Batch Regions: Process Instance Synchronization based on Data

Luise Pufahl, Andreas Meyer, Mathias Weske

Technische Berichte Nr. 86

des Hasso-Plattner-Instituts für
Softwaresystemtechnik
an der Universität Potsdam

Technische Berichte des Hasso-Plattner-Instituts für
Softwaresystemtechnik an der Universität Potsdam

Technische Berichte des Hasso-Plattner-Instituts für
Softwaresystemtechnik an der Universität Potsdam | 86

Luise Pufahl | Andreas Meyer | Mathias Weske

Batch Regions

Process Instance Synchronization based on Data

Universitätsverlag Potsdam

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.dnb.de/ abrufbar.

Universitätsverlag Potsdam 2014
http://verlag.ub.uni-potsdam.de/

Am Neuen Palais 10, 14469 Potsdam
Tel.: +49 (0)331 977 2533 / Fax: 2292
E-Mail: verlag@uni-potsdam.de

Die Schriftenreihe **Technische Berichte des Hasso-Plattner-Instituts für Softwaresystemtechnik an der Universität Potsdam** wird herausgegeben von den Professoren des Hasso-Plattner-Instituts für Softwaresystemtechnik an der Universität Potsdam.

ISSN (print) 1613-5652
ISSN (online) 2191-1665

Online veröffentlicht auf dem Publikationsserver der Universität Potsdam
URL http://pub.ub.uni-potsdam.de/volltexte/2014/6908/
URN urn:nbn:de:kobv:517-opus-69081
http://nbn-resolving.de/urn:nbn:de:kobv:517-opus-69081

Zugleich gedruckt erschienen im Universitätsverlag Potsdam:
ISBN 978-3-86956-280-3

Batch Regions: Process Instance Synchronization based on Data

Luise Pufahl, Andreas Meyer, and Mathias Weske

Hasso Plattner Institute at the University of Potsdam
{Luise.Pufahl,Andreas.Meyer,Mathias.Weske}@hpi.uni-potsdam.de

Abstract. Business process automation improves organizations' efficiency to perform work. In existing business process management systems, process instances run independently from each other. However, synchronizing instances carrying similar characteristics, i.e., sharing the same data, can reduce process execution costs. For example, if an online retailer receives two orders from one customer, there is a chance that they can be packed and shipped together to save shipment costs. In this paper, we use concepts from the database domain and introduce *data views* to business processes to identify instances which can be synchronized. Based on data views, we introduce the concept of *batch regions* for a context-aware instance synchronization over a set of connected activities. We also evaluate the concepts introduced in this paper with a case study comparing costs for normal and batch processing.

Keywords: BPM, batch processing, process instance grouping, data view

1 Introduction

Companies use business process management systems (BPMS) to automate processes, especially those with a high degree of repetition. Therefore, the process is first documented as process model. The process model serves as blueprint for a number of process instances whereby one instance represents the execution of one business case [20]. In existing BPMSs, e.g., [2, 3, 6], instances of a process usually run independently from each other. However, there are scenarios in which process instances can and shall be grouped and processed as one batch. For example, online retailers handle many orders per day. They face the situation that customers place several orders with the same shipping address within a short time-frame. As these orders are then processed completely independent from each other resulting in one package per order, this behavior causes avoidable costs for transportation. By synchronizing the processing of orders from same customers, several articles can be shipped in one package reducing the total shipping costs. Synchronization of instances can be realized by batch processing [1, 7, 11]. Considering the requirements from [11] to integrate batch processing into process models, existing works are not complete. For instance, they they do not support rule-based batch activation, multiple resource allocation of batches, or differentiation of process instances.

In this paper, we introduce *batch regions* to group process instances with similar characteristics expressed as *data views* and synchronize their execution over a number of activities to fulfill the defined requirements. Therefore, we generalize the batch activity

concept from [11]. There, process instances are assumed to be homogenous such that they are grouped into batches based on their arrival ignoring heterogeneous demands. To overcome this limitation, we utilize contextual information to differentiate process instances. Each instance acts on multiple data objects which give an instance the context and characterizes it. However, not all data is relevant to compare process instances and to find relations; only specific attributes of the utilized data objects are of interest. The online retailer may identify related process instances by identical customer identifier and shipping address. In database systems research, the concept of *views* allows to extract relevant data by projection [16]. Summarized, this paper will provide these two contributions:

Data views on process instances: The concept of views from the database domain is applied to business processes to identify related process instances based on data. The introduced concept is called data views for process instances.

Batch regions: We utilize the data views to generalize the batch activity concept allowing to group process instances into batches based on specific characteristics and to synchronize their execution for one or several activities connected by sequence flow.

The remainder of this paper is structured as follows. In Section 2, we introduce the online retailer scenario in detail to discuss the motivation and challenges for grouping and synchronizing process instances. Afterwards, we introduce the foundation of our approach in Section 3 before Section 4 discusses the concept of data views formally and application-wise describing the corresponding algorithms. Section 5 presents the batch region concept and its operational semantics whose application is discussed as case study in Section 6. Section 7 is devoted to related work and Section 8 concludes the paper.

2 Motivating Example

An online retailer receives hundreds of orders per day which are processed as described in the process model shown in Fig. 1. Using this example, we present the opportunities and challenges of synchronizing the execution of related process instances.

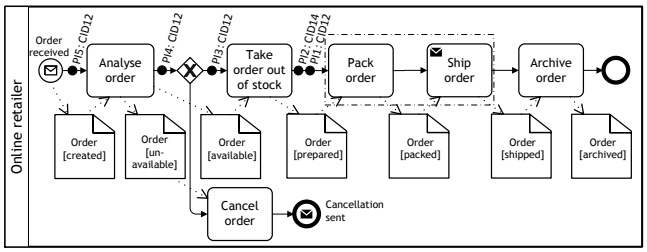
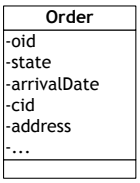


Fig. 1. Process model of an online retailer and exemplary running instances represented as labeled token. The label, e.g., PI1: CID12, references the respecting process instance ID and customer ID.

After receiving an order from a customer, the online retailer analyzes the order. If the ordered articles are available, they are taken from the warehouse. Otherwise – in case of unavailability of the articles – the customer is informed about order cancellation. Next, the taken articles are packed into a package and shipped to the customer. Afterwards, the

order is archived. As mentioned above, the process model serves as blueprint for a set of process instances. Exemplary, five running instances are represented as labeled token in the process model in Fig. 1.

The process utilizes one business object: the *Order*. This is represented by associating *Order* data nodes to the activities as input (read) or output (write); data nodes represent data objects on the model level. For instance, activity *Analyze order* reads object *Order* in data state *created* and updates it to state *analyzed*. All nodes with the same object name reference the same data class which describes the object's structure, i.e., the attributes and allowed data states. Fig. 2a shows the structure of the referenced *Order* class consisting of a unique identifier *oid*, the customer ID *cid* etc. During process execution, a data object of class *Order* is created by the first activity and updated multiple times afterwards. Fig. 2b shows multiple entries, each referring to one distinct *Order* object of the five process instances represented in Fig.1; here the instance ID refers to the order *oid*.



