Context-Aware Querying and Injection of Process Fragments in Process-Aware Information Systems

Klaus Kammerer
Institute of Databases and Information Systems
Ulm University, Germany
klaus.kammerer@uni-ulm.de

Rüdiger Pryss
Institute of Clinical Epidemiology and Biometry
University of Würzburg, Germany
ruediger.pryss@uni-wuerzburg.de

Manfred Reichert
Institute of Databases and Information Systems
Ulm University, Germany
manfred.reichert@uni-ulm.de

***Abstract*—Cyber-physical systems (CPS) are often customized to meet customer needs and, hence, exhibit a large number of hard-/software configuration variants. Consequently, the processes deployed on a CPS need to be configured to the respective CPS variant. This includes both configuration at design time (i.e., before deploying the implemented processes on the CPS) and runtime configuration taking the current context of the CPS into account. Such runtime process configuration is by far not trivial, e.g., alternative process fragments may have to be selected at certain points during process execution of which one fragment is then dynamically applied to the process at hand. Contemporary approaches focus on the design time configuration of processes, while neglecting runtime configuration to cope with process variability. In this paper, a generic approach enabling context-aware process configuration at runtime is presented. With the Process Query Language process fragments can be flexibly selected from a process repository, and then be dynamically injected into running process instances depending on the respective contextual situations. The latter can be automatically derived from context factors, e.g., sensor data or configuration parameters of the given CPS. Altogether, the presented approach allows for a flexible configuration and late composition of process instances at runtime, as required in many application domains and scenarios.**

***Index Terms*—context-awareness, process injection, dynamic process change**

I. Introduction

In Industry 4.0 and mass customization scenarios, the management of cyber-physical processes is a mainstay for successful companies [1]. The involved processes are exposed to a multitude of external and internal factors, which vary from scenario to scenario [2]. One prerequisite to cope with this variability is the flexible support of process variants. The latter are managed by either integrating all possible variants into one model (i.e. *single-model approach*) or by storing each process variant in a dedicated process model (i.e. *multi-model approach*) [3]. Moreover, process variants need to be managed throughout the process lifecycle by a process-aware information system. For example, if the state of a cyber-physical system (e.g., a production machine) changes due to an exceptional situation [4], running process instances have to be adapted accordingly. In general, such behavioral changes, which often cannot be foreseen at the start of a process instance, can be categorized into *ad-hoc changes*, which occur unexpectedly and pose the peculiarity that the affected process parts are not known in advance, and *late modeling* which is predictable to a certain degree (i.e., the region concerned by the structural change is known, but it is not known whether further changes become necessary) [5]. As example consider a production machine line, for which maintenance processes are defined based on standard operating procedures (SOPs) and maintenance instructions. Due to the current status of a particular machine, however, the required tasks and their sequence, i.e., the process steps to be performed by a service technician, cannot be defined in advance for all potential scenarios, i.e., further changes might become necessary.

To provide support in such scenarios, the context-aware process injection approach (CaPI), which we introduced in [6], is enhanced to enable dynamic process configurations and changes as well. CaPI manages the injection of process fragments into running processes based on a set of rules [6]. Note that such approach is pursued by related works as well [7–9]. However, in large-scale scenarios as faced by CPS, a rule-based approach for selecting the process fragments to be injected is often not feasible. In this paper, we enhance CaPI with the declarative process query language PQL [10]. It is shown that PQL enables the selection of process fragments from a process repository, followed by their injection into a running process. This allows for dynamic process extensions in changing situations in particular, and offers a powerful way to model and manage process variants in dynamic environments in general. Based on a sophisticated proof-of-concept prototype, the feasibility of the approach is demonstrated and performance measurements are provided. Experimental results show that the retrieval of process fragments with PQL can be efficiently accomplished, even if a large number of PQL queries are executed. Although the approach is inspired by Industry 4.0 scenarios and cyber-physical processes [11], it may be applied to handle process variants in dynamic environments in general.

Sect. II introduces PQL fundamentals, while Sect. III introduces the Context-aware Process Execution (CaPE) framework for modeling and executing cyber-physical processes. In Sect. IV, the concepts for context-specific process adaptations as well as dynamic selections of process fragments based on PQL are described. Sect. V presents experimental results and Sect. VI discusses related work. Finally, Sect. VII provides a

summary and an outlook on future work.

II. Process Querying with the PQL Language

The Process Query Language (PQL) is a query language that allows describing process model properties, process model abstractions, and process model changes in a declarative way [10]. PQL statements may be applied to a single process model or to a collection of process models with common properties. In general, PQL enables the selection of process instances as well as the creation of process model abstractions. Declarative descriptions of any selection, abstraction, or change of a process model (collection) are denoted as *PQL requests*. A PQL request, in turn, consists of two sections: the *selection section* specifies an expression for selecting the respective process models, whereas the *modification section* defines the abstractions and changes to be applied to the selected process models. This paper focuses on the selection of process models.

Fig. 1 illustrates the processing of a PQL request, which is triggered by an authorized user sending a PQL request to the *PQL interpreter* (Step ①). Then, all process models that match the predicates specified in the selection section of the PQL request are selected and retrieved from the process repository (Step ②). Moreover, PQL enables the modification of the selected process models (Step ③) and their abstractions (Step ④)—the latter two steps are not considered in this paper. Finally, all selected process models are presented to the user (Step ⑤). An example of a PQL request is depicted in Listing 1.

```
1 MATCH  a1:ACTIVITY-[:ET_Control]->a2:ACTIVITY
2        -[:ET_Control]->a3:ACTIVITY
3 RETURN a3
```

Listing 1. Example PQL Request

Line 1 refers to the selection of all process models with a path (i.e., a sequence of edges with type *ET_Control*) containing activities $a1$, $a2$, and $a3$. Note that $a1$, $a2$, and $a3$ are only variables (i.e. placeholders), i.e., the PQL request searches for all process models comprising any sequence consisting of three activities (cf. Lines 1+2) and returns only direct successors of $a2$ (cf. Line 3). Consider the example shown in Fig. 2. When applying the PQL request to it, activity G is returned as the only possible match. Interested readers can find more information on PQL in [10].

