Preprint
Published version: http://dx.doi.org/10.1142/S0218843016500027

International Journal of Cooperative Information Systems

APPLICATION OF DYNAMIC INSTANCE QUEUING TO ACTIVITY SEQUENCES IN COOPERATIVE BUSINESS PROCESS SCENARIOS

JOHANNES PFLUG

Faculty of Computer Science, University of Vienna, johannes.pflug@univie.ac.at Vienna, Austria

STEFANIE RINDERLE-MA

Faculty of Computer Science, University of Vienna, stefanie.rinderle-ma@univie.ac.at Vienna, Austria

Received (Day Month Year)
Revised (Day Month Year)

The optimization of their business processes is a crucial challenge for many enterprises. This applies especially for organizations using complex cooperative information systems to support human work, production lines, or computing services. Optimizations can touch different aspects such as costs, throughput times, and quality. Nowadays, improvements in workflows are mostly achieved by restructuring the process model. However, in many applications there is a huge potential for optimizations during runtime as well. This holds particularly true for collaborative processes with critical activities, i.e., activities that require a high set up or changeover time, typically leading to waiting queues in instance processing. What is usually suggested in this situation is to bundle several instances in order to execute them as a batch. How the batching is achieved, however, has been only decided on static rules so far. In this paper, we feature Dynamic Instance Queuing (DIQ) as an approach towards clustering and batching instances based on the current conditions in the process, e.g., attribute values of the instances. Specifically, we extend our previous work on applying DIQ at single activities towards a queuing approach that spans activity sequences (DIQS). The approach is evaluated based on a real-world case study from the manufacturing domain. We discuss limitations and further applications of the DIQ idea, e.g., with respect to collaborative human tasks.

Keywords: Business Process Management Systems; Process-aware information systems; Process analysis; Process optimization; Cooperative Information Systems

1. Introduction

Enterprises usually regard Business Process Management as an enabler for *optimizing* their business processes. The streamlined and effective implementation of their business processes can constitute the key advantage over their competitors in the market. As business processes can become very complex consisting of several hundreds of business activities with a highly collaborative character during design time

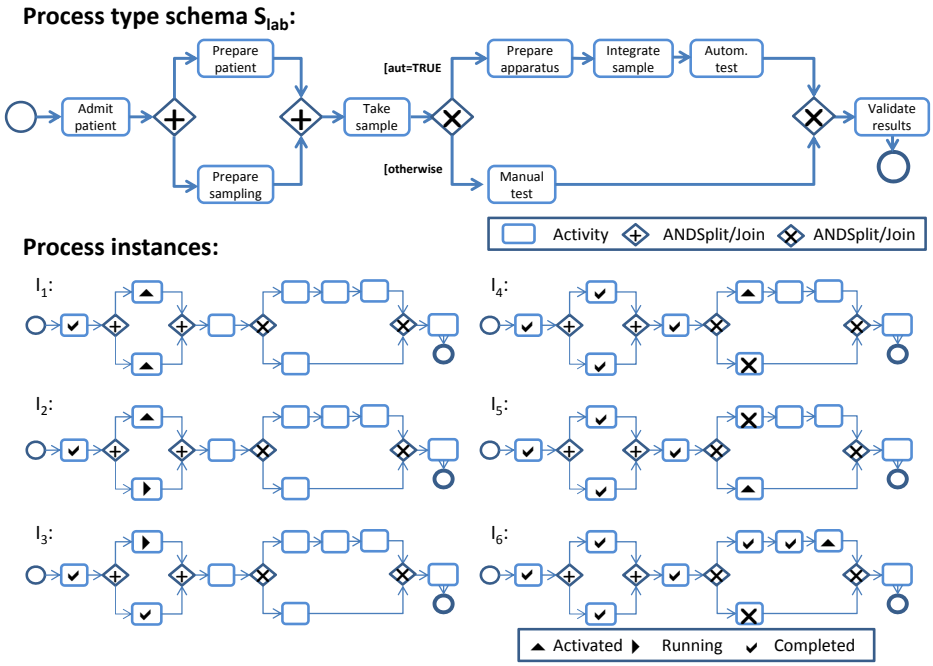

Fig. 1. Laboratory Process in a Hospital [11]

as well as serve as basis for several thousands of active process instances during run time (e.g., in the automotive industry [6]), their manual optimization is no option. In turn, the provision of (tool-supported) optimization strategies during design and run time is crucial.

1.1. *Motivation*

Business processes are captured by process models following a meta model such as BPMN at design time. Figure 1 depicts the process model for a laboratory process in a hospital (slightly simplified case study taken from [11]). The interaction of a high number of solitary human tasks constitutes the overall process. Based on process models, analysis techniques such as simulation can be applied to, for example, identify over-long process chains, to decrease process throughput times, or to balance workload on the resources. Optimizing the process models at design time is crucial in order to optimize the process models before they are actually implemented and enacted within a Process-aware information systems (PAIS).

PAIS foster the implementation, management, execution, and analysis of business processes. At runtime based on a schema describing a certain process type, e.g., a patient treatment process or an insurance claim, process instances can be initiated and executed. Each of these process instances represents a certain case for

this process type, for example, processing a blood sample for patient Smith through process instance I_1 in Figure 1.

Even if the process model has been optimized at design time, further optimization can become necessary during runtime. Let us explain this based on the laboratory example (cf. Figure 1): after running the process for some time it was learned that processing the samples automatically was expensive due to the high preparation expenses (time-wise and monetary). The high *change over times* led to *waiting queues* of process instance at runtime. Such waiting queues mostly arise if a certain number of process instances arrive at an activity with restricted resources, e.g., one apparatus or one clerk. Such process activities are referred to as *critical activities* [27]. To tackle this problem for the laboratory process, the preparation and execution of the automatic sample check was scheduled to certain time frames (e.g., every fourth day at 9 am) in order to batch as many instances as possible and hence achieve a reduction of costs and time.

Although batching instances on the apparatus seems to be promising, the realization by a static rule as proposed might not exploit the full potential for optimization. The reason is that the rule does not take into consideration the actual runtime situation at a certain point in time, for example, that due to a holiday season it might be even better to wait for 5 days to obtain a good efficiency of the laboratory automaton. Particularly for processes that involve human tasks a strict prediction of what will happen during runtime is not possible. Hence, it could be promising to investigate means for *Dynamic* Instance Queuing during runtime of the processes [27].

1.2. *Problem Description and Objectives*

As motivated by the laboratory example, ignoring waiting queues in collaborative information systems during runtime might become expensive. Different approaches have been proposed that offer queuing (also in combination with batching [19]) for process optimization and as a strategy to prevent delays and deadline violations [40]. Although queuing and batching as proposed in [40,19] become effective at runtime, the underlying rules and strategies are still static, i.e., fixed at design time. Liu and Hu [19], for example, proposed First-In-First-Out (FIFO) logic for processing the instances in the queue. Hence, the specifics of the current situation in the system known by the PAIS are again not taken into consideration. Recent approaches advocate queue mining for service processes as ex-post analysis for predicting delays [35]. Service processes are another good example for the necessity to analyze and manage queues in business processes, also dynamically.

In this paper, we elaborate on the principle of *Dynamic Instance Queuing*, i.e., aggregating incoming instances at critical activities based on automatic decisions during runtime [27]. For understanding the conceptual extensions provided in the paper at hand, we want to first draw the reader's attention to the following two cases: either we can think of dynamically queuing instances at one (critical) activity (de-

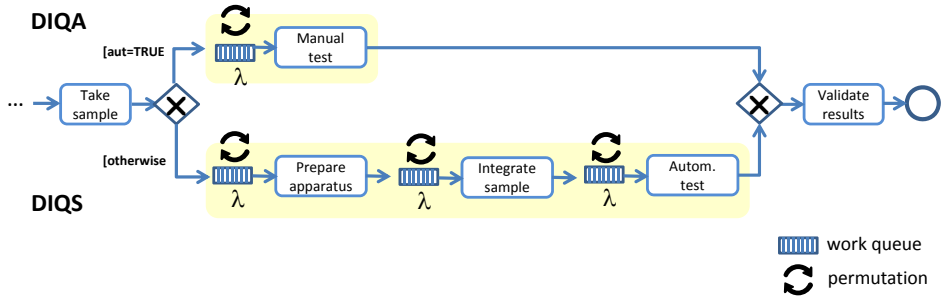

Fig. 2. Applying DIQS versus DIQA

noted as DIQA [27]) or Dynamic Instance Queuing can be applied over a sequence of activities (denoted as DIQS) as we will analyze in this paper. In Figure 2, for example, DIQA would mean to queue at activity `Manual test`, whereas DIQS refers to queuing at any of the activities incorporated by the sequence `Prepare apparatus, Integrate sample, Autom. test` as indicated by the yellow box. In particular, we address the following research objectives:

O1 The integration of DIQ in workflow engines as well as available configuration parameters.
O2 The application of DIQ in the sequence pattern (DIQS).
O3 The evaluation of the performance of DIQS both on a theoretical basis and in a case study from

1.3. *Contribution*

The general idea of DIQ is that process instances at a critical activity or at critical process patterns are collected and automatically batched based on clustering techniques. We will provide an overview of how the DIQ algorithm works in general in Section 2. Note that this paper grounds on our previous work presented in [27], but extends and elaborates it in many ways: first of all, DIQ is applied to activity sequences (DIQS) instead of single activities. Moreover, a detailed description of implementing and integrating DIQ into a PAIS architecture is provided. A new case study from the manufacturing domain has been conducted and DIQS is compared to DIQA on basis of this case study. Finally, discussion and related work are reworked and extended.

