Classifying and Detecting Task Executions and Routines in Processes using Event Graphs

Eva L. Klijn[0000−0001−9270−4774], Felix Mannhardt[0000−0003−1733−777X], and Dirk Fahland[0000−0002−1993−9363]

Eindhoven University of Technology, the Netherlands
{e.l.klijn,f.mannhardt,d.fahland}@tue.nl

Abstract. Business process management organizes work into several interrelated "units of work", fundamentally conceptualized as a task. The classical concept of a task as a single step executed by a single actor in a single case fails to capture more complex aspects of work that occur in real-life processes. For instance, actors working together or the processing of work in batches, where multiple actors and/or cases meet for a number of steps. Established process mining and modeling techniques lack concepts for dealing with these more complex manifestations of work. We leverage event graphs as a data structure to model behavior along the actor and the case perspective in an integrated model, revealing a variety of fundamentally different types of task executions. We contribute a novel taxonomy and interpretation of these task execution patterns as well as techniques for detecting these in event graphs, complementing recent research in identifying patterns of work and their changes in routine dynamics. Our evaluation on two real-life event logs shows that these non-classical task execution patterns not only exist, but make up for the larger share of events in a process and reveal changes in how actors do their work.

Keywords: Task execution patterns · Routines · Event graphs.

1 Introduction

A central goal of Business Process Management (BPM) is organizing into several interrelated "units of work" to achieve shared goals. The formal foundations of BPM, as used in process modeling and mining, conceptualize such a unit of work as a *task*. Tasks are planned, scheduled, distributed to suitable actors such that the overall work can be performed by a collaborating workforce. Most Process-aware Information Systems (PAIS) support this goal by assuming that work is performed in the context of a business process that is executed as a sequence of task executions called a *case*. Each task is executed by a specific actor and the BPM system is responsible that the correct tasks are performed in the correct order. Thereby the actual work happens outside the PAIS itself which only schedules tasks and checks completion [9].

However, this concept of a task in process modeling and mining in BPM — a unit of work is a single step executed by a single actor in a single case —

fails to capture many facets of work that occur in practice. In organizations research, a well-defined (atomic) step in a process is called an activity or *action* [2,19]. In contrast, a *task* is considered a slightly larger "unit of work" that has to be carried out to achieve an objective within the process, e.g., review CVs. Completing or *executing* a task often requires to perform multiple actions (e.g., download, open, take notes), not necessarily limited to a single case (e.g., all CVs received); these actions may be grouped differently depending on the actor(s) the task is assigned to.

This also has been acknowledged in the BPM field from several perspectives. Robotic Process Automation (RPA) uses *task mining* to identify how individual actors perform tasks by recording their desktop interactions revealing tasks spanning more than a single case, e.g., data entry from a spreadsheet to an information system [15]. Individual actors may batch actions in multiple cases, e.g., a manager reviewing and approving requests in different cases, which is still poorly supported by many PAIS [22]. Finally, actors often do not act independently from each other, multiple actors may perform work together even across multiple cases, e.g., the collaborative grading of student reports, and across multiple actions, e.g., delivering and installing a new washing machine. Despite this acknowledgment and many years of BPM research, there is no generally agreed definition of a task that captures such aspects of work in a process and that is compatible with the established process modeling and process mining concepts. In other words, the existing process mining and process modeling concepts are too simplistic.

In this paper, we investigate how to conceptualize task executions beyond the basic definition of an action performed by a single actor in isolated cases with the goal of capturing the various facets of tasks. Our approach combines event data analysis with conceptually modeling behavior in processes and actors as *two behavioral dimensions simultaneously* [10]: (1) the sequence of events recorded in a process case and (2) the sequence of events, across multiple process cases, in which an actor is involved. We use *event graphs* as introduced in [10] as data structure to model relations between events, cases, and actors as paths along cases and along actors over the same events; thereby escaping limitations of classical event logs (Sect. 3). In such a graph, a task execution emerges when a path along an actor meets a path along a case over one or more events.

We then perform a systematic, theoretical analysis of the types of task executions that can be expressed in an event graph depending on (1) how many paths along cases meet (2) how many paths along actors for (3) how many events (Sect. 4.1). From this theoretical analysis we derive a *taxonomy* of task execution types characterized by 5 different parameters (Sect. 4.2); the taxonomy describes 23 task execution types, several have not been described in literature before. We present methods for querying these task execution types in event graphs (Sect. 5) and evaluate the existence of these task execution types in the BPIC'17 and BPIC'14 event logs (Sect. 6). We specifically found that non-trivial task execution patterns over multiple steps, multiple cases, and even multiple actors frequently occur in two real-life event logs, as well as occurrences of several previously unknown patterns. We also observe changes in frequency of task execution patterns over time due to changes in the way actors do their work.

2 Related Work

Related research that also accounts for the more complex aspects of work beyond isolated cases has been conducted from several perspectives.

Process *modeling* literature studies actors performing work in terms of "resources" required for a task. Of the workflow resource patterns [23], only "Simultaneous Execution" and "Additional Resources" consider joint work by multiple actors. Only recently, actor behavior across multiple cases came into focus under *batching* across individual cases [22] and *instance-spanning constraints* [12]. Current BPM systems poorly support these phenomena and existing notations (e.g., BPMN) require extensions [22,13] to support them; but actor behavior is never modeled explicitly. Synchronous proclets [11] allow modeling individual actor behavior across individual cases in a network of Petri nets, each describing a process or an actor [6], that dynamically synchronize on single transition occurrences. The same synchronization principle has also been adopted for DCR graphs [4]. We contribute to this stream of modeling research by showing that actor-case interactions themselves form complex task execution patterns over multiple actions, cases, and actors, that should be supported in modeling.

In *process mining*, social network mining [26] analyzes actor interactions but excludes the control-flow perspective. Other approaches are mining of team composition and work assignment [24], resource skills, utilization, and productivity [21] and resource availability [16]; these works assume tasks to be single actions. Task executions by the same actor over multiple actions can be discovered as local process models [5]. Task mining analyzes behavior that may transcend multiple cases [15] by tapping into desktop interaction logs of actors. These works are limited to single actors in isolated cases due to the use of event logs. For analyzing instance spanning constraints [27], batch activities [17,18] and scheduled processes [25] process mining methods have been extended to consider inter-case relations; in these works actor behavior is described/modeled implicitly, whereas we analyze actor behavior explicitly.

Routines research [20,14,19] studies "work" in terms of a *narrative network* of actors interacting in an organization (of people and devices) to achieve their goals. A narrative [19] thereby is a *path* in the narrative network, i.e., a "coherent, time-ordered sequence of actions or interactions for accomplishing an organizational task" [20,14]. An action pattern that occurs repeatedly at an individual actor is called a *habit*; a recurrent action pattern involving multiple actors is called a *routine* [2]. Habits and routines capture how actors accomplish their tasks. A central question in routine dynamics research is to identify such patterns in the narrative network and how they change [14] and is approached through observations, interviews, and archival materials in a field study.

Our work complements prior work by first transforming an event log into an event graph which can be understood as data-based representation of a narrative network. We use graph theory to detect patterns of task executions (i.e., habits and routines) and their changes over time. Our taxonomy of different types of task executions extends and generalizes existing notions of tasks in BPM and process mining that are tailored towards either isolate cases, e.g., [5], or towards specific aspects of work behavior across cases, e.g., [12,13,15,17,18,22,27,25].

3 Preliminaries

We first discuss relevant concepts of the conventional, single-dimensional representation of event data. We then show how these concepts can be translated to a multi-dimensional representation using a general data model based on labeled property graphs [10], which we use as a foundation for our work.

Single-Dimensional Representation of Event Data. A PAIS can record an action execution as an *event* in an event log. Each event records multiple attributes, including at least the *action* that occurred, the *time* of occurrence, and an *entity identifier* indicating on which entity or *case* the action occurred. Often, the actor executing the action is recorded as *resource*. Tab. 1 shows an example event log containing 10 events occurring on the same day.