III. Context-aware Process Execution

To support cyber-physical processes, the Context-aware Process Execution (CaPE) framework was developed. It enables the runtime integration of sensor data on physical objects (e.g., machines) and the creation of a semantic network to manage data that may influence the execution of cyber-physical processes, e.g., the physical structure and runtime states of a machine. Data of the semantic network allows evolving loosely coupled processes based on context-specific process injections [6]. The CaPE Framework comprises three technical pillars: *CaPE Sensor*, *CaPE Context*, and *CaPE Process* (cf. Fig. 3).

CaPE Sensor provides features for connecting sensors, acquiring raw sensor data, transforming and normalizing the sensor data, and analyzing, storing, and forwarding the derived events from the sensor data. **CaPE Context**, in turn, was specifically developed with the requirements of cyber-physical systems (CPS) in mind. A semantic network is used to map the physical structure of a CPS to a digital shadow [11]. **CaPE Context** allows for a standardized representation of a CPS. In particular, the state of a CPS is represented independent from the respective communication and data structures.

CaPE Process enables runtime process flexibility based on context-aware process injections [6]. The key objective of CaPI is to ease the modeling of a collection of process variants (i.e. a *process family*) at design time and to enable context-aware process variant configuration during runtime. In essence, by taking the current context of a process (e.g., the runtime state of a machine) into account, CaPI enables the context-driven injection (i.e., insertion) of process fragments into a lean base process in a controlled manner.

We illustrate the CaPE approach along a production-process run on a production machine. First, required production data is received and the production steps and their order are determined. Following this, raw materials are prepared, the actual production process is performed, and the machine is manually cleaned (cf. base process in Fig. 4g). Production machines usually require specific format parts to manufacture a product (e.g., injection molds are needed to produce plastic parts). The following situation might occur during process execution (cf. Fig. 4e): by evaluating context factors CF1 and CF2 (cf. Fig. 4a), mapping rules (cf. Fig. 4c) determine whether a required format part is missing. If this is the case, contextual situation *CS_FormatPartsMissing* becomes activated (cf. Fig. 4e). When reaching extension area *EA1* (cf. Fig. 4h) process fragment "PF_OrderFormPrts" is inserted into the base process instance (cf. Fig. 4fj). Another situation that might occur recalibrates a machine if the machine configuration does not meet the production specifications (cf. Fig. 4).

During process execution, context factors are used to determine whether pre-specified contextual situations have occurred. If this applies, process fragments are inserted based on predefined injection specifications linked to the respective contextual situation.

Following the *separation of concerns* principle, the *base process model* solely contains the decisions (i.e., branches and gateways) and activities common to all variants of the process. In particular, these activities need to be known at build time and must not be changed during runtime. By contrast, *extension areas* represent the dynamic (i.e. varying) parts of the process. Accordingly, process modelers may first focus on the modeling of the predictable parts of the base process and its extension areas, and then add the varying process fragments to this extension areas. The latter enables context-specific injections of process fragments into the base process at runtime based on well-defined *injection specifications*. Moreover, an extension area allows for the dynamic injection of any number of same or different process fragments organized in parallel. In turn, contextual situations are defined through conditions expressed in first-order logic, referring to *process parameters*

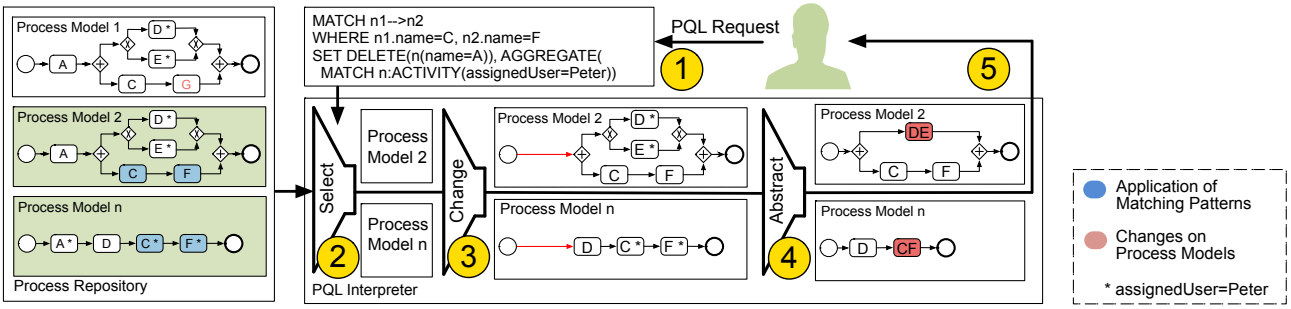

Fig. 1. Processing a PQL Request

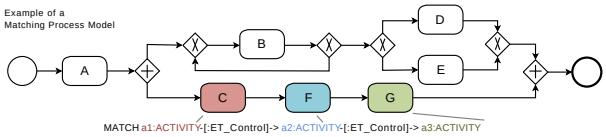

Fig. 2. PQL Request Determining a Sequence of three Activities

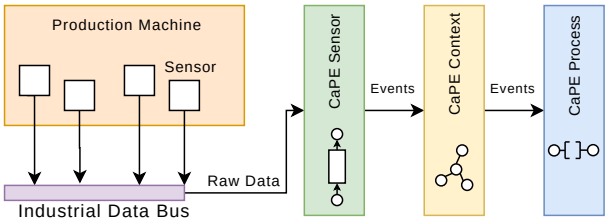

Fig. 3. Overview of the CaPE Framework

as well as data objects of the base process model. Process parameters may be linked to external factors (e.g. availability of a resource) that influence the concrete process injection chosen. When injecting process fragments, CaPI also takes care of the data mapping, i.e., data objects of an injected process fragment are automatically connected to existing data objects of the base process.

