Object-aware Business Processes: Properties, Requirements, Existing Approaches

Vera Künzle, Barbara Weber, Manfred Reichert

Ulmer Informatik-Berichte
Nr. 2010-06
August 2010
Object-aware Business Processes: Properties, Requirements, Existing Approaches

Vera Künzle¹, Barbara Weber², and Manfred Reichert¹

¹Institute of Databases and Information Systems, Ulm University, Germany
{vera.kuenzle,manfred.reichert}@uni-ulm.de
²Institute 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, configuring, enacting, monitoring, and diagnosing business processes [AHW03, HBR10, MRB08, WRWR09]. 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) to assign upcoming activities to authorized users [RiRe07, RiRe09].

Generally, PrMS comprise generic runtime functions for interpreting process models, assigning human tasks to users and invoking application components. As a pre-requisite for the latter, for each activity a corresponding application component must be provided. 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 [Sil09]. 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 [AWG05]. 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 [ABEW00, MRH07]. This, in turn, leads to a "contradiction between the way processes can be modeled and preferred work practice" [SOSS05]. 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 [KüRe09A, KüRe09B, KüRe09C, KüRe10]. Processes similar to the ones we evaluated can be found in many other fields like healthcare [LeRe07, RHD98], software engineering [GOR10] and release management [MHHR06]. 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 at any point in time. 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., on 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. In the application systems we analyzed about eighty-five percent of all activities were form-based.

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 [KüRe09A, KüRe09B, KüRe09C, KüRe10] 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 and discuss why objects are the driver for modeling, executing and monitoring these processes. We first introduce an example of an object-aware process. As illustrated by Fig. 1, we use a (simplified) scenario from our case study in the area of human resource management [KüRe10].

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, birthdate, 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.

In the following we discuss fundamental properties of object-aware business processes along this realistic example. Thereby, we focus on the categories Data, Activities, Processes, User Integration, and Monitoring.

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. 2a, 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. 2b); 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. 2: Data structure at build- and runtime

2.2 Activities

Activities can be divided into form-based and black-box activities. In object-aware processes typically, during activity execution, the values of object attributes can be accessed. 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 depending on the number of object instances they apply to.
**Instance-specific activities** correspond to exactly one object instance (cf. Fig. 3a). 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. 3b). 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. 3c); 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.

---

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**. Both form-based and black-box activities may either be optional or mandatory. In the latter case, their execution is required for process execution.

### 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. 4. 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. 4c: 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.
At runtime, for each object type multiple object instances may exist (cf. Fig. 5a). Generally, the exact number of instances to be processed for a particular object type is not always known at build-time. 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 at a certain point in time; 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. 5b).

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

Object interactions within the recruitment process (cf. Fig. 6): 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 con-
ducts 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. 6: 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. 7); 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. 7. 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 (form-based and black-box) 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. 8). 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.

2.6 Summary

Altogether, for many process-aware business applications, objects act as driver for process modeling and execution. Thus, process modeling must consider object types, attributes and relations, while process execution depends on the processed object instances and their individual state, which reflects the progress of corresponding object behavior. To achieve tight synchronization between object and process state, it is not sufficient to model processes only in terms of black-box activities. Instead, for each process step, pre-conditions regarding object attribute values exist. However, if such conditions are not met during runtime, process execution must not be blocked; i.e., it is not sufficient to only postulate certain attribute values, but it also becomes necessary to dynamically react on available data at any point in time. In particular, activation of an activity does not directly depend on the completion of other activities, but on changes of object attribute values. Note that this is a fundamental difference when compared to activity-centric approaches.
3. Requirements for the IT Support of Object-aware Processes

This section elicits fundamental requirements for the support of object-aware processes. We gathered these requirements in three case studies in which we analyzed processes and objects of applications from human resource management, paper reviewing and healthcare. Though these requirements are not complete in the sense that they cover all aspects of the object and process lifecycle, their fulfillment is indispensable for enabling the aforementioned properties as well as the automatic generation of runtime components like worklists, overview lists and form-based activities. Fig. 9 gives an overview of these requirements. They are described in detail in the following subsections.

<table>
<thead>
<tr>
<th>data</th>
<th>activities</th>
<th>processes</th>
<th>user integration</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1</td>
<td>data integration</td>
<td>F5</td>
<td>form-based activities</td>
</tr>
<tr>
<td>F2</td>
<td>access on data</td>
<td>F6</td>
<td>black-box activities</td>
</tr>
<tr>
<td>F3</td>
<td>cardinalities</td>
<td>F7</td>
<td>variable granularity</td>
</tr>
<tr>
<td>F4</td>
<td>mandatory information</td>
<td>F8</td>
<td>mandatory and optional activities</td>
</tr>
<tr>
<td></td>
<td></td>
<td>F9</td>
<td>control flow within user forms</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>F15</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>F16</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>F17</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>F18</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>F19</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>F20</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Fig. 9: Fundamental requirements for object-aware processes

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 1: 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.