Process mining [1] analyzes event data by grouping events w.r.t. a chosen *case identifier* attribute, e.g., a loan application document or a patient in a hospital. Ordering all events of a case by time yields the *trace* as a sequence of events. Grouping the events in Tab. 1 by *Case* yields the traces $\langle e1, e2, e3, e4, e5 \rangle$ and $\langle e6, e7, e8, e9, e10 \rangle$. A set of such traces is a traditional *single-dimensional* event log along the case perspective [10].

Classically the *Resource* attribute in Tab. 1 is considered as event attribute describing the event further. However, the resource (the actor) is an entity in its own right and we can also study the sequence of events along each resource, defining a second behavioral dimension in the data. Choosing *Resource* as case identifier yields a second event log with traces $\langle e1, e2, e6, e7, e8 \rangle$ (Resource 1), $\langle e3, e4, e9, e10 \rangle$ (R. 5), and $\langle e5 \rangle$ (R. 29).

Each event in Tab 1 is related to 2 entity identifiers: a case identifier and resource identifier. Generally, an event can have multiple case identifiers and/or multiple resource identifiers [10]. The relation of events to multiple entities results in different behavioral dimensions between events that cannot be adequately represented or analyzed using a single-dimensional event log representation.

Multi-Dimensional Representation of Event Data. We use a *labeled property graph* (LPG) to represent multiple behavioral dimensions (case and resource) together over a set of events.

Graph databases use LPGs [3, Ch.2] for modeling various entities (as nodes) and various relationship (as edges) between them. An *event graph* [10] is a specific LPG, which can be obtained from an event table: each event and each entity (i.e., cases and resources) is represented by a node with label *Event* or *Entity*. Event and entity nodes are connected through directed binary *relationships*: a *CORR* relationship from e to n defines that event e is *correlated* to entity n. A

DF relationship from e to e' defines that event e' *directly follows* e from the perspective of a specific entity n to which e and e' are correlated (i.e., e occurs before e' and there is no other event between them). Each node and relationship can hold a number of key-value pairs referred to as *properties*, e.g., whether an entity has $Type = Case$ or $Type = Resource$. As short-hand notation we write $(e, e')^x$ for a *DF*-relationship in G from e to e' of type $x \in \{c, r\}$ (i.e., case or resource). See [10] and App. A.1 for details.

The example in Fig. 1 shows the event graph derived from Tab. 1: each square (white) node is an event node; each circle is an entity node of the corresponding type (blue for *Case*, red for *Resource*). *CORR* relationships are shown as dashed edges, e.g., $e1, e2, e3, e4, e5$ are correlated to case $c3$ and $e1, e2, e6, e7, e8$ are correlated to resource $r1$. *DF* relationships are shown as solid edges. The DF-relationships between the events correlated to the same entity form a *DF-path* for that entity; the graph in Fig. 1 defines 2 DF-paths for case entities, e.g., $\sigma_{c3} = \langle (e1, e2)^c, (e2, e3)^c, (e3, e4)^c, (e4, e5)^c \rangle$ and 3 DF-paths for resource entities, e.g., $\sigma_{r1} = \langle (e1, e2)^r, (e2, e6)^r, (e6, e7)^r, (e7, e8)^r \rangle$.

In the graph in Fig. 1, we observe which resource executed which action in which case, e.g., $r1$ performed A in $c3$ (at event $e1$). However, we can also see that DF-paths for case and resource "flow in parallel" over multiple actions, e.g., σ_{c3} and σ_{r1} both contain $(e1, e2)^x$ meaning $r1$ performed A and B consecutively in $c3$ (events $e1$ and $e2$) forming a larger unit of work captured Fig. 1 as the *connected subgraph* of events $\{e_1, e_2\}$ and the two DF-relationships between them. We can observe more such subgraphs in Fig. 1 where the same resource and case are involved in consecutive events, i.e., larger units of work.

4 Task Execution Patterns

We observed in Sect. 3 that event graphs reveal "units of work" that are not just individual events but are *connected subgraphs* where resources and case meet along several subsequent events. In this section, we conceptualize these connected subgraphs as *task executions* and explore in which forms they can manifest. We explain our approach in Sect 4.1 and present a novel taxonomy of task execution

Table 1: Event table example

Event	Action	Time	Case	Resource
e1	A	12:02	3	1
e2	B	12:04	3	1
e3	E	14:38	3	5
e4	F	14:41	3	5
e5	C	16:21	3	29
e6	A	12:08	4	1
e7	B	12:09	4	1
e8	D	12:15	4	1
e9	E	14:54	4	5
e10	F	14:59	4	5

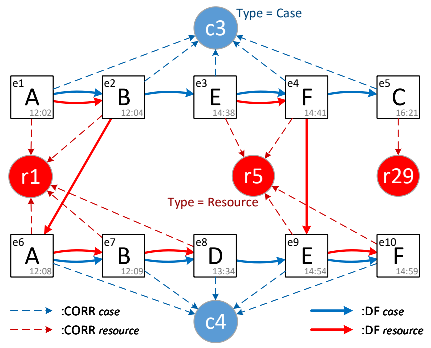


Fig. 1: Event graph containing the events and entities from Tab. 1

patterns and their interpretation in Sect 4.2. We thereby make use of standard graph theory concepts; the formal definitions are available in Appendix A at reviewer discretion.

4.1 Exploring Event Graphs for Forms of Task Executions

In the event graph in Fig. 1, we initially observe two ways in which a task execution manifests. (1) A resource follows a case over multiple events, e.g., $e6, e7, e8$; these event nodes form a subgraph induced by one DF-path σ_c of a case entity and one DF-path σ_r of a resource entity as follows: σ_c and σ_r *enter* the subgraph together (e.g., at $e6$) and *leave* the subgraph together (e.g., at $e8$) and are both *continuous* in this subgraph (all events of the DF-path are within the graph). (2) We also observe an execution of a classical task in the event graph in Fig. 1 consisting of only a single event, e.g., $e5$; the path of the resource and the path of the case synchronize for this step only, i.e., the subgraph is a single node.

We explored whether other subgraphs can be characterized by searching for different configurations of the following concepts in the subgraph: DF-path of a resource, DF-path of a case and their synchronization. We identified the following parameters and values:

1. The subgraphs in Fig. 1 contain at most *one* case DF-path. Are there (meaningful) execution patterns which have *multiple* case DF-paths?
2. If multiple case DF-paths are in the subgraph: are the case DF-paths disjoint (i.e., each event belongs to exactly one case) or can case DF-paths synchronize (i.e., have a shared event)?
3. The subgraphs in Fig. 1 contain at most *one* resource DF-path. Are there (meaningful) execution patterns which have *multiple* resource DF-paths?
4. If multiple resource DF-paths are in the subgraph: are the resource DF-paths disjoint (i.e., each event belongs to exactly one resource) or can resource DF-paths synchronize (i.e., have a shared event)?
5. In Fig. 1, all DF-paths are continuously in the subgraph (i.e., they only enter once and leave once). Are there (meaningful) execution patterns where a DF-path also temporarily leaves the subgraph and re-enters later?

4.2 Taxonomy of Task Execution Patterns

The above questions define a parameter space that allows for a set of different subgraph configurations that can be systematically described within the bounds of this space. We explored this parameter space by modeling abstract *task execution patterns* as subgraphs of event nodes of an event graph. We explain the patterns found, introduce a taxonomy to structure them systematically, and evaluate whether each pattern has real-world interpretation and whether it was already discussed in literature.

We considered subgraphs that emerge from multiple DF-paths synchronizing as some “unit of work”. We identified the following necessary conditions for

$n \geq 2$ DF-paths $\Sigma = \{\sigma_1, \ldots, \sigma_n\}$ to induce a subgraph G that describes a task execution: (**T1**) any two event nodes in G are connected via at least one DF-path $\sigma \in \Sigma$, (**T2**) for each event node e in G exists a case DF-path $\sigma_c \in \Sigma$ and a resource DF-path $\sigma_r \in \Sigma$ that contain e (i.e., G is traversed by at least one case and one resource DF-path), (**T3**) there is at least one DF-path $\sigma_1 \in \Sigma$ that is *continuously* in G (enters G once and leaves G once). We identified two stricter necessary conditions of task executions defining a spectrum:

- *Graph-structure based task execution*: in the strictest form of task executions the subgraph G is induced by $\Sigma = \{\sigma_1, \ldots, \sigma_n\}$ and satisfies (T1)-(T3) and additionally (**T2'**): each event node $e \in G$ is in *each* DF-path $\sigma \in \Sigma$ (i.e., all paths always synchronize in all events in G but some paths may leave in between). As a consequence, all paths *converge* at the first event of the continuous DF-path σ_1 in G and *diverge* at the last event of σ_1 in G, see (T3). All subgraphs in Fig. 2 have this property.
- *Domain-knowledge based task execution*: the paths in G do not converge and diverge at the same start and end events of G, yet are coherent. In addition to (T1)-(T3) the following condition holds: (**T3'**) all DF-paths are continuously in G. All subgraphs in Fig. 3 have this property. T3' requires domain-knowledge to decide whether DF-paths Σ form a valid subgraph G describing a task execution.