Altogether, CaPI enables dynamic configurations and changes of varying processes in a controlled way during runtime. By solely enabling insertions of process fragments, CaPI allows process modelers to focus on the commonalities of all variants (base process) as well as the varying process parts, instead of creating a single complex process model that captures all variants. Process modelers may directly integrate contextual data with the modeling of variants. For this purpose, external context factors may be abstracted through process parameters. CaPI is able to cope with context-driven runtime injections based on the late evaluation of contextual parameters when reaching predefined extension areas. Moreover, a consistent data flow between the newly injected process fragments and the base process is ensured.

IV. Context-aware Selection of Process Fragments with CaPE and PQL

To utilize context for appropriately evolving a process, on one hand, contextual factors need to be made available to the process. On the other, the required actions need to be modeled. CaPE Process distinguishes between two types of actions: (1) the change of a process execution state, e.g., by starting or stopping a process instance, and (2) the adaptation of the schema of a process instance. *Context factors* as well as the two change types are introduced below.

A. Context Factors

The major artifact of *CaPE Process* is the *Context-aware Process Family* (CPF), which consists of a base process model, process parameters (cf. Sect. III), change specifications (cf. Sect. IV-C and IV-D), and context factors that describe external context. Context factors shall express the contextual situations and serve as an immutable interface to the respective process context. Context factors can be regarded as data objects having a type and a value. The latter are set, for example, when receiving specific events.

Definition 1 (Context Factor): A *context factor* is defined as a tuple $C = (t, v, v_d)$, where:

- t is the type (i.e., Boolean, Integer, Float, Double, String, Complex, or user-defined) of the context factor.
- v is the value of the context factor corresponding to t.
- v_d is the default value initially set for v.

Context factors decouple the mapping of contextual situations from the modeling of a Context-aware Process Family (CPF). Consequently, a modeled CPF needs not be changed if a context model changes. During CPF execution, context factors are mapped to process parameters based on mapping rules. When reaching an extension area, process parameters are checked against conditions defined in contextual situations, which are explained in the following.

B. Context Evaluation with Contextual Situations

A process variant may rely on a set of occurring contextual situations based on the combination of process parameters, i.e. their current values. Contextual situations are defined by conditions expressed in a first-order logic relying on the set of process parameters. Contextual situations can have

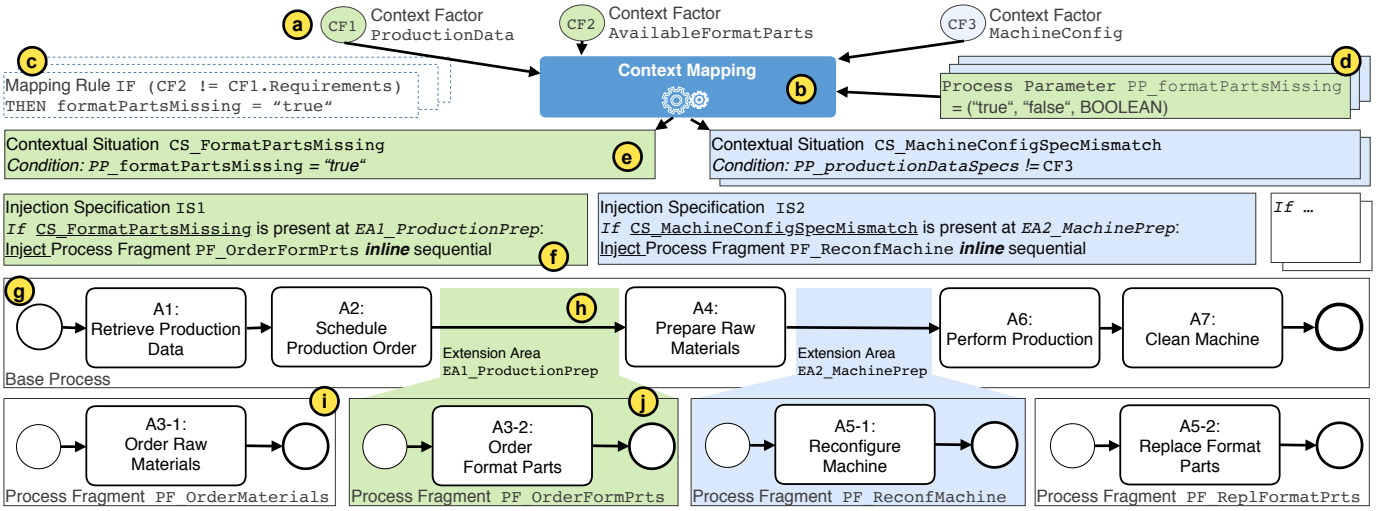

Fig. 4. Illustration of a Context-aware Process Family

different states, i.e., `watched`, `accepted`, `triggered`, `finished`, and `rejected` (cf. Fig. 5). When executing a CPF, first of all, contextual situations are assigned to state `watched`. If a condition is evaluated to `true` (e.g., changing the value of a context factor to which this refers), state `accepted` is set for the contextual situation. In this state, all change specifications defined for a contextual situation are applied to the base process. Two types of change specifications are distinguished: *Execution specifications* allow creating new process instances or changing the execution state of a running one (cf. Sect. IV-C). In turn, *injection specifications* define the process fragments to be injected into extension areas based on a PQL query (cf. Sect. IV-D). Execution specifications are executed immediately when a situation becomes activated. In turn, injection specifications are executed when an extension area referring to the injection is reached during process execution, i.e., when setting the state of the contextual situation to `triggered`. In the latter case, the predefined injections are applied, and the contextual situation enters state `finished`. A contextual situation may be set to state `rejected` at any time, either manually or when it turns out that the associated evaluation condition can never evaluate to `true`. Change specifications are assigned priorities to determine the execution order at the presence of multiple change specifications. Hence, a specification with a higher defined priority is executed prior to one with a lower priority.