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 2: 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.

R3 (Cardinalities). It should be possible to restrict relations between object instances through cardinality constraints; i.e., through the minimal and maximal number of object instances which may be created in the given context.

Example 3: 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.

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 4: 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.

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 5: 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.

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 6: 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.

R7 (Variable granularity). As discussed, support for instance-specific, context-sensitive, and batch activities is required (cf. Section 2). 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 7: 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.

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). It must be possible for users to clearly distinguish between these two types.
Example 8a (Mandatory activity): After a review has been initiated by a personnel officer, the assigned employee either must provide or refuse the review; i.e., a form-based activity needs to be mandatorily performed.

Example 8b (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.

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 9: 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.

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 10: 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.

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) [ABEW00]. Third, the executions of different process instances may be mutually dependent [MRH07, ABEW00]; whether an object instance may switch to a certain state depends on the state of another object instance and vice versa (execution dependency). Consequently, processes should be defined in terms of object interactions. Additionally, the integration of black-box activities should possible.
Example 11: We consider the different kinds of synchronization dependencies in the context of our example: 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.

R12 (Process-oriented view). During process execution some activities must be mandatorily executed by responsible users 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 12: 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. Consequently, access this object instance must be coordinated.

R13 (Flexible process execution). While optional activities can be applied to change object attribute values at any point in time, mandatory activities are bound to certain object instance states. In particular, 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 13: 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 due to the up front provision of the required data.

R14 (Re-execution of activities). While in many cases mandatory activities shall be automatically skipped if related attribute values are set prior to the execution of these activities, other scenarios require that users explicitly commit completion of such activities even if all mandatory information is available; i.e., to explicitly approve the values assigned to mandatory object attributes. In particular, users should be allowed to re-execute a particular activ-
ity (i.e., to update its attributes), even if all mandatory object attributes have been already set.

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

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 15: 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.

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 [Bot02, WSML02]. 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. Altogether, the execution of optional activities cannot be treated independently from normal process execution.

Example 16: 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.

R17 (Process authorization). For each mandatory activity required for process execution 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 [Bot02].

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

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 18:** 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.

**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. We denote this as *vertical authorization*.

**Example 19:** 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' and 'Z'.

### 3.5 Monitoring

**R20 (Aggregated View).** A complex process, which integrates several object instances of the same and of different type, can be defined based on the interactions between these object instances (i.e., each process step refers to one interaction dependency). Corresponding to this, a process monitoring component should provide aggregated views of all involved object instances. In particular, individual object instances are executed concurrently to each other and to corresponding higher-level as well as lower-level object instances. This leads to the asynchronous execution of different parts of the process. Process monitoring should provide an aggregated view of all object instances involved in a process as well as their interdependencies.

**Example 20:** 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.
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. In addition, there are approaches allowing for the partial support of certain requirements based on workarounds. As illustrated in Fig. 10, only limited support is provided in respect to the support of object-aware processes.

![Diagram showing evaluation of existing approaches]

**Fig. 10: Evaluating of existing approaches**

### 4.1 Imperative Approaches

There is a long tradition of modeling business processes in an imperative way. Process languages supporting this paradigm include BPMN and BPEL. Usually, processes are specified as directed graphs [WRR08]. Process steps correspond to different activities [TRI09] which are connected to express precedence relations (cf. Fig. 11). For control flow modeling a number of patterns exists, e.g., sequential, alternative and parallel routing, and loop backs [AHKB03].
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: 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. 12).

**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. 11). 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 (and object attributes respectively). Consequently, it remains hidden which data is actually accessed or changed during activity execution. Altogether, this characteristic affects requirements of categories **Data, Activities, User Integration,** and **Monitoring.**
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. Opposed 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. There exist sophisticated approaches for enabling the monitoring of imperative processes [BBR06, BRB05]. However, due to the 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 not chosen for execution). 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). This characteristic affects requirements of categories Data, Activities, Processes and User Integration.

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). However, such data access can be simulated using the following workaround.

Workaround 1 (Optional activities). “Optional” activities can be added as conditional branches in different regions of a process model (cf. Fig. 13). As a drawback, end-users cannot distinguish between optional and mandatory work items emerging in their worklists. Furthermore, complex and difficult to maintain spaghetti-like process models might result.
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).

Workaround 2 (Flexible process execution). Process data elements could be evaluated using XOR-splits before and after activity execution (cf. Fig. 14a). If required object attribute values have already become available before activity execution, the process model can simulate the skipping of the respective activity by embedding it in an alternative path. Note that there also exist approaches like ADEPT2 [RHD98, Rei00, RRKD05, RD09], which enable process flexibility by supporting dynamic process changes (e.g., to add or move activities) during runtime. However, issues related to such dynamic process changes are outside the scope of this paper (see [RRD09, WRR08, WSR09] for recent surveys).

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).

Workaround 3 (Re-execution and user decisions). User decisions and commitments are encapsulated within black-box activities which write certain data elements as specified within the data flow (cf. Fig. 14b). These data elements are then evaluated using an XOR-split to decide whether to proceed with process execution or to initiate a backward jump using a loop.