Next, we differentiate different types of subgraphs further by the following 4 parameters over a subgraph G (i) the number of case DF-paths, (ii) the number of resource DF-paths, and (iii) how often they enter and leave, and (iv) how they synchronize in G. We start with graph-structure based tasks executions.

Taxonomy of Graph-Structure-Based Task Execution Patterns Fig. 2 shows the graph-structure-based task execution patterns arranged according to the parameters identified in Sect. 4.1.

The taxonomy categorizes the subgraphs on the x-axis based on them containing a single DF-path from a single case (SC) or multiple DF-paths from multiple cases (MC). The patterns are categorized along the y-axis based on them containing a single DF-path from a single resource (SR) or multiple DF-paths from multiple resources (MR). This structures our taxonomy into four major quadrants: (SR,SC), (SR,MC), (MR,SC), (MR,MC).

Next, subgraphs within each of these quadrants are arranged based on the configuration of the paths they contain. A path is (1) *single step* (s) if it only contains a single event node within the subgraph, (2) *continuous* (c) if the path contains > 1 event node and is continuously within the subgraph, i.e., it only enters and leaves once, and (3) *interrupted* (i) if it contains > 1 event node and it leaves and enters the subgraph more than once. A single step path and an interrupted path are both *non-continuous.*

The set of resource DF-paths are configured separately from the set of case DF-paths. For each quadrant the subgraphs are arranged on the x-axis into columns for continuous and non-continuous case DF-paths and on the y-axis into rows for continuous and non-continuous resource DF-paths.

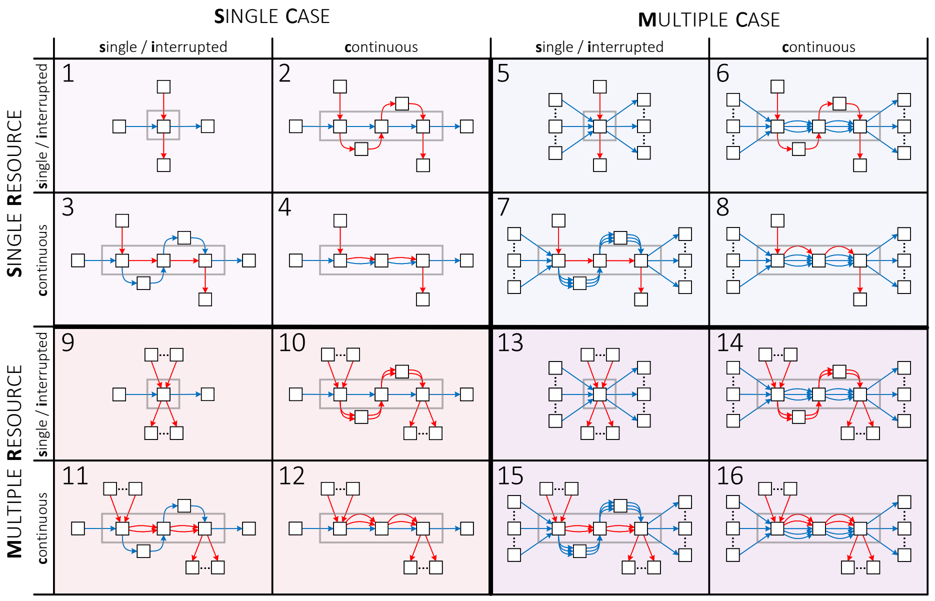


Fig. 2: Taxonomy of graph-structure-based task execution patterns

The bold letters in Fig. 2 indicate the short-hand notation we use in the following, e.g., $\mathsf{MR_i,SC_c}$ denotes pattern P10: Multiple Resource interrupted, Single Case continuous.

Structural properties. A fundamental property of graph-structure-based execution patterns is that all entities (cases and resources) are involved in each step of the task execution.

A fundamental property of our taxonomy is that adding/removing resource or case DF-paths results in a corresponding pattern in another quadrant, e.g., adding more resource DF-paths to P3 returns P11 and adding both more resource and case DF-paths returns P15.

A second fundamental property of the taxonomy is that it distinguishes *elementary task execution patterns* that cannot be decomposed further into subgraphs fitting the parameter space (1, 4, 5, 8, 9, 12, 13, 16) and *non-elementary task execution patterns* that are compositions thereof (2, 3, 6, 7, 10, 11, 14, 15). By combining instances of elementary patterns along either a continuous resource DF-path or a continuous case DF-path, we end up with a non-elementary instance belonging to the same quadrant. For instance, if we take three instances of P1 for the same case c and resource r and combine them along the case DF-path, i.e., the case is continuous in the composition (only leaves and enters once) while the DF-path for r can be arbitrary, we end up with P2.

The subgraphs in Fig. 2 that are not single-step are only one of the many variations possible within each specific cell. For instance, if we take two instances of P1 and one instance of P4 and again combine along a continuous case DF-path, we end up with a variation of P2.

Conceptual evaluation. Within each quadrant, the top-left cell (1, 5, 9, 13) describes a single-step task. The bottom-right cell (4, 8, 12, 16) describes a task continuously involving all cases and resources over multiple steps. The top-right and bottom-left cell of a quadrant describes tasks interrupted by either the resource(s) or the case(s), respectively, e.g., a resource attending an urgent task in another case or a resource requiring input from another resource not involved in the task. While some single step and continuous patterns have been observed in other works [5,6,17], at present, no work has systematically studied interrupted patterns.

SR,SC patterns portray tasks that can only be executed for a single case by a single resource at a time, e.g., writing out a speeding ticket (single step, i.e., P1), or finalizing a loan offer: ⟨call client about loan offer, create offer, send offer⟩ (multiple steps, i.e., P4). P2 describes an actor interrupting and returning to a larger unit of work in a case, e.g., compiling a report that is interrupted by other duties. P3 describes an actor continuously being concerned with the same case while other actors have to be involved as well, e.g., due checks based on a four-eyes principle. P1 has been extensively studied in traditional process analysis and P4 has been observed in [5].

SR,MC patterns portray tasks where a single resource handles *multiple cases together*, such as batch processing, e.g., lecturing a classroom of students (single step, i.e., P5) or analyzing a batch of blood samples, which involves transporting them to the lab, scanning them and analyzing them (multiple steps, i.e., P8). Only single step (P5) and multi-step batching (P8) have both been observed in [17].

Conversely, MR,SC patterns portray tasks where *multiple resources* work on a single case *together*. For instance in collaborative decision making, e.g., a master's defense (single step, i.e., P9), or due to practical or technical requirements, e.g., delivering, carrying, and installing a washing machine requires two people (multiple steps, i.e., P12). Only P9 has been observed in [6], where queues and conveyor belts synchronize as distinct resources for the same (single step) events.

Finally, MR,MC patterns portray tasks executed by multiple resources on multiple cases together. While theoretically possible, it is very unlikely multiple cases and resources synchronize that strongly in real-life processes especially over multiple steps. A very relaxed interpretation of such a task would be the co-chairing of a panel for a conference (single step, i.e., P13). We discuss more realistic manifestations of MR,MC task configurations when discussing domain-knowledge based patterns (shown in Fig. 3). At present, no work has systematically studied patterns involving both multiple cases and multiple resources.

