Enhancing Business Process Flexibility
by Flexible Batch Processing*

Luise Pufahl\(^1\) and Dimka Karastoyanova\(^2\)

\(^1\) Hasso Plattner Institut, University of Potsdam, Potsdam, Germany
luise.pufahl@hpi.de
\(^2\) University of Groningen, Groningen, The Netherlands
d.karastoyanova@rug.nl

Abstract. Business Process Management is a powerful approach for the automation of collaborative business processes. Recently concepts have been introduced to allow batch processing in business processes addressing the needs of different industries. The existing batch activity concepts are limited in their flexibility. In this paper we contribute different strategies for modeling and executing processes including batch work to improve the flexibility i) of business processes in general and ii) of the batch activity concept. The strategies support different flexibility aspects (i.e., variability, looseness, adaptation, and evolution) of batch activities. The strategies provide a systematic approach to categorize existing and future batch-enabled BPM systems. Furthermore, the paper provides a system architecture independent from existing BPM systems, which allows for the support of all the strategies. The architecture can be used with different process languages and existing execution environments in a non-intrusive manner.

Keywords: batch activities · business processes · flexibility · flexibility strategies · separation of concerns · modular architecture

1 Introduction

Business process management (BPM) and the automation of business processes allows organizations to execute their own as well as collaborative business processes more effectively and efficiently. The main artifact of BPM are process models which help to document, analyze, improve, and automate/executive business processes [6]. A process model represents a blueprint for a set of process instances whereby each process instance represents the execution of a single business case [35]. Each process instance has "an independent existence and typically executes without reference to each other" [26]. However, a common phenomenon in operational business processes is batch processing [29]. Batch processing implies that several cases are collected at specific activities to process them as a

* The research leading to these results has been partly funded by the BMWi under grant agreement 01MD18012C, Project SMile.
group to reduce costs or processing time [12]. For example, in the administration, several invoices are collected first before they are approved and checked [1], or in logistics, several individually ordered goods are processed and packaged as one batch by a package delivery service.

Recent research efforts have introduced batch activities [12,14,19] to model and automate batch work in business processes. A batch activity as presented by Pufahl et al. [19] provides several parameters, such as an activation rule, a maximum batch size etc. to allow for configuration of a batch activity individually, which then can be executed automatically by a BPM system. A prototypical implementation of the concept for batch activities in such an existing system is described in [17]. Existing implementation concepts of batch activities are static and limited in their flexibility. Once a process model with a batch activity and its batch configuration is deployed on a BPM system, it can only be adapted by changing the process model and re-deploying it. However, in practice, more flexible approaches are needed. For instance, in the SMile project\(^3\), innovating the last mile logistics of parcels, batch processing is used for the consolidation of parcels during the last mile delivery. Parcels are provided in batches to different carriers whereas different variations of batching are required depending on the individual carrier (e.g., batch size). Also the adaptation of batches is important in case of exceptions, such as the sudden unavailability of a carrier.

Flexibility is besides time, cost, and quality, one important performance dimension [6] and multiple research efforts focus on enabling the flexibility of business processes. These efforts fall into one of the following groups according to the flexibility needs they address: variability, looseness, adaptation, and evolution, which were proposed by Reichert and Weber [23]. In order to address the need for flexibility, in this paper we introduce the so called flexibility strategies for batch activities – the modeling, deployment, and execution strategy. Inspired by the principle of separation of concerns [5], the flexibility strategies for batch activities presented prescribe different levels of modularity for the design of the business processes that comprise batch configuration.

Our contribution regarding flexibility is twofold: on the one hand we improve the flexibility of batch activities in business processes by specifying different strategies for their possible adaptation during modeling and execution. On the other hand, by introducing the flexible batch activities we contribute another mechanism for process adaptation and therefore enhance process flexibility in general.