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, which enables process coordination among users. Further, when a human activity becomes enabled, a corresponding work item is added to worklists of authorized users. This property affects a requirement of category **User Integration**.

**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. This property affects a requirement of category **Activities**.

**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*. This has effects on requirements of categories **Data** and **Processes**.

**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 [ABEW00]. 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 [MRH07, MRHP07]. Thereby, synchronization constraints are defined based on object states [MRH06, MRH07, MRH08]. 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 [BHS09, GeSu07, KRG07, RDHI07, RDHI10, NiCa03, LBW07]. 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; i.e., process execution is blocked. Neither relations between object types nor the varying number of object instances are considered at runtime.

Process support involving different object instances can be provided by using subprocesses. 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 [AHKB03, RiRe06], 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. 15a), 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. 15b). Finally, interdependencies between sub-processes, which are executed asynchronously to each other (cf. Fig. 15c), cannot be taken into account.

4.2 Declarative Approaches

Declarative approaches suggest a fundamentally different way of describing business processes [AaPe06, APS09]. 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. 16, activities A2 and A3 can only be executed after finishing A1. Finally, A2 and A3 are mutually exclusive.

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, declarative approaches rely on a constraint-based activation (B) (cf. Fig. 17). 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.
Constraint-based activation of activities (B)

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 [Pes08]. 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 [SSO05, AaPe06, APS09]: 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, time difference between activities, etc.). This property affects requirements of categories Activities and Processes.

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).

Processes. The declarative approach does not directly support flexible process execution, i.e., mandatory activities cannot be skipped if the required data are already available (i.e., R13 is not met). However, the following workaround is conceivable:

Workaround 4 (Flexible process execution). Specific data constraints could be introduced to check whether all required data are available and to prohibit activity execution for this case.
Like in imperative approaches, activities cannot be re-executed based on user commitments (i.e., R9 is not met).

**Workaround 5 (Re-execution of activities).** Using a specific constraint, re-execution of a particular activity is possible as long as another activity that captures the user commitment has not been executed.

Similar to imperative approaches, such workarounds lead to models which are difficult to comprehend and to maintain.

**Arbitrary process granularity (E)**

Partial support for integrating process instances can be achieved based on sub-processes. This affects one requirement of category 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 [Bid02] 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 [RLA03, VRA08, MRH06, MRH07, AWG05]. Since Case Handling (CH) [AWG05, GRA08] 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 [PA02] in the context of our evaluation. Compared to imperative and declarative approaches the main differences lie in the integration of application data (A), the data-driven execution paradigm (B), and the advanced role concept (C). Like in imperative/declarative approaches, processes can be defined at arbitrary level of granularity (E) (cf. Fig. 18). In the following we discuss CH along these characteristics.

**Data Integration (A)**

Opposed to activity-centric approaches, CH enables a tighter integration of processes, activities and data [MWR08, WMR10]. Thereby, CH differentiates between free, restricted and mandatory data elements (cf. Fig. 19). 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. This affects requirements of categories Data, Activities, User Integration and Monitoring.
Fig. 18: Evaluating the data-driven approach

Fig. 19: Data-driven modeling approach - Case Handling [AWG05]
**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* [PA02]. 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., the same optional activity is provided to all users (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 (B)**

Imperative and declarative approaches are both activity-centric, i.e., activation of an activity depends on the completion of preceding activities. In data-driven approaches (like CH), activities become enabled when data changes. An activity is completed if all mandatory data elements have assigned values; i.e., mandatory information is provided. This affects one requirement of the category *Processes*.

**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 [PA02] (i.e., R15 is met).

**Advanced Role Concept (C)**

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. This affects requirements of categories *Processes* and *User Integration*.

**Processes.** Based on the execute-role, like in imperative and declarative approaches, 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). Consider the following example.

**Example 21 (Re-execution).** As illustrated in Fig. 20, 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 20b) 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. 20c). Otherwise, A1 cannot be redone any longer.

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)

Like in activity-centric approaches, in CH each activity belongs to exactly one process instance, and the granularity of activities is fixed at build-time. This affects one requirement of category Activities.

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) [PA02]. 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". This affects requirements of categories Data and Processes.

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., that their quantity lies between the minimal and maximal cardinality (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 exe-
cution 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

Some approaches apply an object-oriented paradigm for modeling processes. The object-process methodology (OPM) [Dor02], 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 [SoWa05], decision-oriented, and conversation-oriented process modeling approaches [Nur08] 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 tight integration of business processes, business data, and users. We denote such paradigm as Object-aware Process Management and consider related research as fundamental for the further maturation of process management technology. We believe that a better understanding of object-aware process management will stimulate other research activities on, for example, integrated process and data mining, data-centred process monitoring, and business process compliance. Finally, a well-defined granularity of processes in respect to data constitutes the basis for any modeling methodology as well as for ensuring consistency between process and data evolution.