Section 3 will explain all phases of the DIQ algorithm in detail and further elaborate on a) incorporating sequences into queuing, i.e., considering DIQS on top of DIQA and b) analyzing the usage of different clustering algorithms within the algorithm. By contribution a) we address objective O2. Section 3 will also comment on how the algorithm was implemented.

Tackling objective O3 we will provide a real-world case study from the manufac-

turing domain in Section 4. By contrast to the case study presented in [27] based on one activity requiring a printer as critical resource, in the manufacturing case study we consider a sequence of activities. It is first evaluated which reduction of instance processing time can be achieved by applying DIQS when compared to the current real-world processing time (FIFO). Then we evaluate DIQS over DIQA in Section 5. An interpretation of the results is provided in order to explain the reasons for the processing time reduction.

In the discussion (cf. Section 6), we elaborate on extensions of DIQ in PAIS such as supporting further process patterns[a] as well as taking behavior of resources into account. Moreover, our assumptions will be discussed: (a) a process that contains at least one *activity* or *sequence*-workflow pattern at a critical activity (b) finishing times of the instances are not fixed (c) resource behavior is deterministic. Related approaches from the area of PAIS research as well as from other areas, e.g., in messaging systems, are demarcated in Section 7. Finally, Section 8 concludes and shows future research directions.

Employing process technology to conduct daily work more efficiently is one of the most convincing arguments for the application of these systems in practice. This paper provides the next brick when building comprehensive support for runtime optimization in PAIS. Moreover, DIQ in combination with techniques for instance synchronization [21,15,30] constitutes a means to support cooperation in PAIS. Take as an example a lab setting where several organizational entities and their business processes share a resource. Here, DIQ with instance synchronization can support the cooperative usage of the resources in an efficient and effective way.

2. Foundations of Dynamic Instance Queuing

In this section, we will first present the general concept of the Dynamic Instance Queuing approach, followed by a more detailed elaboration on its different phases.

2.1. *General Approach*

Dynamic Instance Queuing (DIQ) is an algorithm based on artificial intelligence techniques that aims at reducing the processing time at critical activities by optimizing the maximum throughput. We refer to an activity as being critical if due to restricted resources assigned to the activity, a significant number of instances cannot be processed momentarily after arriving in the resource's work list continuously throughout the workflow execution and thus leading to a waiting queue. So far, queuing has been adopted for process optimization in a merely static manner, i.e., the strategy in which order the instances are processed from the queue is fixed. However, in PAIS, it is quite difficult to foresee dynamic behavior, particularly in the context of long running processes and at the presence of a multitude of process

[a]Typical process patterns comprise of sequences, parallelism, or alternative branches. For an overview see `www.workflowpatterns.com`.

instances. This hampers the definition of strategies for processing instances at critical activities at design time. We argue that determining the processing strategy for instance queues at runtime (dynamic queuing) offers the potential to reduce the processing time in a flexible, easy-to-implement way that is appropriate for a lot of application domains.

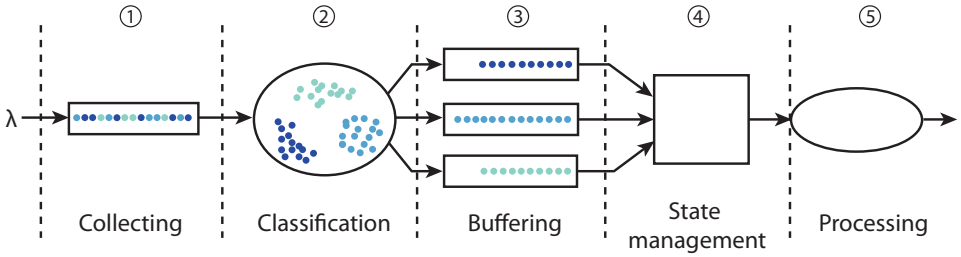

Fig. 3. Different phases of Dynamic Instance Queuing (adapted from [27])

Figure 3 shows the generic concept of Dynamic Instance Queuing. The core idea is that similar instances can be processed faster in a row by making use of optimizations such as reduced changeover times, caching or gaining of routine by human resources than randomly distributed instances. Incoming instances at a critical activity are at first collected (step 1) and, when the first resource shifts to idle state due to a lack of instances, are classified into groups (step 2). The classification is based on instance attributes such as patient age or color mode depending on the use case. The instance classes are then transferred to the buffers of the queuing system where each buffer represents one instance cluster (step 3). The instances from one buffer are then processed at once by the resource (step 5). The state management service provides a continuous assessment of the effectiveness of the current setting which is required for an effective mapping of instance clusters on resources (step 4).

By classifying the instances in a logic way, the algorithm uses the fact that processing times are often not independent from the instance order: their processing times decrease when similar instances are handled sequentially. Saving potentials arise from

- the elimination of changeover times by machines when handling similar materials/components or
- optimized performance by computer technologies such as caching
- lower processing times by humans when working on similar work items, e.g., gaining routing, acquiring experience

One or more of the above mentioned reasons exist in nearly every application scenario. For the laboratory process displayed in Figure 1, instances reflecting the processing of samples on one laboratory apparatus could be clustered based on

instance attributes analysis technique or sample size. In the industrial domain, optimization potential arises from eliminating changeover times at machines. As reordering instances in the resource's work list does not have negative impact on temporal parameters such as the throughput time both on a instance and a cumulated basis for the single-tasks and sequence pattern [28], positive effects of DIQ directly affect the overall performance.

2.2. *Dynamic Instance Queuing applied to activities (DIQA) and sequences (DIQS)*

In a previous work, we investigated how DIQ works on critical activities (DIQA) [27]. Usually business processes are composed of different structural patterns such as sequences, parallel/alternative branchings, or loops [41]. Could throughput times be further reduced when applying DIQ on process patterns instead of single activities? Figure 2 displays a fragment of the health care process presented in the introduction. In the top figure, DIQA is applied to an activity `Prepare apparatus`. By contrast, on the bottom the entire sequence of activities `Prepare apparatus`, `Integrate sample`, and `Autom.test` is covered by DIQS.

The major difference between a sequence and an activity (i.e. single-task pattern) is the optimum alignment. For a single-task, there is exactly *one* ideal alignment in terms that all similar instances for a certain processing rule are arranged in a row. In a sequence pattern, where several *different* single-tasks are orchestrated, there is a different optimum alignment for each task (note that if *equal* single-tasks were orchestrated, a loop-pattern arises). In consequence, this optimum arrangement can only be obtained by reordering the work items for each individual resource's work list that is associated to the incorporated tasks of the sequence. Therefore, DIQ is expected to perform better on sequences than on single-tasks. In Section 5, we provide a quantitative comparison of DIQA and DIQS based on a manufacturing case study.

2.3. *Temporal Concurrency Concept*

Figure 3 shows one iteration of the Dynamic Instance Queuing approach. One iteration contains the collection of arriving instances as long as all resources are busy, followed by the classification step and finally the processing of the instances. However, during runtime, iterations might overlap since new instances are arriving steadily, hence necessitating a concurrency concept as depicted in Figure 4. The horizontal layers represent the phases of Dynamic Instance Queuing (Collecting, Classification, Buffering, Processing and the State Management Service) as explained in Section 2.1. All phases are being executed constantly, however they are assigned to different iterations within the workflow. An iteration is constituted by the sum of instances between two subsequent classification steps from the collecting to the processing phase. In Figure 4, the entirety of same-colored/patterned intercepts represents one iteration in our Dynamic Instance Queuing approach.

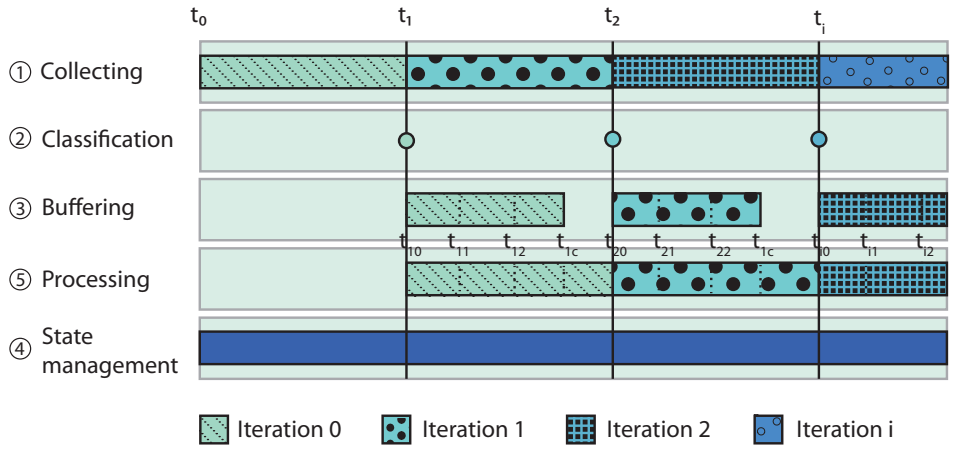

Fig. 4. Temporal concurrency approach

In the following, let t_i represent the time of the i-th classification, while t_0 is considered to be the start of Dynamic Instance Queuing, i.e. the time when the first instance arrives. At first, arriving instances need to be collected for a certain time. The classification of instances represents a point in the time line, visualized as the dot in phase 2. From then on, the classified instances are buffered and, concurrently, the processing of the first instances from the buffers begins. The process of buffering is finished, when all contained instances are moved to a resource and hence, all buffers are empty. The time when the last instance is processed also constitutes the end of one iteration. This means, e.g., iteration *0* starts at time t_0 and ends at t_2. In general, iteration i starts at time t_i and is finished by time t_{i+2}.

