contrast, an object instance which directly or transitively references another object instance is denoted as *lower-level* object instance (e.g., reviews are lower-level object instances of to an application object).

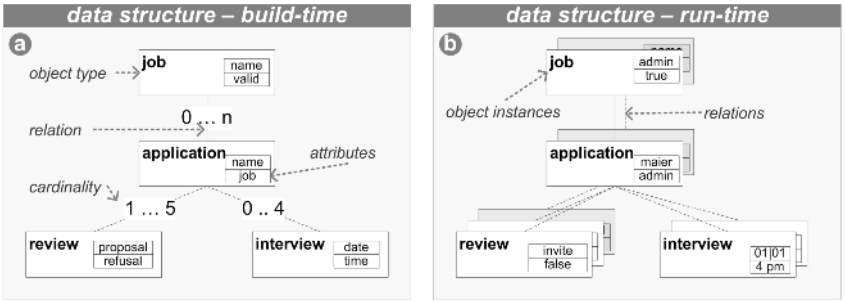

Fig. 3: Data structure at build- and runtime

2.2 Activities

Activities can be divided into *form-based* and *black-box* activities. While form-based activities provide *input fields* (e.g., text-fields or checkboxes) for writing and *data fields* for reading selected attribute values of object instances, black-box activities enable complex

computations or integration of advanced functionalities (e.g., sending e-mails or invoking web services).
Form-based activities can be further divided into instance-specific activities, batch activities and context-specific activities. *Instance-specific activities* correspond to exactly one object instance (cf. Fig. 4a). When executing such activity, attributes of that object instance can be read, written or updated using a form (e.g., the form an applicant can use for entering his `application` data). A *context-sensitive activity* additionally includes fields corresponding to higher-level or lower-level object instances (cf. Fig. 4b). When integrating lower-level object instances, usually, a collection of object instances is considered. For example, 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). Furthermore, `employees` may change the value for attribute `comment` of the `application` object instance. Finally, *batch activities* allow users to change a collection of object instances in one go, i.e., attribute values are assigned to all selected object instances using one form (cf. Fig. 4c); e.g., a `personnel officer` might want to flag a collection of `reviews` as "evaluated" in one go. Or 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.

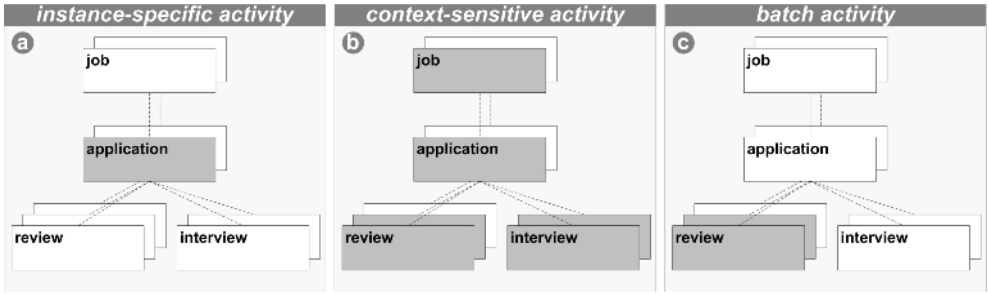

Fig. 4: Basic types of form-based activities

Object-aware processes provide a process-oriented view in which *mandatory activities* are assigned to responsible users at the right point in time as well as a data-oriented view in which object instances can be accessed at any point in time using *optional activities*.

2.3 Processes

In addition to the *structure* of object types (i.e., their attributes and inter-relations), their *behavior* needs to be considered. Basically, object behavior determines in which order and by whom object attributes have to be (mandatorily) written, and what valid attribute settings are. Thereby, for each object type a set of *states* needs to be defined of which each postulates specific attribute values to be set. More precisely, a state can be expressed in terms of a particular data condition referring to a number of attributes of the respective object type. As example consider object type `review` and its states as depicted in Fig. 5. In state `accepted` a value for attribute `appraisal` must be assigned and the value of attribute `proposal` must either be set to 'reject' or 'invite'. Further, object behavior restricts possible *state* sequences using *transitions*. In particular, for each state possible successor states are defined. Consider the processing of a `review` in Fig. 5c: First, the `review` must be `initiated` by a `personnel officer`. Following this, the `employee` may either `refuse` or `accept` the `review`. In the latter case, he `submits` the `review` back to the `personnel officer`.

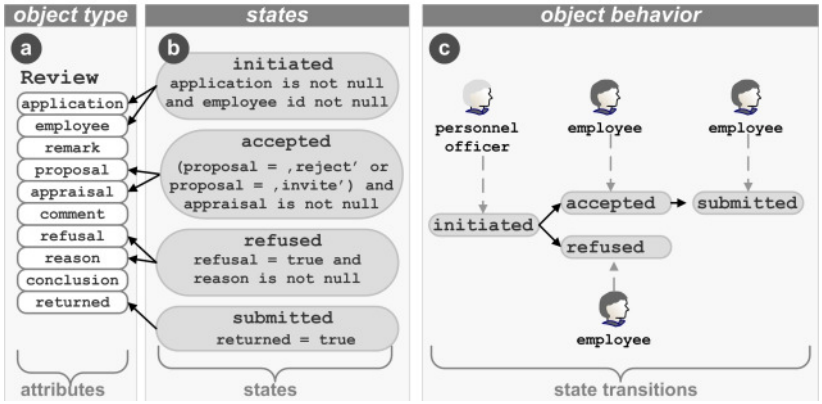

Fig. 5: Object behavior defined based on states and transitions

At runtime, for each object type multiple object instances may exist (cf. Fig. 6a). These object instances may be created or deleted at arbitrary points in time; i.e., the *data structure* dynamically evolves depending on the type and number of created object instances as well as on their relations. Consequently, the individual object instances may be in different states; e.g., several `reviews` may be requested for a particular `applicant`. While one of them might be in state `initiated`, others might have already reached state `submitted`. Taking the behavior of individual object instances into account, we obtain a complex *process structure* in correspondence to the given data structure (cf. Fig. 6b).

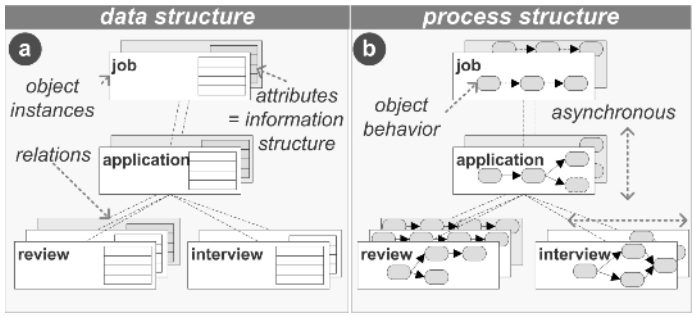

Fig. 6: Data structure and corresponding process structure

Generally, complex *processes* result from the *interactions between instances of different object types*:

Object interactions within the recruitment process (cf. Fig. 7): A `personnel officer` announces a `job`. Following this, `applicants` may init `applications` for this `job`. After submitting an `application`, the `personnel officer` requests internal `reviews` for it. If an `employee` acting as referee proposes to invite the `applicant` the `personnel officer` conducts an `interview`. Based on the results of `reviews` and `interviews` the `personnel officer` decides in the `application`. In case of acceptance the `applicant` is hired.

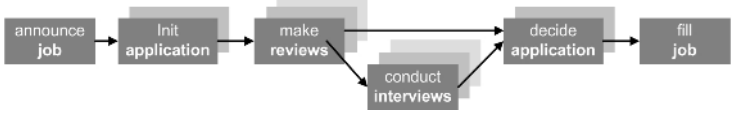

Fig. 7: Process definition based on object interactions

As can be seen from this scenario, behavior of individual object instances (of same and of different type) needs to be coordinated considering their inter-relations as well as their asynchronous execution. In this context, the dynamic number of object instances must be taken into account (cf. Fig. 8); e.g., a `personnel officer` is not allowed to read the result of a `review` before the `employee` has `submitted` it. Further, the `personnel officer` may only reject an `application` immediately if all `reviewers` propose its rejection.