---

1 Process, Humans and Information Linkage for harmonic Business Flows
References


<table>
<thead>
<tr>
<th></th>
<th>Title</th>
</tr>
</thead>
</table>
| 91-01 | Ker-I Ko, P. Orponen, U. Schöning, O. Watanabe  
Instance Complexity                      |
| 91-02* | K. Gladitz, H. Fassbender, H. Vogler  
Compiler-Based Implementation of Syntax-Directed Functional Programming |
| 91-03* | Alfons Geser  
Relative Termination                        |
| 91-04* | J. Köbler, U. Schöning, J. Toran  
Graph Isomorphism is low for PP             |
| 91-05  | Johannes Köbler, Thomas Thierauf  
Complexity Restricted Advice Functions     |
| 91-06* | Uwe Schöning  
Recent Highlights in Structural Complexity Theory |
| 91-07* | F. Green, J. Köbler, J. Toran  
The Power of Middle Bit                       |
| 91-08* | V. Arvind, Y. Han, L. Hamachandra, J. Köbler, A. Lozano, M. Mundhenk, A. Ogiwara, U. Schöning, R. Silvestri, T. Thierauf  
Reductions for Sets of Low Information Content |
| 92-01* | Vikraman Arvind, Johannes Köbler, Martin Mundhenk  
On Bounded Truth-Table and Conjunctive Reductions to Sparse and Tally Sets |
| 92-02* | Thomas Noll, Heiko Vogler  
Top-down Parsing with Simultaneous Evaluation of Noncircular Attribute Grammars |
| 92-03  | Fakultät für Informatik  
17. Workshop über Komplexitätstheorie, effiziente Algorithmen und Datenstrukturen |
| 92-04* | V. Arvind, J. Köbler, M. Mundhenk  
Lowness and the Complexity of Sparse and Tally Descriptions |
| 92-05* | Johannes Köbler  
Locating P/poly Optimally in the Extended Low Hierarchy |
| 92-06* | Armin Kühnemann, Heiko Vogler  
Synthesized and inherited functions - a new computational model for syntax-directed semantics |
| 92-07* | Heinz Fassbender, Heiko Vogler  
A Universal Unification Algorithm Based on Unification-Driven Leftmost Outermost Narrowing |
92-08*  Uwe Schöning  
On Random Reductions from Sparse Sets to Tally Sets

92-09*  Hermann von Hasseln, Laura Martignon  
Consistency in Stochastic Network

92-10  Michael Schmitt  
A Slightly Improved Upper Bound on the Size of Weights Sufficient to Represent Any Linearly Separable Boolean Function

92-11  Johannes Köbler, Seinosuke Toda  
On the Power of Generalized MOD-Classes

92-12  V. Arvind, J. Köbler, M. Mundhenk  
Reliable Reductions, High Sets and Low Sets

92-13  Alfons Geser  
On a monotonic semantic path ordering

92-14*  Joost Engelfriet, Heiko Vogler  
The Translation Power of Top-Down Tree-To-Graph Transducers

93-01  Alfred Lupper, Konrad Froitzheim  
AppleTalk Link Access Protocol basierend auf dem Abstract Personal Communications Manager

The COCOON Object Model

93-03  Thomas Thierauf, Seinosuke Toda, Osamu Watanabe  
On Sets Bounded Truth-Table Reducible to P-selective Sets

93-04  Jin-Yi Cai, Frederic Green, Thomas Thierauf  
On the Correlation of Symmetric Functions

93-05  K.Kuhn, M.Reichert, M. Nathe, T. Beuter, C. Heinlein, P. Dadam  
A Conceptual Approach to an Open Hospital Information System

93-06  Klaus Gaßner  
Rechnerunterstützung für die konzeptuelle Modellierung

93-07  Ullrich Keßler, Peter Dadam  
Towards Customizable, Flexible Storage Structures for Complex Objects

94-01  Michael Schmitt  
On the Complexity of Consistency Problems for Neurons with Binary Weights

94-02  Armin Kühnemann, Heiko Vogler  
A Pumping Lemma for Output Languages of Attributed Tree Transducers

94-03  Harry Buhrman, Jim Kadin, Thomas Thierauf  
On Functions Computable with Nonadaptive Queries to NP

94-04  Heinz Faßbender, Heiko Vogler, Andrea Wedel  
Implementation of a Deterministic Partial E-Unification Algorithm for Macro Tree Transducers
V. Arvind, J. Köbler, R. Schuler
On Helping and Interactive Proof Systems

Christian Kalus, Peter Dadam
Incorporating record subtyping into a relational data model

Markus Tresch, Marc H. Scholl
A Classification of Multi-Database Languages

Friedrich von Henke, Harald Rueß
Arbeitstreffen Typtheorie: Zusammenfassung der Beiträge

Construction and Deduction Methods for the Formal Development of Software

Axel Dold
Formalisierung schematischer Algorithmen

Johannes Köbler, Osamu Watanabe
New Collapse Consequences of NP Having Small Circuits