Consider now the phases of buffering and processing. These two intercepts are designed to be concurrent within one iteration. The buffering begins right after the classification has taken place. For iteration i, buffering begins at time t_{i+1}. Concurrently, the processing of the instances within the first buffers begins. In the following, the instances from one buffer are transferred to the resource when all instances from a previous class have been fully processed. This takes place at time t_{ij} where i represents the i-th classification and j represents the j-th buffer to be processed. For example, at time $t_{2,1}$, all the instances from the first class (buffer) in iteration 2 have already been processed and the instances from the next unprocessed buffer will be transferred to the resource. This implicates that the processing step always takes longer than the process of buffering.

The processing logic remains the same during the entire process life cycle. This includes the handling of the initial elements, i.e., the instances that are classified first. When the first instance arrives, the resources are in idle state. Hence, classification is executed instantly. It is not performed earlier since any classification works best with the highest possible number of elements. The emerging cluster consist-

ing of this single instance will be transferred to one of the resources and processed momentarily. No waiting times arise, the throughput time and processing time are the same. With respect to the process model, no queue arises. This is intuitively understandable, since queuing is necessary only if more instances are arriving than the resources are able to process. Dynamic Instance Queuing ends when the overall process is finished.

3. Generic Concept of DIQ

This section describes the components of the DIQ algorithm in a generic way.

3.1. *Generic classification component*

The classification component classifies incoming work items into groups of similar instances based on certain instance attributes. As described in Section 2.1, the emerging instance classification is mapped onto the buffers and, finally, onto the resources, i.e., there will be a 1:1 mapping between clusters and buffers hence making use of the fact that similar instances might be processed faster in a row than randomly distributed instances. The key advantage is that the classification component defines the processing strategy dynamically during runtime, which makes any pre-specification of rules and conditions obsolete. In our DIQ approach, the classification component is designed in a generic way so it doesn't provide a predefinition of the specific classification technique (e.g. clustering algorithms such as k-Means, OPTICS or supervised learning algorithms) to be used. Consequently, any method that is appropriate for the specific nature of the particular application scenario can be implemented.

Algorithm 3.1 shows the context of the classification component within steps 1 and 2 of the overall approach as depicted in Figure 3, i.e., the classification of arriving instances at a critical activity and the subsequent distribution onto buffers. Buffers serve as data structures to store the instances between classification and processing. The number of buffers is determined by the number of clusters of the previous classification step. As different classifications within the process execution may result in different numbers of clusters, the number of buffers may vary over time. The generic notion of DIQ does not only appear in the classification algorithm to be used but also in its actual specification. Depending on the application scenario, one could impose an adequate lower or upper bound to the number of clusters in order to achieve the best performance. The order in which the instances are queued within the buffers is adjustable as well. Besides basic logic such as *First-in-first-out*, one could envisage a certain priority order on the buffers to work well with the later processing behavior of the resources.

Note that Algorithm 3.1 reflects the first two steps of Dynamic Instance Queuing within *one* iteration. In Section 2.3, the temporal concurrency concept of Dynamic Instance Queuing has been introduced where it has been shown how the algorithm works over several iterations. In the case study (Section 4), a specification of the

Algorithm 3.1 Classification and queuing concept

```
1: while Buffers are not empty do
2:     gather arriving instances in dataset d;
3: end while
4: C := Classify(d)                    ▷ Individual classification algorithm to be used
5: for classes c ∈ C do                ▷ Individual lower/upper bound for classes
6:     select free buffer b;
7:     move instances from class c to buffer b; ▷ Based on adequate priority order
8: end for
```

classification component for a manufacturing process in the industrial area will be provided.

3.2. *State management service*

The state management service is responsible for the assignment of instance clusters on resources. It represents a permanent evaluation of the effectiveness of the current setting, i.e., the dynamic evaluation of the performance based on runtime variables to react on changes in the process environment. A state management service is required if the setting includes at least one activity which can be processed by two or more resources.

The performance evaluation is always both depending on the specific scenario in which DIQ is meant to be implemented and on the trade-off between time, costs, flexibility, and quality of service (cf. critical parameters for business process redesign as stated in [22]). Some of these parameters seem to be hard to reach at the same time; especially time and costs appear to be at conflict. A generic decision function should incorporate all relevant parameters for the specific application scenario. From a *temporal* point of view, especially the throughput time, waiting time, processing time tend to be most relevant, while *cost variables* typically include costs per time unit, energy consumption or initialization costs. Flexibility and quality of service parameters are hard to quantify and often not included in performance analyses. However, indicators such as prioritization demands, compliance, or number of errors might be concerned.

Based on the described parameters, the state management service fulfills the following functions:

- **Assignment of instance clusters to resources** refers to the mapping of instances being processed at a certain activity by the best available resource. The evaluation of the *best* resource is based on a scenario-dependent individual decision function. This becomes necessary if the setting includes at least one activity that can be processed by two or more resources.
- **Management of resource states** controls the state (active or inactive) of resources dependent from the process environment during runtime.

As for the classification component, a specification for the generic state management service will be provided in the case study (Section 4). In [27] we describe a decision function that encounters both time and costs parameters and is applicable for a lot of scenarios.

3.3. *Implementation of Dynamic Instance Queuing*

As DIQ follows a generic approach, it can be integrated in any conventional PAIS. Figure 5 shows the architecture of a typical workflow engine [3,5]. The workflow engine as the core element receives notification about completed work items (step 1). A work item corresponds to an instance in a business process. The workflow engine examines the workflow specification to determine what work node needs to be activated (step 2). In steps 3 and 4, the workflow engine consults a resource broker in order to evaluate a resource for the activated work node. Finally, the work will be assigned to the appropriate resource's work queue (step 5).

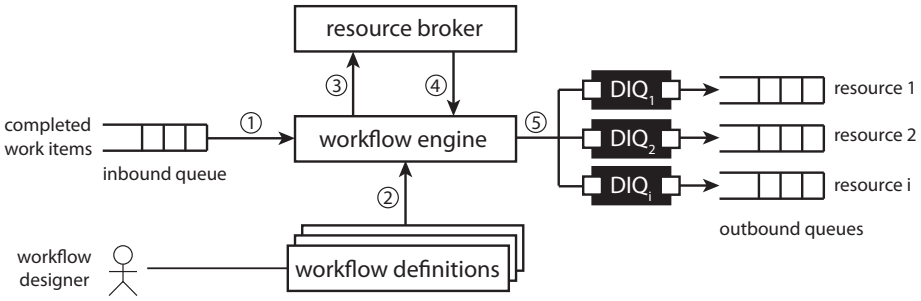

Fig. 5. Integration of DIQ in a generic workflow engine (based on [3])

The DIQ component is located between workflow engine and the resource's working queues. Based on the results from the classification (as described in Section 3.1), the DIQ component arranges the work items in a way that similar instances can be processed in a row. As one can see, our implementation of DIQ follows a self-contained approach. This keeps the DIQ component independent from the implementation of the host workflow engine.

DIQ includes two interfaces that share the runtime status with the host workflow engine. Figure 6 offers a schema for the operation of this design. It shows a draft of a prototypical process containing i resources with DIQ activated. The left side of the figure represents a global view of the process. The little dots represent instances within the process; their color represent similarities among instances. Instances within the ovals are currently processed by the *i-th* resource.

During runtime, global variables are captured by any PAIS, such as number of instances within the system, the current performance (as assessed by the state

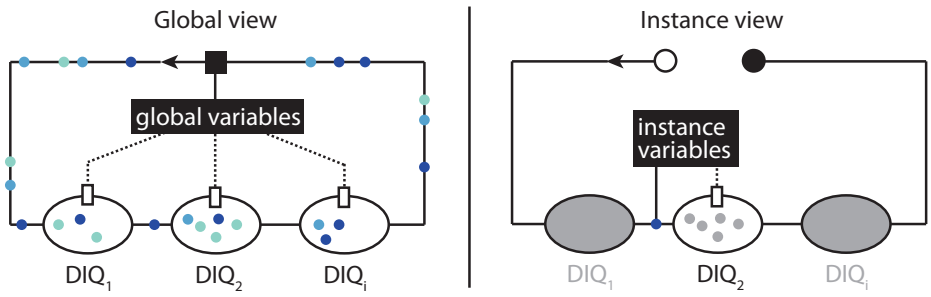

Fig. 6. Runtime schema

management service) or the global time. These variables are read by each DIQ component via an interface, as represented by the rectangle. However, one needs to distinguish the instance view of the process as shown on the right side of Figure 6. The instance to be considered is located right *before* the second resource that has DIQ activated. Each instance possesses instance attributes. These instance attributes are dependent of the application domain. In the scenario described in a subsequent section of this article, for example, these would be the length, width, and depth of a single raw material. Based on these attributes, the classification of similar instances is executed by the DIQ component. The instance variables are read independently by any DIQ component when the specific instance arrives at the respective pattern.