Taxonomy of Domain-Knowledge-Based Task Execution Patterns So far we discussed our taxonomy for the strictly synchronized graph-structure based patterns shown in Fig. 2. We now discuss our taxonomy for the domain-knowledge based patterns at the other end of the spectrum shown in Fig. 3. The domain-knowledge based task execution patterns allow that only some case/resource paths synchronize per event but require all paths to be uninterrupted; this yields

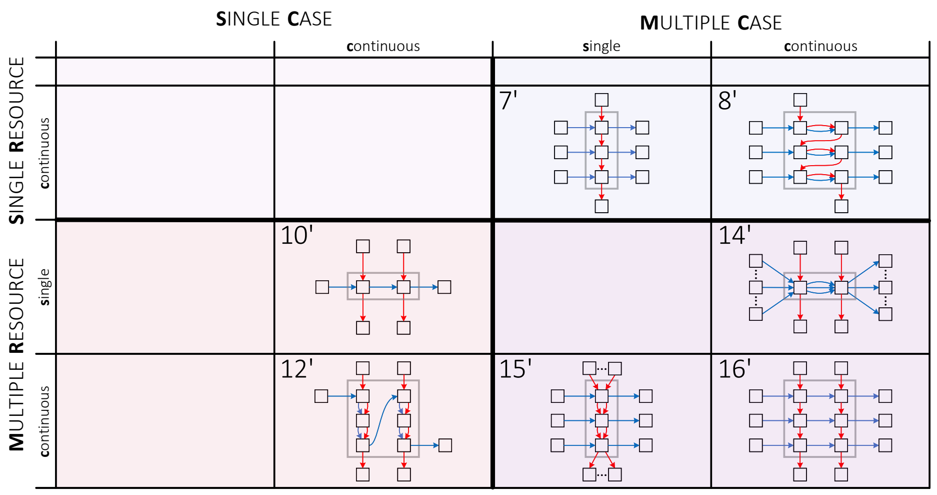


Fig. 3: Taxonomy of domain-knowledge-based task execution patterns

"units of work" that are more distributed. While there is a considerable amount of other subgraphs that fit Fig. 3, we limit ourselves to discuss those that require only basic domain knowledge.

Structural properties. We first observe that we can derive all the configurations in Fig. 3 by composing multiple of the same elementary pattern from Fig. 2. For instance, P7' and P10' are essentially multiple instances of P1 composed along the resource and case DF-path, respectively. P8' and P12' can be composed similarly using instances of P4. P14' can be composed of P5 instances along the case DF-path and P15' with P9 instances along the resource DF-path. In Fig. 3, P16' is composed of multiple instances of P1, both along the resource and case dimension. The pattern properties of this particular cell 16' allow for basically every combination of elementary patterns composed along both the resource and case dimension. In contrast to graph-structure-based non-elementary patterns, the conditions for composing the patterns in Fig. 3 require domain knowledge.

Conceptual evaluation. In Fig. 3, P7', P8', P15' portray different forms of sequential batching, i.e., the same step is executed for a sequence of cases one after the other, with a single resource (P7', $\mathsf{SR_c}$,$\mathsf{MC_s}$), with multiple resources (P15', $\mathsf{MR_c}$,$\mathsf{MC_s}$) and with one resource executing multiple steps per case (P8', $\mathsf{SR_c}$,$\mathsf{MC_c}$). Sequential batching that involves a single resource (P7' and P8') has been observed in [17].

We also identify a subset of patterns in which multiple resources are separately involved, each performing a set of steps for a case after which it moves to the next resource for the next step(s) in a pipe-lined fashion (e.g., P10' and P12', $\mathsf{MR_{s,c}}$,$\mathsf{SC_c}$), resembling a factory/production type of setting. Such a setting could also be realized for (simultaneous) batches of cases (P14', $\mathsf{MR_s}$,$\mathsf{MC_c}$). At present, no work has systematically studied these patterns.

P16' ($\mathsf{MR_c}$,$\mathsf{MC_c}$) is a combination of the former two types; it portrays sequential batching being performed in a pipe-lined fashion. In Sect. 6 we show that we can identify an instance of P16' in the BPIC'17 data.

5 Detecting Task Execution Patterns in Event Graphs

In this section, we present a technique for detecting instances of the task execution patterns of Section 4 as subgraphs in an event graph. We *query* the graph to retrieve all instances (subgraphs) of *elementary* task execution patterns (P1, P4), which we then materialize and store as new *"task instance" nodes* (Sect. 5.1). Later, we use these task instance nodes for querying *elementary task instances* (Sect. 5.2) and for detecting and querying *non-elementary task instances* (Sect. 5.3). All conceptual queries presented here are implemented in the graph query language Cypher on the graph database Neo4j; see Sect. 6 and App. B.

5.1 Modeling Elementary Task Instances as High Level Events

We assume the data to be given in an event graph G (see Sect. 3). We describe how to detect in G subgraphs describing task instances (TIs) of elementary patterns P1 and P4 and how to materialize these as new nodes with label TI in the event graph. Finally, we lift the DF-edges from the *Event* nodes in the task instance subgraph to the corresponding TI node. Fig. 4 shows the result of constructing the TI nodes and all corresponding relationships for the event graph from Fig. 1.

We first search the graph for all pairs of events (e_i, e_{i+1}) that have both a case DF-edge $(e_i, e_{i+1})^c$ and a resource DF-edge $(e_i, e_{i+1})^r$ and create a new *"joint"* DF-edge $(e_i, e_{i+1})^j$. We then detect any task instance of elementary P4 as a maximal sequence of events $ti = \langle e_m, ..., e_n \rangle$ where for each $e_i, e_{i+1} \in ti$ there exists $(e_i, e_{i+1})^j$ and there exists no joint DF-edges $(e', e_m)^j$ or $(e_n, e')^j$. An instance of P1 is $ti = e$ without $(e', e)^j$, $(e, e')^j$. We materialize ti as a new node h_{ti} with label TI and a *contains* relationship from h_{ti} to each $e \in ti$. We treat h_{ti} as a "high-level" event and set properties $h_{ti}.time_{start} = e_m.time$

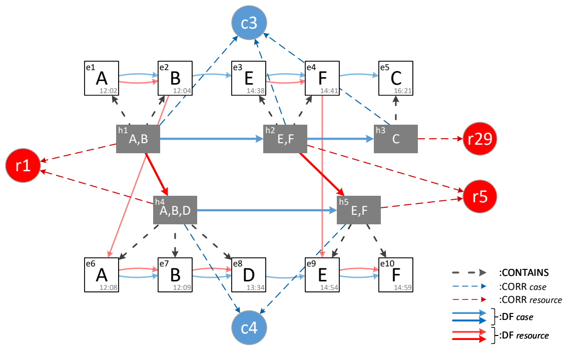


Fig. 4: Event graph containing the h_{ti} nodes constructed from the events from Fig. 1

and $h_{ti}.time_{end} = e_n.time$ and correlate h_{ti} to each entity n to which $e_m, ..., e_n$ are correlated by adding *CORR* relationships. Finally, we sort all TI nodes $h_1, \ldots, h_k$ correlated to the same entity n of type x by $time_{end}$ and introduce corresponding DF-edges $(h_i, h_{i+1})^x$ of type x, which lifts DF-paths from events to task instances. See Appendix B.1 for the Cypher query. For example, we detect in Fig. 1 $ti_1 = e_1, e_2$ and $ti_4 = e_6, e_7, e_8$ resulting in nodes $h1$ and $h4$ and $(h1, h4)^r$ in Fig. 4. Instances of the other elementary patterns (5, 8, 9, 12, 13, 16) can be found by checking for multiple $(e_i, e_{i+1})^c$ and/or $(e_i, e_{i+1})^r$ relationships over all events in ti.

The *elementary task execution* T described by an elementary task instance $ti = e_m, ..., e_n$ is its sequence of action names $T = e_m.action, ..., e_n.action$; we set $ti.task = T$ for easier querying.