Rainer Schuler
On Average Polynomial Time

Rainer Schuler, Osamu Watanabe
Towards Average-Case Complexity Analysis of NP Optimization Problems

Wolfram Schulte, Ton Vullinghs
Linking Reactive Software to the X-Window System

Alfred Lupper
Namensverwaltung und Adressierung in Distributed Shared Memory-Systemen

Robert Regn
Verteilte Unix-Betriebssysteme

Helmut Partsch
Again on Recognition and Parsing of Context-Free Grammars: Two Exercises in Transformational Programming

Helmut Partsch
Transformational Development of Data-Parallel Algorithms: an Example

Oleg Verbitsky
On the Largest Common Subgraph Problem

Uwe Schöning
Complexity of Presburger Arithmetic with Fixed Quantifier Dimension

Harry Buhrman, Thomas Thierauf
The Complexity of Generating and Checking Proofs of Membership

Rainer Schuler, Tomoyuki Yamakami
Structural Average Case Complexity

Klaus Achatz, Wolfram Schulte
Architecture Independent Massive Parallelization of Divide-And-Conquer Algorithms
95-06  Christoph Karg, Rainer Schuler  
Structure in Average Case Complexity

95-07  P. Dadam, K. Kuhn, M. Reichert, T. Beuter, M. Nathe  
ADEPT: Ein integrierender Ansatz zur Entwicklung flexibler, zuverlässiger kooperierender Assistenzsysteme in klinischen Anwendungsumgebungen

95-08  Jürgen Kehrer, Peter Schultethess  
Aufbereitung von gescannten Röntgenbildern zur filmlosen Diagnostik

95-09  Hans-Jörg Bartschick, Wolfgang Lindner  
On Sets Turing Reducible to P-Selective Sets

95-10  Boris Hartmann  
Berücksichtigung lokaler Randbedingung bei globaler Zieloptimierung mit neuronalen Netzen am Beispiel Truck Backer-Upper

95-12  Klaus Achatz, Wolfram Schulte  
Massive Parallelization of Divide-and-Conquer Algorithms over Powerlists

95-13  Andrea Mößle, Heiko Vogler  
Efficient Call-by-value Evaluation Strategy of Primitive Recursive Program Schemes

95-14  Axel Dold, Friedrich W. von Henke, Holger Pfeifer, Harald Rueß  
A Generic Specification for Verifying Peephole Optimizations

96-01  Ercüment Canver, Jan-Tecker Gayen, Adam Moik  
Formale Entwicklung der Steuerungssoftware für eine elektrisch ortsbediente Weiche mit VSE

96-02  Bernhard Nebel  
Solving Hard Qualitative Temporal Reasoning Problems: Evaluating the Efficiency of Using the ORD-Horn Class

96-03  Ton Vullinghs, Wolfram Schulte, Thilo Schwinn  
An Introduction to TkGofer

96-04  Thomas Beuter, Peter Dadam  
Anwendungsspezifische Anforderungen an Workflow-Mangement-Systeme am Beispiel der Domäne Concurrent-Engineering

96-05  Gerhard Schellhorn, Wolfgang Ahrendt  
Verification of a Prolog Compiler - First Steps with KIV

96-06  Manindra Agrawal, Thomas Thierauf  
Satisfiability Problems

96-07  Vikraman Arvind, Jacobo Torán  
A nonadaptive NC Checker for Permutation Group Intersection

96-08  David Cyrluk, Oliver Möller, Harald Rueß  
An Efficient Decision Procedure for a Theory of Fix-Sized Bitvectors with Composition and Extraction