The described interface architecture carries on the generic approach of DIQ to the implementation level. That way, DIQ becomes compatible with any conventional PAIS without implying the need for adaptions in its workflow engine necessary.

4. Case study

In this section, we will describe the application of DIQ in a real-world scenario. Our case studio focuses on a scenario from the manufacturing domain. We already conducted a simulation of a health care scenario, where we analyzed a specific task referring to a print job. In this simulation, we were able to reduce the processing time in an amount of 13% [27]. Different to that, we now focus on a *sequence* of tasks, which is more representative for a variety of different scenarios. The described case study is based on real-world data.

4.1. *Application scenario*

The production line of a retailer with an own production line comprises individual construction materials based on standardized, stocked raw materials. More concretely, different types of wooden blank are being transformed to the end product in subsequent stages of production based on the individual ad-hoc orders from customers. A single production process starts with an employee fetching the raw

material needed for the order from the stock. In an automated process, the wooden blank is being individually cut and milled. If necessary, holes are being drilled. The automated part of the production process finishes with a sealing step of the raw material. The end-product is finally being packed by another employee from the company. This process is visualized in Figure 7.

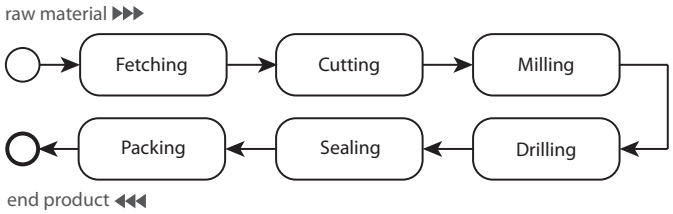

Fig. 7. Case study - Sequence fragment from the Manufacturing Process

Each order represents the transformation of one or more raw materials into the requested product. Incoming orders occur continuously, mostly within the store of the retailer. As customers wait for their order to be completed, detailed production schedules cannot be computed in advance. Reducing waiting times for the customers is a central target of the company.

The described sequence of activities is embedded in a larger production process and implemented through a PAIS. Due to restrictions in scope, we focus on the subprocess which will serve as the scenario for the implementation of DIQS. All further investigations and numbers relate to this subprocess.

4.2. *Scenario analysis*

The case study is based on a dataset that provides detailed information about each product order, its characteristics, and times for each activity within the manufacturing process described in Section 4.1. In this scenario, we consider one product as an instance within the process of transforming the raw material into an end product. A product is manufactured individually respecting the customer's needs. Hence, the activities' temporal sequence is dependent on the orders to be given. In fact, besides *fetching* and *packing*, the remaining activities can be executed optionally, i.e., certain activities might be skipped for some instances. This results in a number of $\binom{4}{1} + \binom{4}{2} + \binom{4}{3} + \binom{4}{4} = 15$ possible workflow executions. We focus on the production of one day as a characteristic model for the overall manufacturing process. Note that the potential variations in the process instance executions are captured by the realistic data set. On the day of interest, 50 orders are triggered with the first one incurring at 8:08:13 and the last one at 18:12:32.

The throughput time of one instance turns out as the sum of the activities' *processing times* and *waiting times*. The *processing time* constitutes as the duration

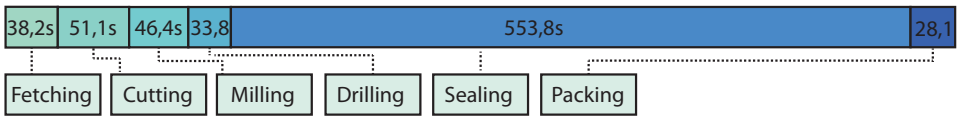

Fig. 8. Case study - Average processing times within the manufacturing scenario

of the respective activity (fetching, cutting, milling, drilling, sealing or packing) for one instance. However, due to limited resources within the production line, periods also include some waiting times when certain machines are busy. This is the case when more instances (i.e., orders) are being triggered than the existing machines are able to handle and hence results in the arising of waiting queues. The time an instance is kept within a waiting queue is being considered the *waiting time*.

The mean average processing time of one instance is $751,4$ seconds, which divides into average times for fetching ($8,2$ sec.), cutting ($51,1$ sec.), milling ($46,4$ sec.), drilling ($33,8$ sec.), sealing ($553,8$ sec.) and packing ($28,1$ sec.). A graphical representation of the average processing time of one instance is provided in Figure 8. The throughput time, however, includes waiting times within the production line. Figure 9 shows the average throughput times, i.e. the sum of waiting time (shaded area) and processing time (colored area) per activity. One can see that massive waiting times occur. We refer to those activities as being critical (cf. Section 1), as due to restricted resources assigned to the activity, a significant number of instances cannot be processed momentarily after arriving in the resource's work list. Note that activities with $waitingtime + processingtime = 0$ exist since not every production step is always executed for each single instance.

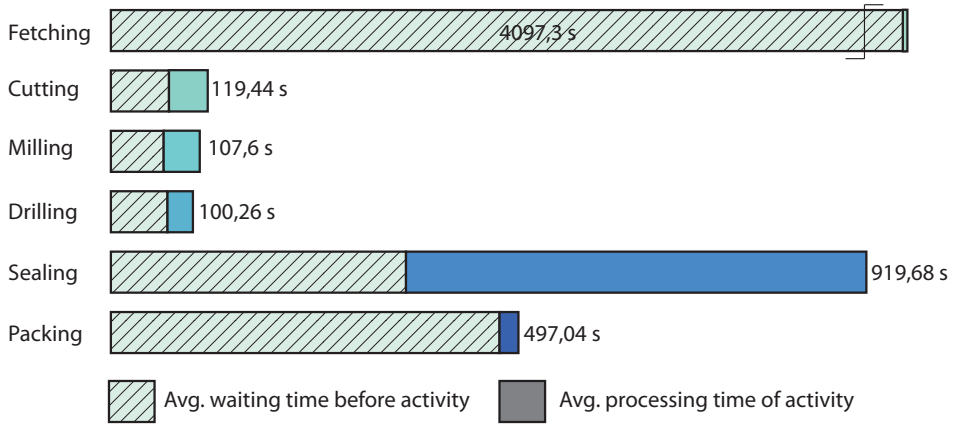

Fig. 9. Case study - Average throughput times of the production tasks

4.3. *Applying DIQ in the manufacturing scenario*

In Section 4.2 we illustrated that the process from a real-world scenario includes *critical* activities. Dynamic Instance Queuing offers a way to reduce the throughput time of instances at these kind of activities. The actual *processing* time of an activity is the time that is needed by the respective machine to execute the corresponding production step, i.e., the processing time is deterministic and immutable from the process environment. However, the waiting time of an instance before an activity is not only dependent of the kind of work to do, but might result from so called changeover times. These periods need to pass if instances with different properties are meant to be processed subsequently. This means that a proper alignment of the instances can reduce changeover times. DIQ estimates an optimized alignment of instances based on a classification of the instance properties.

Consider Figure 10. As an instance corresponds to a product, it possesses its relevant properties: kind of material, its length, depth and width. The properties of the raw material depend on the individual end product to be manufactured, which involves the target dimensions (matrix of length, width and depth), the information whether the material shall be milled or drilled, and if so, the number of holes, their depth and circumstance. A declaration whether the product needs to be sealed (*sealing_ bool*) and in which way (*sealing_ both-sided*) must be provided. Furthermore, an end-product is always associated to a distinct order.

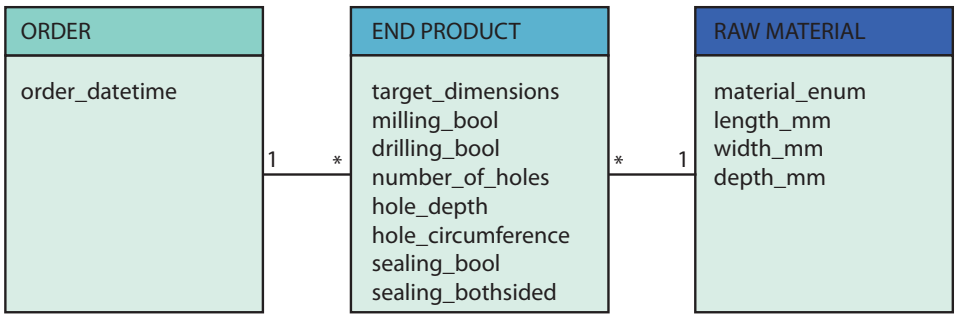

Fig. 10. Case study - Instance properties

Each single instance property is relevant to the throughput time (sum of waiting time, idle time and processing time) of at least one activity (as shown in the mapping provided by Table 1). For example, the processing time of *milling* is dependent from the kind of material to be manufactured. This means that the overall throughput time of one instance is subject to the *linear combination* of twelve parameters. This number of combinations makes it impossible to foresee their implications on instance processing time during runtime. Moreover, the complexity cannot be tackled by a rule-based system where the property conditions need to be defined a-priori.