5.2 Querying Elementary Task Instances

Having materialized all elementary task instances into TI nodes, we can query the graph of TI nodes and DF-relationships between them for insights. This allows for the following kinds of queries: (1a) Retrieve a *subset* of TIs based on a specific property, e.g., all TIs correlated to $r1$ ($h1$ and $h4$ in Fig. 4), or (1b) the subset of TIs of the most frequently executed tasks ($h2$ and $h5$ in Fig. 4). (2) Query for DF-paths between TI nodes, for instance the DF-path of TI nodes correlated to a specific case ($\langle h4, h5 \rangle$ for $c1$ in Fig. 4) or to a specific resource ($\langle h3 \rangle$ for $r29$ in Fig. 4). (3) Querying the DF-path of TIs of a specific resource on a specific day could give insight into habits [2] followed by this resource. Next, we query DF-paths between TI nodes along cases and resources to detect larger, non-elementary task execution patterns.

5.3 Querying Non-Elementary Task Instances

We materialized elementary task instances as TI nodes connected through DF-edges in Sect. 5.1. We now show how to detect instances of non-elementary task execution patterns (NTI for short) as shown in Figures 2 and 3 as compositions of TIs by querying for paths of TI nodes along DF-edges.

We detect any *interrupted* NTI (Fig. 2) involving resources $r_1, ..., r_l$ and cases $c_1, ..., c_m$ by querying for a maximal sequence of TI nodes $h_1, ..., h_k$ with $(h_i, h_{i+1})^{r_1}, ..., (h_i, h_{i+1})^{r_l}$ or $(h_i, h_{i+1})^{c_1}, ..., (h_i, h_{i+1})^{c_m}, 1 \leq i < k$ so that all underlying *Event* nodes are correlated to the same resource entities $n_{r_1}, ..., n_{r_l}$ and case entities $n_{c_1}, ..., n_{c_m}$. For detecting the *domain-knowledge-based* NTIs (Fig. 3), we also query a maximal sequence of TI nodes along either the resource-path or case-path, but this time require only one of the entity types (cases or resources) to be correlated to all TI nodes. For the patterns that describe batching behavior (7', 8', 15', 16'), we additionally require all TI nodes $h_1, \ldots, h_k$ to describe the same elementary task, i.e. $h_i.task = h_{i+1}.task$, and a maximum time difference Δt_{batch} between two subsequent TIs, i.e. $h_{i+1}.time_{start} - h_i.time_{end} < \Delta t_{batch}$; see Appendix B.2 for a Cypher query that detects NTIs of sequential batching P8'. Using such domain knowledge for a time gap is commonly done in batch identification [17]. For patterns 10', 12', 14' and 16' additional domain-knowledge is required to determine if multiple TIs along the case-path form a task execution. Examples of NTIs of P2, P3, P7' and P8' are shown in Fig. 5.

6 Evaluation

We performed an exploratory analysis to investigate the occurrence of task execution patterns in two real-life event logs BPIC'14 [7] (only the events correlated to a resource) and BPIC'17 [8]. We realized the approach of Sect. 5 in Cypher queries invoked via Python scripts on the Graph DB Neo4j; available at `https://github.com/multi-dimensional-process-mining/event-graph-task-pattern-detection`. Creating all TI constructs (Sect. 5.1) took 42.09s for the BPIC'14 log and 73.59s for the BPIC'17 log on an Intel i7 CPU @ 2.2GHz machine with 32GB RAM.

We applied queries for detecting all patterns that did not require specific domain knowledge, i.e., all patterns except 10', 12', 14' and 16', in the event graphs of the BPIC'14 and BPIC'17 data. We found TIs of patterns 1, 2, 3, 4, 7' and 8'; TIs involving multiple resources and/or multiple cases per event simply do not occur in the data.

Fig. 5 shows for the BPIC'17 data for each detected pattern type a task instance annotated with resource and case identifiers. The P1 instance in Fig. 5 shows actor $r14$ executing a single step in a case before moving to a different case and the P4 instance shows $r95$ executing four actions in a case before moving to the next. The P2 instance shows $r3$ executing ten actions in a case with an interruption after the first step W_2, executing the same action W_2 also in another case before completing the task in the former case. The P3 instance shows $r107$ continuously working on a case, performing the same task execution ($\langle V_4, I_1, I_2, A_6\rangle$) twice while $r128$ performs other actions in between. The P7' and P8' instances show $r35$ and $r126$ performing the same actions W_2 and $\langle I_4, V_1, V_2, A_9\rangle$, respectively, for the same five cases in a sequential batch. We observe a min/avg/max time difference of 0/12/613min and 0/4.8/512min between any two subsequent steps in a batch for BPIC'17 and BPIC'14, respectively.

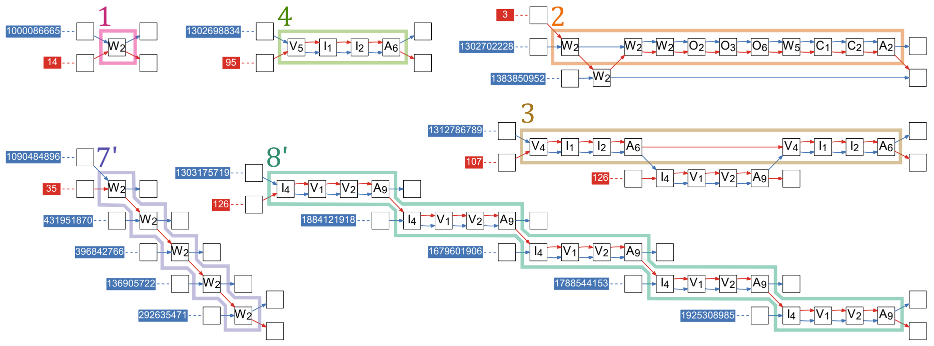


Fig. 5: Instances of task execution patterns 1, 2, 3, 4, 7' and 8' found in the BPIC'17 data

Table 2: General task execution pattern measurements for BPIC'17 and BPIC'14 with <30m between all TIs in NTI or >30m between at least one TI in NTI

log	pat.	# of TIs	# of events	% of events	length		duration (min)		# of TIs(1,4) in the NTIs	
					avg.	st.dev	avg.	st.dev	avg.	st.dev
BPIC'17	**1**	11 995	11 995	1.4	1.0	0.0	0.0	0.0		
without	**4**	125 472	703 000	81.8	5.6	2.8	2.9	61.5		
User_1	$2^{>30}$	1 174	10 640	1.2	9.1	3.0	140.0	104.0	2.0	0.1
	$3^{>30}$	45	354	0.0	7.9	2.4	121.6	92.9	2.0	0.0
	$2^{<30}$	431	3 755	0.4	8.7	3.7	19.9	10.5	2.0	0.3
	$3^{<30}$	55	419	0.0	7.6	3.4	11.8	12.8	2.0	0.1
	$7'^{<30}$	269	1 297	0.2	4.8	1.1	23.8	16.1	4.8	1.1
	$8'^{<30}$	2 385	64 974	7.6	27.2	16.4	43.8	38.8	6.2	3.7
BPIC'17	***1***	*27*	*27*	*0.0*	*1.0*	*0.0*	*0.0*	*0.0*		
only	***4***	*33 706*	*144 683*	*16.8*	*4.3*	*1.6*	*0.2*	*0.4*		
User_1	$2^{<30}$	*1 788*	*10 687*	*1.2*	*6.0*	*0.3*	*0.8*	*0.8*	*2.0*	*0.1*
	$3^{<30}$	*255*	*1 530*	*0.2*	*6.0*	*0.0*	*0.7*	*0.5*	*2.0*	*0.0*
	$8'^{<30}$	*1 350*	*43 641*	*5.1*	*32.3*	*29.2*	*24.2*	*33.1*	*9.0*	*14.3*
BPIC'14	**1**	107 069	107 069	22.9	1.0	0.0	0.0	0.0		
	4	138 002	359 668	77.1	2.6	0.9	21.6	580.9		
	$2^{>30}$	16 489	80 585	17.3	4.9	2.0	116.1	102.5	2.4	0.8
	$3^{>30}$	1 631	7 364	1.6	4.5	2.0	103.3	87.5	2.1	0.3
	$2^{<30}$	16 963	70 965	15.2	4.2	1.5	10.7	14.9	2.2	0.4
	$3^{<30}$	11 085	40 029	8.6	3.6	1.4	6.5	18.0	2.0	0.1
	$7'^{<30}$	3 274	18 320	3.9	5.6	2.6	8.9	12.6	5.6	2.6
	$8'^{<30}$	228	1 988	0.4	8.7	1.5	10.4	12.4	4.3	0.7