The presented flexibility strategies for batch activities contribute a systematic approach for evaluating and categorizing batch capabilities of existing BPM systems in terms of both architecture and implementation. For each of these strategies, we identify the degree of flexibility supported, the requirements they impose on a BPM system for enacting individual strategies, the corresponding advantages, and challenges. Furthermore, each strategy is compared to existing research works. Finally, we contribute a generic architecture of a supporting sys-

\(^3\) [http://smile-project.de]
tem, which can guide the realization of all three or only some of the strategies for a concrete implementation depending on the individual requirements.

In the remainder of the paper, first related work in Sect. 2 regarding batch processing in business processes and existing support for process flexibility in service-oriented environments is discussed. After the batch activity concept is revisited in Sect. 3, the three flexibility strategies are introduced and discussed in detail in Sect. 4. Based on the BPM system requirements described in each flexibility strategy, a generic BPM system architecture for flexible batch processing is presented and discussed in Sect. 5. Finally, the paper concludes in Sect. 6 and highlights directions for future work.

2 Related Work

The related work overview will focus on several aspects. We will review related works in batch processing, works focusing on addressing flexibility needs, and we will present some of the approaches investigating how flexibility, modularity and reusability can be enhanced by using different strategies.

Batch Processing in Business Processes. The need for batch processing in business processes was discussed by several works, such as by Reijers and Mansar [24] as redesign heuristic, by Aalst et al. [2] as an escalation strategy to avoid deadline violations, by Fdhila et al. [7] as an instance-spanning constraint, or in process mining by Martin et al. [13]. A first proposal to integrate activities with a batching behavior is the Compound activity by Sadiq et al. [27], which is mainly manually driven by the task performers. Concepts to model batch activities in business processes and execute them automatically can be found in the works of Liu et al. [12] and Natschläger et al. [14] having certain limitations, such as the focus on single resources, the merge of instances during the batch execution, and no proof-of-concept implementation. The batch activity concept by Pufahl [17] is designed based on a requirement analysis, and also provides a prototypical implementation in Camunda\(^4\), an open-source, java-based BPM system. This concept will provide the basis for the work of this paper.

Although flexibility of batch activities is mentioned in the requirements analysis on integrating batch processing in business processes [20], none of previous mentioned batch activity concepts provide sufficient means to support different flexibility needs. Pflug and Rinderle-Ma [16] provide a sequential batch processing approach for activities with long waiting queues with a dynamic classification of instances at runtime. Still, the critical activity and the clustering algorithm has to be selected at design time. The work by Pufahl et al. [18] provides a concept for flexible batch activity configuration. This approach allows adapting the configuration of a batch activity during runtime if certain events occur. This work focuses only on the adaptation of batch configurations; variability, looseness, and evolution aspects are not targeted.

\(^4\) \url{www.camunda.org}
Flexibility of Business Processes. Multiple research efforts have focused on enhancing the flexibility of business processes. Reicht and Weber [23] distinguish four major flexibility needs, namely a) variability, i.e., the need to have different variants of a process model to handle different customer groups, product groups etc., b) looseness, which is the need to leave some aspects unspecified during modeling because cases are unpredictable etc., c) adaptation as the need to react to exceptions or special cases by adapting the process, and d) evolution being the need to change process models over time due to changed business processes and which also involves the migration of instances. Regarding the flexibility need variability introduced by [23] the survey presented by [25] gives an overview of available approaches and recommendations as about which approach can be applied for specific context. The looseness need has been addressed in works like [4] using the so called node activities, [28] using the concept of pockets of flexibility, [9] using parameterized service compositions, and [10] where Semantic Web Services are dynamically discovered during process deployment or execution and others. The adaptation need has been the subject of abundant literature; some examples are for choreographies [23], [22], [33]. As for the evolution need, interesting results have been published in works such as [3], [32], [30] and others.