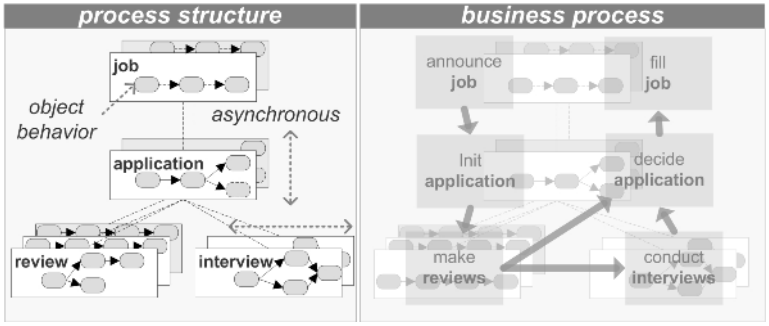

Fig. 8. Process structure at build- and runtime

Activity execution depends on the *behavior* of the processed object instances as well as on their *inter-relations* and thus requires modeling at two abstraction levels.

2.4 User integration

Taking the data-oriented view users may optionally access object instances at any point in time and create, read and write them (i.e., executing *optional activities*). The process-oriented view, in turn, provides worklists; i.e., it allows assigning mandatory activities to the right users at the right point in time. If mandatorily required information is missing during process execution, a form-based activity is automatically generated by the system and added to the worklist of the responsible user; e.g., if a `review` needs to be filled out by an `employee` a form-based activity with input fields for attributes `proposal` and `appraisal` is generated.

2.5 Monitoring

The overall state of the process, which is defined in terms of interactions between object instances, should be made transparent. Generally, monitoring the overall process state should provide an aggregated view on the corresponding object instances (cf. Fig. 9). Since each object instance may be in a different state, object behavior of each involved object instance needs to be considered in a fine-grained manner.

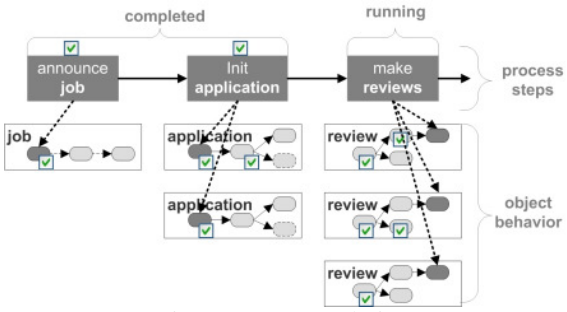

Fig. 9: Aggregated view

3. Requirements for the IT Support of Object-aware Processes

This section elicits fundamental requirements for the support of object-aware processes. We categorize these requirements along the main building blocks of a PrMS (cf. Fig. 10). We believe that the poor integration of these building blocks in existing PrMS constitutes a major reason for the insufficient support of object-aware processes and their properties in existing PrMS.

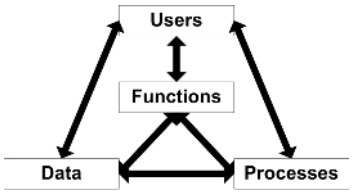

Fig. 10: Integration of building blocks

We gathered these requirements in case studies in which we analyzed processes and objects of applications from human resource management, paper reviewing, order handling, and healthcare (cf. Fig. 11). Though these requirements are not complete in the sense that they cover all aspects of the object and process lifecycle, their fulfilment is indispensable for enabling the aforementioned properties as well as the automatic generation of runtime components like worklists, overview lists and form-based activities.

data		activities		processes		user integration	
R1	data integration	R5	form-based activities	R10	object behavior	R16	data authorization
R2	access on data	R6	black-box activities	R11	object interactions	R17	process authorization
R3	cardinalities	R7	variable granularity	R12	process-oriented view	R18	differentiation
R4	mandatory information	R8	mandatory and optional activities	R13	flexible process execution	R19	vertical authorization
		R9	control-flow within user forms	R14	re-execution of activities		**monitoring**
				R15	explicit user decisions	R20	aggregated view

Fig. 11: Fundamental requirements for object-aware processes

We illustrate the requirements along the introduced recruitment example. We further undergird them using an order handling process in which customers may order different products in an online shop; after such an order is submitted and the resulting bill is paid the seller initiates the shipping of the ordered products. To distinguish between the two scenarios for each requirement we annotate the given examples with 'a' (recruitment process) and 'b' (order handling process) respectively.

3.1 Data

R1 (Data integration). Data should be managed in terms of object types comprising object attributes and relations to other object types.

Example 1a: For each `job` a set of `applications` may be created. For each `application`, in turn, several `reviews` may exist, each having attributes like `application`, `employee`, `remark`, `proposal` and `appraisal`.

Example 1b: A `shop` offers different `products`. A particular `order` is always directed to one `shop` and may comprise several `products` offered by this `shop`. Important attributes of a `product` include its `label`, `price` and `place of manufacture`.

R2 (Access to data). Access to data should be granted at any point given the required authorizations; i.e., not only during the execution of a particular activity.

Example 2a: The `personnel officer` should be allowed to access an `application` even if

no activity is contained in his worklist. Furthermore, if an applicant contacts him to change her address, he should be allowed to update corresponding attributes.

Example 2b: The customer should be allowed to access an order even if no activity is contained in his worklist. For example, he should be allowed to change attribute delivery address even if he has already submitted his order.

R3 (Cardinalities). It should be possible to restrict relations between object instances through *cardinality constraints*.

Example 3a: For each application at least one and at most five reviews may be requested. While for an application two reviews exists, for another one three reviews may be requested.

Example 3b: For each order exactly one payment method must be specified.

R4 (Mandatory information). To reach a particular object instance state from the current one, certain attribute values must be set. For this, a form-based activity with mandatory input fields needs to be assigned to the worklists of authorized users. When executing it, specific input fields referring to mandatorily required attributes have to be filled. Other input fields may be optionally set.

Example 4a: The mandatory form-based activity for requesting a review is accomplished by a personnel officer. When working on this activity, values for object attributes application and employee are mandatory, while other attributes (e.g., remark) are optional.

Example 4b: The mandatory form-based activity for initiating the shipping is accomplished by the seller. When executing this activity, values for object attributes weight and height are mandatory, while other attributes (e.g., express transmission) are optional.

3.2 Activities

R5 (Form-based activities). A form-based activity comprises a set of atomic *actions.* Each of them corresponds to either an *input field* for writing or a *data field* for reading the value of an object attribute. Which attributes may be written or read in a particular form-based activity may depend on the user invoking this activity and the state of the object instance. Consequently, a high number of form variants exists. Since it is costly to implement them all, it should be possible to automatically generate form-based activities at runtime.

Example 5a: An employee needs a form-based activity to edit a review; i.e., to assign values to attributes proposal and appraisal. In addition, she can access attributes of the application to which the review refers. As soon as she has submitted her review she may only read attributes proposal and appraisal. If the responsible personnel officer wants to edit the review at the same point in time, he may only write attribute remark.

Example 5b: A customer needs a form-based activity to create an order; e.g., to assign values to attributes shipping address and order date. In addition, the customer may add several products to the order. However, as soon as he has submitted the order he may only change attribute shipping address.

R6 (Black-box activities). To ensure proper execution of black-box activities, we need to be able to define pre-conditions on attribute values of processed object instances. If their

input parameters belong to different object instances, their inter-relationships should be controllable. Opposed to form-based activities, which should be automatically generated by the runtime system (cf. R5), for each black-box activity an implementation is required.

Example 6a: Consider a *black-box activity* which compares the `skills` of an `applicant` with the `requirements` of the `job`. This activity requires input parameters referring to (objects) `application`, `skills`, `job`, and `job requirements`. It should be ensured that the `job` is exactly the one for which the `applicant` applies. Finally, the `requirements` must comply to the ones of the `job` and the `skills` relate to the ones of the `applicant`.