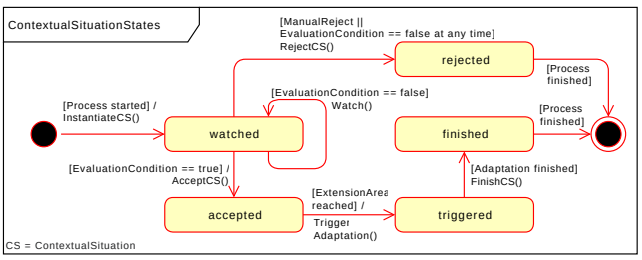

Fig. 5. UML State Diagram of Contextual Situation States

C. Modifying Process States with Execution Specifications

The execution state of a process instance may be adjusted if a contextual situation is in state `activated`. Corresponding execution specifications can be defined for contextual situations, and *start* new process instances or *suspend*, *abort*, or *terminate* running process instances as well as *resume* suspended process instances. When aborting a process instance, a pre-specified compensation process is executed. Finally, a termination leads to an immediate stop of an instance without compensation.

Execution specifications include various parameters. *ExecutionType* expresses the kind of instance change (i.e., START, SUSPEND, RESUME, ABORT, TERMINATE), and *ExecutionQuery* specifies the PQL query to be executed. Moreover, Parameter *Singleton* defines whether only exactly one instance of a process model may be running (*Singleton* = *'true'*). In this case no further instance is started. *ExecutionRate* defines the number of instances that may be started and *ExecutionTrigger* the point in time at which *ExecutionSpecification* shall be applied, e.g., immediately or deferred by a timer. If multiple process models are selected in a PQL query, *QueryStrategy* is used to specify whether the execution specification is to be applied to all selected models (*'multiple'*), only to the last updated model (*'single_newest'*), the process model with the highest priority defined in its attributes (*'single_prio'*), or a manually selected process model (*'single_manual'*).

Example 1 (Instantiation of an Order Process within a Production Process): Fig. 6 shows the example of a CPF that starts an order process if signs of wear and tear are detected on a machine part. The wearing status is represented by context factor "CF1" (cf. Fig. 6a) and the context mapping defines a process parameter "PP_wearalert", which is set to `true` if the value of "CF1" is 'true' (cf. Figs. 6b-d). Contextual situation "CS_WearingPartOrder" defines conditions setting the

contextual situation to `accepted` if the value of process parameter "PP_wearAlert" is set to `true` (cf. Fig. 6e). If the contextual situation becomes activated, execution specification "ES_1" is applied by starting *exactly one* instance of the newest process model (cf. parameters *Singleton* and *QueryStrategy*) selected by PQL query "PQ_1" (cf. Fig. 6f). In the example, PQL query "PQ_1" is executed on a process repository (cf. Fig. 6g) returning the newest process model including "Order Part BallBearing" (cf. Fig. 6h) in its name.

D. Context-aware Extensions with Injection Specifications

Injection specifications allow adding a process fragment to an extension area of a base process in a given contextual situation (cf. green elements in Fig. 6). An injection is based on three components: a contextual situation, an injection specification, and a PQL query.

An injection specification is assigned to a contextual situation. It describes the possible injections of a process model in terms of change operations and refers to the extension area to which the injection shall be applied. Finally, *InjectionQuery* defines the PQL query for retrieving the process to be injected from a process repository. *QueryStrategy* determines whether all process fragments selected by the PQL query are injected (*multiple*), or only the newest (*single_newest*), the most prioritized (*single_prio*), or whether one is to be selected manually (*single_manual*, cf. Sect. IV-C). Parameter *InjectionType* defines how a process fragment is inserted into an extension area, i.e., inline or as sub-process. Parameter *InjectionPattern* denotes whether a process fragment is injected in parallel or sequentially into the respective extension area. Furthermore, *InjectionRate* denotes whether a process fragment is injected once or multiple times into the extension area. Finally, *InjectionTrigger* determines the point in time an injection shall be triggered. In general, injection specifications and PQL queries may be used by several *Contextual Situations* within a CPF.

Example 2 (Injection of a Printer Installation Process Fragment): Fig. 6 shows the CPF of a production process running on a machine, in which a decision is made based on context factor "DeviceStatus" (cf. Fig. 6a). Specifically, it is decided whether to install and calibrate a printer, which is required for a production run. The contextual situation "CS_PrinterRequired" will be triggered if the production specifications require a printer (cf. Fig. 6i). When reaching Extension Area EA1_ProductionPrep (cf. Fig. 6l), the injection specification IS_1 is evaluated (cf. Fig. 6j). In turn, this triggers the execution of PQL query PQ_2 (cf. Fig. 6k) returning all processes from the repository that contain activity "Install Printer printerName" directly succeeded by activity "Calibrate Printer printerName". Note that "printerName" is a process parameter being replaced by the real name of the printer. In this example, "Install_and_Calibrate_Printer_Delco1" is inserted into the extension area of the running base process (cf. Fig. 6m). Note that a process fragment may contain extension areas and react to contextual situations as well (cf. Fig. 6n).