In related work dedicated to adaptation of processes in service-oriented environments, several approaches have been introduced that specify different strategies for modularization of process models and as a result enhancements in flexibility in the different process life cycle phases. For example, in [31] four possible service binding strategies (i.e., the way of selecting services in workflows have been identified and mapped to the process life cycle), and their impact on the modular structure of the workflow-based applications, the service middleware and its architecture have been discussed. The strategies allow for static service binding and three flavors of dynamic binding: 1) traditional binding where the service middleware/bus selects the endpoint, 2) dynamic binding with service deployment and 3) with software stack provisioning. The concept of Semantic Service Bus (SSB) [10] was introduced with the purpose of enabling service selection at deployment or runtime of processes based on semantic information. Both approaches enable flexibility on the functional dimension of processes and the improved modularity of the process model facilitates the enhancement of the flexibility of the processes. The adaptation of the control flow of processes has been in the focus of the BPEL’n’Aspects approach by Karastoyanova and Leymann [8]. This work introduces a non-intrusive approach for adapting the control flow of processes during their execution and on per-instance basis. The adaptation operations supported are: insert, delete, and skip activity or activities. Inspired by the separation of concerns principle, the adaptation steps are enabled by modular process models that specify adaptation steps separately from the control flow logic. This allows for dynamically switching on and off the adaptations that need to be performed on process instances, even after they have been started and as long as the change happens (in some cases) at the wavefront of the process instance or after it, i.e. elements in the process that have not been
executed yet. Note that the approach can be extended with these adaptation operations so that they can be performed on part of the control logic that has already been executed, i.e. before the wavefront, by using the so called rewinding of processes [33]. The advantages of BPEL’n’Aspects over other approaches sum up to 1) runtime adaptation of processes and specification of the adaptation is enabled on a per-instance basis 2) in a generic manner since adaptation operations are mapped to WS operation calls rather than any other language-specific implementation; 3) a general purpose mechanism to specify when and how a process and an adaptation operation are connected; 4) the approach and the infrastructure can be used with legacy process models and process execution environments. The approach we introduce in this work is inspired by these works on process adaptation and are applied and transfered to the problem of insufficient flexibility of batch processing.

3 Background: The Batch Activity Concept

In this section, we revisit briefly the batch activity concept presented in [17,19] as it is the basis for the rest of the paper. Batch activities can be used to collect several process instances and to process them as a group in order to reduce costs or processing time.

For example, Fig. 1a shows a health care process with two batch activities as a BPMN process diagram [15]. In this process, a blood sample is taken from a patient, if a blood test is needed. Then, the sample is brought to the laboratory where the blood sample is prepared for testing. The actual test is conducted by a blood analysis machine. After the test, the results are published in the central hospital information system where they are accessible by the physicians for evaluation in the respective ward. There are two batch activities in this process. The activity Transport sample and order to lab models the fact that a ward nurse does not bring each blood sample separately to the lab but instead she delivers several samples together to save transportation time. The second batch activity is the sub-process which consists of two activities: it collects multiple blood samples before a test on a blood analysis machine is started to save some machine costs.

As the health care process presented above, a process model consists in general of nodes that can be activities, events, or gateways and edges [35]. Each process activity can be a task (i.e., non-decomposable activity) or sub-process (i.e., an activity with internal behavior). Different types of tasks exist, such as user task where a task performer gets a work item via worklist manager provided, or a service task where a service is called. For integrating batch activities, conceptually activities are extended by a so-called batch model [17] as shown in Fig. 1b comprising different batch configuration parameters. A process designer can specify which instances are grouped in a batch (groupedBy), when a batch is started (activationRule), what is the maximum allowed size of a batch in number of instances (maxBatchSize), and how the batch is executed: in parallel or sequential (executionOrder). In the case of the batch activity Transport sample

Batch Activity

Activity

- type a: Enumeration
- type t: Enumeration

Activity Instance

1 0..*

Batch Model

- groupedBy: List<DataAttribute>
- activationRule: ActivationRule
- maxBatchSize: Integer
- executionOrder: Enumeration

Batch Cluster

1 0..*

(a) Blood testing process with two batch activities.

(b) Batch activity concept.

Fig. 1. Batch activity concept and an example process with batch activities from the health care domain.

and order to lab, the blood samples are grouped into batches based on their hospital ward where they were requested (i.e., Order.ward). A batch is initialized if at least 20 blood samples are collected or at most after 1 hour waiting time. At maximum 150 blood samples can be processed in one batch and the batch is executed in parallel which means that all blood samples of a batch are transported at the same time to the laboratory. In contrast, in sequential batch execution the activity instances are still executed one after another but it is used to save setup costs or time.