Example 6b: Consider a *black-box activity* which calculates the `total price` of an `order`. This activity requires input parameters referring to the `order`, `products`, and `shipping method`.

R7 (Variable granularity). As discussed, support for instance-specific, context-sensitive, and batch activities is required. Regarding *instance-specific* activities, all actions refer to attributes of one particular object instance, whereas *context-sensitive* activities comprise actions referring to different, but related object instances (of potentially different type). Since *batch activities* involve several object instances of the same type, for them each action corresponds to exactly one attribute. Consequently, the attribute value must be assigned to all referred object instances. Depending on their preference, users should be allowed to freely choose the most suitable activity type for achieving a particular goal. Finally, executing several black-box activities in one go should be supported.

Example 7a: An `employee` may choose a *context-sensitive activity* to edit a `review`; i.e., to write attributes `proposal` and `appraisal)` and to read attributes of the `application`. A `personnel officer`, in turn, may choose a *batch activity* to update several `reviews` in one go; e.g., to set attribute `evaluated` for all `reviews` relating to an `application`.

Example 7b: A `customer` may choose a *context-sensitive activity* to edit an `order`. A `seller`, in turn, may choose a *batch activity* to initiate the `shipping` of several `products` with same `weight` and `height` in one go.

R8 (Mandatory and optional activities). Depending on the state of object instances certain activities are mandatory for progressing with the control-flow. At the same time, users should be allowed to optionally execute additional activities (e.g., to write certain attributes even if they are not required at the moment).

Example 8a (Mandatory activity): After a `review` has been initiated, the assigned `employee` either must provide or refuse the `review`; i.e., a form-based activity needs to be mandatorily performed. **(Optional activity)** After a `review` request has been triggered by the `personnel officer` (i.e., attributes `application` and `employee` are set), he should be further allowed to update object attribute `remark`. Generally, he may update certain object attributes, while an `employee` fills in a `review`.

Example 8b (Mandatory activity): After an `order` is submitted and the corresponding `bill` is paid, the `seller` must initiate the `shipping`. **(Optional activity)** However, as long as the `shipping` has not been initiated by the `seller`, the `customer` may optionally change the `shipping address`.

R9 (Control-flow within user forms). Whether certain object attributes are mandatory when processing a particular activity might depend on other object attribute values; i.e., when filling a form certain attributes might become mandatory on-the-fly.

Example 9a: When an `employee` receives a `review` request, she either fills the `review` form as requested by the `personnel officer` or refuses this task. Consequently, a value needs to be assigned to at least one of the two attributes `proposal` or `refusal`. If the `employee` decides to set attribute `proposal`, additional object attributes will become mandatory; e.g., if she wants to *invite* the `applicant` for an `interview` she has to set attribute `appraisal` as well. This is not required if she assigns value *reject* to attribute `proposal`.

Example 9b: If a `customer` chooses `express delivery` he must additionally specify a `delivery date`.

3.3 Processes

R10 (Object behavior). It should be possible to determine in which order and by whom object attributes have to be (mandatorily) written, and what valid attribute value settings are. In addition, when executing black-box activities the involved object instances need to be in certain states. Consequently, for each object type its behavior should be definable in terms of states and transitions. In particular, it should be possible to drive process execution based on data and to dynamically react upon attribute value changes. Therefore, it is crucial to map states to attribute values.

Example 10a: An `employee` may only provide a `review` for a particular `application` if the state of the `review` is `initiated`. This state is automatically entered as soon as values for attributes `employee` and `application` are assigned.

Example 10b: A `customer` may only order a `product` if the state of the `product` is `stocked`.

R11 (Object interactions). Generally, a process deals with a varying number of object instances of the same and of different object types. In addition, for each processed object instance its behavior must be considered. In this context, it should be possible to process instances in a loosely coupled manner, i.e., *concurrently* to each other and to *synchronize* their execution where needed. More precisely, any process modeling paradigm should allow defining processes with a dynamic number of object instances. First, it should be possible to make the creation of a particular object instance dependent on the state of the related higher-level object instance (*creation dependency*). Second, during the execution of a higher-level object instance, aggregated information from its lower-level object instances should be accessible; amongst others this requires the aggregation of attribute values from lower-level object instances (*aggregative information*) (van der Aalst et al., 2000). Third, the executions of different process instances may be mutually dependent (Müller & Reichert & Herbst, 2007; van der Aalst et al., 2000); whether an object instance may switch to a certain state depends on the state of another object instance (*execution dependency*). Consequently, processes should be defined in terms of object interactions. Additionally, the integration of black-box activities should possible.

Example 11a: 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 when other initiated reviews are submitted. Finally, if the personnel officer decides to hire one of the applicants, all others must be rejected (*execution dependency*). In this context, black-box activities become relevant as well (e.g., sending an acknowledgement to an applicant after rejecting his application or comparing the skills of the applicant with the requirements of the job before reviews are initiated.

Example 11b: A seller must not initiate the shipping as long as the corresponding bill has not been paid by the customer (*creation dependency*). A customer should be able to access information about ordered products (*aggregative information*). Finally, if an *ordered* product is not in stock, a reorder by the manufacturer should be initiated (*execution dependency*).

R12 (Process-oriented view). During process execution some activities must be mandatorily executed while others are optional. To ensure that mandatory activities are executed at the right point in time, they must be assigned to the worklists of authorized users.

Example 12a: When a review enters state accepted (i.e., its request was accepted by an employee), a workitem is added to her worklist. When processing it, she has to mandatorily set attributes proposal and appraisal. Furthermore, a personnel officer may optionally change attribute remark of the review.

Example 12b: When an order enters state submitted, a work item is added to the worklist of the customer who has to pay the corresponding bill.

R13 (Flexible process execution). Mandatory activities are obligatory for process execution; i.e., they enforce the setting of object attribute values as required for progressing with the process. In principle, respective attributes can be also set up front by executing optional activities; i.e., before the mandatory activity normally writing this attribute becomes activated. In the latter case, the mandatory activity can be automatically skipped when it is activated.

Example 13a: After the personnel officer has set values for object attributes application and employee, a mandatory activity for filling the review form is assigned to the specified employee. Even if the personnel officer has not completed this review request (i.e., he has specified the respective employee, but not the corresponding application), the selected employee may optionally edit certain attributes of the review. For example, he may refuse the review and set object attribute comment. If the personnel officer has assigned the application, the mandatory activity for providing the review is automatically skipped.

Example 13b: The seller may initiate the shipping before the bill is paid by the customer. If the bill is paid afterwards the mandatory activity for initiating the shipping will be automatically skipped.

R14 (Re-execution of activities). Users should be allowed to re-execute a particular activity (i.e., to update its attributes), even if all mandatory object attributes have been already set.

Example 14a: An employee may change his proposal arbitrarily often until he explicitly agrees to submit the review to the personnel officer.

Example 14b: A customer may change the order arbitrarily often until he explicitly agrees to submit the order to the seller.

R15 (Explicit user decisions). Generally, different ways for reaching a process goal may exist. Usually, the selection between such alternative execution paths is based on history data; i.e., on completed activities and available process-relevant data. In our context, this selection might be also based on explicit user decisions.

Example 15a: A personnel officer may decide whether reviews are requested for a particular application. Only if a review is initiated, a mandatory activity for finalizing the reviews is invoked; i.e., execution of the second activity depends on user a decision.

Example 15b: A customer may decide how many different products he wants to order.

3.4 User Integration

R16 (Data authorization). To provide access to data at any point in time, we need to define permissions for creating and deleting object instances as well as for reading/writing their attributes. However, attribute changes contradicting to object behavior should be prevented. For this, the progress of the process has to be taken into account when granting permissions to change objects attributes (Botha, 2002; Wu et al., 2002). Otherwise, if committed attribute values were changed afterwards, object instance state would have to be adjusted to cope with dirty reads. Generally, data permissions should be made dependable on the states as captured by object behavior. This is particularly challenging for context-sensitive and batch activities, since attribute changes have to be valid for all selected instances.