(a) Data class.

Order					
oid	**state**	**arrivalDate**	**address**	**cid**	**...**
1	prepared	9.10.2013	Anystreet 1	CID12	...
2	prepared	9.10.2013	Street 5	CID14	...
3	available	10.10.2013	Teststreet 2	CID12	...
4	available	10.10.2013	Anystreet 1	CID12	...
5	created	10.10.2013	Anystreet 1	CID12	...
...	...	...	...	...	...

(b) Database table representation.

Fig. 2. Data class *Order* for the *online retailer* process with exemplary data objects of five instances.

As discussed in Section 1, customers may place multiple orders with the same shipping address shortly after another. For example, the orders relating to process instances *PI1*, *PI3*, *PI4* and *PI5* refer to one customer with ID *CID12*. Usually, those orders are processed independently – each resulting in an individual package which is shipped to the customer. However, due to the same shipping address, instances *PI1*, *PI4*, and *PI5* can be synchronized while packing and shipping the articles. This leads to a reduction of total shipping costs for the online retailer by leveraging synergy effects. Process instance *PI3* cannot be processed with *PI1*, because the order has to be shipped to another address. Analogously, *PI2* is processed independently from the other instances due to different customer ID and shipping address. This example implicates that we need to select groups of process instances to be processed within one batch based on their data. This batch must comprise multiple activities and requires means to decide when to activate it. E.g., if the waiting time for the first arrived instance is still acceptable and another similar one is in progress, both can be grouped into one batch. If the waiting time gets too long, batch processing of the already arrived instances starts while the other one initializes a new batch. Further, an instance may take a path through the model which bypasses the batch processing as, for instance, the xor split in Fig. 1 allows. Extending the requirements from [11], these conclusions can be summarized in four requirements:

RQ1—Identify related process instances based on values of relevant data attributes.

RQ2—Determine the relevant attributes.

RQ3—Synchronize process instance execution for multiple connected activities.

RQ4—Use information of running process instances to reasoning about the activation of batch processing.

3 Foundation on Process and Data Modeling

We proceed with introducing formalisms for process and data modeling which we then use for defining the technique for process instance grouping in Section 4. First, we give a generic process model definition and require it to be syntactically correct with respect to the used modeling notation. Behaviorally, we require that it terminates for all execution paths of the model in exactly one of probably multiple end events and that every node participates in at least one execution path, i.e., the process model must be lifelock and deadlock free. Formally, we define a process model as follows.

Definition 1 (Process Model).
A *process model* $m = (N, D, \mathfrak{C}, \mathfrak{F}, type)$ consists of a finite non-empty set $N \subseteq A \cup E \cup G$ of control flow nodes being activities A, events E, and gateways G (A, E, and G are pairwise disjoint) and a finite non-empty set D of data nodes (N and D are disjoint). $\mathfrak{C} \subseteq N \times N$ is the control flow relation specifying the partial ordering of activities and $\mathfrak{F} \subseteq (A \times D) \cup (D \times A)$ is the data flow relation specifying input and output data dependencies of activities. Function $type : G \rightarrow \{and, xor\}$ gives each gateway a type. $\diamond$

Fig. 1 shows a process model with one start event, two end events, six activities, and multiple data nodes. Each data node has a name, e.g., *Order*, and a specific data state, e.g., *analyzed* or *shipped*. All these nodes share the same name referencing the data class *Order* which describes the structure of data objects and the data states; a data node maps to exactly one data class. A data state denotes a situation of interest for the execution of the business process. For instance, state *shipped* of object *Order* indicates that all pre-steps like analyzing the order as well as packing and handing it over to the postal service are successfully executed and that the package is on the way to the customer.

Definition 2 (Data Class).
A *data class* $c = (J, S)$ consists of a finite set J of attributes and a finite non-empty set S of data states (J and S are disjoint). Each attribute $j \in J$ is fully qualified allowing to determine the actual attribute and the corresponding data class. C denotes the set of data classes utilized in the process model. $\diamond$

We use subscripts, e.g., N_m and J_c to denote the relation of sets and functions to process model m and data class c respectively and omit subscripts where the context is clear. The same holds for the instance level concepts we introduce below. On instance level, an arbitrary set of data objects exists. Each data object maps to exactly one data node and therefore one data class. Formally, a data object is defined as follows.

Definition 3 (Data Object).
Let D be a set of data nodes and let C be a set of data classes, then is $o = (T_S, \alpha_D, \alpha_C)$ a *data object* consisting of a sequence $T_S =< s_1, s_2, \ldots, s_n >$ of data states where each $s_i \in S_c$ with S_c denoting the set of data states of the corresponding data class $c = \alpha_C(o)$. Functions $\alpha_D : O \rightarrow D$ and $\alpha_C : O \rightarrow C$ refer each data object o to the corresponding data node $d \in D$ and data class $c \in C$ respectively. O denotes the set of data objects utilized during process model execution. $\diamond$

At any point in time, a data object is in exactly one data state. The state may change over time by being updated by activities which is represented by data nodes. For example, the activity *Take order out of stock* updates the *Order* data object to the state *prepared* (cf. Fig. 1). Each data state of a specific data object refers to a set of values for specific attributes defined as follows.

Definition 4 (Data State).
A *data state* $s_O = (V, \gamma)$ of a data object o consists of a finite set V of values. Function $\gamma : J_C \rightarrow V$ refers each attribute of data class c to a value $v \in V$. Thereby, data state $s \in T_S$ of the corresponding object o and $c = \alpha_C(o)$ hold. $\diamond$

At any point in time, each attribute can get assigned a value. If it is not defined, the value is set to $\perp$. In our example, the *Order* object with *oid* = 1 in state *prepared* consists of multiple values relating to attributes; for instance, the value *CID12* refers to the attribute *cid*. Executions of process models are represented by process instances with each instance belonging to exactly one model. At any point in time, the process instance has a current *process instance state* z which consists of a finite non-empty set H of states of data objects $O_Z \subseteq O$, where each object $o \in O_Z$ belongs to a different data class c, i.e., $\bigcap_{O_Z} \alpha_C(o_i) = \emptyset$. Thus, the current state z of the process instance *PI1* in the example process consists of its corresponding *Order* object state *prepared* presented above. A sequence of process instance states describes a process instance which we define as follows.

Definition 5 (Process Instance).
Let m be a process model from the set M of process models, then is $i = (pid, T_Z, \mu)$ a *process instance* consisting of a process instance identifier pid, a sequence $T_Z =< z_1, z_2, \ldots, z_m >$ of process instance states where each $z_i \in Z_m$ with Z_m denoting the set of process instance states of the corresponding process model $m = \mu(i)$. Function $\mu : I \rightarrow M$ refers each process instance i from the set of process instances I to its corresponding process model m. $\diamond$

For an implementation, the correlation between a process instance and its corresponding data objects can be realized via the object's primary and foreign keys as discussed in [10].