If an activity has a batch model assigned, its enabled activity instances are disabled at runtime, i.e., their execution is paused in order to collect/assign them in/to batches clusters based on the groupedBy-parameter. Batch clusters (see Fig. 1b) are responsible for the batch execution. Each batch cluster has at least one activity instance assigned or at most the maxBatchSize. A batch cluster is enabled as soon as the activation rule is fulfilled, after which it is provided to the responsible task performer or service. As long as a batch cluster is not started, activity instances can be added to it. After the termination of a batch
cluster, the process instances continue with their individual process execution, thus regaining their autonomy.

4 Flexibility Strategies for Batch Activities

In order to enhance the flexibility of batch processing and also be able to use batch processing as an adaptation mechanisms we identify three possible strategies for process modeling and execution: a) Modeling strategy, b) Deployment strategy and c) Execution strategy. This section is used to describe each of the strategies, to identify the requirements they impose on the BPM systems in terms of architectural components and the necessary interactions between them, as well as to highlight their advantages and disadvantages. For each strategy we consider only the relevant process life cycle phases, i.e., design time, deployment, and execution time (a.k.a. runtime phase).

4.1 Modeling strategy

The modeling strategy postulates that a batch activity is specified in a process model at design time (Fig. 2). During deployment this process model is parsed, transformed into the BPM system internal representation and added to the process repository of the system. If the BPM system is capable of executing batch activities as described in [20], then anytime a batch activity is reached in a process instance, it will be executed following the execution semantics and under consideration of the batch model that specifies the batch activity. The requirements, this strategy poses on a BPM system are that a) it has to be able to parse, transform and store process models containing batch activities and their corresponding configuration information into an internal representation, b) the execution engine has to implement the execution semantics of a batch activity and extend the life cycle management functionality to be able to deal with instances that belong to a batch cluster: pause the instances in their execution, assign them to a cluster, and execute the specific batch behavior as soon as the batch cluster is enabled.

The main advantage of the modeling strategy is its simplicity. It only requires the implementation of the parsing and execution of batch activities. However, if there is any change needed in the batch configuration, then the complete process model has to be redesigned and re-deployed. Using this strategy neither improves
modularity of the processes nor their flexibility. Despite these disadvantages, existing works on adding a new modeling element to business processes to model and execute batch activities, such as [12, 14, 19] support only this strategy because of its simplicity. In the case of commercial BPM systems, which do not provide batching support yet, the deployment strategy is considered the most feasible and easy to implement.

4.2 Deployment strategy

The deployment strategy calls for a modular design and requires keeping the process model and the batch model (batch activity configuration) separate. For the process designer, the modeling of batch activities (i.e., at design time) would not change. What changes is the way the process model and the batch model are stored by the business process modeler. If a batch activity is included in a process model, then its specification, the batch model, is stored in an extra deployment artifact as shown in Fig. 3 as well as a connector describing to which process activity a batch model applies.

This means that the batch activity concept is extended by a connector which realizes an m-to-n relationship between a process activity and a batch model as shown in Fig. 4. Since one batch model can be assigned to multiple activities, and one activity can have multiple batch models, the connector specifies the process context to which a batch configuration applies.

During deployment, the process model as well as its batch models and connectors are deployed on the BPM system, all as a bundle. As a result, the process model is stored in the process repository, whereas the batch model and connector are stored as configuration artifacts of the BPM system. If an update on the batch model or process model is necessary, all three artifacts – the process model, the batch model, and the connector – have to be deployed again together as bundle.