Example 16a: After submitting her review, the employee still may change her comment. However, attribute proposal must not be changed anymore. The personnel officer might have already performed the proposed action. Further, using a batch activity, he may flag several reviews in one go (i.e., assign value *true* to object attribute evaluated). Finally, it must be ensured that the employee can only access reviews she submitted before.

Example 16b: After submitting the order the customer may still change the shipping address. However, since the total price of the order depends on the ordered products, the latter must not be changed anymore.

R17 (Process authorization). For each mandatory activity at least one user or user role should be assigned to it at runtime. Regarding a form-based activity, each user who may execute it must have the permissions for reading/writing corresponding attribute values (Botha, 2002).

Example 17a: An employee who has to fill a review also needs the permissions to set attributes proposal, appraisal, refusal, and appraisal.

Example 17b: A customer who wants to place an order needs the permissions to set corresponding attributes (e.g., shipping address).

R18 (Differentiating authorization and user assignment). When executing mandatory activities particular object attributes have to be set. To determine which user shall execute a pending mandatory activity, her permissions for writing object attributes need to be evaluated. 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. In particular, a mandatory activity should be only added to the worklists of users having "mandatory permissions". Users with "optional permissions", in turn, may change the corresponding attributes when executing optional activities.

Example 18a: An `employee` must write attribute `proposal` if she has accepted the `review` request. 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`.

Example 18b: The `seller` must write attribute `price` before a `product` can be ordered by a `customer`. However, the `manager` of the `shop` may optionally set this attribute as well. The mandatory activity for filling the `product` form, in turn, should be only assigned to the `seller`, but not to the `manager`.

R19 (Vertical authorization and user assignment). Usually, human activities are associated with actor expressions (e.g., user roles). We denote this as *horizontal authorization*. Users who may work on respective activities are determined at runtime based on these expressions. For object-aware processes, however, the selection of potential actors should not only depend on the activity itself, but also on the object instance processed by it (Rosemann & zur Mühlen, 1997; Rosemann & zur Mühlen, 2004). We denote this as *vertical authorization*.

Example 19a: A `personnel officer` may perform activity `make decision` only for `applications` for which the `name` of `applicants` starts with a letter between 'A' and 'L', while another `officer` may perform this activity for `applicants` whose `name` starts with a letter between 'M' und 'Z'.

Example 19b: An `employee` of the `seller` may initiate the `shipping` for `products` having a `price` lower than 500 euro, while another `employee` of the `seller` may perform this activity for `products` whose `price` is higher than 500 euro.

3.5 Monitoring

R20 (Aggregated View). Process monitoring should provide an aggregated view of all object instances involved in a process as well as their interdependencies.

Example 20a: Consider the decision about a particular `application` as expressed with attribute `decision` (based on the results of the `reviews`). While some `reviews` might have been already submitted, others might be still processed by an `employee`. Further, additional `reviews` might be requested at a later point in time.

Example 20b: Consider different `orders` referring to the same `product`. While some `orders` may have already been completed (i.e., the `shipping` is completed), others might be still processed. Furthermore, additional `orders` might be requested at a later point in time.

4. Evaluating Existing Process Support Paradigms

We evaluate existing approaches along the requirements introduced in Section 3. We focus on imperative, declarative and data-driven process support paradigms. Other approaches, which are related to our requirements, constitute **extensions** of these paradigms. We base our evaluation on the main characteristics of the approaches. As illustrated in Fig. 12, for this purpose we number them consecutively using letters 'A' to 'I'.

	characteristic	imperative	declarative	data-driven
A	hidden information flows	X	X	
B	flow-based activation	X		
C	actor expressions	X	X	
D	fixed granularity of actvities	X	X	
E	arbitrary granularity of processes	X	X	X
F	constraint-based activation		X	X
G	data integration			X
H	data-driven activation			X
I	advanced role concept			X

Fig. 12: Characteristics of existing approaches

As illustrated in Fig. 13, only limited support is provided in respect to the support of object-aware processes.

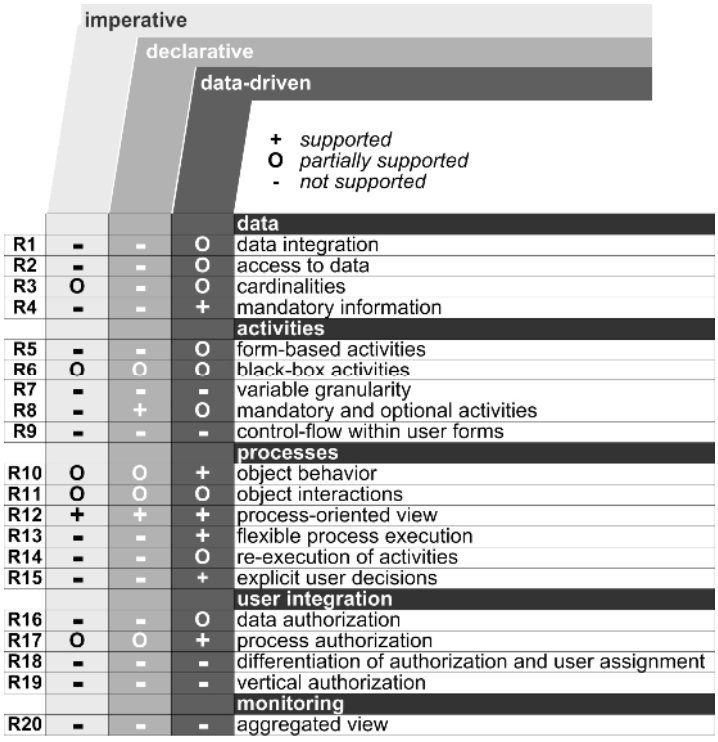

Fig. 13: Evaluating existing paradigms

4.1 Imperative Approaches

There is a long tradition of modeling business processes in an imperative way. Usually, processes are specified as directed graphs (Weber & Reichert & Rinderle-Ma, 2008). Process steps correspond to different activities which are connected to express precedence relations (cf. Fig. 14). For control flow modeling a number of patterns exists, e.g., sequential, alternative and parallel routing, and loop backs (van der Aalst et al., 2003).

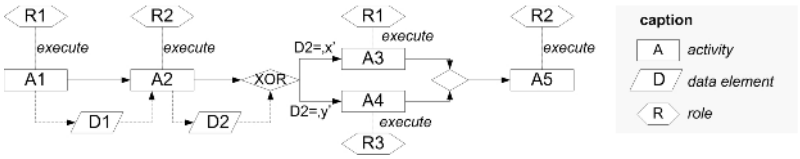

Fig. 14: Imperative modeling approach

Imperative approaches only provide limited support regarding the requirements raised by object-aware processes. In the following, we discuss the imperative approach along its main characteristics which are numbered using letters 'A' to 'E': hidden information flows (A), flow-based activation of activities (B), actor expressions (C), fixed activity granularity (D), and arbitrary process granularity (E). We evaluate the requirements along these characteristics (cf. Fig. 5).

imperative approach

+	*supported*	A	*hidden information flows*
O	*partially supported*	B	*flow-based activation*
-	*not supported*	C	*actor expressions*
		D	*fixed granularity of actvities*
		E	*arbitrary granularity of processes*

			characteristic
		data	
R1	-	data integration	A
R2	-	access on data	B
R3	O	cardinalities	E
R4	-	mandatory information	A
		activities	
R5	-	form-based activities	A,E
R6	O	black-box activities	A
R7	-	variable granularity	D
R8	-	mandatory and optional activities	B
R9	-	control-flow within user forms	A
		processes	
R10	O	object behavior	E
R11	O	object interactions	E
R12	+	process-oriented view	C
R13	-	flexible process execution	B
R14	-	re-execution of activities	B
R15	-	explicit user decisions	B
		user integration	
R16	-	data authorization	B
R17	O	process authorization	A
R18	-	differentiation of authorization and user assignment	B
R19	-	vertical authorization	A
		monitoring	
R20	-	aggregated view	A