4 Process Instance Grouping based on Data Views

Utilizing the process and data modeling concepts, we introduce our approach to group process instances based on their data characteristics and synchronize their execution at predefined positions inside a process model – at so-called batch regions which are discussed in Section 5. We use the concept of *data views* for the grouping, which is an abstracted view on the process instance data. At run-time, each process instance gets a data view assigned which is updated dynamically with instance progressing and changing data objects. First, we introduce the *data view function* in Section 4.1 upon which *data view clusters* can be created; each containing a set of related process instances. Afterwards, we describe the algorithms to create a data view for a given process instance and its assignment to a data view cluster in Section 4.2.

4.1 Data View Formalisms

The data view is a projection on the values of multiple data object attributes in a specific state of one process instance presenting only the relevant ones as specified by the stakeholder. Therefore, the function requires as input a set of attributes being of interest for a business situation as well as the current state of a process instance being executed in this business environment. The attributes of interest indicate upon which aspects instances should be grouped. Formally, we define the function as follows.

Definition 6 (Data View Function).
Let X be a set of fully qualified data class attributes denoting the attributes of interest and let Z be a set of process instance states, then the *data view function* $\varphi : 2^X \times Z \rightarrow 2^V$ returns a set of relevant values V, referred to as *data view*, for a given process instance state $z \in Z$ and a given set of fully qualified data object attributes X, referred to as *data view definition*. 2^X and 2^V denote the power set of sets X and V respectively. $\diamond$

The data view function utilizes function γ of Definition 4. As γ assigns one value to each attribute of a data object depending on its current data state, φ can be partitioned into a conjunction of many functions γ – one for each attribute x_i comprised by X, the set of fully qualified attributes of interest. Let $\psi(o_1)$ return the current state h_1 of the object o_1. Then, let $z = \{h_1, h_2, \ldots, h_n\}$ be a process instance state consisting of multiple data states h_i of different data objects. Via data class matching, only those γ_{h_i} of the data states in z are selected whose data state refers to a relevant data class used in the set X. Formally, we partition the data view function such that $\varphi(X, z) := \bigcup_{x_j \in c = \alpha_C(o_{h_i})} \gamma_{h_i}(x_j), h_i \in z, x_j \in X \subseteq \bigcup_{c \in C} J_c$. In the online retailer process, it is aimed to group the process instances with respect to the customer identifier and the customer address (cf. Fig. 2). Thus, the set of fully qualified interesting attributes X comprises $\mathtt{x_1}$ `= Order.cid` and $\mathtt{x_2}$ `= Order.address`; we reference this data view definition as *OrdersBySameCustomer* (cf. Fig. 3). Considering process instance *PI1*, after execution of activity *Take order out of stock*, state z consists of data state $\mathtt{h_1}$ `= ψ(Order) = prepared`, the current state of the corresponding *Order* object. For both attributes of interest in X, only γ_{h_1} of object *Order* is needed so that $\varphi(X, z) = \gamma_{h_1}(x_1) \cup \gamma_{h_1}(x_2)$ holds. The resulting data view for *PI1* is {`Anystreet 1, CID12`} which is highlighted together with the remaining instance data views in the table in Fig. 3.

Data View Definition:
OrdersBySameCustomer

Order					
oid	**state**	**arrivalDate**	**address**	**cid**	**...**
1	prepared	9.10.2013	Anystreet 1	CID12	...
2	prepared	9.10.2013	Street 5	CID14	...
3	available	10.10.2013	Teststreet 2	CID12	...
4	available	10.10.2013	Anystreet 1	CID12	...
5	created	10.10.2013	Anystreet 1	CID12	...
...	...	...	...	...	...

Fig. 3. Example of a data view definition and the resulting data view clusters.

Instances of one process model can be grouped based on their data view by assigning each to one *data view cluster*. Process instances with identical data views are collected in the same cluster. A data view cluster is defined as follows.

Definition 7 (Data View Cluster).
Let I be a set of process instances, then is a *data view cluster* $q = (k, W)$ a set of related process instances $W \subseteq I$ of the same process model $m = \mu(w), w \in W$ characterized

by a data view k. Relatedness of two process instances is denoted by equality of their data views, i.e., both instances return the same set of values for their respective current process instance state. Q and K denote the set of all data view clusters and the corresponding keys respectively. $\diamond$

The set of all data view clusters for one process model is a key value store such that each cluster represents one entry. As illustrated in Fig. 3, grouping the process instances *PI1* to *PI5* based on data view definition *OrdersBySameCustomer*, process instances *PI1*, *PI4*, and *PI5* belong to one cluster while *PI2* and *PI3* belong to two separate clusters because either the customer identifier (*PI2*) or the customer address (*PI3*) differs.

4.2 Algorithms

We utilize the above introduced formalisms to introduce the two algorithms allowing to create data views and cluster them appropriately. To ease algorithm presentation, we require five additional functions which we introduce first. Function $\eta : Z \rightarrow 2^O$ returns the set of all data objects $o_i \in O$ for a given process instance state $z \in Z$. Function $\xi : X \rightarrow C$ returns the data class $c \in C$ for a given data attribute $x \in X$. Function $\psi : O \rightarrow S$ returns the current data state $s \in S$ for a given data object $o \in O$. Function $\lambda : Q \rightarrow K$ returns the key $k \in K$ for given data view cluster $q \in Q$. Function $\kappa : Z \rightarrow I$ returns the process instance $i \in I$ for a given process instance state $z \in Z$.

Algorithm 1: Data view creation.

Input: X, z
Output: $dataView = \varphi(X, z)$
1: $dataView \leftarrow$ null;
2: $object \leftarrow$ null;
3: $value \leftarrow$ null;
4: $O \leftarrow \eta(z)$;
5: **for all** $x \in X$ **do**
6: **for all** $o \in O$ **do**
7: **if** $\alpha_C(o) == \xi(x)$ **then**
8: $object \leftarrow o$;
9: break;
10: **end if**
11: **end for**
12: $value \leftarrow \gamma_{\psi(object)}(x)$;
13: $dataView$.add(v);
14: **end for**

Algorithm 2: Cluster assignment.