Tab. 2 shows the occurrence and other statistics of patterns 1, 2, 3, 4, 7' and 8' in the BPIC'14 and BPIC'17 data. We observe that P4 (multiple actions by same actor) makes up for the largest share of events in both logs (77.1% and 98.6% for BPIC'14 and BPIC'17, respectively) and has an average duration of 21.6 and 2.9 minutes and an average waiting time of 16.5 and 22.6 minutes between every pair of successive P4 instances for BPIC'14 and BPIC'17, respectively. It is therefore likely that these instances are composed of a single task execution as opposed to multiple tasks executions, rejecting the general assumption that a task execution is a single step executed by a single actor in a single case. We see P3 (actors interrupt a task execution and switch to another case) more often in BPIC'14 (1.6%+8.6%) than in BPIC'17 (0.2%), showing that actors work differently in different processes. Of these P3 interruptions in the BPIC'14 data, 86% lasted less than 10 seconds. We observe that almost half of the interruptions P2 in BPIC'14 last more than 30 minutes, indicating that actors often switch context for long periods at a time. Task executions interrupted by waiting for another actor (P3) make up 10.2% of the BPIC'14 data, with a minor part lasting longer than 30 minutes, indicating either very long breaks or tasks in another process context not recorded in the data. TIs of P2 and P3 contain on average 2 other elementary TIs of P1 or P4 (last column). Executions of batch patterns P7' and P8' comprise 0.2+7.6% of the BPIC'17 data; although multi-step batches (P8') have > 5 times as many steps as single-step batches (P7') they only take twice as long in duration; indicating large deviations in executions of batching tasks in the BPIC'17 data. We observe a mean duration of 21.6m for analyzing problems in IT components (BPIC'14) and 2.9m for handling loan applications (BPIC'17) for elementary TIs, confirming our intuition that a task execution is short.

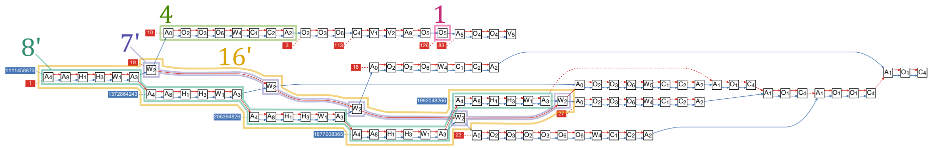
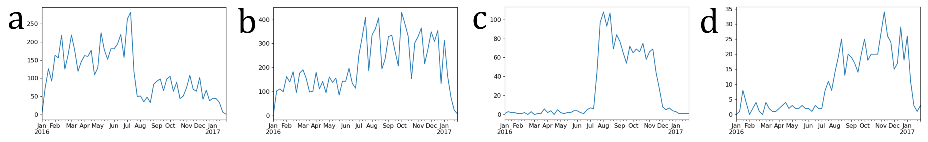


Fig. 6: Event graph of loan applications 1111458873, 1372864243, 206394826, 1877008365 and 1992048266 in BPIC'17 revealing five different task execution patterns

To investigate the manifestation of different task execution patterns on a case level, we visualized the events of five process executions in Fig. 6, revealing instances of five different task execution patterns P1, P4, P7', P8', and notably the most complex P16'. In Fig. 6, we observe six instances of P1 (one separately and five as part of P7'). All other task instances are of type P4, meaning that most actors perform tasks over multiple steps in the same case before handing the case over to the next actor. Existing process discovery techniques lack the resource perspective necessary to actually structure a trace into a sequence of P4 (and P1) instances, required to reveal these handovers of work. We observe $r1$ executing $\langle A_4, A_8, H_1, H_3, W_1, A_3\rangle$ in a batch for five cases in row (P8') and $r19$ executes W_2 for the same fives cases in a batch (P7') directly afterwards. The instances of P8' and P7' together form an instance of P16' along the cases. While domain knowledge is required to verify whether this instance of P16' is intentional, we show that structured task executions involving multiple resources and cases exist in the data. This particular type of structured collaboration over multiple steps suggests a routine; confirming this requires further investigation.

We finally explored whether we find evidence for task executions changing over time, as stipulated in [14]. We queried the frequency of all task executions in BPIC'17 over time; Fig. 7 shows 4 selected task executions T_a-T_d and their frequency. T_a changes into T_b in July 2016 by changing only the first action from V_5 (lifecycle complete) to V_4 (lifecycle abort), suggesting a minor change in directive but not a change in the way people work. We see similar lifecycle changes in other task executions of the BPIC'17 data (not shown here). Around the same time, T_c and T_d emerge, which both contain T_a at the end suggesting that also



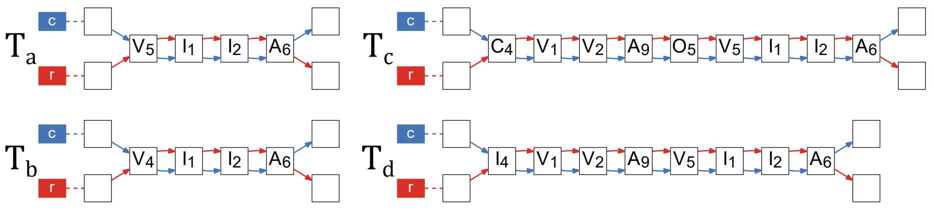


Fig. 7: Trends and subgraphs of four elem. task executions T in BPIC'17 showing concept drift

the way people worked changed. In total we found 6 more task executions with this characteristic that also show this change.

7 Conclusion

In this paper, we considered event data along both the case and actor dimension. Doing this in event graphs revealed different ways in which non-trivial tasks manifest as patterns in the data and uncovered dynamics that have not been described before in established process mining and modeling; these range from interruptions and batching to the more complex production-type settings.

This lays the foundation for a fundamentally different way of conceptualizing processes as the interplay of cases and actors engaging in recurrent patterns of work, i.e., routines and habits as studied routines research [2]. We found evidence in existing real-life event logs that such patterns make up the larger share of events. We believe the taxonomy of task patterns aids in task mining and many other process analysis problems where actor and case perspectives meet. For example, the relation between certain actor behavior and process performance or process outcomes, the adherence to queuing policies [25] as well as questions related to the study of complete systems of processes, resources and queues together [6].

Our approach is limited in that our taxonomy does not cover the entire spectrum of possible task patterns; we currently do not account for graph-structure-based patterns with a less strict synchronization of paths, while real-life manifestations thereof do exist. This includes other possible patterns or aspects that may have been overlooked. Some patterns of the taxonomy involving multiple actors and cases were not found in the data as such data was not available. However, such patterns do exist, e.g., delivering and installing a washing machine by two actors does happen. A next step is finding and exploring other data for the existence of these patterns. Important to note is that our work does not identify a task itself but only the patterns of actions used to achieve a task. Finding out what these patterns mean and what real-life tasks they portray is future work.

References

1. van der Aalst, W.M.P.: Process Mining: Data Science in Action. Springer (2016)
2. Becker, M.C.: Organizational routines: a review of the literature. Industrial and Corporate Change **13**(4), 643–678 (2004)
3. Bonifati, A., Fletcher, G.H.L., Voigt, H., Yakovets, N.: Querying Graphs. Synthesis Lectures on Data Management, Morgan & Claypool Publishers (2018)
4. Debois, S., López, H.A., Slaats, T., Andaloussi, A.A., Hildebrandt, T.T.: Chain of events: Modular process models for the law. In: IFM. LNCS, vol. 12546, pp. 368–386. Springer (2020)
5. Delcoucq, L., Lecron, F., Fortemps, P., van der Aalst, W.M.P.: Resource-centric process mining: clustering using local process models. In: SAC. pp. 45–52. ACM (2020)