The BPM system requirements we identify are: a) the process modeling tool should be able to produce separate files/artifacts for the batch model, the process model, and the connector showing which process and which batch configuration go together, as well as the process bundle and a deployment descriptor; b) the deployment component of the execution environment has to be able to accept the bundle as input and parse, transform and store it accordingly; c) the execution environment has to support the execution behavior of batch processing. Depending on the implementation approach, the actual connection between the process model and the batch model can be realized either already at deployment
time or during the instantiation of processes. The actual redeployment is done by the user. If there is a need to keep track of versions of process and batch models, then the BPM system has to enable this. Similarly, it is an implementation and domain specific concern if the redeployment would lead to process instance migration to the new versions or not, and what would happen with the currently running instances. Existing work on this topic like [30], [32] and [3] present different ways of addressing this issue. It is out of the scope of the strategy definitions to prescribe which approach to use.

Through the concept of separation of concerns (i.e., separating the batch model from the process model), the advantages of the deployment strategy are that it improves the modularity, reusability, and also the flexibility of process artifacts. It addressed the variability need [23], and allows the batching over several process models. In the following, these two aspects are presented in more detail:

- **Batch activity variability**: Several variants of a batch configuration can be defined for a batch activity which is valid for different resources or case types. For instance, a premium customer group can require that its members are only handled optionally as batch, whereas for all other customer groups the batch processing mechanism is always applied [21]. Hence, the connector includes the conditions, a set of data values which the process instances have to fulfill, under which a batch model is connected to a specific process activity. Please note, the other shown parameters of the batch connector are relevant for the execution strategy presented in the following sub-section.

- **Multi-Process Batching**: Batch processing across multiple business processes is enabled by this strategy if one batch model is assigned to several, similar activities in different process models.

As stated, the deployment strategy improves the modularity, maintainability, and reusability which are crucial characteristics from software engineering perspective. However, quick adaptations on the batch model to react on changes in the business environment on *per-instance basis* are not possible with this strategy, since a redeployment is required.

Currently, this strategy is partly supported in the work of [20] in which the batch model is centrally stored in an object life cycle5 independently from a process model. Independent of the batch processing, a deployment strategy is supported by most BPM systems supporting service orchestrations with or

---

5 Object lifecycles complement process models and describe allowed actions of business processes on data artifacts across the process-model boundaries.
without flexibility support, like e.g. [8,10,31,33] that are based on open source versions of ActiveBPEL\(^6\) and Apache ODE\(^7\).

### 4.3 Execution strategy

The execution strategy allows for assigning a batch model to one or more (or all) running instances of a process model dynamically at execution time. Similar to the deployment strategy, at design time, the process modeling tool has to be able to generate three artifacts for a business process involving batch activities: the process model, the batch model, and the connector. However, the execution strategy postulates that at deployment time, these artifacts are not deployed together as a bundle, but rather independently from each other. It allows for deploying a process model to which only at a later point in time of its life cycle, deployment or execution, a batch model can be designed and/or an existing one assigned by using a separate artifact specifying the connector between them.

The connector is used to define the exact process instances that will be affected by batch processing. For this, several parameters in the connector shown in Fig. 4 can be defined. The *conditions* consisting of a list of data values defines for which type of process instances a connector between an activity and a batch model is valid. Furthermore, a *trigger* might be given to define under which circumstances a batch model is activated. Therefore, an event is specified that has to occur to activate the corresponding batch model; for instance, the scheduling of a maintenance on the blood testing machine used in the above presented blood testing process. If no trigger is specified, the batch model is immediately activated upon its deployment until its validity ends. The *validity* might be used to specify a point in time or a duration when the assignment of a batch model to an activity or a set of activities ends.

Deleting the assignment of a batch model to instances can also be done at any time during the execution phase. This will affect all process instances which are not yet assigned to a batch cluster or in a batch cluster which is not yet being executed. The clusters that are already in state *running* will still be executed as described by the batch model. Additionally, an update of the process model is possible independently of the batch model and the connector, and updates of a

---

\(^6\) http://www.activebpel.org/

\(^7\) http://ode.apache.org/
batch model or connector are possible at any time independently of the process model since all these are separate artifacts. When updating the process model, all possible adaptation mechanisms are allowed at any stage for which abundant literature exists. If a change on the batch activity configuration is necessary, then only the batch model has to be adapted and re-deployed. Same as in the case of deleting an assignment of a batch model; this affects all batch clusters which have not started with their execution yet. The change on the batch configuration can be done manually or triggered automatically.