V. Evaluation

Motivation. In practice, a process repository may contain thousands of process models. Thus, it is crucial that PQL queries can be executed efficiently even on large process repositories. Compared to single-model approaches, the presented approach requires additional runtime, as process models are dynamically assembled at runtime. To demonstrate the performance of PQL-based injections, we developed a sophisticated prototype, that allows retrieving process fragments from a process repository with a PQL query and adding them to an extension area of a CPF. To be more precise, a PQL query is required for each injection specification.

Data Set. To evaluate the performance of the PQL query processes, we use process models from the process model matching contest (PMMC) dataset [12]. The latter contains process models and different variants manually created from them. We use the *heavy revision* data set that comprises 150 process models, where all process activity labels are rephrased to ease the search for keywords. The data set includes process models stemming from various domains: administration (27), booking (10), insurance (3), manufacturing (7), medicine (11), order processing (50), quality assurance (7), support & service (13), and academia (15). Due to quality problems and duplicates, we removed 7 process models from the data set. The process models following the single-model approach are manually split up by us into base process models and associated process fragments. We integrate those process elements into a base process that are common to all variants of a process. Process elements not common to all variants are shifted into separate process fragments. Furthermore, contextual situations and PQL-based injection specifications are created for every variant. We excluded 3 process models that feature no execution variants, i.e., they neither contain exclusive gateways nor do they show any variability. All other process models remain in the data that we use for our performance measurements experiments.

Prototype. The prototype architecture comprises the CPF execution component *CaPE Process*, a PQL interpretation component, and a Neo4j database (cf. Fig. 7). The already existing prototype that implements CaPI is extended with the PQL query processor [13]. Process models enacted by *CaPE Process* are stored in a Neo4j database–activities, gateways and data elements are stored as Neo4j nodes and process control flows as Neo4j edges. The Neo4j database serves as an in-memory index structure based on which efficient queries can be performed with the Cypher Query Language (CQL). When a contextual situation becomes activated, the PQL query of an injection/execution specification is translated by the interpretation component into a CQL expression and executed in Neo4j. Any CQL query returns a set of process models for which appropriate process fragments from the CaPE repository

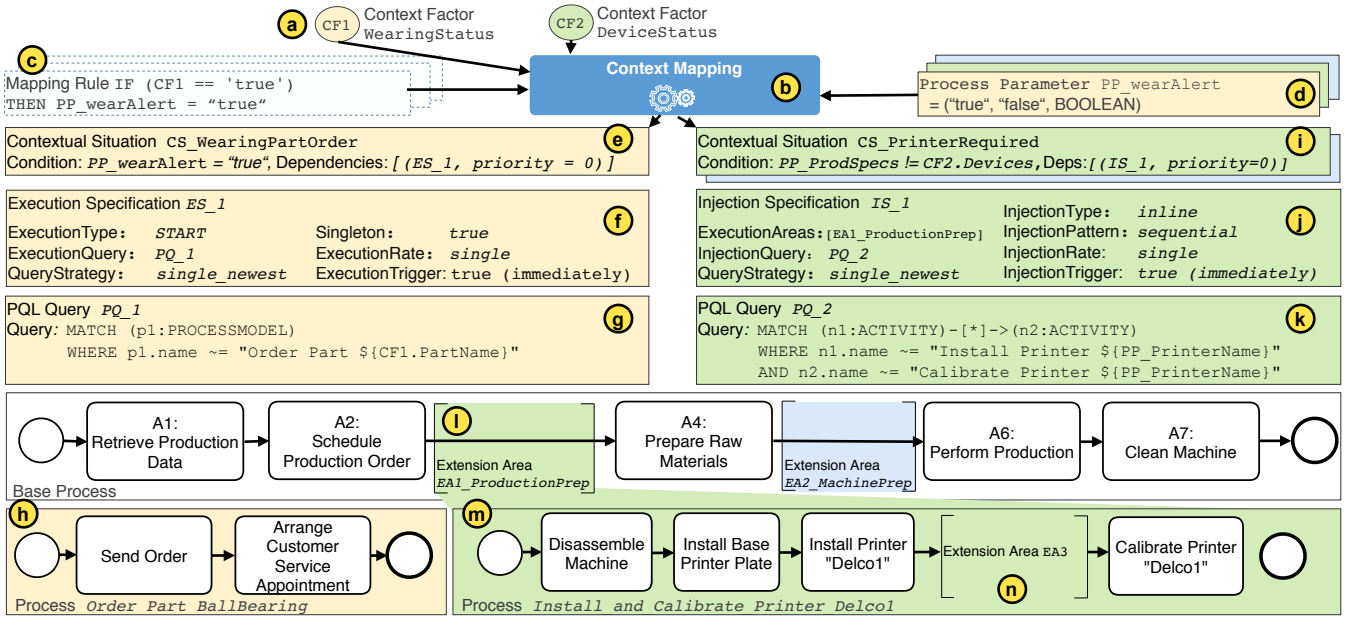

Fig. 6. Example of Change Specifications with PQL

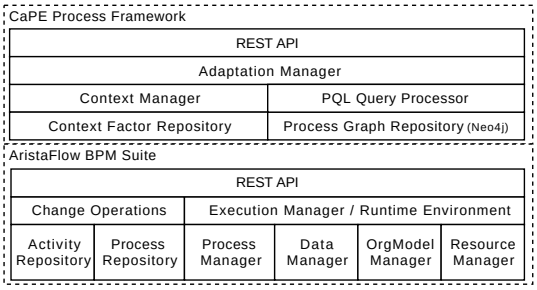

Fig. 7. CaPE Process Architecture

can be selected and, for example, be injected into an extension area.