6. Denisov, V., Fahland, D., van der Aalst, W.M.P.: Repairing event logs with missing events to support performance analysis of systems with shared resources. In: PetriNets. LNCS, vol. 12152, pp. 239–259. Springer (2020)
7. van Dongen, B.F.: BPI Challenge 2014. Dataset (2014), `https://doi.org/10.4121/uuid:c3e5d162-0cfd-4bb0-bd82-af5268819c35`
8. van Dongen, B.F.: BPI Challenge 2017. Dataset (2017), `https://doi.org/10.4121/12705737.v2`
9. Dumas, M., Rosa, M.L., Mendling, J., Reijers, H.A.: Fundamentals of Business Process Management. Springer, 2 edn. (2018)
10. Esser, S., Fahland, D.: Multi-dimensional event data in graph databases. J. Data Semant. **10**, 109–141 (2021)
11. Fahland, D.: Describing behavior of processes with many-to-many interactions. In: PetriNets. LNCS, vol. 11522, pp. 3–24. Springer (2019)
12. Fdhila, W., Gall, M., Rinderle-Ma, S., Mangler, J., Indiono, C.: Classification and formalization of instance-spanning constraints in process-driven applications. In: BPM. LNCS, vol. 9850, pp. 348–364. Springer (2016)
13. Gall, M., Rinderle-Ma, S.: Visual modeling of instance-spanning constraints in process-aware information systems. In: CAiSE. LNCS, vol. 10253, pp. 597–611. Springer (2017)
14. Goh, K., Pentland, B.: From actions to paths to patterning: Toward a dynamic theory of patterning in routines. The Academy of Management Journal **62**, 1901–1929 (12 2019)
15. Leno, V., Polyvyanyy, A., Dumas, M., La Rosa, M., Maggi, F.M.: Robotic Process Mining: Vision and Challenges. BISE (2020)
16. Martin, N., Depaire, B., Caris, A., Schepers, D.: Retrieving the resource availability calendars of a process from an event log. Inf. Syst. **88** (2020)
17. Martin, N., Pufahl, L., Mannhardt, F.: Detection of batch activities from event logs. Inf. Syst. **95**, 101642 (2021)
18. Martin, N., Swennen, M., Depaire, B., Jans, M., Caris, A., Vanhoof, K.: Retrieving batch organisation of work insights from event logs. Decis. Support Syst. **100**, 119–128 (2017)
19. Pentland, B., Feldman, M.: Narrative networks: Patterns of technology and organization. Organ. Sci. **18**, 781–795 (2007)
20. Pentland, B., Feldman, M., Becker, M., Liu, P.: Dynamics of organizational routines: A generative model. J. of Mngmt. Studies **49**, 1484–1508 (12 2012)
21. Pika, A., Leyer, M., Wynn, M.T., Fidge, C.J., ter Hofstede, A.H.M., van der Aalst, W.M.P.: Mining resource profiles from event logs. ACM Trans. Manag. Inf. Syst. **8**(1), 1–30 (2017)
22. Pufahl, L., Weske, M.: Batch activity: enhancing business process modeling and enactment with batch processing. Computing **101**(12), 1909–1933 (2019)
23. Russell, N., van der Aalst, W.M.P., ter Hofstede, A.H.M., Edmond, D.: Workflow resource patterns: Identification, representation and tool support. In: CAiSE. LNCS, vol. 3520, pp. 216–232. Springer (2005)
24. Schönig, S., Cabanillas, C., Ciccio, C.D., Jablonski, S., Mendling, J.: Mining resource assignments and teamwork compositions from process logs. Softwaretechnik-Trends **36**(4) (2016)
25. Senderovich, A., Rogge-Solti, A., Gal, A., Mendling, J., Mandelbaum, A., Kadish, S., Bunnell, C.A.: Data-driven performance analysis of scheduled processes. In: BPM. LNCS, vol. 9253, pp. 35–52. Springer (2015)
26. Song, M., van der Aalst, W.M.P.: Towards comprehensive support for organizational mining. Decis. Support Syst. **46**(1), 300–317 (2008)

27. Winter, K., Stertz, F., Rinderle-Ma, S.: Discovering instance and process spanning constraints from process execution logs. Inf. Syst. **89**, 101484 (2020)

A Formal Definitions

This appendix provides formal definitions of the graph concepts used in Sect. 3 and 4 and is available at the reviewers' discretion.

A.1 Event Graphs

The following definitions summarize the concepts of event data in labeled property graphs [10] as they are used in the paper an introduced in Section 3.

A *labeled property graph* (LPG) $G = (N, R, \lambda, \#)$ is a graph with nodes N, and relationships R where each relationship $r \in R$ defines a directed edge $\overrightarrow{r} \in N \times N$ between two nodes. Each node $n \in N$ carries a label $\lambda(n) \in \Lambda_N$; each relationship $r \in R$ carries a label $\lambda(r) \in \Lambda_R$. We write $N^\ell = \{n \in N \mid \lambda(n) = \ell\}$ and $R^\ell = \{r \in R \mid \lambda(r) = \ell\}$ for the nodes and relationships with label ℓ, respectively. Any node n and relationship r can carry properties via function $\# : (N \cup R) \times \mathit{Attr} \to \mathit{Val}$ of attribute-value pairs. We write $x.a = v$ for $\#(x, a) = v$ and $x.a = \perp$ if a is undefined for x.

An *event graph* is an LPG $G = (N, R, \lambda, \#)$ with node labels $\Lambda_N \in \{\mathit{Event}, \mathit{Entity}\}$ and relationship labels $\Lambda_R = \{\mathit{df}, \mathit{corr}\}$ with the following properties.Every event node $e \in N^{\mathit{Event}}$ records an action name $e.\mathit{action} \neq \perp$ and a timestamp $e.\mathit{time} \neq \perp$. Every entity node $n \in N^{\mathit{Entity}}$ has an entity type $n.\mathit{type} \neq \perp$; in this paper $n.\mathit{type} \in \{\mathit{Case}, \mathit{Resource}\}$ is either a case (of the process) or an actor (working in the process).

Every correlation relationship $r \in R^{\mathit{corr}}$ is defined from an event node to an entity node $\overrightarrow{r} = (e, n), e \in N^{\mathit{Event}}, n \in N^{\mathit{Entity}}$. The set of entities to which e is correlated is $\mathit{corr}(e) = \{n \mid \exists r \in R^{\mathit{corr}} \overrightarrow{r} = (e, n)\}$; note that this set can contain any number of entity nodes and even be empty. The set of events correlated to entity n is $\mathit{corr}(n) = \{e \mid \exists r \in R^{\mathit{corr}} \overrightarrow{r} = (e, n)\}$

Any directly-follows relationship $r \in R^{\mathit{df}}$ is defined between event nodes $\overrightarrow{r} = (e_1, e_2), e_1, e_2 \in N^{\mathit{Event}}$ and has an entity type $r.\mathit{type} \neq \perp$ so that (1) e_1 and e_2 are correlated to the same entity $n \in \mathit{corr}(e_1) \cap \mathit{corr}(e_2)$ with $n.\mathit{type} = r.\mathit{type}$, and (2) e_1 occurs before e_2: $e_1.\mathit{time} < e_2.\mathit{time}$, and (3) there is no other event $e_3 \in N^{\mathit{Event}}$ related to $n \in \mathit{corr}(e_3)$ that occurs in between $e_1.\mathit{time} < e_3.\mathit{time} < e_2.\mathit{time}$. In this paper we assume R to be complete wrt. all *df* relationships, i.e., it contains all *df* relationships for the given *corr* relationships.

A.2 DF-Paths

The following definitions formalize the notion of *df*-paths as introduced in Sect. 3.

Let $G = (N, R, \lambda, \#)$ be an event graph. A *df-path for entity* $n \in N^{\mathit{Entity}}$ is a sequence $\sigma = \langle r_1, \ldots, r_k \rangle$ of *df*-relationships $r_i \in R^{\mathit{df}}$ with $r_i.\mathit{type} = n.\mathit{type}$ so that there exist events $e_0, \ldots, e_k \in N^{\mathit{Event}}$ with

1. $\overrightarrow{r_i} = (e_{i-1}, e_i), 1 \leq i \leq k$, and
2. $\{e_0, \ldots, e_k\} \subseteq \mathit{corr}(n)$

We write $\sigma^{\mathit{Event}} = \langle e_0, \ldots, e_k \rangle$. A *df*-path does not have to be maximal, but it has to be continuous. A *df*-path σ for entity n is *complete* iff $\sigma^{\mathit{Event}} = \langle e_0, \ldots, e_k \rangle$ and $\{e_0, \ldots, e_k\} = \mathit{corr}(n)$.