The BPM system requirements are more extensive. Conceptually, the modeling tool is not much different from the one needed for the deployment strategy. The deployment component of the system has to be able to parse, transform and store processes, batch models and connectors. The execution engine has to implement the functionality that will support the dynamic assignment of batch models to process models and/or instances and ensure that the batch processing is activated and performed for exactly these process models and/or instances. Similarly, the functionality enabling the dynamic undeployment of connectors has to be realized. Additionally, the execution environment needs a component dealing with the life cycle management of the batch models and connectors.

In addition to batch activity variability and multi-process batching, the execution strategy supports also the flexibility needs evolution, looseness and adaptation which will be explained in the following:

– **Evolution**: By separating the deployment of a batch model from the process models, the ability to update the process model or a batch configuration due to changes in the business is further facilitated.

– **Looseness and Process Adaptation**: Batch work does not need to be specified before process execution. A batch configuration can be flexibly assigned and deleted during the execution phase of a business process. Thus, looseness where aspects can be unspecified during modeling is supported. Furthermore, the adaptability of processes is improved since we introduce and allow the use of batch processing as a new runtime adaptation mechanism.

– **Adaptation of Batch Activity**: The batch activity concept itself is also made flexible since batch configurations can be adapted during execution time in case of exception or special cases.

The execution strategy is a non-intrusive approach improving flexibility, modularity, re-usability, and maintainability of process models and also their batch activities due to the separation of concerns. It will allow batch processing independently of the functionality of a BPM system by providing it as an extension functionality. More details on this are discussed in the following section on the architecture. The disadvantage of the execution strategy is that it is much more complex to implement and requires higher effort in terms of investment in technologies and skills of the software implementation team.

To the best of our knowledge this strategy is only partly supported by the work of Pufhal et al. [18] in which adaptation of running batch clusters can be applied by defining batch adaptation rules explicitly which are triggered by
external events. In the same work, the BPM system includes explicit functionalities for the batch execution and the adaptation of batch clusters. The approach is less modular and does not support the evolution of batch activities or process models. The execution strategy is realized for control flow adaptation of processes by the BPEL’n’Aspects approach [8]; however it has not been applied for batch processing and does not consider human users as participants in the activity execution but rather only conventional Web Services.

5 Architecture

Based on the requirements on a BPM system identified in the previous section, this section presents the conceptual architecture of a BPM system that supports the introduced flexibility strategies for batch activities. We start with a description of the main components of the architecture in Sect 5.1 and present their interactions enabling the flexibility strategies in Sect. 5.2. Advantages, disadvantages, and the feasibility of the realization of the architecture are discussed in Sect. 5.3.

Fig. 6. Architecture for a BPM system realizing the flexibility strategies for batch activities

5.1 Components and their Functionality

The components of the BPM System architecture supporting the flexibility strategies are presented in Fig. 6. In this figure some of the components are grouped in components on a higher abstraction level (surrounded by red dashed lines) in order to bring structure to the discussion and to denote their higher-level functionality. The system supports the complete life cycle of business processes, namely the modeling, deployment, execution, monitoring and analysis phases; components used for monitoring and analysis are neither discussed nor depicted. Next, the components are presented:
The Process Modeling Tool is used to create process models, batch models, and connectors. It has to be capable of producing the deployment artifacts necessary to support the different strategies (see Sect. 4 for specific details), like deployment descriptor, which typically contains configuration information necessary during deployment, and deployment bundles, or process models separately from the batch models and connectors.

The BPM Enactment System is the execution environment that accepts as input the artifacts produced in the modeling phase. The Process Deployment Component parses and stores the models into the system database and prepares the models for instantiation, typically by generating additional artifacts and deploying endpoints at which the model can be instantiated and used. How these steps are performed by the deployment component is strategy- and implementation-specific. The Process Execution Engine executed the process behavior and tracks the execution status of all instances of all process model elements. It also delegates the invocation of activity implementations to a corresponding middleware, called the Service Bus; we do not show this connection here for brevity.