96-09  Bernd Biechele, Dietmar Ernst, Frank Houdek, Joachim Schmid, Wolfram Schulte  
Erfahrungen bei der Modellierung eingebetteter Systeme mit verschiedenen SA/RT-Ansätzen
<table>
<thead>
<tr>
<th>Year</th>
<th>Title</th>
<th>Authors</th>
</tr>
</thead>
<tbody>
<tr>
<td>96-10</td>
<td>Formalizing Fixed-Point Theory in PVS</td>
<td>Falk Bartels, Axel Dold, Friedrich W. von Henke, Holger Pfeifer, Harald Rueß</td>
</tr>
<tr>
<td>96-11</td>
<td>Mechanized Semantics of Simple Imperative Programming Constructs</td>
<td>Axel Dold, Friedrich W. von Henke, Holger Pfeifer, Harald Rueß</td>
</tr>
<tr>
<td>96-12</td>
<td>Generic Compilation Schemes for Simple Programming Constructs</td>
<td>Axel Dold, Friedrich W. von Henke, Holger Pfeifer, Harald Rueß</td>
</tr>
<tr>
<td>97-01</td>
<td>Pattern Matching in Trace Monoids</td>
<td>Jochen Messner</td>
</tr>
<tr>
<td>97-02</td>
<td>A Small Span Theorem within P</td>
<td>Wolfgang Lindner, Rainer Schuler</td>
</tr>
<tr>
<td>97-03</td>
<td>A Distributed Execution Environment for Large-Scale Workflow Management Systems with Subnets and Server Migration</td>
<td>Thomas Bauer, Peter Dadam</td>
</tr>
<tr>
<td>97-04</td>
<td>Interaction Expressions - A Powerful Formalism for Describing Inter-Workflow Dependencies</td>
<td>Christian Heinlein, Peter Dadam</td>
</tr>
<tr>
<td>97-05</td>
<td>On Pseudorandomness and Resource-Bounded Measure</td>
<td>Vikraman Arvind, Johannes Köbler</td>
</tr>
<tr>
<td>97-06</td>
<td>Punkt-zu-Punkt- und Mehrpunkt-basierende LAN-Integrationsstrategien für den digitalen Mobilfunkstandard DECT</td>
<td>Gerhard Partsch</td>
</tr>
<tr>
<td>97-07</td>
<td>ADEPT&lt;sub&gt;flex&lt;/sub&gt; - Supporting Dynamic Changes of Workflows Without Loosing Control</td>
<td>Manfred Reichert, Peter Dadam</td>
</tr>
<tr>
<td>97-08</td>
<td>The Project NoName - A functional programming language with its development environment</td>
<td>Hans Braxmeier, Dietmar Ernst, Andrea Mößle, Heiko Vogler</td>
</tr>
<tr>
<td>97-09</td>
<td>Grundlagen von Interaktionsausdrücken</td>
<td>Christian Heinlein</td>
</tr>
<tr>
<td>97-10</td>
<td>Graphische Repräsentation von Interaktionsausdrücken</td>
<td>Christian Heinlein</td>
</tr>
<tr>
<td>97-11</td>
<td>Sprachtheoretische Semantik von Interaktionsausdrücken</td>
<td>Christian Heinlein</td>
</tr>
<tr>
<td>97-12</td>
<td>Proving Properties of Finite Enumerations: A Problem Set for Automated Theorem Provers</td>
<td>Gerhard Schellhorn, Wolfgang Reif</td>
</tr>
</tbody>
</table>
Dietmar Ernst, Frank Houdek, Wolfram Schulte, Thilo Schwinn
Experimenteller Vergleich statischer und dynamischer Softwareprüfung für eingebettete Systeme

Wolfgang Reif, Gerhard Schellhorn
Theorem Proving in Large Theories

Thomas Wennekers
Asymptotik rekurrenter neuronaler Netze mit zufälligen Kopplungen

Peter Dadam, Klaus Kuhn, Manfred Reichert
Clinical Workflows - The Killer Application for Process-oriented Information Systems?

Mohammad Ali Livani, Jörg Kaiser
EDF Consensus on CAN Bus Access in Dynamic Real-Time Applications

Johannes Köbler, Rainer Schuler
Using Efficient Average-Case Algorithms to Collapse Worst-Case Complexity Classes

Daniela Damm, Lutz Claes, Friedrich W. von Henke, Alexander Seitz, Adelinde Uhrmacher, Steffen Wolf
Ein fallbasiertes System für die Interpretation von Literatur zur Knochenheilung

Thomas Bauer, Peter Dadam
Architekturen für skalierbare Workflow-Management-Systeme - Klassifikation und Analyse

Marko Luther, Martin Strecker
A guided tour through Typelab

Heiko Neumann, Luiz Pessoa
Visual Filling-in and Surface Property Reconstruction

Ercüment Canver
Formal Verification of a Coordinated Atomic Action Based Design

Andreas Küchler
On the Correspondence between Neural Folding Architectures and Tree Automata

Heiko Neumann, Thorsten Hansen, Luiz Pessoa
Interaction of ON and OFF Pathways for Visual Contrast Measurement

Thomas Wennekers
Synfire Graphs: From Spike Patterns to Automata of Spiking Neurons

Thomas Bauer, Peter Dadam
Variable Migration von Workflows in ADEPT

Heiko Neumann, Wolfgang Sepp
Recurrent V1 – V2 Interaction in Early Visual Boundary Processing

Frank Houdek, Dietmar Ernst, Thilo Schwinn
Prüfen von C–Code und Statmate/Matlab–Spezifikationen: Ein Experiment
Gerhard Schellhorn
Proving Properties of Directed Graphs: A Problem Set for Automated Theorem Provers

Gerhard Schellhorn, Wolfgang Reif
Theorems from Compiler Verification: A Problem Set for Automated Theorem Provers

Mohammad Ali Livani
SHARE: A Transparent Mechanism for Reliable Broadcast Delivery in CAN

Mohammad Ali Livani, Jörg Kaiser
Predictable Atomic Multicast in the Controller Area Network (CAN)

Susanne Boll, Wolfgang Klas, Utz Westermann
A Comparison of Multimedia Document Models Concerning Advanced Requirements