PQL Query Definition. Various PQL queries are specified for the performance measurements. On one hand, simple queries are used for 18 injection specifications, i.e. the latter are simply selected based on an exact match of the label of a process fragment (cf. Listing 2). On the other, more complex PQL queries are defined for 25 further injection specifications (cf. Listing 3). 10 PQL queries include a search with variable path length between the individual nodes, i.e., these nodes need not be direct successors.

```
1 MATCH a:ProcessModel
2 WHERE a.name="(p11)payment receipt process1010(HR)"
```

Listing 2. PQL Exact Match Query Example

After conducting performance measurements with 140 process models, we duplicate them to a total of 648, 1193, 11003, 16453, and 21903 models and extend all identifiers and designators with a random UUID suffix to avoid collisions.

```
1 MATCH a1:ACTIVITY-[:ET\_Control]->a2:ACTIVITY
2 -[:ET\_Control]->a3:ACTIVITY
3 WHERE not (a1-[:ET\_Control]->a3) AND a1.name CONTAINS
4 "send feedback" AND a3.name CONTAINS "determine feedback"
```

Listing 3. PQL Control Flow Query Example

Results. The Neo4j database contained 3231 nodes (one per process model, activity, or gateway) and 13366 edges. Two process fragments are selected for 30 base processes in 62.5% of the process instances, and exactly one process fragment for 37.5% of the process instances. After 20 runs on a standard laptop with 16GB RAM and an Intel I7-6700HQ, the average selection time of the *exact model name* PQL queries is less than 2.446ms, whereas PQL queries with control flow selections take on average 4.72ms. After model duplication, the subsequent execution of the control flow based PQL queries takes 15.43ms (648 models), 23.99ms (1193 models), 201.56ms (11003 models), 310.30ms (16453 models), 404.81ms (21903 models) on average for 20 runs (cf. Fig. 8). Execution time of exact matching PQL queries takes 2.22ms (648 models), 2.15ms (1193 models), 2.33ms (11003 models), 1.91ms (16453 models), and 1.87ms (21903 models).

Discussion. Execution time of exact matching PQL queries remains below 2.33ms, which is enabled by node label indexes in Neo4j. Furthermore, execution time of control flow based PQL queries increases linearly with the number of nodes stored in the database. PQL execution time remains at 404.81ms for a process repository size of 21009 process models, which allows for a fast injection of process fragments even for larger process repositories. Performance measurements with CQL and Neo4j show similar results [14].

Threats to validity include the total number of process fragments injected and the total number of process models stored in the database. If many injection specifications are

defined for a base process, selection time increases. Furthermore, a Spring Data Neo4j implementation is used [15]–the measurements include PQL interpretation time, transmission delay between Spring implementation and Neo4j database, database query, and Spring Data object-relational mapper conversion time. No optimizations except Btree indexes on node IDs and node labels were performed.

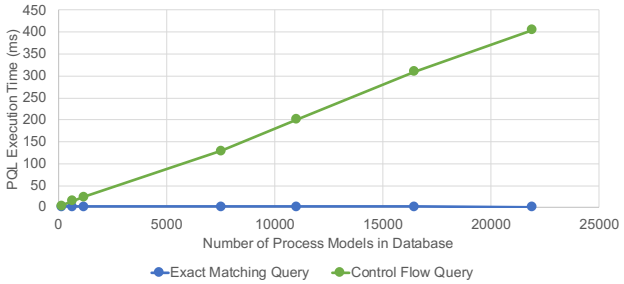

Fig. 8. PQL Query Duration for different Process Repository Sizes

In summary:

- The declarative language PQL is used for the dynamic selection of the process fragments to be injected into running processes.
- A proof-of-concept prototype enables the practical use of PQL.
- Experimental results show that the PQL-based retrieval of process fragments from a process repository can be accomplished in an efficient manner.

The limitations of the approach are as follows:

- The approach allows retrieving process fragments that may be added to an extension area of a process, where parameter *InjectionStrategy* determines the permitted quantity and the selection strategy of selected fragments (cf. Sect. IV-D). However, further sophisticated selection strategies are missing.
- If no suitable process fragments can be retrieved for a PQL query, selection must be accomplished manually.

VI. Related Work

Several approaches for coping with process variability based on late selection and dynamic process changes exist. Overviews of methods enabling process variability can be found in [3,16,17].

Worklets present a rule-based approach enabling late process fragment selection [7]. Each activity of a process is associated with a set of process fragments out of which one is selected at runtime based on the evaluation of complex rules. Similarly, [18] proposes a goal-based late selection approach, which defines alternative activities for different Quality of Service process goals. At runtime, one of the alternatives is automatically selected, according to the actual goals of the process.

Various approaches for the late selection of activity implementations exist [19–21]. These allow dynamically selecting suitable services from a service repository when executing activities. Moreover, the dynamic assignment of resources in a distributed grid execution environment is described in [22]. The approach allows for the dynamic selection and linkage of web services to activities of a process instance. Thereby, web service calls are selected without changing the schema of a process instance rather than allowing for injections of process fragments. Other approaches for dynamically assigning resources in distributed environments can be found in [23,24]. [25] deals with data-aware interactions of distributed and collaborative workflows.

[26] proposes an object-oriented process modeling language that enables the event-driven selection of process components during runtime. In turn, [27] proposes automated workflow adaptions based on case-based reasoning. The core concept is a powerful approach to automatically evaluate the execution point in time of workflow changes. As opposed to CaPI, no pre-specified extension areas exist for process instances, but the time of a change may be expressed with rules. Process instance changes are defined in terms of change operations instead of adding pre-specified process fragments. PHILharmonicFlows enables ad-hoc changes to data-centric processes and provides a scalable and flexible approach for this [28,29].