Input: $dataView = \varphi(X, z)$, z, Q
Output: Q
1: $clusterFound$ = false;
2: **for all** $q \in Q$ **do**
3: **if** $dataView == \lambda(q)$ **then**
4: q.add($\kappa(z)$);
5: $clusterFound$ = true;
6: break();
7: **end if**
8: **end for**
9: **if** $clusterFound$ == false **then**
10: $q \leftarrow$ null;
11: $q.k \leftarrow dataView$;
12: q.add($\kappa(z)$);
13: Q.add(q);
14: **end if**

Algorithm 1 describes the implementation of the data view function introduced in Definition 6. As discussed, the data view function requires a set $X \subseteq \bigcup_{c \in C} J_c$ of relevant and fully classified data attributes and a process instance state $z \in Z$ to compute the data view of the corresponding process instance $\kappa(z)$. First, variables to hold the data view, object information, and a data attribute value are initialized for later usage (lines 1 to 3). In line 4, function η returns all data objects that are utilized in the given process instance state z. Lines 5 to 14 iterate over each single relevant attribute $x \in X$

to compute the current value of this attribute x_i in the given process instance state z. Lines 7 to 10 check for each object $o_i \in O$ whether its data class corresponds to the data class of attribute x_i currently processed until it finds a correspondence. Then, the object is stored in the above initialized variable (line 8) and the iteration is aborted (line 9). Remember that each object utilized within a process instance state refers to a different data class. After identifying the corresponding *object*, function $\gamma_{\psi(object)}$ retrieves the current value of attribute x_i and stores it in the prepared variable (line 12) before line 13 adds this value to the data view. After iterating over all attributes x_i, the data view is fully computed with the values corresponding to the order of the attributes in set X. Thereby, we assume that X is an ordered set and the iterations are ordering preservative.

After computing the data view of a process instance based on its current state z and the set X of relevant data attributes, Algorithm 2 assigns this data view to a corresponding data view cluster (cf. Definition 7). In addition to the data view, Algorithm 2 requires the state z of the corresponding process instance and the set Q of currently existing data view clusters for the respecting process model $m = \mu(\kappa(z))$; this set may also be empty. The output of this algorithm will be the updated set Q of data view clusters. Either it is extended by one cluster $q \in Q$ containing the given process instance $\kappa(z)$ or it consists of the same number of clusters with one of them now containing the mentioned process instance $\kappa(z)$. For computation, first, a Boolean variable indicating whether a matching cluster was found, gets initialized with value *false* (line 1). Lines 2 to 8 iterate over all input cluster $q \in Q$ to check whether there exists one with a key k being equal to the input data view. If such a cluster q_i is found, the process instance $\kappa(z)$ is added to that cluster (line 4). The corresponding Boolean variable is set to *true* (line 5), and the iteration is aborted (line 6). In this case, the algorithm already succeeded in identifying the fitting data view cluster and terminates. Otherwise, if no corresponding cluster q_i was found, lines 9 to 14 create a new cluster q (line 10). The new cluster q gets initialized with the given data view as key k (line 11) and gets the process instance $\kappa(z)$ assigned (line 12). Finally, this newly created cluster is added to the set Q of data view clusters (line 13) resulting in satisfactory algorithm completion.

By applying these two algorithms to all instances of a process model at a certain point in time, the process instances in their respective states can be grouped into data view clusters based on the data information they carry at that point in time. However, the algorithm may also be applied on a subset of process instances only; for instance those a stakeholder is interested in. Referring to batch processing and the batch region which will be discussed in the next section, this subset may be all process instances which have not yet reached the batch region, i.e., a specific process instance state. Such filtering may also be applied on other criteria, e.g., process instances started between 5am and 2pm each day.

5 Batch Region

In [11], the concept of batch activities was introduced to synchronize the execution of several instances of an atomic activity. Thereby, the respective activity gets a batch model assigned with different configuration parameters allowing to configure the batch execution by selecting an activation rule and by defining the maximum batch size

(capacity) as well as the way of execution (parallel vs. sequential). The configured batch activity describes the behavior for a set of batch instances, where each batch instance manages one batch execution. Upon enablement of an activity instance of a batch activity, it is assigned to an available batch instance. Thereby, the instances are not differentiated; in sequence of their arrival, they are simply added to the available batch instance, i.e., the batch instance which has neither started its execution nor reached its capacity.

We generalize this concept with respect to the requirements discussed in Section 2 by introducing *batch region*s. Thereby, we preserve the functionality of the original concept. Section 5.1 presents the concept of batch regions and describes its integration in process modeling and its configuration parameters. In Section 5.2, we present the corresponding execution semantics.

5.1 Modeling

A process model consists of control flow nodes connected by the control flow relation represented by edges. As the batch handling shall be extended to a set of connected control flow nodes, we propose the concept of a *batch region* which flexibly surrounds a specified number of connected nodes with a single entry, i.e., one specific node defines the entry to the batch region. It describes the conditions for the batch execution and can be configured by the process designer based on the parameters *grouping characteristic*, *activationRule*, *maxBatchSize*, and *executionOrder*.

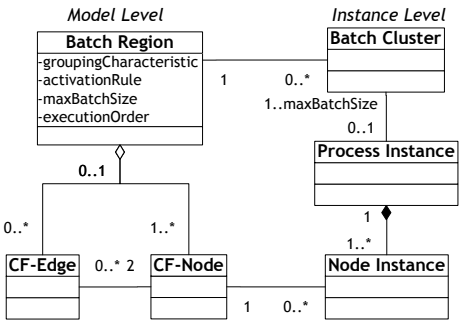


Fig. 4. Conceptual model for batch regions.

Fig. 4 illustrates the concept of batch regions. At model level, a batch region consists of at least one control flow node and a correlating number of control flow edges such that two nodes are connected by one edge. Within a batch region, activities, events, and gateways may appear with the limitation that conditions on exclusive gateways (type = xor) must be designed such that all process instances of one batch follow the same path. Next, we introduce the configuration parameters of a batch region:

The *groupingCharacteristic* defines how the process instances are grouped by specifying the relevant attributes for identifying related instances (cf. data view definition). We assume that process instances do not change their data view within the batch region, i.e., no batch region activity updates an attribute specified in the data view definition. If no data view definition is provided, process instances are grouped upon their arrival order.

The *activationRule* specifies when a batch cluster is enabled for execution allowing to balance costs and waiting times. Analogously to [11], the process designer selects an activation rule type and provides the required user inputs.

The *maxBatchSize* limits the capacity of a batch cluster by specifying the maximum number of activity instances processed in a batch. It can be used to incorporate limits of involved resources, e.g., at most three articles fit in one package.

The *executionOrder* describes whether the process instances comprised within one batch cluster are synchronized parallelly or sequentially. Parallel execution means that all

instances for one activity are executed simultaneously and get terminated before the next activity is executed the same way. Sequential execution can be activity-based, the sequential variant of parallel execution, or case-based where the activities within a batch region are executed for one process instance (case) before the next one can be started.

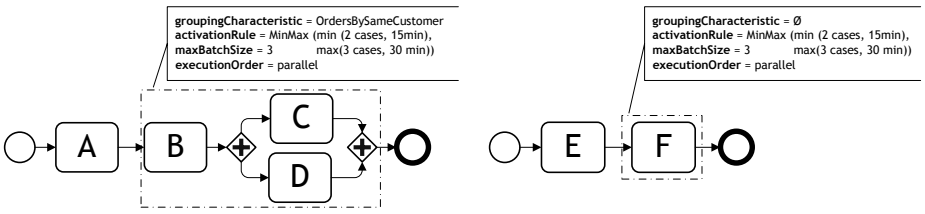
A batch region has an arbitrary number of batch clusters. A cluster's behavior (instance level) is defined by the batch region's configuration. Each batch cluster comprises a positive natural number of process instances having the same data view with *maxBatchSize* specifying the upper limit. A process instance consists of node instances from those only the ones are relevant which are surrounded by the batch region.