The Service Bus is the middleware responsible for the interaction with services, including discovery and invocation, and the interactions with the Worklist Manager, which allows for human users to participate in the processes and perform certain tasks.

The Event Publishing System contains an Event Publishing Component directly connected to the BPM Enactment System that observes the execution of each process instance and sends a notification about each state change to a message-oriented middleware (MOM) supporting point-to-point interaction via queues. The MOM allows in turn for filtering and reacting to only relevant state changes. The Controller component participating in the event publishing system is a generic component that defines an interface allowing to implement plug-ins for different process extension functionalities that should not be hard-coded in the process language/engine itself [11]. It helps to improve system modularity. An example of such an extension is the runtime adaptation functionality presented in the BPEL’n’Aspects approach [8] that supports inserting new activities in the control flow of a process or deleting activities while process instances are running. For enabling batch processing, a new controller is introduced, called Batch Activity Controller which keeps the batching functionality separate from the Process Execution Engine. For enabling the interaction between the BPM Enactment System and the batch activity implementation, the Batch Activity Controller is part of the Broker component.

The Broker is a complex component responsible for: 1) enabling the publishing of events of interest from and to the BPM Enactment System so that a reaction upon such events can be defined, 2) notifying the batch implementations (either services or a Worklist Manager) to execute a batch and return the results via the Notification Component to the Service Bus, and 3) implementing a reaction
to events through the Batch Activity Controller. The Batch Activity Controller implements the behavior as specified in [19], which reacts to the ready state of activity instances of batch activities. This means that the Batch Activity Controller will subscribe itself only to those state changes that are pertinent to activities that have a batch model assigned. To enable this, the Batch Activity Controller stores the batch model and the connectors between the batch model and process model, i.e., it possesses the information about the batch configuration. For the modeling and deployment strategy, the configuration information is provided by the Process Deployment Component of the BPM Enactment System, because the batch configuration is either in-line in the process model, or a part of the process deployment bundle. In the case of the execution strategy, the configuration information is provided by the Batch Activity Management Tool.

The Batch Activity Management Tool is used to interact with the process modeling tool to create and edit the batch activity models and connectors, and with the Batch Activity Controller to delete, deploy, and undeploy the batch configurations also during process execution. The operations that these architectural components support are described in the following: Deploying a batch configuration means that a connector is created that assigns a batch model to process activities; the undeployment operation deletes the connector. Assigning batch models to activities using the connectors results in interpreting the batch model together with the connector, i.e., the batch configuration, and creating or deleting a subscription to process instance events (produced by the execution engine). In addition, the controller checks the parameters of the connector definition, like condition, validity and the trigger, and performs an activation step in the case the trigger is evaluated to true. Since the trigger identifies an external event the controller has to be able to subscribe to such events.

5.2 Component Interactions

Regarding the interaction of the components for realizing the flexibility strategies, in Fig. 6, all interactions are depicted by a bi-directional arrows to denote the bi-directional nature.

After a batch configuration is deployed, the Batch Activity Controller initiates a subscription to events of activity instances whose activity is assigned to a batch model, more precisely the events indicating that an activity instance is ready for execution. If such an event is received, the Batch Activity Controller informs the engine via the Engine Adapter to pause the respective activity instance of the batch activity and assigns it to an existing or new batch cluster according to the batch configuration. As described in the Sect. 3, the batch cluster itself checks regularly whether its activation rule is fulfilled and in case a batch cluster is enabled the batch execution is started. This functionality is implemented by the Batch Activity Controller for all three strategies.

For the batch execution, the Broker deals with the bi-directional communication between the Process Execution Engine and the Service Bus. Therefore,
it interacts with the Service Bus using notifications and delegates to it the execution of the batch clusters through service invocations, or to humans via the interface provided by Worklist Manager. Additionally, it takes care of returning the results of batch executions via the batch activity controller to the concrete process instances.