An extension of late selection is late modeling: instead of choosing from a set of predefined process fragments or activity implementations, fragments themselves may be dynamically modeled or composed during runtime, e.g., Pockets of Flexibility [30] allow modeling loosely specified processes, in which placeholder activities are replaced by process fragments at runtime. There also exist language-based approaches [31], which follow different goals than our work. Due to the late selection of process fragments and multi-model process compositions, our approach is able to consider every process fragment as a self-contained process.

[32] proposes a framework to conceive process querying methods and gives an overview of existing languages. The framework describes methods and concepts for managing large process repositories, including the optimization of queries (”prepare”), the optimal execution of queries (”execute”), the formalization and simulation of queries (”model, simulate, record and correlate”), and the interpretation of query results (”interpret”). Many query languages are based on structural descriptions, including BP-QL [33], BPMN-Q [34], FNet [35], process matching [36], DMQL [37] and PQL [10]. The BPMN-Q framework stores process models in relational databases [38]. BPMN-Q queries can be modeled graphically and then converted into SQL scripts using a query processor. However, BPMN-Q queries cannot be executed as efficiently on graph-based process models as in the CaPE approach. PQL is based on the Cypher Query Language for querying graph databases and, thus, enables efficient querying. Note that PQL is the only query language that allows for changes to process models.

VII. Summary & Outlook

In the physical world, the integration of different systems and the support of cyber-physical processes are both crucial.

The CaPE framework allows integrating sensor data into cyber-physical systems and building a semantic network used to map the physical structure of a CPS to a digital shadow. The concept of context-aware process injections (CaPI) was introduced to evolve the modeling and execution of context-driven processes. Additionally, concepts were presented that enable changing runtime states of process instances in a context-aware manner. The modular concept supports the modeling and execution of process variants, which are required in many cyber-physical systems. By linking CaPI with PQL, process fragments can be selected from process repositories and dynamically injected into running process instances. This allows for a flexible modeling and late composition of process instances at runtime. In future work, we evaluate and optimize context-aware modeling of process families. Especially, this includes the correct modeling of context-aware process families. Furthermore, a user study investigating PQL comprehensibility is planned.

REFERENCES

[1] B. Hoppenstedt *et al.*, "Techniques and Emerging Trends for State of the Art Equipment Maintenance Systems—A Bibliometric Analysis," *Applied Sciences*, vol. 8, no. 6, p. 916, 2018.

[2] K. Kammerer *et al.*, "Anomaly Detections for Manufacturing Systems Based on Sensor Data—Insights into Two Challenging Real-World Production Settings," *Sensors*, vol. 19, no. 24, p. 5370, 2019.

[3] A. Hallerbach, T. Bauer, and M. Reichert, "Configuration and Management of Process Variants," in *International Handbook on Business Process Management*. Springer, 2010, pp. 237–255.

[4] ——, "Managing Process Variants in the Process Lifecycle," in *10th Int'l Conf on Enterprise Inf Sys (ICEIS 2008)*, 2008, pp. 154–161.

[5] M. Reichert and B. Weber, *Enabling Flexibility in Process-Aware Information Systems: Challenges, Methods, Technologies*. Springer, 2012.

[6] N. Mundbrod, G. Grambow, J. Kolb, and M. Reichert, "Context-Aware Process Injection: Enhancing Process Flexibility by Late Extension of Process Instances," in *23rd Int'l Conf on Coop Inf Sys (CoopIS 2015)*, ser. LNCS, no. 9415. Springer, 2015, pp. 127–145.

[7] M. Adams, A. H. Ter Hofstede, D. Edmond, and W. M. Van Der Aalst, "Worklets: A Service-oriented Implementation of Dynamic Flexibility in Workflows," in *Int'l Conf "On the Move to Meaningful Internet Systems" (OTM 2006)*. Springer, 2006, pp. 291–308.

[8] C. Ayora, V. Torres, B. Weber, M. Reichert, and V. Pelechano, "VIVACE: A Framework for the Systematic Evaluation of Variability Support in Process-aware Information Systems," *Information and Software Technology*, vol. 57, pp. 248–276, 2015.

[9] A. Murguzur, X. De Carlos, S. Trujillo, and G. Sagardui, "Context-aware Staged Configuration of Process Variants@Runtime," in *26th Int'l Conf Advanced Inf Sys Engineering*. Springer, 2014, pp. 241–255.

[10] K. Kammerer, J. Kolb, and M. Reichert, "PQL-A Descriptive Language for Querying, Abstracting and Changing Process Models," in *Enterprise, Business-Process and Information Systems Modeling*. Springer, 2015, pp. 135–150.

[11] K. Kammerer, R. Pryss, K. Sommer, and M. Reichert, "Towards Context-aware Process Guidance in Cyber-Physical Systems with Augmented Reality," in *4th Int'l Workshop on Requirements Engineering for Self-Adaptive, Collaborative, and Cyber Physical Systems (RESACS'18)*. IEEE Computer Society Press, 2018, pp. 44–51.

[12] G. Antunes, M. Bakhshandeh, J. Borbinha, J. Cardoso, S. Dadashnia, C. Di Francescomarino, M. Dragoni, P. Fettke, A. Gal, C. Ghidini *et al.*, "The Process Model Matching Contest 2015," *GI-Edition: Lecture Notes in Informatics.*, vol. 248, pp. 127–155, 2015.

[13] K. Kammerer, N. Mundbrod, and M. Reichert, "Demonstrating Context-aware Process Injection with the CaPI Tool," in *BPM Demo Session 2017, co-located with 15th Int'l Conf Business Process Management. CEUR Workshop Proceedings*, no. 1920. CEUR-WS.org, 2017.