In the remainder of this section, we discuss the activation rule in detail and provide two example configurations explaining concept application. The activation rule allows to optimize the batch processing. The later a batch is started, the more process instances are synchronized resulting in lower total execution costs at the price of increasing waiting times. In customer relations, more waiting times increases the risk of losing customers [17]. Therefore, specifying the activation rule requires to find an optimal trade-off between reduced execution costs and increased waiting times. The optimal configuration settings are derived from expert knowledge, simulations, or statistical evaluations. Different types of activation rules are provided by process engine suppliers with each activation rule type relying on the concept of Event-Condition-Action (ECA) rules. An action A is performed if event E is recognized and condition C is satisfied. In this paper, the action is always the activation of the respective batch cluster. Earlier proposed activation rule types focus on the instances in the scope of the batch activity only [11], e.g., the *threshold rule* which enables batch processing as soon as a certain number of process instances is available. In this paper, we introduce an activity rule type where information about running process instances is considered additionally.

This information may be integrated into the event or condition definition of an activation rule. Below, we provide a general example of such *MinMax* activation rule. This rule activates a batch cluster if a minimal number of instances are assigned to it and no other related instances being before the batch region run. Otherwise, if at least one process instance with the same data view can be observed, the activation is postponed until the cluster's size equals a maximum. The rule of a batch cluster `bc` contains a composite event that triggers the check of the condition if a new process instance `PI` was added to `bc` or if no instance `PI` was added for `w` time units. The condition is a logical expression requiring that either the minimum condition is true while there exists no other related instance or the maximum condition is true to trigger the action. The existence of other process instances is checked with function `ExistingEqualPI()`. The function first creates the data views for all running process instances in front of the the batch region by utilizing Algorithm 1 and then assigns each one to a data view cluster as described in Algorithm 2. If there exists a data view cluster with the same key as the *bc*, the function returns *true*. Here, the minimum condition requires that the batch cluster size is greater or equal a given *minimal number* or a defined *minimal waiting time* passed. The maximum condition is satisfied if the batch cluster size is equal to the *maximal number* or if the *maximal waiting time* passed. The configuration of the two conditions is up to the process designer. We propose to include timing constraints within the minimum and maximum conditions in order to avoid deadlocks.

```
ActivationRule MinMax rule
 On Event      (PI added to bc) OR (No PI since w time units)
 If Condition  ((Minimum condition) AND !(ExistingEqualPI())) OR
               (Maximum condition)
 Do Action     Enable batch cluster bc
End ActivationRule
```

Fig. 5 shows exemplary batch region configurations for two abstract processes. The batch region in Fig. 5a consists of three activities *B*, *C*, and *D* with *B* being the single entry point. As *groupingCharacteristic*, the process designer chose the data view definition *OrderBySimilarCustomer* introduced in Section 4. Thus, the process instances of process *P1* are grouped based on the customer ID and the shipping address of their processed *Order* object. Furthermore, the *MinMax rule* is selected with two cases (i.e., instances) or 15 minutes for the minimum condition and three cases or 30 minutes for the maximum condition. The capacity (*maxBatchSize*) of a batch cluster is set to three. In case, the maxBatchSize is higher than the maximal threshold, further instances can be added to the batch cluster although it was already activated as long as the processing is not started by the task performer. We will present details on this during execution semantics discussion in Section 5.2. Finally, all instances are processed in parallel. The process *P2* presented in Fig. 5b illustrates the compatibility to the batch activity concept [11] with the *groupingCharacteristic* being unspecified resulting in batch cluster assignment based on the arrival time only. The other parameters are filled similar to process *P1*.



(a) P1. (b) P2.

Fig. 5. Exemplary configurations of batch regions in two abstract processes.

5.2 Execution Semantics

We utilize the online retailer process from Fig. 1 and the batch region configuration as described for Fig. 5a as example to discuss the execution semantics of our approach. Fig. 6 visualizes the setup as condensed version with activity *Pack order* being the entry point to the batch region. If a process instance, e.g., *PI5*, reaches this entry point, its execution is interrupted by transferring the activity of the entry point into the *disable* state; a disabled activity instance is temporarily deactivated [21]. Then, the batch region configuration is evaluated. If a *groupingCharacteristic* is specified, the data view of this process instance is created and it is added to the corresponding batch cluster. In the given example, the grouping characteristic *OrderingBySameCustomer* is specified leading to the data view `CID12, Anystreet 1` for process instance *PI5*. It is assigned to the batch cluster *BC1* sharing the same data view as key. If a process instance does not match to any, as the second arrived instance *PI3*, a new batch cluster is created and

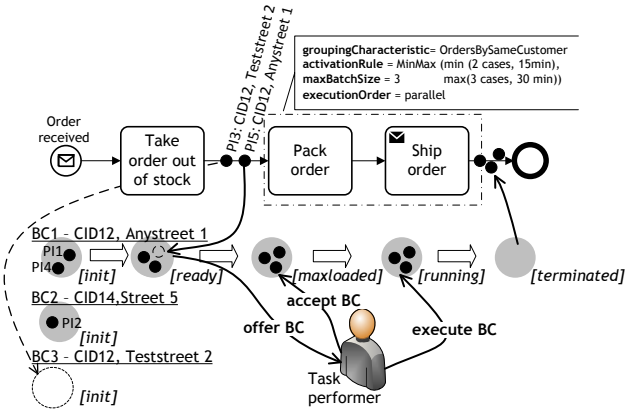


Fig. 6. Execution semantics using a condensed version of the online retailer process from Fig. 1.

initialized with the data view of the respective instance as key, here *BC3* with `CID12, Teststreet 2` as key.

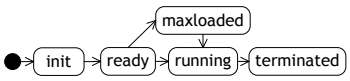


Fig. 7. Life cycle of batch cluster.

Fig. 7 summarizes the states and the transitions between them for a batch cluster. Upon arrival of a process instance requiring a new batch cluster, it gets initialized by transitioning to state *init*, where arriving instances sharing the same data view can now be assigned to the cluster. On instance addition or after certain time durations, the activation rule is checked. In Fig. 11's example, we use the *MinMax Rule* introduced above. With process instances *PI1*, *PI2*, and *PI4* having already arrived at the batch region, the batch clusters *BC1* and *BC2* are in state *init*. Checking the activation rule for cluster *BC1* reveals that the minimum rule is satisfied but evaluating function `ExistingEqualPI()` shows that there is another instance running with the same data view – *PI5* – and that the maximum condition with three arrived instances is not yet satisfied. Thus, the activation rule is not yet fulfilled.