Fig. 15: Evaluating the imperative approach

Hidden information flows (A)
Usually, imperative approaches enable the explicit definition of data flows between activities based on atomic data elements. The latter are connected with activities (and their parameters) or with routing conditions (cf. Fig. 14). Activities themselves are regarded as black-boxes; i.e., application data is usually managed within invoked applications. In particular, there is no explicit link between activities and the object instances they manipulate.
Data. *Data integration* based on object types, attributes and relations is not supported (i.e., R1 is not met); i.e., the PrMS is unaware of the object instances being accessed during process execution. Further, it cannot control whether required data changes are actually accomplished; i.e., *mandatory information* cannot be realized (i.e., R4 is not met).
Activities. A particular activity usually requires data that has to be provided by preceding activities. Ideally, this is accomplished according to the modeled data flow. If accessed data elements are not written by previous activities, process execution might be blocked. Op-

posed to this, if consumed data is not explicitly considered in the modeled data flow, the process instance might proceed though required data is missing. Consequently, it is not possible to automatically invoke a *form-based activity* for requesting missing data from users. Furthermore, the *internal control-flow* of a form-based activity cannot be expressed (i.e., R5 and R9 are not met). Regarding *black-box activities*, in turn, different parameters may belong to attributes of different object instances. However, we cannot control the relations between the object instances to which the parameters of an activity refer (i.e., R6 is not fully supported).

User integration. It cannot be guaranteed that users who own the permission for executing an activity are also authorized to read/write attributes processed by this activity. Thus, *process authorization* is only enabled at activity level (i.e., R17 is not fully met). *Vertical authorization* (i.e., assigning different permissions for the same activity depending on the state of the processed object instance) is not supported (i.e., R19 is not met).

Monitoring. Due to hidden information flows one cannot provide an *aggregated view* on processed object instances (i.e., R20 is not met).

Flow-based activation of activities (B)

Each process step corresponds to one activity being mandatory for process execution (except it is contained in a conditional path). Moreover, activity activation depends on the state of preceding activities, i.e., a particular activity becomes enabled if its preceding activities are completed or cannot be executed anymore (except loop backs).

Data. *Data access* is only possible when executing activities according to the defined control-flow; i.e., data cannot be accessed independently from process execution (i.e., R2 is not fully met).

Activities. There is no support for *optional activities* enabling data access at any point in time (i.e., R8 is not met).

Processes. Since the activation of an activity solely depends on the completion of other activities, *flexible process execution* (e.g., skipping certain activities if required output data is already available) is not explicitly supported (i.e., R13 is not met).

There is no direct support for *re-executing an activity* as long as the user does not commit its completion (i.e., R14 is not met). Since activity activation only depends on the completion of other activities there is no explicit support for *user decisions* (i.e., R15 is not met).

User integration. Since data can only be accessed when executing mandatory activities, imperative approaches lack sophisticated support for coordinating processed data and executed processes. Thus, neither proper *data authorization* (i.e., R16 is not met) nor the *distinction between process and data authorization* are considered (i.e., R18 is not met).

Actor expressions (C)

Human activities are associated with actor expressions (e.g., roles). Based on these expressions activities can be assigned to authorized users at runtime. Further, when a human activity becomes enabled, a corresponding work item is added to worklists of authorized users.

User Integration. A *process-oriented view* is provided enabling the execution of activities by the right users at the right point in time (i.e., R12 is met).

Fixed activity granularity (D)

Activities are associated with a specific business function implemented at buildtime, thus having a fixed granularity.

Activities. Support of different work practices by enabling instance-specific, context-sensitive and batch activities is not provided; i.e.; a *variable granularity of activities* is not possible (i.e., R7 is not met).

Arbitrary process granularity (E)
Imperative approaches do not distinguish between the behavior of individual object instances and the processes coordinating them. Business functions associated with the activities of a process model can be implemented at different levels of granularity. While certain activities are only processing one object instance, others may process several object instances of same/different type. Generally, there exists no elaborated modeling methodology giving advice on the number of object types to be handled within one process definition. Consequently, a process is either defined at a **coarse- or fine-grained level**.
Data. When applying a **coarse-grained process modeling style**, an activity may be linked to several object types. Since object flows are hidden, it is difficult to ensure consistency between process and data modeling. In particular, when modeling a process the creation of object instances cannot be restricted to a varying and dynamic number of object instances based on *cardinalities* (i.e., R3 is not fully met).
Activities. When applying a **fine-grained process modeling style**, activity execution is associated with exactly one process instance. Consequently, only instance-specific activities can be realized, but no context-sensitive or batch activities. Further, it is not possible to automatically generate a *form-based activity* if required data is missing (i.e., R5 is not met).
Processes. When choosing a **fine-grained modeling style** each process definition is aligned with exactly one object type. This way one can ensure that corresponding process instances access one particular object instance of the respective object type at runtime. For this purpose, either one data element for routing the object-ID or several data elements (of which each relates to one attribute) are added to the process model. The activity-centred paradigm of imperative approaches is not appropriate for supporting *object behavior* (i.e., R10 is not fully met). Hidden information flows and the flow-based activation of activities inhibit the dynamic adaptation of the control-flow based on available data. Further, interdependencies between process models cannot be expressed and process instances are executed in isolation to each other. Thus, the definition of *interactions between object instances* is not captured (i.e., R11 is not met). To deal with these requirements the following extensions exist.

Extension 1 (Proclets). Proclets enable process communication and asynchronous process coordination based on message exchanges (van der Aalst et al., 2000). Using Proclets, however, process coordination cannot be explicitly based on the underlying data structure or on specific data element values. Further, messages can only be exchanged at specific points during process execution (e.g., based on send/receive activities).

Extension 2 (Data-driven process structures). In Corepro, the coordination of processes instances can be based on the relations between involved object instances. Thereby, synchronization constraints are defined based on object states (Müller & Reichert & Herbst, 2007). However, states are not connected to object attributes. Further, each invoked process is defined imperatively. This leads to the discussed disadvantages like hidden information flows, fixed activity granularity, and arbitrary process granularity.

A **coarse-grained modeling style**, in turn, prohibits **fine-grained control** in respect to *object type behavior* (i.e., R10 is not met). Processes are only defined based on activities and interactions between *object instances* are not considered (i.e., R11 is not met). An interesting extension are object life cycles.

Extension 3 (Object life cycles). To integrate object behavior with processes, an extension of the imperative approach based on object life cycles (OLC) has been proposed (Gerede & Su, 2007; Küster & Ryndina & Gall, 2007; Redding et al., 2007; Nigam & Caswell, 2003; Liu & Bhattacharya & Wu, 2007). In particular, the introduction of OLCs target at consistency between process models and process data. For this purpose, an OLC

defines the states of an object and the transitions between them in a *separate model*. Activities, in turn, are associated with pre-/post-conditions in relation to objects states. However, states are not mapped to attribute values. Consequently, if certain pre-conditions cannot be met during runtime, it is not possible to dynamically react to this. Neither relations between object types nor the varying number of object instances are considered.

Process support involving different object instances can be provided by using sub-processes. Thereby, a sub-process is associated with an activity of the higher-level process instance. However, it is not possible to define relations and synchronization dependencies between different sub-process definitions of the same level. Consequently, processes which are defined based on *object interactions* are not supported (i.e., R11 is not met). This limitation can be addressed by multiple-instantiation patterns (van der Aalst et al., 2003), which allow specifying the number of instances for a respective activity either at build- or runtime.

Extension 4 (Multiple-instantiation patterns). Regarding multiple-instance activity patterns, new sub-process instances can only be created as long as subsequent activities have not been started; e.g., additional `reviews` can be instantiated as long as the corresponding `application` is not further processed. Thus, lower-level process instances (i.e., sub-process instances) can only be created at a specific point during the execution of the higher-level process instance. Furthermore, except for one variant of the multiple-instantiation pattern, sub-process instances cannot be executed asynchronously to the higher-level process instance. Using multiple-instantiation patterns with synchronization (cf. Fig. 16a), each sub-process instance must either be completed or skipped before subsequent activities of the higher-level process instance can be triggered. Using multiple-instantiation without synchronization, in turn, the results of these sub-process executions are not relevant for progressing the higher-level process instance (cf. Fig. 16b). Finally, interdependencies between sub-processes, which are executed asynchronously to each other (cf. Fig. 16c), cannot be taken into account.