Using the DIQ approach, it is not necessary to make these specifications a-priori. The alignment of properties is done dynamically during runtime using a artificial intelligence classification methods (as described in the section beneath).

Table 1. Case study - Mapping of relevant properties on activities

Activity	Relevant properties
Fetching	material_enum, length_mm, width_mm, depth_mm
Cutting	target_dimensions
Milling	milling_bool, material_enum
Drilling	drilling_bool, number_of_holes, hole_depth, hole_circumstance
Sealing	sealing_bool, sealing_both-sided
Packing	target_dimensions

4.4. *Specification of the classification component*

Applying DIQ necessitates the specification of the classification component at first (cf. Figure 3). The classification component aims at finding similar instances according to their properties and joins them into groups. The rationale behind is that similar jobs can be processed in a row more efficiently than in an arbitrary manner probably causing changeover times. Changeover times occur when a machine has to switch between instances with different attributes, e.g., by adjusting the drilling hole measures. A classification strategy offers the potential to find an optimized instance distribution that reduces changeover times significantly, which directly affects the overall throughput times. As described in Section 4.3, the instances in our scenario possess 13 different properties reflected by numerical parameters which enables the application of a variety of classification techniques.

In the given example, we embark the use of a clustering method. Clustering techniques meet the flexible character of DIQ since they do not require a training set and thus are applicable without further preparatory work. However, there is a variety of clustering approaches that can be considered for the implementation. Basically, there are two kinds of algorithms. In centroid clustering methods, groups are represented by a central vector. They base on a given order of objects and thus a fixed number of clusters. Using a replacement algorithm, the individual elements are exchanged until a certain target criteria is met. Density based clustering methods separate a set into areas of higher and lower density. Some objects are considered to be more related to nearby objects than to objects farther away.

We opt to use a centroid based clustering method since it allows a pre-specification of the numbers of clusters. The number of clusters represents the maximum batch size, i.e. the number of raw materials that can be processed in a batch. One of the most common centroid based clustering methods is the *k-Means* algorithm. This algorithm targets to find the k cluster centers and assigns

the objects to the nearest cluster center in a way that the squared distances from the clusters are minimized. In the k-Means algorithm, the variables to be considered must not be known a-priori. This contributes to the dynamic character of the proposed approach since determining a number of variables would be needed to be done at design time or by the user during runtime. Both options are less favorable than the automatic detection by the k-Means algorithm. For DIQ, we implement the k-Means algorithm as described by MacQueen in 1967 [20]. The problem is computationally difficult (NP-hard); however, we MacQueen's implementation employs heuristics such that the algorithm performs efficiently.

4.5. *Specification of the state management service*

As a next step, the state management service is specified for the application scenario. As shown in Figure 9, massive waiting occur within the process of transforming a raw material into a construction material. This makes the reduction of the changeover times the most essential task, aiming at decreasing the overall throughput times. State management offers the possibility to assess the most efficient number of resources at any time. Based on a decision function, the number of resources to be applied can be automatically increased or decreased. However, in the setting of our real-world scenario, the number of resources is fixed as there is exactly one set of machines available for the processing of the orders. In order to guarantee full comparability to the simulation, we don't apply the dynamic handling of resources in this scenario. Instead, we enable the person responsible for the process execution to take actions on the resource handling.

4.6. *Implementation*

DIQ was simulated under the exact same parameters as the current setting from the case study based on detailed logs. That way, full comparability is ensured. The algorithm was implemented in Java using high level concurrency techniques (as shown in Figure 4). The centerpiece of the DIQ implementation is a *QueuingSystem* class that is responsible for the initialization and coordination of all components. The threads for existing resources are managed by a *ResourceManager* class which offers functions to assign the buffers to the resources and to adapt the number of active resources as calculated by the decision function from the state management system. Buffers are similarly managed by a *BufferManager*. Since it is a real time simulation, the runtime environment does not have impact on the performance of the simulation. As proposed before, the k-Means algorithm was chosen for clustering. For that purpose we used the Java Machine Learning Library (Java-ML) [1].

4.7. *Simulation results*

Assessing the performance of DIQ in the given scenario means comparing its temporal efficiency with traditional approaches. In the given example, traditional approach means the status quo technique, which is a first-in-first-out (FIFO) logic.

This means, jobs are processed in the same order they have been triggered before. As described in Section 4.6, comparability between the simulation of the two techniques is guaranteed.

Figure 11 shows some details of the temporal performance of DIQ. In the first diagram, the cumulated throughput time of all jobs is compared. The cumulated throughput time is the sum of the periods between the time of a job being triggered and its processing end, which is the packing of the construction material. This period takes 291966 seconds using the status quo FIFO-logic, while applying the DIQ approach, the sum of the throughput times is only 257248 seconds. This corresponds to a reduction of almost 13%.

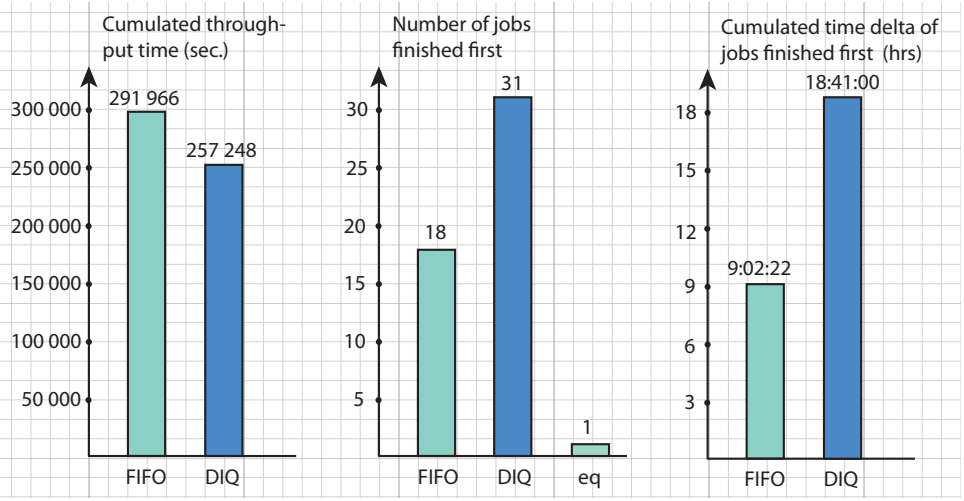

Fig. 11. Case study - Performance of DIQ compared to FIFO approach

Considering the endpoints of the instances, more specifically the time a construction material has been successfully packed, the situation is as follows: applying DIQ, 31 of the 50 jobs are finished faster than applying FIFO logic, which finishes first for only 18 instances. One job is equally fast processed by both approaches. Considering the extent of the durations the instances were processed faster, the analysis is even more obvious (right plot of Figure 11): The 31 instances that were processed more efficiently by DIQ were in a sum finished more than 18 : 41 : 00 hours faster than applying FIFO. Respecting the correspondent number for the first-in-first-out approach (appr. 9 hours), DIQ is in a total 9 : 38 : 38 hours faster.

An important parameter for a lot of scenarios is the total throughput time, i.e. the duration between the arrival of the first order and the completion of the last item. This time span is 40669 seconds for the FIFO approach, while DIQ reduces the throughput time to 39494 seconds. But how good can you be at all? For that matter, we applied another simulation. An optimum schedule represents an ideal

processing order in terms of processing efficiency. In this theoretical simulation, we evaluated a near optimum throughput time of 36559 seconds, if instances are processed in a setting optimized for reduced changeover times. However, the validity of this comparison is limited as evaluating an optimum schedule requires a-posteriori knowledge, as the times of new orders cannot be known in advance. This means an optimum schedule can only be known subsequent to the workflow execution. For a workflow execution in which the trigger time of jobs is unknown in advance as the case study at hand, an optimum schedule is never available. A processing logic could only evaluate a schedule based on knowledge from the past, e.g. the orders from previous days. This might be referred to as a different approach, optimizing before runtime (contrary to DIQ which optimizes during runtime without a-priori knowledge needed).