The arrival of *PI5* adds this instance to cluster *BC1*. The batch cluster changes into state *ready*, because the maximum condition is now satisfied. In this state, the batch cluster is offered to the *task performer* of the entry activity into the batch region (cf. *BC1* in Fig. 6). The task performer can be either a software service, a human, or a non-human resource. We propose that a batch is assigned to the same employee for all user activities within the batch region (i.e., case handling resource pattern [13]) to ensure that the batch is performed uninterruptedly. However, other resource allocation patterns can also be applied. Newly arriving process instances can still be added to a batch cluster in state *ready* independently from resource allocation. As the arrival of *PI5* also satisfies the *maxBatchSize* of three, the state is transitioned to *maxloaded* and no further instance addition is allowed. Once allocated to a resource, the performer may decide to start execution. Then, the batch cluster transforms into state *running*. The batch cluster's state changes to *terminated* as soon as all control flow node instances of the process instances assigned to the batch cluster have terminated. The process instances then continue their execution individually for all control flow nodes beyond the batch region.

Next, we present the execution details starting from the acceptance of a batch cluster by a task performer. With the configuration parameter *executionOrder*, the

process designer selects the type of batch execution. She may choose *parallel* as in the above example, *sequential per activity*, or *sequential per case*. Their different execution behaviors are illustrated in Fig. 8, Fig. 9, and, Fig. 10 using an example where two process instances are synchronized within one batch cluster.

In *parallel* execution, shown in Fig. 8, the batch cluster changes into state *running*, when the assigned task performer decides to start the execution. Then, the *disabled* instances *PI1.AI1* and *PI2.AI1* of the entry activity into the batch region are enabled, one of each assigned process instance *PI1* and *PI2*. With enablement, an activity instance usually directly offers the work item to its task performer. Here, the batch cluster acts as interface between the activity instance and the task performer to organize the batch execution. Thus, each activity instance provides the work item to the batch cluster. The batch cluster aggregates them into one *batch task* and provides it to the task performer for parallel execution. As soon as the task performer starts the batch task, all activity instances are started by the batch cluster. With termination of the batch task, the combined result R is sent to the batch cluster which then provides each activity instance with the individual outputs R_i of the task execution. The activity instance is terminated which results in activation of its outgoing sequence flow leading to the enablement of the subsequent activity instances – *PI1.AI2* and *PI2.AI2* in Fig. 8. These instances again provide their work items to the batch cluster and the above described steps are repeated until all control flow node instances of the batch region are terminated. Then, also the batch cluster terminates. The activity instances which gets enabled beyond the batch region are again executed individually. They follow the usual activity semantics and provide their work item directly to the task performer specified in the activity description. Each activity instance knows whether its corresponding activity is part of a batch region and thus whether to provide the work item to a batch cluster or directly to a task performer.

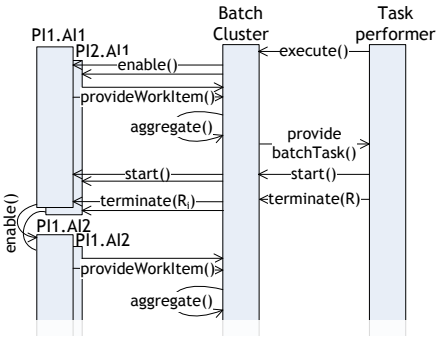


Fig. 8. *Parallel* execution of two instances.

Similar as in the *parallel* execution, in the *sequential per activity* execution, shown in Fig. 9, the batch cluster enables all *disabled* instances of the first activity as soon as the task executor decides to execute the cluster. Again, all work items of the activity instances are provided to the batch cluster. This time, they are arranged in a list specifying the order in which the batch cluster provides the work items one after another to the task performer. In Fig. 9, first, the work item of activity instance *PI1.AI1* is provided. With its termination, the work item of the first

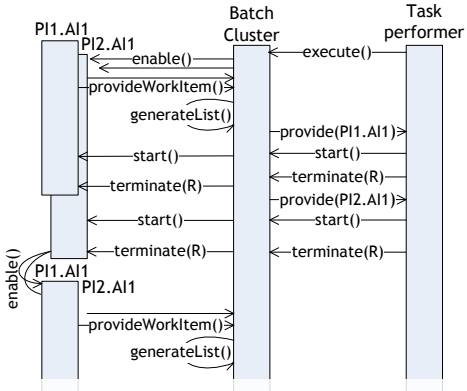


Fig. 9. *Sequential per activity* execution.

activity instance of the next process instance is provided – *PI2.AI1*. When all instances of the first activity are terminated and the work items of the subsequent activity are provided to the batch cluster, a new list is generated specifying the order in which the instances of the second activity of all process instances are processed one after another. This continues for all activities in the batch region.

In the *sequential per case* execution, shown in Fig. 10, all nodes in the batch region are executed for the first process instance assigned to the batch cluster before the nodes of the second process instance can be started. Thus, only the *disabled* activity instance *PI1.AI1* of the first process instance *PI1* is enabled when the task performer decides to execute the batch cluster. Then, the batch cluster provides the work item of *PI1.AI1* to the task performer. If it is finished, the work item of the subsequently enabled activity instance *PI1.AI2* of the same process instance is provided. When all nodes of the first process instance *PI1* are terminated, the *disabled* activity instance *PI2.AI1* of the second process instance *PI2* is enabled by the batch cluster and all activity instances of this process instance are processed as described above for the first process instance. The batch cluster terminates, if all assigned process instances are processed.

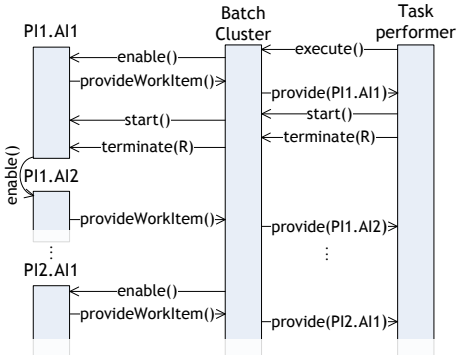


Fig. 10. *Sequential per case* execution.

6 Case Study

In this section, we apply the presented approach on the online retailer process presented in Section 2 and show its impact in terms of increasing waiting time and decreasing costs. Therefore, we compare the execution without a batch region and the execution with a batch region having two different configurations. As indicated in Fig. 1, the batch region comprises the activities *Pack order* and *Ship order*. For this case study, we assume that the online retailer provides a 24h service receiving ten orders per hour resulting in 240 orders per day. From those, 2% are canceled due to article unavailability. In practice, customers may send further orders shortly after their first one. These orders can be used to synchronize their execution with the earlier placed orders to pack them into one package within the batch region in order to save shipment costs, i.e., parallel batch execution. Easing this case study, we assume that customers only place one second order. However, our approach can handle multiple orders of one customer increasing the potential for cost savings. In this case study, 10% of the customers send their second order equally distributed within one hour after the first one (24 orders per day). Afterwards, the probability to observe a second order of a customer reduces to 2.5% per hour until six hours after the first order (in sum, 30 orders per day). After 6 hours, observation for second orders is stopped; they are treated as first ones. The total package and shipment costs of our online retailer are 3.00 € per package. The first two process activities take together one hour. This results in an hour waiting time between order placement and

order packaging. The two activation rules the online retailer may use for batch region configuration are the following ones:

MinMax Rule (min (1 instance), max (2 instances, 1 hour)): This rule activates the batch cluster after arrival of one process instance in case that no other related instance is observed. Otherwise, it waits for one hour. Therefore, only process instances having the same data view (customer ID and address are identical) can be synchronized where the second order was placed at most one hour after the first one. As a process instance takes one hour until it arrives at the batch region, the 10% second orders arriving within the first hour can be used for synchronization.