Thomas Bauer, Peter Dadam
Verteilungsmodelle für Workflow-Management-Systeme - Klassifikation und Simulation

Uwe Schöning
On the Complexity of Constraint Satisfaction

Ercument Canver
Model-Checking zur Analyse von Message Sequence Charts über Statecharts

Johannes Köbler, Wolfgang Lindner, Rainer Schuler
Derandomizing RP if Boolean Circuits are not Learnable

Utz Westermann, Wolfgang Klas
Architecture of a DataBlade Module for the Integrated Management of Multimedia Assets

Peter Dadam, Manfred Reichert

Vikraman Arvind, Johannes Köbler
Graph Isomorphism is Low for ZPP^NP and other Lowness results

Thomas Bauer, Peter Dadam
Efficient Distributed Workflow Management Based on Variable Server Assignments

Thomas Bauer, Peter Dadam
Variable Serverzuordnungen und komplexe Bearbeiterzuordnungen im Workflow-Management-System ADEPT

Gregory Baratoff, Christian Toepfer, Heiko Neumann
Combined space-variant maps for optical flow based navigation

Wolfgang Gehring
Ein Rahmenwerk zur Einführung von Leistungspunktsystemen
2000-05  Susanne Boll, Christian Heinlein, Wolfgang Klas, Jochen Wandel
Intelligent Prefetching and Buffering for Interactive Streaming of MPEG Videos

2000-06  Wolfgang Reif, Gerhard Schellhorn, Andreas Thums
Fehlersuche in Formalen Spezifikationen

2000-07  Gerhard Schellhorn, Wolfgang Reif (eds.)

2000-08  Thomas Bauer, Manfred Reichert, Peter Dadam
Effiziente Durchführung von Prozessmigrationen in verteilten Workflow-Management-Systemen

2000-09  Thomas Bauer, Peter Dadam
Vermeidung von Überlastsituationen durch Replikation von Workflow-Servern in ADEPT

2000-10  Thomas Bauer, Manfred Reichert, Peter Dadam
Adaptives und verteiltes Workflow-Management

2000-11  Christian Heinlein
Workflow and Process Synchronization with Interaction Expressions and Graphs

2001-01  Hubert Hug, Rainer Schuler
DNA-based parallel computation of simple arithmetic

2001-02  Friedhelm Schwenker, Hans A. Kestler, Günther Palm
3-D Visual Object Classification with Hierarchical Radial Basis Function Networks

2001-03  Hans A. Kestler, Friedhelm Schwenker, Günther Palm
RBF network classification of ECGs as a potential marker for sudden cardiac death

2001-04  Christian Dietrich, Friedhelm Schwenker, Klaus Riede, Günther Palm
Classification of Bioacoustic Time Series Utilizing Pulse Detection, Time and Frequency Features and Data Fusion

2002-01  Stefanie Rinderle, Manfred Reichert, Peter Dadam
Effiziente Verträglichkeitsprüfung und automatische Migration von Workflow-Instanzen bei der Evolution von Workflow-Schemata

2002-02  Walter Guttmann
Deriving an Applicative Heapsort Algorithm

2002-03  Axel Dold, Friedrich W. von Henke, Vincent Vialard, Wolfgang Goerigk
A Mechanically Verified Compiling Specification for a Realistic Compiler

2003-01  Manfred Reichert, Stefanie Rinderle, Peter Dadam
A Formal Framework for Workflow Type and Instance Changes Under Correctness Checks

2003-02  Stefanie Rinderle, Manfred Reichert, Peter Dadam
Supporting Workflow Schema Evolution By Efficient Compliance Checks

2003-03  Christian Heinlein
Safely Extending Procedure Types to Allow Nested Procedures as Values
Stefanie Rinderle, Manfred Reichert, Peter Dadam
On Dealing With Semantically Conflicting Business Process Changes.

Christian Heinlein
Dynamic Class Methods in Java

Christian Heinlein
Vertical, Horizontal, and Behavioural Extensibility of Software Systems

Christian Heinlein
Safely Extending Procedure Types to Allow Nested Procedures as Values
(Corrected Version)

Changling Liu, Jörg Kaiser
Survey of Mobile Ad Hoc Network Routing Protocols

Thom Frühwirth, Marc Meister (eds.)
First Workshop on Constraint Handling Rules

Christian Heinlein
Concept and Implementation of C++, an Extension of C++ to Support User-Defined Operator Symbols and Control Structures

Susanne Biundo, Thom Frühwirth, Günther Palm (eds.)
Poster Proceedings of the 27th Annual German Conference on Artificial Intelligence

Armin Wolf, Thom Frühwirth, Marc Meister (eds.)
19th Workshop on (Constraint) Logic Programming

Wolfgang Lindner (Hg.), Universität Ulm, Christopher Wolf (Hg.) KU Leuven
2. Krypto-Tag – Workshop über Kryptographie, Universität Ulm