[14] F. Holzschuher and R. Peinl, "Performance of Graph Query Languages: Comparison of Cypher, Gremlin and Native Access in Neo4j," in *Joint EDBT/ICDT 2013 Workshops*, 2013, pp. 195–204.

[15] M. Hunger and O. Gierke, *Good Relationships: The Spring Data Neo4j Guide Book*. C4Media, 2012.

[16] A. Hallerbach, T. Bauer, and M. Reichert, "Capturing Variability in Business Process Models: The Provop Approach," *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 22, no. 6-7, pp. 519–546, 2010.

[17] B. Aysolmaz, D. M. Schunselaar, H. A. Reijers, and A. Yaldiz, "Selecting a Process Variant Modeling Approach: Guidelines and Application," *Software & Systems Modeling*, vol. 18, no. 2, pp. 1155–1178, 2019.

[18] J. Klingemann, "Controlled Flexibility in Workflow Management," in *12th Int'l Conf on Advanced Inf Sys Engineering*, 2000, pp. 126–141.

[19] R. Aggarwal, K. Verma, J. Miller, and W. Milnor, "Constraint Driven Web Service Composition in METEOR-S," in *IEEE Int'l Conf on Services Computing 2004 (SCC 2004)*, 2004, pp. 23–30.

[20] G. Canfora, M. Di Penta, R. Esposito, and M. L. Villani, "A Framework for QoS-aware Binding and Re-binding of Composite Web Services," *Journal of Systems and Software*, vol. 81, no. 10, pp. 1754–1769, 2008.

[21] F. Casati and M.-C. Shan, "Dynamic and Adaptive Composition of E-Services," *Information Systems*, vol. 26, no. 3, pp. 143–163, 2001.

[22] S. Coppens, R. Verborgh, E. Mannens, and R. Van de Walle, "A Semantic Workflow Engine Powered by Grid Reasoning," in *IEEE Int'l Conf on Pervasive Comp and Comm Workshops*, 2014, pp. 460–465.

[23] R. Pryss, M. Reichert, M. Schickler, and T. Bauer, "Context-based Assignment and Execution of Human-centric Mobile Services," in *IEEE Int'l Conference on Mobile Services (MS 2016)*, 2016, pp. 119–126.

[24] C. Cabanillas, D. Knuplesch, M. Resinas, M. Reichert, J. Mendling, and A. Ruiz-Cortes, "Ralph: A graphical notation for resource assignments in business processes," in *27th Int'l Conf on Advanced Inf Sys Engineering*, ser. LNCS, no. 9097. Springer, 2015, pp. 53–68.

[25] D. Knuplesch, R. Pryss, and M. Reichert, "Data-aware Interaction in Distributed and Collaborative Workflows: Modeling, Semantics, Correctness," in *8th Int'l Conf on Collab Comp (CollaborateCom 2012)*. IEEE, 2012, pp. 223–232.

[26] R. Seiger, C. Keller, F. Niebling, and T. Schlegel, "Modelling Complex and Flexible Processes for Smart Cyber-physical Environments," *Journal of Computational Science*, vol. 10, pp. 137–148, 2015.

[27] M. Minor, R. Bergmann, S. Görg, and K. Walter, "Towards Case-based Adaptation of Workflows," in *18th Int'l Conference on Case-Based Reasoning (ICCBR 2010)*. Springer, 2010, pp. 421–435.

[28] K. Andrews, S. Steinau, and M. Reichert, "Enabling Runtime Flexibility in Data-Centric and Data-Driven Process Execution Engines," *Information Systems*, p. 101447, 2019.

[29] V. Künzle and M. Reichert, "PHILharmonicFlows: Towards a Framework for Object-aware Process Management," *Journal of Software Maintenance and Evolution*, vol. 23, no. 4, pp. 205–244, 2011.

[30] S. W. Sadiq, M. E. Orlowska, and W. Sadiq, "Specification and Validation of Process Constraints for Flexible Workflows," *Information Systems*, vol. 30, no. 5, pp. 349–378, 2005.

[31] D. Knuplesch and M. Reichert, "A Visual Language for Modeling Multiple Perspectives of Business Process Compliance Rules," *Software & Systems Modeling*, vol. 16, no. 3, pp. 715–736, 2017.

[32] A. Polyvyanyy, C. Ouyang, A. Barros, and W. M. van der Aalst, "Process Querying: Enabling Business Intelligence through Query-based Process Analytics," *Decision Support Systems*, vol. 100, pp. 41–56, 2017.

[33] C. Beeri, A. Eyal, S. Kamenkovich, and T. Milo, "Querying business processes with BP-QL," *Inf Sys*, vol. 33, no. 6, pp. 477–507, 2008.

[34] A. Awad, "BPMN-Q: A Language to Query Business Processes," *Enterprise Modelling and Information Systems Architectures–Concepts and Applications*, 2007.

[35] Z. Yan, R. Dijkman, and P. Grefen, "FNet: An Index for Advanced Business Process Querying," in *10th Int'l Conference on Business Process Management (BPM 2012)*. Springer, 2012, pp. 246–261.

[36] J. Zhu and H. K. Pung, "Process Matching: A Structural Approach for Business Process Search," in *Computation World*, 2009, pp. 227–232.

[37] P. Delfmann, D. Breuker, M. Matzner, and J. Becker, "Supporting Information Systems Analysis through Conceptual Model Query–the Diagramed Model Query Language (DMQL)," *Communications of the Association for Information Systems*, vol. 37, no. 1, p. 24, 2015.

[38] S. Sakr and A. Awad, "A Framework for Querying Graph-Based Business Process Models," in *19th Int'l Conference on World Wide Web (WWW 2010)*. ACM, 2010, p. 1297–1300.