MinMax Rule (min (2 instance, 5 hours), max (2 instances, 6 hours)): This rule activates the batch cluster after arrival of two process instances with the same data view or five hours after arrival of the first instance. The waiting time for instances increases to six hours if a second instance with the same data view is observed. Thus, all orders arriving within six hours after their corresponding first one can be synchronized.

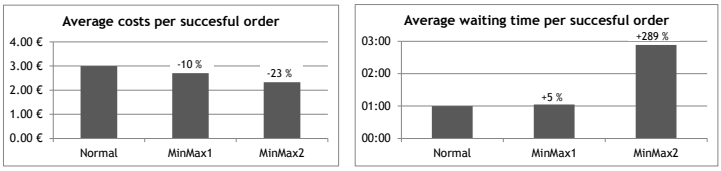


Fig. 11. Average costs and waiting time per successful order for three settings: (i) without batch region, (ii) batch region with 1st activation rule, and (iii) batch region with 2nd activation rule.

The remaining configuration parameters are identical to the ones presented in Fig. 6. Fig. 11 visualizes the impact of our approach. It shows the online retailer's average shipping costs (left) and the average waiting time a customer experiences (right) per successful, i.e., not canceled, order. In both diagrams, the most left bar represents the usual setting without a batch region. The middle and right bars illustrate the results for the batch region application with the first and second *MinMax* rule respectively. The percentages shown above the bars indicates the change towards the usual setting. In the usual setting, each order is sent in a separate package costing 3.00 € each and having a waiting time of one hour until the articles get packed and shipped. Using the first activation rule allows to synchronize on average 23.5 process instances per day (24 instances - 2% instances being canceled) with another instance reducing the average shipping cost per order to 2.70 € – 10% cost savings. As consequence of synchronization, these 23.5 orders have to wait at average 30 minutes additionally until packaging takes place. This results in an increase of the average waiting time by 5% to 1 hour and 3 minutes. The second activation rule allows to synchronize on average 52.9 instances per day ((24 + 30 instances) - 2% instances being canceled) resulting in the same amount of packages less to send. Thus, the costs per order decrease by 23% to 2.30 €. The average waiting time increases by 289% to almost three hours, because also those process instances which cannot be synchronized with another one have to wait for five additional hours. However, this waiting time increase is no issue in practice as the logistics service provider comes only once a day picking up the packages meaning that the customer gets

her order without delay except the waiting time causes a miss of the pick up resulting in one day delivery delay.

In the case study, we discussed the application of the batch region to an online retailer process. In practice, there exist multiple scenarios in which such synchronization improves process execution. For instance, for factorization, goods of the same type or customer shall be handled together – and goods from the competitor shall not be put into the same shipment. Insurance and banking companies can bundle work with respect to their customers by handling multiple claims together to use possible interrelations for decision taking or to send one letter instead of many to inform a customer about happenings with respect to her account.

7 Related Work

Few works provide means for batch processing of multiple process instances. [1] and [14] describe the design for batch execution but limit configuration options to the predefined maximum capacity based on which a batch is started. Rule-based activation is important to balance costs and waiting time for batch execution but is not supported by these approaches. [11] overcomes these limitations by proposing an approach for a flexible and configurable batch activity design which supports rule-based batch activation. Additionally, [11] provides execution semantics for batch activities. In the work, instances are assumed to be completely independent such that no differentiation is enabled. This leads to an instance grouping based on the order of their arrival at the batch activity. In [7], grouping characteristics for batch processing are introduced. A central buffer at the batch processing activity receives and stores multiple process instances as they arrive. If the central buffer exceeds a specified threshold value, waiting instances are grouped based on data equality [8] and then one group is selected for execution. Optimizing batch execution is not possible following this approach, because batch activation only results from the central buffer threshold excess and the group selection process is not described. [9] introduces another mechanism using rule-based synchronization to group process instances. An individual synchronization service is defined for each process instance type handling synchronization and triggering execution. Often missing a-priori knowledge about the types results in an inflexible batch handling. In PHILharmonicFlows [5], a data-oriented modeling approach, batch activities are used to process multiple data objects in one go. The *batch region* approach described in this paper combines and evolves on the aforementioned concepts providing flexible batch configuration and execution for a set of connected control flow nodes. It allows to group instances based on their data characteristics and a rule-based activation considering related running instances.

In literature, different approaches deal with identification of related process instances for various application scenarios. [15] proposes a similarity measure which identifies activities requiring similar work as the worker's current one to reduce context switches. Similarity identification uses instance data and context information and therefore requires access for the BPMS to that context data as well as efficient analysis techniques and computational resources. In the field of adaptive processes, [19] introduces an approach to identify former process instances with deviations from the pre-defined model for situations similar to the current one. Identification bases on instance data manually

provided by the executor including specification which data is triggering the deviation. In process mining, clustering and classification techniques are used to improve the mining results [18], for decision mining [12], and for prediction of instance behavior [4]. In contrast to our work, the data attributes determining instance similarity are not known a-priori but get extracted from run-time knowledge. Thus, those techniques can also be used to support the definition of our grouping characteristic for the data view function.

8 Conclusion

In this paper, we generalize the concept of batch activities to *batch regions* which allow to group process instances based on their data characteristics and to synchronize the execution of such a group over a connected process fragment. Multiple process instances may depend on each other as they share some common data, e.g., equal customer identifier and shipping address of processed orders. We allow the grouping of process instances utilizing the concept of *data views* adapted from database domain. A data view is a projection on the process instance data based on relevant attributes specified by the process designer. Further, this paper introduces a new type of activation rule which optimizes batch region enablement by considering running instances which have not yet reached the batch region. Summarized, we presented means to model and configure batch regions and we described their execution semantics. The batch regions concept fulfills the requirements discussed in [11] for integrating batch processing in business processes and the requirements from Section 2 to handle process instance heterogeneity. Currently, the determination of relevant attributes for the data view (RQ2) is done manually by the process designer. Further run-time and context information can be used to extend the reasoning about batch processing activation (RQ4). In future, we will cope with these restrictions and support the process designer by automatically providing relevant attributes for the data view. We also plan to optimize batch region execution by activating a batch region only if necessary.