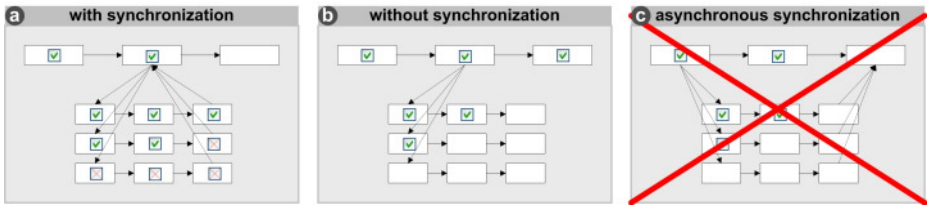

Fig. 16: Sub-process execution based on multiple-instantiation

4.2 Declarative Approaches

Declarative approaches suggest a fundamentally different way of describing business processes (van der Aalst & Pesic, 2006). While imperative models specify how things have to be done, declarative approaches only focus on the logic that governs the interplay of actions in the process by describing (1) the *activities* that can be performed and (2) the *constraints* prohibiting undesired behavior. In the example from Fig. 17, activities A2 and A3 can only be executed after finishing A1 (Pesic, 2008). Finally, A2 and A3 are mutually exclusive.

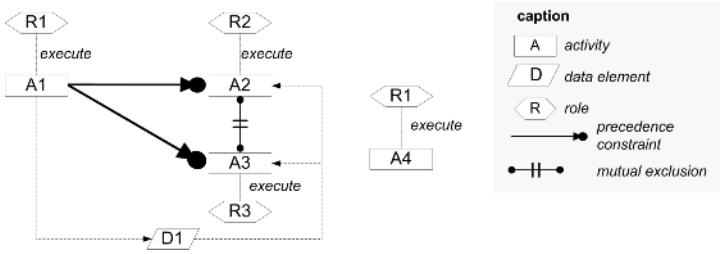

Fig. 17: Declarative modeling approach

Declarative approaches provide limited support for object-aware processes. Many of their characteristics correspond to the ones of imperative approaches: hidden information flows (A), actor expressions (C), fixed activity granularity (D), and arbitrary process granularity (E). However, they differ in respect to activity activation. While imperative approaches pursue a flow-based activation (B), declarative approaches rely on a constraint-based activation (F) (cf. Fig. 18). This leads to better support of optional activities in comparison to imperative approaches. However, the extensions introduced for imperative approaches are not applicable to declarative ones. To avoid redundancies, we only discuss the main differences between imperative and declarative approach.

declarative approach

\+ *supported*
O *partially supported*
\- *not supported*

A *hidden information flows*
C *actor expressions*
D *fixed granularity of actvities*
E *arbitrary granularity of processes*
F *constraint-based activation*

			characteristic
		data	
R1	-	data integration	A
R2	-	access on data	F
R3	-	cardinalities	E
R4	-	mandatory information	A
		activities	
R5	-	form-based activities	A,E
R6	O	black-box activities	A
R7	-	variable granularity	D
R8	+	mandatory and optional activities	F
R9	-	control-flow within user forms	A
		processes	
R10	O	object behavior	E
R11	O	object interactions	E
R12	+	process-oriented view	C
R13	-	flexible process execution	F
R14	-	re-execution of activities	F
R15	-	explicit user decisions	F
		user integration	
R16	-	data authorization	F
R17	O	process authorization	A
R18	-	differentiation of authorization and user assignment	F
R19	-	vertical authorization	A
		monitoring	
R20	-	aggregated view	A

Fig. 18: Evaluating the declarative approach

Constraint-based activation of activities (F)
Imperative models take an "inside-out" approach by requiring all execution alternatives to be explicitly specified in the model. Declarative models, in turn, take an "outside-in" approach: constraints implicitly specify execution alternatives as all valid alternatives have to satisfy the constraints (Pesic, 2008). Adding more constraints means discarding some execution alternatives. This results in a coarse up-front specification of a process, which can be refined iteratively during runtime. Typical constraints can be roughly divided into three

classes (Sadiq et al., 2005; van der Aalst & Pesic, 2006): constraints restricting the *selection* of activities (e.g., minimum/maximum occurrence of activities, mutual exclusion), the *ordering* of activities and the use of *resources* (e.g., execution time of activities).
Activities. Adequate support for *optional activities* is provided, i.e., activities can be considered as optional as long as no constraint enforces their execution (i.e., R8 is met).

Arbitrary process granularity (E)
Partial support for integrating process instances can be achieved based on sub-processes.
Processes. Since most declarative approaches do not support multiple instantiations, *cardinalities* to higher-level process definitions cannot be expressed (i.e., R3 is not met). It is further not possible to define processes based on *object interactions* (i.e., R11 is not met).

Extension 5 (State-oriented business process modeling). In the state-based extension provided by (Bider, 2002) a state does not necessarily correspond to an object instance. Instead, it rather belongs to a process instance comprising a set of atomic attributes or repeated groups (e.g., lists). States are used to specify the activities which should, can or must be executed; i.e., opposed to declarative modeling, conditions for executing activities are defined based on states rather than on activities. The disadvantages known from declarative approaches still hold: hidden information flows, fixed activity granularity, and arbitrary process granularity. Finally, this approach focuses on modeling functionalities without defining operational semantics; i.e., models cannot be generated.

4.3 Data-driven Approaches

There exist several approaches which support a tighter integration of processes and data (Reijers & Liman & van der Aalst, 2003; Vanderfeesten & Reijers & van der Aalst, 2008, Müller & Reichert & Herbst, 2007; van der Aalst & Weske & Grünbauer, 2005). Since Case Handling (CH) (van der Aalst & Weske & Grünbauer, 2005) satisfies the requirements for object-aware processes best, we focus on CH when evaluating data-driven approaches. Additionally, we refer to the Flower CH tool (Pallas Athena, 2002) in the context of our evaluation. Compared to imperative and declarative approaches the main differences lie in the integration of application data (G), the data-driven execution paradigm (H), and the advanced role concept (I). Like in imperative and declarative approaches, the processes can be defined at arbitrary level of granularity (E). However, the granularity of activities is fixed (D) (cf. Fig. 19).

data-driven approach

+	*supported*	D	*fixed granularity of actvities*
O	*partially supported*	E	*arbitrary granularity of processes*
-	*not supported*	G	*data integration*
		H	*data-driven activation*
		I	*advanced role concept*

			characteristic
		data	
R1	O	data integration	G
R2	O	access on data	G
R3	O	cardinalities	E
R4	+	mandatory information	G
		activities	
R5	+	form-based activities	G
R6	O	black-box activities	G
R7	-	variable granularity	D
R8	O	mandatory and optional activities	G
R9	-	control-flow within user forms	G
		processes	
R10	+	object behavior	E
R11	O	object interactions	E
R12	+	process-oriented view	I
R13	+	flexible process execution	H
R14	O	re-execution of activities	I
R15	+	explicit user decisions	H
		user integration	
R16	O	data authorization	G
R17	+	process authorization	I
R18	-	differentiation of authorization and user assignment	I
R19	-	vertical authorization	I
		monitoring	
R20	-	aggregated view	G

Fig. 19: Evaluating the data-driven approach

Data Integration (G)

Opposed to activity-centric approaches, CH enables a tighter integration of processes, activities and data (Mutschler & Weber & Reichert, 2008). As illustrated in Fig. 20, CH differentiates between free, restricted and mandatory data elements (van der Aalst & Weske & Grünbauer, 2005). Based on *free data elements*, business data not directly relevant for process control or activity inputs can be added to the process model. Free data elements are assigned to the *case description* (i.e., process model) and can be changed at any point in time while modeling the *case* (i.e., the process instance). All other data elements are associated with one or more activities, and are further subdivided into two categories. *Restricted data elements* can only be written in the context of the activities they are assigned to. *Mandatory data elements* require a value to complete the activity to which they belong.