The communication between the Batch Activity Controller and the Web Services or the Worklist managers is enabled in terms of a pub/sub infrastructure. The actual service calls are done by the so-called wrappers. Wrappers are 1) forwarding batch clusters to the right service invocation or the worklist manager and 2) returning the results of a batch execution back to the Batch Activity Controller. For each of the deployed batch configurations the Batch Activity Controller creates a topic and there is one wrapper that consumes the messages. Upon publication of a message to the topic the wrapper executes the service invocation [11] or delegates the batch cluster to a worklist manager. Upon a batch implementation response, the wrapper publishes the message on a single topic which collects the responses of all batch implementations. The Batch Activity Controller is a subscriber to this topic and processes the responses of the batch implementations as described above.

5.3 Discussion

Recalling the related work, batch processing functionalities are currently designed and implemented as inseparable part of the BPM system. In the modular BPM system architecture we propose here, this functionality is outsourced to support all three proposed flexibility strategies for batch activities. Even though the execution strategy caters for highest degree of flexibility, modularity and maintainability, still in some cases the requirements on a solution do not call for as much flexibility, or the effort and budget requirements to implement such a system would not allow its development. Specifically, the effort related extending the Process Modeling Tool and Process Deployment Component of a BPM System are increasing for each strategy: while for the modeling strategy only a new modeling element has to be added to the modeler, which needs to be parsed by the deployment component, for the deployment strategy different modeling artifacts have to be handled by both components, and for the execution strategy the Batch Activity Management Tool is additionally necessary to (un)deploy batch models during process execution. Still, even in the case of the simple modeling strategy, the advantage of the modular batch activity component is that it allows for improved maintainability and flexibility at a later point in time if requirements change.

For enhancing an existing BPM system with the Batch Activity Controller functionality two things have to be available: 1) the implementation of the Batch Activity Controller and 2) a mapping between the event model of the process engine and the event model of the presented architecture. The architecture we presented here is designed from its onset with the objective to allow for and encourages the reuse of the Batch Activity Controller implementation as described in [17]. The minimum remaining prerequisite for being able to use the batch
processing functionality is the mapping between the event model of the process engine with the event model used in the Controller component. Even though this brings certain implementation complexity, such an approach has been successfully applied for example in the work [11] on a pluggable, generic architecture for enabling extending behavior of BPEL engines and in works like [8] and [34] that separate the control flow adaptation functionality from a process engine for service compositions and choreographies.

6 Conclusion

In this paper, we presented an approach that has the goal to improve the flexibility of batch activities in business processes, while also improving the flexibility of processes in general and increasing the degree of modularity and the maintainability of process artifacts. To this end, we introduced the concept of flexibility strategies for batch activities, namely the modeling, deployment and execution strategies. These strategies help to realize in a systematic way batch processing in business processes with different flexibility degree: from simply allowing batch activities, through batch activity variants and multi-process batching, to flexible batch activities as a dynamic process adaptation mechanism. The individual strategies address one or more flexibility needs for batch work in business processes. Each of the three strategies imposes different requirements on the BPM system and exhibit different advantages and disadvantages. Based on the identified requirements, a conceptual architecture for a BPM system was contributed to support all introduced flexibility strategies.

While we recommend to use the execution strategy for highest degree of flexibility and modularity, we also advise to select the strategy that fits best the business context and the existing BPM system in an enterprise. Our approach can support such a decision since it provides the systematic framework to compare the strategies based on the requirements they impose and the advantages they give.

Our plan for future work is to focus on several aspects. On the one hand, we will work on finalizing the proof-of-concept prototype supporting the flexibility strategies introduced in this paper. In addition to that, we will work towards enabling the monitoring of processes and batch processing, and will consider the different strategies. So far our focus was on the runtime adaptation of a batch configuration targeting batches which have not been started yet. In future, runtime adaptation of batches even after their execution has been started is of particular interest. Since this topic is closely related to the type and properties of participating resources, we will investigate how a resource management can be incorporated into the presented approach in terms of concepts, realization and optimization mechanisms. In this respect, the life cycles of resources and batch activities have to be identified, and the way they intertwine with the life cycle of processes and their instances researched.
References

15. OMG: Business Process Model and Notation (BPMN), V. 2.0 (2011)