References

1. van der Aalst, W.M.P., Barthelmess, P., Ellis, C.A., Wainer, J.: Proclets: A framework for lightweight interacting workflow processes. IJCIS 10(4), 443–481 (2001)
2. Activiti: Activiti BPM Platform. `https://www.activiti.org/`
3. Bonitasoft: Bonita Process Engine. `https://www.bonitasoft.com/`
4. Ghattas, J., Soffer, P., Peleg, M.: A formal model for process context learning. In: Business Process Management Workshops. pp. 140–157. Springer (2010)
5. Künzle, V., Reichert, M.: PHILharmonicFlows: Towards a Framework for Object-aware Process Management. J SOFTW MAINT EVOL-R 23(4), 205–244 (2011)
6. Lanz, A., Reichert, M., Dadam, P.: Robust and flexible error handling in the aristaflow bpm suite. In: CAiSE Forum 2010. LNBIP, vol. 72, pp. 174–189. Springer (2011)
7. Liu, J., Hu, J.: Dynamic batch processing in workflows: Model and implementation. Future Generation Computer Systems 23(3), 338–347 (2007)
8. Liu, J., Wen, Y., Li, T., Zhang, X.: A data-operation model based on partial vector space for batch processing in workflow. Concurrency and Computation 23(16), 1936–1950 (2011)

9. Mangler, J., Rinderle-Ma, S.: Rule-based synchronization of process activities. In: CEC. pp. 121–128. IEEE (2011)
10. Meyer, A., Pufahl, L., Fahland, D., Weske, M.: Modeling and Enacting Complex Data Dependencies in Business Processes. In: BPM. pp. 171–186. Springer (2013)
11. Pufahl, L., Weske, M.: Batch Activities in Process Modeling and Execution. In: ICSOC. pp. 283–297. Springer (2013)
12. Rozinat, A., Mans, R., Song, M., van der Aalst, W.M.: Discovering simulation models. Information Systems 34(3), 305–327 (2009)
13. Russell, N., van der Aalst, ter Hofstede, A., Edmond, D.: Workflow resource patterns: Identification, representation and tool support. In: CAiSE. pp. 216–232. Springer (2005)
14. Sadiq, S., Orlowska, M., Sadiq, W., Schulz, K.: When workflows will not deliver: The case of contradicting work practice. In: BIS. vol. 1, pp. 69–84. Witold Abramowicz (2005)
15. Shkundina, R., Schwarz, S.: A similarity measure for task contexts. In: ICCBR Workshops. pp. 261–270. Citeseer (2005)
16. Silberschatz, A., Korth, H.F., Sudarshan, S.: Database System Concepts, 4th Edition. McGraw-Hill Book Company (2001)
17. Simons Jr., J.V., Burke, G., Russell, G.R.: A cost-based model for customer batching in mass service operations. Journal of Service Science Research 3(2), 123–151 (2011)
18. Song, M., Günther, C.W., van der Aalst, W.M.P.: Trace clustering in process mining. In: Business Process Management Workshops. pp. 109–120. Springer (2009)
19. Weber, B., Reichert, M., Rinderle-Ma, S., Wild, W.: Providing integrated life cycle support in process-aware information systems. Int. J. Cooperative Inf. Syst. 18(01), 115–165 (2009)
20. Weske, M.: Business Process Management: Concepts, Languages, Architectures. Second Edition. Springer (2012)
21. Weske, M., Hündling, J., Kuropka, D., Schuschel, H.: Objektorientierter Entwurf eines flexiblen Workflow-Management-Systems. Inform., Forsch. Entwickl. 13(4), 179–195 (1998)

Aktuelle Technische Berichte des Hasso-Plattner-Instituts

Band	ISBN	Titel	Autoren / Redaktion
85	978-3-86956-276-6	**HPI Future SOC Lab: Proceedings 2012**	Christoph Meinel, Andreas Polze, Gerhard Oswald, Rolf Strotmann, Ulrich Seibold, Bernhard Schulzki (Hrsg.)
84	978-3-86956-274-2	**Anbieter von Cloud Speicherdiensten im Überblick**	Christoph Meinel, Maxim Schnjakin, Tobias Metzke, Markus Freitag
83	978-3-86956-273-5	**Proceedings of the 7th Ph.D. Retreat of the HPI Research School on Service-oriented Systems Engineering**	Christoph Meinel, Hasso Plattner, Jürgen Döllner, Mathias Weske, Andreas Polze, Robert Hirschfeld, Felix Naumann, Holger Giese, Patrick Baudisch (Hrsg.)
82	978-3-86956-266-7	**Extending a Java Virtual Machine to Dynamic Object-oriented Languages**	Tobias Pape, Arian Treffer, Robert Hirschfeld
81	978-3-86956-265-0	**Babelsberg: Specifying and Solving Constraints on Object Behavior**	Tim Felgentreff, Alan Borning, Robert Hirschfeld
80	978-3-86956-264-3	**openHPI: The MOOC Offer at Hasso Plattner Institute**	Christoph Meinel, Christian Willems
79	978-3-86956-259-9	**openHPI: Das MOOC-Angebot des Hasso-Plattner-Instituts**	Christoph Meinel, Christian Willems
78	978-3-86956-258-2	**Repairing Event Logs Using Stochastic Process Models**	Andreas Rogge-Solti, Ronny S. Mans, Wil M. P. van der Aalst, Mathias Weske
77	978-3-86956-257-5	**Business Process Architectures with Multiplicities: Transformation and Correctness**	Rami-Habib Eid-Sabbagh, Marcin Hewelt, Mathias Weske
76	978-3-86956-256-8	**Proceedings of the 6th Ph.D. Retreat of the HPI Research School on Service-oriented Systems Engineering**	Hrsg. von den Professoren des HPI
75	978-3-86956-246-9	**Modeling and Verifying Dynamic Evolving Service-Oriented Architectures**	Holger Giese, Basil Becker
74	978-3-86956-245-2	**Modeling and Enacting Complex Data Dependencies in Business Processes**	Andreas Meyer, Luise Pufahl, Dirk Fahland, Mathias Weske
73	978-3-86956-241-4	**Enriching Raw Events to Enable Process Intelligence**	Nico Herzberg, Mathias Weske
72	978-3-86956-232-2	**Explorative Authoring of ActiveWeb Content in a Mobile Environment**	Conrad Calmez, Hubert Hesse, Benjamin Siegmund, Sebastian Stamm, Astrid Thomschke, Robert Hirschfeld, Dan Ingalls, Jens Lincke
71	978-3-86956-231-5	**Vereinfachung der Entwicklung von Geschäftsanwendungen durch Konsolidierung von Programmier-konzepten und -technologien**	Lenoi Berov, Johannes Henning, Toni Mattis, Patrick Rein, Robin Schreiber, Eric Seckler, Bastian Steinert, Robert Hirschfeld
70	978-3-86956-230-8	**HPI Future SOC Lab - Proceedings 2011**	Christoph Meinel, Andreas Polze, Gerhard Oswald, Rolf Strotmann, Ulrich Seibold, Doc D'Errico

ISBN 978-3-86956-280-3
ISSN 1613-5652