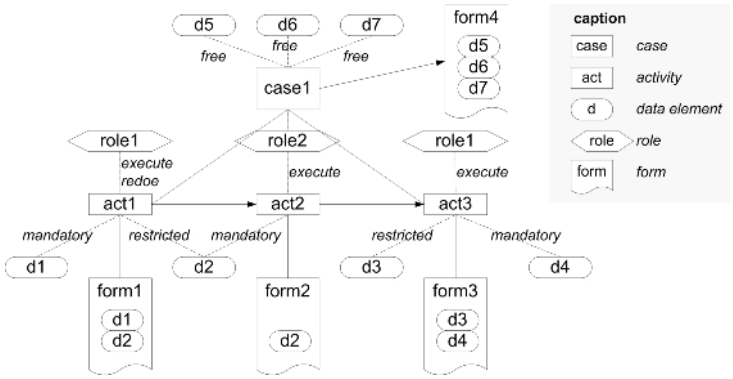

Fig. 20: Case Handling

Data. CH only provides atomic data elements; *data integration* in terms of object types and their inter-relations is not considered (i.e., R1 is not fully met). All users involved in a case are allowed to *read* its data elements. Based on a query mechanism, users may access active and completed cases. This enables *access to data* at any point in time. However, the composition of atomic data elements to object types is not considered (i.e., R2 is not fully met). By specifying certain data elements as mandatory, *mandatory information* is supported (i.e., R4 is met).
Activities. *Form-based activities* can be explicitly defined. However, provided form fields cannot be made dependent on the current process state and user (i.e., R5 is not fully met). Besides this, CH fosters application integration of *black-box activities* (Pesic & van der Aalst, 2002). However, if activity input parameters refer to different object instances their inter-relations cannot be controlled (i.e., R6 is not fully met). Free data elements enable optional activities, but one cannot define specific *optional activities* for different user roles (i.e., R8 is not fully met). Since dependencies between fields cannot be expressed, no support for controlling the *control-flow within a form* exists (i.e., R9 is not met)
User Integration. Regarding *data authorization*, it is not possible to define different access rights for a particular user depending on the progress of the case (i.e., R16 is not met).
Monitoring. Since neither object types/instances nor the relations between them are considered, *aggregated views* cannot be provided (i.e., R20 is not met).

Data-driven activation of activities (H)
Imperative and declarative approaches are both activity-centric. In data-driven approaches (like CH), activities become enabled when data changes. An activity is completed if all mandatory data elements have assigned values.
Processes. Activities can be automatically skipped at runtime if their data elements are provided by other activities; i.e., mandatory data elements are provided by preceding activities. This, in turn, enables *flexible process execution* (i.e., R13 is met). In addition, *user decisions* are supported (Pesic & van der Aalst, 2002) (i.e., R15 is met).

Advanced Role Concept (I)
CH allows to define who shall work on an activity and who may redo or skip it; for this purpose separate roles exist. Using the redo-role, for example, CH allows actors to execute activities multiple times.
Processes. Based on the execute-role, actors can be assigned to human activities. Further, users may select all cases for which they have to perform an activity. This enables a *process-oriented view* (i.e., R12 is met).
User Integration. Since the data elements that are processed during an activity execution are known, fine-grained *process authorization* at the level of single data elements becomes possible (i.e., R17 is met). Despite the introduction of the redo-role, *re-executing* activities arbitrarily often is not possible (i.e., R9 is not fully met).

Example 21 (Re-execution). As illustrated in Fig. 21, role R1 may execute or redo activity A1. If all mandatory data elements of a particular activity are available, subsequent activities become enabled immediately. Regarding our example (cf. Fig 21) as long as A2 is not completed (i.e., a value for data element D2 is not set), R1 may redo activity A1. However, after completing subsequent activity A1, redo is only possible if the user is authorized to redo A2 (cf. Fig. 21c). Otherwise, A1 cannot be redone any longer.

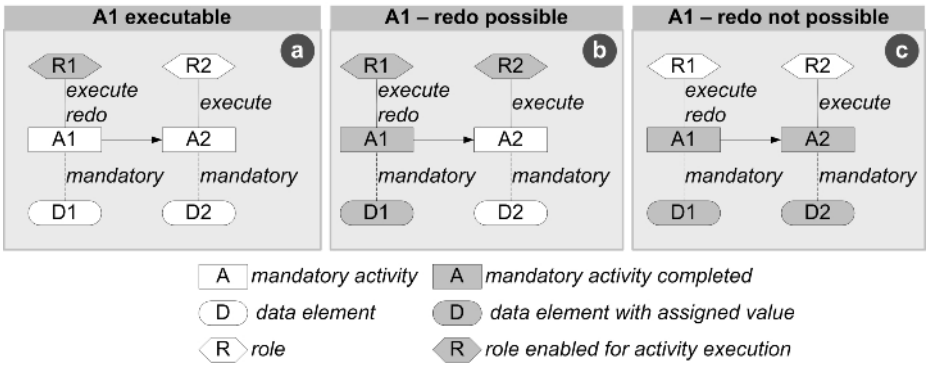

Fig. 21: Re-execution of activities in CH

Regarding mandatory activities, any user owning the *execution-role* for such activity must execute it mandatorily; i.e., no *differentiation between authorization and user assignment* is made (i.e., R18 is not met). Finally, *vertical authorization* based on data element values is not provided (i.e., R19 is not met).

Fixed activity granularity (D)
In CH each activity belongs to exactly one process instance, and the granularity of activities is fixed at build-time.
Activities. Users cannot access data element values of other relating cases; i.e., a *variable granularity* of activities to support preferred work practices (e.g., context-sensitive vs. batch activities) is not supported (i.e., R7 is not met).

Arbitrary process granularity (E)
CH allows to model processes at arbitrary level of granularity. When described at a **coarse-grained level**, a case definition includes data elements corresponding to different objects. Interdependencies between different cases can be defined based on sub-cases. Further, CH supports multiple-instantiation patterns through dynamic sub-plans (i.e., sub-process instances) (Pesic & van der Aalst, 2002). This enables instantiation of a dynamically fixed number of sub-process instances. Alternatively, when processes are described at a **fine-grained level**, a "case" can be manually treated in tight accordance with an "object".
Data. Using multiple-instantiation patterns as extension, *cardinalities* between higher-level and lower-level process instances can be taken into account. However, it cannot be ensured that the correct number of sub-processes instances is actually created at runtime (i.e., R3 is not fully met).
Processes. Modeling processes in a **fine-grained manner** means we need to consider each case as object type. This enables support of *object behavior* (i.e., R10 is met). Even if object behavior is not considered in terms of states and transitions, the data-driven execution allows to dynamically react to data changes. As limitation activities always refer to the execution of one particular case. Thus, only instance-specific activities can be realized, but no *varying granularity*; i.e., users cannot choose their preferred work practice (i.e., R7 is not met). Further, we cannot define *interactions between object instances* (i.e., R11 is not met).
When modeling processes at a **coarse-grained level**, similar restrictions for the asynchronous coordination of sub-process instances hold as for imperative approaches. Compared to imperative approaches CH additionally allows users to access data from multiple instances through data containers with a variable number of data elements. Taking literature we cannot conclude that these data enable the data-driven execution of activities that belong to the higher-level process instance; i.e., *interdependencies between asynchronously executed object instances* are not fully supported (i.e., R11 is not fully met).

4.4 Further approaches

The object-process methodology (OPM) (Dori, 2002), considers object types and their inter-relations. Furthermore, object behavior can be defined in terms of states and processes enable transitions between them; i.e., states are used as pre-/post-conditions for process execution. However, states are not mapped to individual attribute values what leads to hidden information flows. OPM further allows for different levels of aggregation using zooming functions. There exists no methodology to define the resulting abstraction layers in correspondence to the data structure; i.e., each layer can be defined at an arbitrary level of granularity. In addition, the granularity of activities is fixed. Though OPM considers some properties of object-aware processes, it is not suitable for their support. It is also not appropriate for defining operational semantics based on which data- and process-oriented views as well as form-based activities can be automatically generated.
Finally, there exist goal-based (Soffer & Wand, 2005), decision-oriented, and conversation-oriented process modeling approaches (Nurcan, 2008) which are outside the scope of our evaluation.