4.8. *Runtime behavior*

In the previous section, we described that applying DIQ results in a significant performance gain, represented by a reduction of the overall throughput time at the critical activity of around 13%. Having a closer look at the runtime behavior of DIQ, it becomes more transparent how the algorithm transforms its potential to decrease throughput times into real performance gains. To illustrate this effect, Figure 12 shows the evolution of the cumulated throughput time (vertical axis) in the course of the given period (horizontal axis). The solid line represents application of FIFO logic, while the dotted one represents application of DIQ (cumulated throughput times of 291966 sec. and 257248 sec. respectively after processing all 50 instances, see

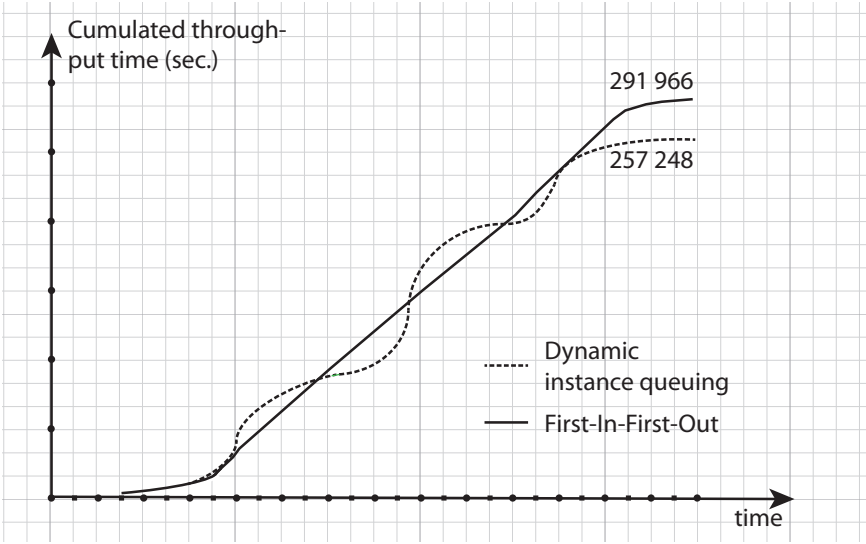

Fig. 12. Case study - Runtime behavior of DIQ

Section 4.7). However, Figure 12 illustrates that the performance of DIQ is varying during the course of the observed period: During some interactions, DIQ and FIFO perform similarly, while in others FIFO is more efficient or DIQ excels. Only at the end of the period under investigation, DIQ remains working more efficiently and finally culminates in the overall throughput time of 257248 seconds.

The reason for this runtime behavior is as follows: DIQ groups instances with similar properties in order to reduce the processing time. The more unprocessed instances are available for clustering (step one "collecting" from Figure 3), the better the classification can work. This means the order is modified more intensely. Compared to FIFO-logic, some instances have a significant lower waiting time, while others have a higher one. In the overall process, these temporary deviations balance out.

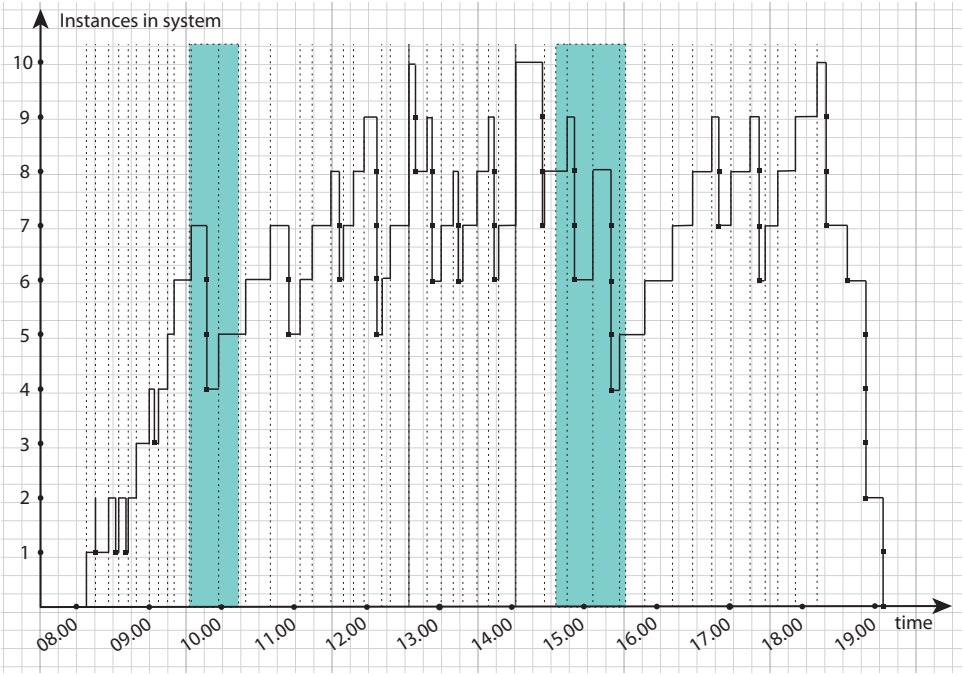

Fig. 13. Case study - Instance distribution during workflow execution

This observation is supported by Figure 13. The diagram shows the number of instances in the system at certain times within the period under consideration applying DIQS. Dark dots represent finished instances, i.e. the jobs that are currently processed are included in the number of instances. The intervals around the turning points after which DIQS runs less efficiently than FIFO (as shown in Figure 12) are highlighted. One can see that in both periods, the number of instances in the system decreases significantly, which means a high number of instances are com-

pleted. When an instance is completed, its numbers incorporate the statistics, i.e. the instance's throughput time is added to the cumulated throughput time.

The course of the indicators highlights the difference between DIQ and FIFO processing: while DIQ optimizes the workflow as a whole, FIFO optimizes the throughput time for each single instance.

5. Evaluation of DIQS over DIQA

The case study provided an implementation of the Dynamic Instance Queuing approach in the context of a *sequence*-pattern. This corresponds to *DIQS* as described in Section 1.2. DIQA, in contrast, applies Dynamic Instance Queuing on *activity* patterns. This technique is embarked in a simulation from the health care domain [27]: The setting covers the printing management in a hospital, where 15 to 40 employees shares two common network units to print medical documents and material. Each print job is sent to one of the two printers, depending on the circumstances of the load. DIQ was applied to evaluate a suitable order of these print jobs according to the diverse number of characteristics these instances possess. By doing so, average changeover times could be reduced by 14%.

To evaluate DIQS over DIQA, we conduct the case study from Section 4 again. However, we apply DIQ on just one of the activities instead on the whole sequence of activities. For this, we chose *fetching*, as it is the most critical activity in our scenario (cf. Figure 9). Relevant properties to this activity are the the kind of material to be processed as well as its length, width and depth (cf. Table 1). In contrast, DIQS evaluated a suitable instance order in the work lists of all activities' resources based on the respective relevant instance attributes. This means DIQA optimizes the most critical activity *fetching*, while DIQS optimizes the sequence as a whole.

The results are shown in Table 2. Applying DIQA, the cumulated throughput time rises to 279269 seconds, which is 7,89% worse than the corresponding result based on DIQS, but still 4,55% better than FIFO. Comparing just the average throughput time for the activity *fetching*, DIQA and DIQS do not differ significantly (33/35 seconds respectively). Both perform better than FIFO (38 seconds). So in this case, DIQS works best for the whole process and equal for the most critical activity.

	FIFO	DIQA	DIQS
cumulated throughput time (sec)	291966	279269	257248
average throughout time at activity *fetching* (sec)	38,2	33, 1	34, 5

Table 2. Comparing strategies FIFO, DIQA and DIQS

6. Discussion

Reijers et al. [32] evaluate the overall performance of a process by means of four criteria, i.e., time, costs, flexibility, and quality of service. Our concept allows a reduction of throughput times in a considerable extent. The decrease is based on the classification of elements to groups of similar instances which offers potentials to reduce the processing times. DIQ works best in scenarios with an improper ratio between number of instances and resource capacity. The more instances are queued, the better results can be obtained. For that matter, DIQ should be considered in application scenarios where the process environment is dynamically changing, important process specifications, i.e. arrival distributions and capacity development, cannot be estimated in an adequate quality, and economic or temporal realities do not allow to develop, test and implement complex static rule systems.

Our approach for DIQ meets several requirements. It is applicable to any conventional Process-Aware Information System since it is a self-contained component that does not imply the need for adaptations in other parts of the PAIS. The optimization takes place during runtime, so no previous specifications have to be made. This distinguishes DIQ from classic scheduling and production planning approaches. They promise exceptional results, but require detailed information or predictions about future parameters. In contrast DIQ works without any a-priori knowledge.

In fact, every process is different and has unique characteristics. DIQ therefore represents a framework that can be adapted on the specific circumstances of the application scenario: Individual decision functions might be defined that support the trade-off between the particular variables of the workflow comprehensively. Moreover, any classification technique might be chosen in order to cope the instance attributes from the process best. That way, universal applicability is guaranteed without neglecting the individuality of each workflow scenario.

DIQ allows the reduction of throughput times at critical activities. Applying DIQ might have implications on subsequent parts of the PAIS. Our algorithm permutes the order of the instances based on their similarity, which possibly influences the processing efficiency of resources that do not apply DIQ. These implications have not been part of this investigation, but are analyzed in [28].

The following assumptions have been made for the implementation of DIQ so far: (a) considering processes that contain at least one *activity* or *sequence*-workflow pattern at a critical activity (b) finishing times of the instances are not fixed (c) resource behavior is deterministic.

The case study and its results described in Section 4 refer to an ideal scenario with respect to the above assumptions, i.e., the process contains a sequence of activities, the finishing times of the jobs are not specified and the resources are machines whose behavior is deterministic (in opposite to, for example, a scenario where humans are involved). As we know from other case studies, e.g., in the health care domain [27], this kind of scenario can often be found in practice. However, the assumptions still leave room for further investigations.