Walter Guttmann, Markus Maucher
Constrained Ordering

Stefan Sarstedt
Model-Driven Development with ACTIVECHARTS, Tutorial

Alexander Raschke, Ramin Tavakoli Kolagari
Ein experimenteller Vergleich zwischen einer plan-getriebenen und einer leichtgewichtigen Entwicklungsmethode zur Spezifikation von eingebetteten Systemen

Jens Kohlmeyer, Alexander Raschke, Ramin Tavakoli Kolagari
Eine qualitative Untersuchung zur Produktlinien-Integration über Organisationsgrenzen hinweg

Thorsten Liebig
Reasoning with OWL - System Support and Insights –

H.A. Kestler, J. Messner, A. Müller, R. Schuler
On the complexity of intersecting multiple circles for graphical display
2008-02  Manfred Reichert, Peter Dadam, Martin Jurisch, Ulrich Kreher, Kevin Göser, Markus Lauer  
Architectural Design of Flexible Process Management Technology

2008-03  Frank Raiser  
Semi-Automatic Generation of CHR Solvers from Global Constraint Automata

2008-04  Ramin Tavakoli Kolagari, Alexander Raschke, Matthias Schneiderhan, Ian Alexander  
Entscheidungsdokumentation bei der Entwicklung innovativer Systeme für produktlinien-basierte Entwicklungsprozesse

2008-05  Markus Kalb, Claudia Dittrich, Peter Dadam  
Support of Relationships Among Moving Objects on Networks

2008-06  Matthias Frank, Frank Kargl, Burkhard Stiller (Hg.)  
WMAN 2008 – KuVS Fachgespräch über Mobile Ad-hoc Netzwerke

2008-07  M. Maucher, U. Schöning, H.A. Kestler  
An empirical assessment of local and population based search methods with different degrees of pseudorandomness

2008-08  Henning Wunderlich  
Covers have structure

2008-09  Karl-Heinz Niggl, Henning Wunderlich  
Implicit characterization of FPTIME and NC revisited

2008-10  Henning Wunderlich  
On span-P^{cc} and related classes in structural communication complexity

2008-11  M. Maucher, U. Schöning, H.A. Kestler  
On the different notions of pseudorandomness

2008-12  Henning Wunderlich  
On Toda’s Theorem in structural communication complexity

2008-13  Manfred Reichert, Peter Dadam  
Realizing Adaptive Process-aware Information Systems with ADEPT2

2009-01  Peter Dadam, Manfred Reichert  
The ADEPT Project: A Decade of Research and Development for Robust and Flexible Process Support - Challenges and Achievements

2009-02  Peter Dadam, Manfred Reichert, Stefanie Rinderle-Ma, Kevin Göser, Ulrich Kreher, Martin Jurisch  

2009-03  Alena Hallerbach, Thomas Bauer, Manfred Reichert  
Correct Configuration of Process Variants in Provop
2009-04  
Martin Bader
On Reversal and Transposition Medians

2009-05  
Barbara Weber, Andreas Lanz, Manfred Reichert
Time Patterns for Process-aware Information Systems: A Pattern-based Analysis

2009-06  
Stefanie Rinderle-Ma, Manfred Reichert
Adjustment Strategies for Non-Compliant Process Instances

2009-07  
Statistical Computing 2009 – Abstracts der 41. Arbeitstagung

2009-08  
Ulrich Kreher, Manfred Reichert, Stefanie Rinderle-Ma, Peter Dadam
Effiziente Repräsentation von Vorlagen- und Instanzdaten in Prozess-Management-Systemen

2009-09  
Dammertz, Holger, Alexander Keller, Hendrik P.A. Lensch
Progressive Point-Light-Based Global Illumination

2009-10  
Dao Zhou, Christoph Müssel, Ludwig Lausser, Martin Hopfensitz, Michael Kühl, Hans A. Kestler
Boolean networks for modeling and analysis of gene regulation

2009-11  
J. Hanika, H.P.A. Lensch, A. Keller
Two-Level Ray Tracing with Recordering for Highly Complex Scenes

2009-12  
Stephan Buchwald, Thomas Bauer, Manfred Reichert
Durchgängige Modellierung von Geschäftsprozessen durch Einführung eines Abbildungsmodells: Ansätze, Konzepte, Notationen

2010-01  
Hariolf Beth, Frank Raiser, Thom Frühwirth
A Complete and Terminating Execution Model for Constraint Handling Rules

2010-02  
Ulrich Kreher, Manfred Reichert
Speichereffiziente Repräsentation instanzspezifischer Änderungen in Prozess-Management-Systemen

2010-03  
Patrick Frey
Case Study: Engine Control Application

2010-04  
Matthias Lohrmann und Manfred Reichert
Basic Considerations on Business Process Quality

2010-05  
HA Kestler, H Binder, B Lausen, H-P Klenk, M Schmid, F Leisch (eds):
Statistical Computing 2010 - Abstracts der 42. Arbeitstagung

2010-06  
V. Künzle, B. Weber, M. Reichert:
Object-aware Business Processes: Properties, Requirements, Existing Approaches