5. Summary and Outlook

We discussed fundamental requirements in respect to the provision of an integrated view on processes and data. Such integration is needed for many applications like *enterprise resource planning* and *customer relationship management*. Further, we showed that the identified requirements go beyond the features of existing modeling approaches. Especially activity-centric approaches show an inherent weakness in respect to object-aware process management. Data-driven approaches, in turn, are more expressive, but have not reached the required maturity level yet. To our knowledge, there exists no approach which provides a well-defined modeling methodology for defining object behavior and interactions. Regarding behavior, object states must be mapped to attribute values and inter-relations must be considered.
To tackle the discussed challenges and requirements, in the *PHILharmonic Flows* project we target at a framework that enables a tight integration of business processes, business data, and users. In our future work we will report on this framework. On the one hand, it retains the well-established principle for separation of concerns; on the other hand, it explicitly considers the relationships between processes, functions, data and users. Furthermore, PHILharmonicFlows will address process modeling, execution and monitoring, and will provide generic functions for the model-driven generation of end user components (e.g., form-based activities). Opposed to existing approaches on process and data integration, we want to consider all components of the underlying data structure; i.e., objects, relations and attributes. For this purpose, we will enable the modeling of processes at different levels of granularity. In particular, we will combine object behavior based on states with data-driven process execution. Further, we will provide advanced support for process coordination as well as for the integrated access to business processes, business functions and business data. Our overall vision is to overcome many of the limitations of contemporary PrMS.

References

van der Aalst, W.M.P., Barthelmess, P., Ellis, C.A., & Wainer, J. (2000). Workflow Modeling using Proclets. In *Proc. CoopIS'00: LNCS 1901* (pp. 198-209). Springer.

van der Aalst, W.M.P., Hofstede, A., Kiepuszewski, B., & Barros, A. (2003). Workflow Patterns. *Distributed & Parallel Databases, 14(1)*, 5-51.

van der Aalst, W.M.P., ter Hofstede, A.H.M., & Weske, M. (2003). Business Process Management: A Survey. In *Proc. BPM'03: LNCS 2678* (pp. 1-12). Springer.

van der Aalst, W.M.P., & Pesic, M. (2006). DecSerFlow: Towards a Truly Declarative Service Flow Language. In *Proc. Dagstuhl Seminar* (pp. 1-23). Springer.

van der Aalst, W.M.P., Weske, M., & Grünbauer, D. (2005). Case Handling: A new Paradigm for Business Process Support. *Data and Knowledge Engineering, 53(2)*, 129-162.

Bider, I. (2002). *State-oriented Business Process Modeling: Principles, Theory and Practice*. PhD thesis, Royal Institute of Technology, Stockholm.

Botha, R.A. (2002). *CoSAWoE - A Model for Context-sensitive Access Control in Workflow Environments*. PhD thesis, Rand Afrikaans University.

Dori, D. (2002). *Object-Process Methodology*. Springer.

Dadam, P., & Reichert, M. (2009). The ADEPT Project: A Decade of Research and Development for Robust and Flexible Process Support - Challenges and Achievements. *Computer Science - R & D, 23(2)*, 81-97.

Gerede, C.E., & Su, J. (2007). Specification and Verification of Artifact Behaviors in Business Process Models. In *Proc. ICSOC'07: LNCS 4749* (pp. 181-192). Springer.

Künzle, V., & Reichert, M. (2009a). Towards Object-aware Process Management Systems: Issues, Challenges, Benefits. In *Proc. BPMDS'09: LNBIP 29* (pp. 197-210). Springer.

Künzle, V., & Reichert, M. (2009b). Integrating Users in Object-aware Process Management Systems: Issues and Challenges. In *Proc. BPM'09 Workshops: LNBIP 43* (pp. 29-41). Springer.

Küster, J., Ryndina, K., & Gall, H. (2007). Generation of Business Process Models for Object Life Cycle Compliance. In *Proc. BPM'07: LNCS 4714* (pp. 165 -181). Springer.

Liu, R., Bhattacharya, K., & Wu, F.Y. (2007). Modeling Business Contexture and Behavior Using Business Artifacts. In *Proc. CAiSE'07: LNCS 4495* (pp. 324-339). Springer.

Müller, D., Reichert, M., & Herbst, J. (2007). Data-driven Modeling and Coordination of Large Process Structures. In *Proc. CoopIS'07: LNCS 4803* (pp. 131-149). Springer.

Mutschler, B., Weber, B., & Reichert, M. (2008). Workflow Management versus Case Handling: Results from a Controlled Software Experiment. In *Proc. SAC'08* (pp. 82-89). ACM Press. Brazil. Springer.

Nigam, A., & Caswell, N.S. (2003). Business Artifacts - An Approach To Operational Specification. *IBM Systems Journal, 42(3)*, 428-445.

Nurcan, S. (2008). A Survey on the Flexibility Requirements Related to Business Processes and Modeling Artifacts. In *Proc. HICSS'08* (pp. 378). Springer.

Pallas Athena. (2002). *Flower User Manual*. Pallas Athena BV, Apeldoorn, The Netherlands.

Pesic, M. (2008). *Constraint-Based Workflow Management Systems: Shifting Control to Users*. PhD thesis, Eindhoven University of Technology.

Redding, G., Dumas, M., ter Hofstede, A.H.M., & Iordachescu, A. (2008). Transforming Object-oriented Models to Process-oriented Models. In *Proc. BPM'07 Workshops: LNCS 4928* (pp. 132-143). Springer.

Reijers, H.A., Liman, S., & van der Aalst, W.M.P. (2003). Product-based Workflow Design. *Management Information Systems, 20(1)*, 229-262.

Rosemann, M., & zur Mühlen, M. (1998). Modellierung der Aufbauorganisation in Workflow-Management-Systemen: Kritische Bestandsaufnahme und Gestaltungsvorschläge. *EMISA-Forum*, 3(1), 78-86.

Rosemann, M., & zur Mühlen, M. (2004). Organizational Management in Workflow Applications: Issues and Perspectives. *Inf. Technol. and Management*, 5(3-4), 271-291.

Rinderle-Ma, S., & Reichert, M. (2007). A Formal Framework for Adaptive Access Control Models. *Journal on Data Semantics IX, LNCS 4601*, 82-112.

Silver, B. (2009). *Case Management: Addressing Unique BPM Requirements*. (Industry Trend Reports) BPMS Watch.

Wu, S., Sheth, A., Miller, J., & Luo, Z. (2002). Authorization and Access Control Of Application Data In Workflow-Systems. *JIIS, 18(1)*, 71-94.

Sadiq, S., Sadiq, W., & Orlowska, M. (2005). A Framework for Constraint Specification and Validation in Flexible Workflows. *Inf. Sys., 30(5)*, 349-78.

Sadiq, S., Orlowska, M.E., Sadiq, W., & Schulz, K. (2005). When Workflows Will Not Deliver: The Case of Contradicting Work Practice. In *Proc. BIS'05*. Springer.

Soffer, P., & Wand, Y. (2005). On the Notion of Soft-goals in Business Process Modeling. *Business Process Management Journal, 11(6)*, 663-679.

Vanderfeesten, I., Reijers, H.A., & van der Aalst, W.M.P. (2008). Product-based Workflow Support: Dynamic Workflow Execution. In *Proc. CAiSE '08: LNCS 5074* (pp. 571-574). Springer.

Weber, B., Reichert, M., & Rinderle-Ma, S. (2008). Change Patterns and Change Support Features - Enhancing Flexibility in Process-Aware Information Systems. *Data and Knowledge Engineering, 66(3)*, 438-466.