(1) *Extension of workflow patterns:* DIQ is also applicable in the context of further workflow patterns such as parallelism. In the industrial domain, for instance, products are typically manufactured out of several components that are produced parallelly and are being joined just in the end (supply chain management). As Dynamic Instance Queuing relates to the instances involved rather than the specifications of the workflow, it offers the potential to improve the processing time in the context of other control-flow patterns as well.
(2) *Extension of time aspects:* The reduction of processing times is an asset especially for instances at critical activities. However, we presumed that the finishing time of the instances are not fixed. Eventually, it is even possible to fulfill deadlines that could not have been met with a different approach. For that matter, it would be profitable to include fixed finishing times for instances based on a prioritization mechanism. This scenario can also be found in Manufacturing Planning Systems.
(3) *Extension of resource behavior:* It cannot be assumed that the resource behavior is always deterministic. This is especially the case when work is done by humans, whose individuality cannot be formalized. Different humans may have a specific behavior, different specializations and strengths which may render them more or less suitable to performing certain activities. With respect to the current trend towards human-oriented PAIS [10], the behavior of human resources might be interesting in connection with optimization techniques such as queuing.
(4) *Improvement of classification techniques:* At the moment, DIQ is designed in a way that no a-priori definition is needed. However, with little pre-specification, many prospects occur: Instance properties relevant for the similarity could be explicitly defined, so that the classification is only applied on relevant attributes. Furthermore, in reality, not all instance properties have equal impact on the similarity measure. It might be promising to introduce variables to weigh these instance properties.

7. Related Work

The core idea of Dynamic Instance Queuing is that similar instances can be processed faster in a row by making use of optimizations such as caching or gaining of routine by human resources than randomly distributed instances. We consider our approach transcendent to several research topics (cf. Figure 14).

Queuing theory: Queuing theory is a vital research topic in mathematics. The latest models can cover a variety of factors including uncertainty. Most important parameters of any model are the average arrival times and processing times, the number of resources and the capacity. Little's theorem [18] allows a relation between the long-term average number of instances, the arrival rate, and the average sojourn time. Optimization of queues is basically achieved by estimating probability distributions of incoming elements or processing times. However, in PAIS, the prediction of parameters like arrival times is often impossible due to the dynamic environment

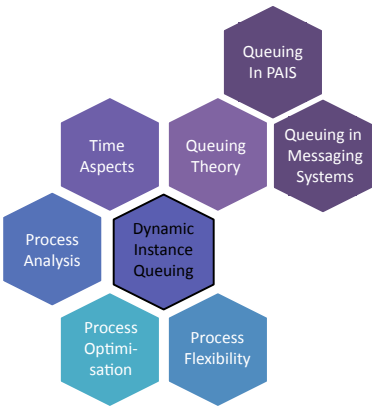

Fig. 14. Related research areas

the instances are executed in. Therefore adaptations of mathematical optimizations to PAIS are limited in ways of flexibility. However mathematical theorems have served a basis for DIQ.

Queuing in PAIS: Queuing is part of basically any PAIS, mostly implicitly, sometimes explicitly. Existing approaches, as described in [42], cover arising queues as a result of an imbalanced ratio between available resources and the number of process instances to be handled by the PAIS. Explicit queuing is addressed by Liu and Hu [19], who apply dynamic batch processing to Workflow Management Systems. A recent approach to model batch activities in BPMN is provided in [30]. Van der Aalst et al. [40], however, understand queues as a mean to handle escalations in PAIS. A similar idea to Dynamic Instance Queuing is pursued by combining *queuing with batching* [42,19]. However, first of all, batching requires that the resources are able to process batches, i.e., the parallel processing of a set of instances. Secondly, batching at one critical activity often results in batch-wise arrival of instances at preceding activities in the process that are not designed for batch processing. Hence, a reduction of processing time by batching at one critical activity might cause subsequent queues and subsequently lead to no reduction or even increase of the overall processing time. Therefore, in this paper, we adopt a *sequential* processing strategy rather than a batch-based one. More precisely, even though instances are grouped together based on similar features, they are not processed as batches but within separated queues. An approach for queue mining is proposed by [35]. Contrary to the runtime optimization idea of DIQ, queue mining features an ex-post analysis technique for predicting future delays in instance execution.

Queuing in Messaging Systems and Middleware: Kumar et al. [12] describe a novel self-adaptation algorithm that has been designed to scale efficiently for thousands of streams and aims to maximize the overall business utility attained from running middleware-based applications. In a subsequent work, Kumar [13] enhances

his approach by a dynamic element to react on the resources available ("resource awareness"). The approaches culminate in a distributed stream processing middleware that provides sharing-aware component composition [33]. In this approach, optimizations are achieved a-posteriori by implicitly evaluating each iteration. Our approach, however, strives to achieve optimizations during runtime. Regarding middleware systems, load balancing is a major topic. As load is represented by a queue of tasks, dynamic processing strategies are a possibility for optimization. An exemplary approach is presented by Drougas [7] as well as by [37]. Amini et al. [4] describe an algorithm that is designed to meet the challenges of extreme-scale stream processing systems, where over-provisioning is not an option, by making the best use of resources even when the proffered load is greater than available resources. This scenario is similar to the approach presented in this paper in the way that the need for DIQ also arises when a resource does not have the capacity to handle all instances in time. However, Dynamic Instance Queuing is a process oriented approach, while load balancing has kind of a static character by definition as it is a methodology that applies typically in multilayer architectures.

Time aspects in PAIS: Since time aspects constitute a major challenge for the modeling and the execution of business processes. Different approaches address this issue [8,9,16,17,24,34,29]. It has been investigated, for example, how to capture time aspects at design time, in particular, to cover uncertainty of processing times and to determine critical paths. At runtime, adherence of the process instances to imposed time restrictions such as deadlines is monitored. Escalation strategies [40] provide means to deal with violations of time restrictions and deadlines. All these questions become more challenging when considered for process choreographies [9]. An analysis of existing approaches based on time patterns can be found in [14]. The work presented in this paper does not directly relate to the above mentioned approaches. Since the presented approach reduces the processing time for all process instances within a certain time frame, it might be even counter-productive for handling possible deadline violations of single process instances, but is more suited for optimizing the processing time of a set of process instances being executed within the same time frame.

Process Analysis: Business process analysis ranges from model verification at design-time to the monitoring of processes at run-time [39]. Especially process verification has matured to a level where it can be used in practice over the last years. Especially petri nets seem to be a way to seize compliance, as initially shown in the approach of van der Aalst [38]. Subsumed the term "business process change", techniques to cover corporate transformations have been developed. In that context, business process reengineering is the task of adapting the business' workflows to the organization's needs. Process monitoring, however, is a task that is done during runtime. In DIQ, process monitoring is implemented within the state management component (Section 3.2).

Process Optimization: Techniques for optimizing business processes are of par-

ticular interest for business process re-engineering. The core challenge when optimizing business process is to find the redesign strategies. As described in [23], in practice, they are often engineered within an expert workshop. For different reasons it would be more beneficiary to provide strategies that can be applied in certain situations in order to support the user. In [32], an overview of existing approaches on best practices or heuristics for business process redesign is provided that address different optimization factors such as time, costs, quality, or flexibility. This paper narrows the optimization factors down to time (and as a side-effect probably costs). However, more factors can be included into the considerations. However, the main difference is that this approach enables an automatic optimization of throughput times at runtime that do not require any redesign measures.

Process Flexibility: Multiple research efforts have focused on introducing a higher degree of flexibility in processes. Reichert [31] characterizes flexible process support by four major flexibility needs, namely support for variability, looseness, adaptation, and evolution. Pesic et al. [25] and van der Aalst [26] address the evolution of processes and offer recommendations based on past experiences and additionally on a specific process goal. Approaches in which ontologies or semantic rules serve as a basis for the recommendations have been addressed as well [2]. A different strategy is the introduction of ad-hoc workflows. An ad-hoc workflow is a workflow in which deviations from the pre-defined process flow are allowed. Ad-hoc workflows have been investigated e.g. by [36]. In DIQ, compliance is considered an essential goal; so process deviations don't represent a mean to achieve a higher degree of flexibility. Instead, we aim to improve processing strategy by leaving it to the runtime.

8. Summary & Future Work

This paper centers on optimizing collaborative business processes by dynamically queuing process instances at critical activities during runtime. Critical activities are characterized by high changeover times and limited resources and might lead to an accumulation of waiting process instances during runtime. They tend to appear especially in systems with different human tasks involved whose collaborative interactions characterize the overall process. So far, approaches have tackled this phenomenon by bundling instances for batch execution based on static rules. DIQ exploits runtime information on arriving instances and bundles them based on those attributes that influence their processing at the critical activity, e.g., number of holes at a drilling machine or layout options at a printer. In this paper we extended the DIQ concept for single activities (DIQA) to DIQ on sequences patterns (DIQS). We discussed the implementation details on a DIQ component including its embedding into a PAIS. Further on, the DIQS concept was extensively evaluated based on a real-world case study from the manufacturing domain. The results showed that DIQS leads to a reduction of 13% of the throughput time when compared to a first-in-first-out processing strategy. In addition, we compared the application of DIQA and DIQS on the same case study where DIQS excelled DIQA by $7,89\%$.

As discussed in the paper, the DIQ approach offers several ways of extension. First of all, it has to be investigated how DIQ performs on more advanced process patterns such as parallel and alternative branchings as well as loops. Secondly, the mechanistic characteristics of the case studies conducted so far, i.e., printer and manufacturing, will be extended to considering processes with human interaction as well. We regard this as particularly interesting since critical manual activities are very likely as well, e.g., a full patient waiting room at a clinic.

Finally, we plan to integrate the DIQ component with a workflow engine, i.e., the Cloud Process Execution Engine CPEE (`cpee.org`) in order to test and demonstrate DIQ online, i.e., during instance execution. One real-world scenario would be a virtual factory setting such as considered within the EU FP7 project ADVENTURE (`http://www.fp7-adventure.eu/`).

ACKNOWLEDGEMENTS

This work was partially supported by the Commission of the European Union within the ADVENTURE FP7-ICT project (Grant agreement no. 285220).

References

1. T. Abeel, Y. V. de Peer, and Y. Saeys. Java-ml: A machine learning library. *Journal of Machine Learning Research*, 10:931–934, 2009.
2. T. Almeida, S. Vieira, and M. A. Casanova. Flexible workflow execution through an ontology-based approach. In *Workshop on Ontologies as Software Engineering Artifacts*, 2004.
3. G. Alonso, F. Casati, H. Kuno, and V. Machiraju. *Web Services: Concepts, Architectures and Applications*. Springer, Berlin, 2004.
4. L. Amini, N. Jain, A. Sehgal, J. Silber, and O. Verscheure. Adaptive control of extreme-scale stream processing systems. In *In ICDCS 2006*, pages 71–79, 2006.
5. K. Anderson. Web services and related technologies, 2006.
6. R. Bobrik, M. Reichert, and T. Bauer. View-based process visualization. In *Business Process Management*, volume 4714 of *LNCS*, pages 88–95. Springer Berlin Heidelberg, 2007.
7. Y. Drougas, T. Repantis, and V. Kalogeraki. Load balancing techniques for distributed stream processing applications in overlay environments. In *Int'l Symp. on Object and Component-Oriented Real-Time Distributed Comp.*, pages 33–42, 2006.
8. J. Eder, E. Panagos, and M. Rabinovich. Time constraints in workflow systems. In *Int'l Conf. on Advanced Information Systems Engineering*, pages 286–300, 1999.
9. J. Eder and A. Tahamtan. Temporal conformance of federated choreographies. In *Database and Expert Systems Applications*, pages 668–675. Springer-Verlag, 2008.
10. S. Kabicher-Fuchs and S. Rinderle-Ma. Work experience in PAIS - concepts, measurements and potentials. In *Int'l Conf on Advanced Information Systems Engineering*, pages 678–694, 2012.
11. I. Konyen, B. Schultheiss, and M. Reichert. Design of a laboratory process. Technical report, University of Ulm, 1996.
12. V. Kumar, B. Cooper, and K. Schwan. Distributed stream management using utility-driven self-adaptivemiddleware. In *Int'l Conf. on Autonomic Computing*, pages 3–14, 2005.

13. V. Kumar, B. F. Cooper, Z. Cai, G. Eisenhauer, and K. Schwan. Resource-aware distributed stream management using dynamic overlays. In *Proc. Int'l Conf. on Distributed Computing Systems*, pages 783–792, 2005.
14. A. Lanz, B. Weber, and M. Reichert. Workflow time patterns for Process-Aware information systems. In *Enterprise, Business-Process and Information Systems Modeling*, volume 50, pages 94–107. Springer Berlin Heidelberg, 2010.
15. M. Leitner, J. Mangler, and S. Rinderle-Ma. Definition and enactment of instance-spanning process constraints. In *Web Information Systems Engineering - WISE 2012*, pages 652–658, 2012.
16. H. Li and Y. Yang. Dynamic checking of temporal constraints for concurrent workflows. *Electronic Commerce Research and Applications*, 4(2):124–142, 2005.
17. J. Li, Y. Fan, and M. Zhou. Timing constraint workflow nets for workflow analysis. *IEEE Trans. on Systems, Man, and Cybernetics*, 33(2):179–193, Mar 2003.
18. J. D. C. Little. A proof for the queuing formula: L= λw. *Operations Research*, 9(3):383–387, 1961.
19. J. Liu and J. Hu. Dynamic batch processing in workflows: Model and implementation. *Future Generation Computer Systems*, 23(3):338 – 347, 2007.
20. J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Berkeley Symposium on Mathematical Statistics and Probability*, pages 281–297, 1967.
21. J. Mangler and S. Rinderle-Ma. Rule-based synchronization of process activities. In *2011 IEEE 13th Conference on Commerce and Enterprise Computing(CEC)*, pages 121–128. IEEE, 2011.
22. S. L. Mansar and H. A. Reijers. Best practices in business process redesign: Validation of a redesign framework. *Computers in Industry*, 56(5):457–471, 2005.
23. M. Netjes, I. Vanderfeesten, and H. Reijers. "intelligent" tools for workflow process redesign: A research agenda. In *Business Process Management Workshops*, volume 3812, pages 444–453. Springer Berlin Heidelberg, 2006.
24. Y. Pan, Y. Tang, H. Ma, and N. Tang. Workflow analysis based on fuzzy temporal workflow nets. In *Computer Supported Cooperative Work in Design II*, volume 3865, pages 545–553. Springer Berlin Heidelberg, 2006.
25. M. Pesic and W. Aalst. A declarative approach for flexible business processes management. In *Business Process Management Workshops*, volume 4103, pages 169–180. Springer Berlin Heidelberg, 2006.
26. M. Pesic, M. Schonenberg, N. Sidorova, and W. Aalst. Constraint-based workflow models: Change made easy. In *On the Move to Meaningful Internet Systems 2007: CoopIS, DOA, ODBASE, GADA, and IS*, volume 4803, pages 77–94. Springer Berlin Heidelberg, 2007.
27. J. Pflug and S. Rinderle-Ma. Dynamic instance queuing in process-aware information systems. In *Proc. 28th Annual ACM Symposium on Applied Computing (SAC '13)*, pages 1426–1433, 2013.
28. J. Pflug and S. Rinderle-Ma. Analyzing the effects of reordering work list items for selected control flow patterns. In *IEEE 19th International Enterprise Distributed Object Computing Workshop*, pages 14–23, September 2015.
29. H. Pichler, M. Wenger, and J. Eder. Composing Time-Aware web service orchestrations. In *Advanced Information Systems Engineering*, pages 349–363, 2009.
30. L. Pufahl and M. Weske. Batch activities in process modeling and execution. In *Service-Oriented Computing*, pages 283–297. Springer, 2013.
31. M. Reichert and B. Weber. *Enabling Flexibility in Process-Aware Information Systems: Challenges, Methods, Technologies*. Springer, Berlin-Heidelberg, 2012.

32. H. A. Reijers and S. Liman Mansar. Best practices in business process redesign: an overview and qualitativeevaluation of successful redesign heuristics. *Omega*, 33(4):283–306, 2005.
33. T. Repantis, X. Gu, and V. Kalogeraki. Synergy: Sharingaware component composition for distributed streamprocessing systems. In *Middleware*, pages 322–341, 2006.
34. S. Sadiq, O. Marjanovic, and M. Orlowska. Managing change and time in dynamic workflow processes. *IJCIS*, 9(1&2):93–116, 2000.
35. A. Senderovich, M. Weidlich, A. Gal, and A. Mandelbaum. Queue mining for delay prediction in multi-class service processes. *Inf. Syst.*, 53:278–295, 2015.
36. T. Stoitsev, S. Scheidl, and M. Spahn. A framework for light-weight composition and management of ad-hoc business processes. In *Task Models and Diagrams for User Interface Design*, volume 4849 of *LNCS*, pages 213–226. Springer Berlin Heidelberg, 2007.
37. R. Strom, C. Dorai, G. Buttner, and Y. Li. Smile: distributed middleware for event stream processing. In *Int'l Conf. on Information processing in sensor networks*, pages 553–554. ACM, 2007.
38. W. van der Aalst. Verification of workflow nets. In *Application and Theory of Petri Nets 1997*, volume 1248 of *LNCS*, pages 407–426. Springer Berlin Heidelberg, 1997.
39. W. van der Aalst. Challenges in business process analysis. In *Enterprise Information Systems*, volume 12 of *LNCS*, pages 27–42. Springer Berlin Heidelberg, 2009.
40. W. van der Aalst, M. Rosemann, and M. Dumas. Deadline-based escalation in process-aware information systems. *Decision Support Syst.*, 43(2):492–511, 2007.
41. W. van der Aalst, A. ter Hofstede, B. Kiepuszewski, and A. Barros. Workflow patterns. *Distributed and Parallel Databases*, 14(1):5–51, 2003.
42. W. van der Aalst and K. van Hee. *Workflow Management*. MIT Press, 2002.