We also call a *df*-path σ for an entity n with $n.\mathit{type} = T$ a T-*df*-path or just T-path, e.g., a *Resource*-path or a *Case*-path; we write $\sigma.\mathit{type} = T$ to denote the type.

A.3 DF-Paths entering and leaving Event Subgraphs

The following definitions formalize the notion of df-paths entering and leaving subgraphs of an event graph as discussed in Sections 3 and 4.1.

An event graph $G' = (N', R', \lambda', \#')$ is an *event subgraph* of $G = (N, R, \lambda, \#)$ if $N' \subseteq N, R' \subseteq R, \lambda' \subseteq \lambda, \#' \subseteq \#$ so that all $n \in N'$ are event nodes $\lambda(n) = \mathit{Event}$. Let G' be an event subgraph of an event graph G.

- A *df*-path σ with $\sigma^{\mathit{Event}} = \langle e_0, \ldots, e_k \rangle$ *enters* (*leaves*) G' at event e_i iff $e_i \in N'$ and either $i = 0$ or $e_{i-1} \notin N'$ (either $i = k$ or $e_{i+1} \notin N'$).
- σ *enters* G' *first* (*leaves* G' *last*) at event e_i iff e_i enters (leaves) G' at e_i and there is no $e_j, j < i$ $(j > i)$ where σ enters (leaves) G'.
- σ *enters (leaves)* G' *once* at event e_i if there is no other event e_j where σ enters (leaves) G'.
- σ is *single* in G' iff $\sigma = \langle e_1 \rangle$; σ is *continuous* in G' iff σ enters and leaves G'; otherwise σ is *interrupted* in G' (enters and leaves G' more than once)

If $\sigma = \langle r_1, \ldots, r_k \rangle$ is a continuous path in G' then we write $\sigma \cap G' = \langle r_i, \ldots, r_j \rangle$ for the path within G', i.e., σ enters G' at e_{i-1}, $\overrightarrow{r_i} = (e_{i-1}, e_i)$ and leaves G' at e_j, $\overrightarrow{r_j} = (e_{j-1}, e_j)$. We write $\sigma^{\mathit{Event}} \cap G' = \langle e_{i-1}, \ldots, e_j \rangle$.

A.4 Task Execution Subgraphs formed by Synchronizing DF-Paths

The following definitions formalize the notion of task execution patterns in an event graph as discussed in Section 4.2.

Let $\sigma_1, \ldots, \sigma_n, n \geq 2$ be a set of complete *df*-paths in event graph G. Let G' be an event subgraph of G.

Paths $\Sigma = \{\sigma_1, \ldots, \sigma_n\}$ *synchronize in a task execution* described by subgraph G' iff all following conditions hold:

(T1) For any two different event nodes $e, e' \in N'Event, e \neq e'$ in G' exists a *df*-path $\sigma_i, 1 \leq i \leq n$ containing both events $e, e' \in \sigma_i^{\mathit{Event}}$.
(T2) For each event node $e \in N'Event$ in G' exists a case *df*-path $\sigma_c, 1 \leq c \leq n$ with $\sigma_c.\mathit{type} = \mathit{case}$ and resource *df*-path $\sigma_r, 1 \leq r \leq n$ with $\sigma_r.\mathit{type} = \mathit{resource}$ so that $e \in \sigma_c^{\mathit{Event}}$ and $e \in \sigma_r^{\mathit{Event}}$.
(T3) There is at least one $\sigma \in \Sigma$ continuous in G'.

Paths $\Sigma = \{\sigma_1, \ldots, \sigma_n\}$ *strongly synchronize in a graph-structure based task execution* described by subgraph G' iff all following conditions hold (T1), (T2), (T3) and additionally

(T2') For each event node $e \in N'Event$ in G' holds: $e \in \sigma^{Event}$ of each $\sigma \in \Sigma$ (e occurs in each path in Σ.

Paths $\Sigma = \{\sigma_1, \ldots, \sigma_n\}$ *weakly synchronize in a domain-knowledge based task execution* described by subgraph G' iff all following conditions hold (T1), (T2), (T3) and additionally

(T3') Each *df*-path $\sigma \in \Sigma$ is continuous in G'

B Cypher Queries

This appendix provides the Cypher implementations of the queries for task instances presented in Sect. 5 and is available at the reviewers' discretion.

B.1 Query for Materializing Elementary Task Instances

The following Cypher query constructs the TI nodes and all corresponding relationships as described in Sect. 5.1.

```
1   CALL {
2      MATCH (e1:Event)-[:DF {EntityType:"joint"}]->()
3        WHERE NOT ()-[:DF {EntityType:"joint"}]->(e1)
4      MATCH ()-[:DF {EntityType:"joint"}]->(e2:Event)
5        WHERE NOT (e2)-[:DF {EntityType:"joint"}]->()
6      MATCH p=(e1)-[:DF*]->(e2) WHERE all(r IN relationships(p)
7        WHERE (r.EntityType = "joint"))
8      RETURN p, e1, e2
9   UNION
10     MATCH (e:Event) WHERE NOT ()-[:DF {EntityType:"joint"}]->(e)
11       AND NOT (e)-[:DF {EntityType:"joint"}]->()
12     MATCH p=(e) RETURN p, e AS e1, e AS e2
13  }
14  WITH [event IN nodes(p) | event.Action_Lifecycle] AS path,
15    nodes(p) AS events, e1, e2
16  CREATE (h:TaskInstance {path:path, rID:e1.resource, cID:e1.case,
17    start_time:e1.timestamp, end_time:e2.timestamp})
18  WITH h, events
19  UNWIND events AS e
20  CREATE (h)-[:CONTAINS]->(e)
```

Lines 1-17 are for detecting the TI subgraphs. The *CALL* {} clause (lines 1-14) evaluates two subqueries: one to detect all subgraphs of P4 (lines 2-8) and one to detect all subgraphs of P1 (lines 10-12); the results of these subqueries are then combined using the *UNION* clause (line 9). The *MATCH* clause in line 2 retrieves all event nodes `e1` for which there exists an outgoing joint DF-edge and the *WHERE* clause in line 3 restricts those nodes to not have an ingoing joint DF-edge (e.g., $e1, e3, e6$ and $e9$ in Fig. 4). Conversely, the query in lines 4-5 retrieves all event nodes `e2` for which there exists an ingoing joint DF-edge and no outgoing joint DF-edge (e.g., $e2, e4, e8$ and $e10$ in Fig. 4). Lines 6-7 then retrieve all possible paths from a node `e1` to a node `e2` of an arbitrary amount of joint DF-edges (e.g., $\langle e1, e2\rangle$, $\langle e3, e4\rangle$, $\langle e6, e7, e8\rangle$ and $\langle e9, e10\rangle$ in Fig. 4). The *RETURN* statement in line 8 defines what to include in the subquery result. We similarly detect the subgraphs of P1 (e.g., $\langle e5\rangle$ in Fig. 4) in lines 10-12 and return the results using the same format in line 13. *WITH* in lines 14-15 then manipulates the output before it pipes the results to the next query, specifically carrying over all event nodes included in the path. Next, we create a h_{ti} node for each TI that we detect (the grey rectangular nodes in Fig. 4). The query in lines 16-17 creates a node with the label *TaskInstance* and adds properties (path, resource, case, start_time, end_time) of the subgraph it corresponds to using the results of the previous query. We then carry over the created TI nodes and the previously carried over list of event nodes (line 18), transform the latter back into individual events (line 19) and, finally, correlate each TI to its corresponding events using a *CONTAINS* relationship (line 20) (the grey dashed edges in Fig. 4).

B.2 Example query for detecting NTIs of P8'

The following Cypher query detects all instances of non-elementary task execution pattern 8'. We assume a maximum of 30 minutes between cases for sequential batch processing.

```
1   MATCH (h1:TaskInstance) WHERE NOT (:TaskInstance {task:h1.task})
2     -[:DF {EntityType:"resource"}]->(h1) AND size(h1.path) > 1
3   MATCH (h2:TaskInstance) WHERE NOT (h2)
4     -[:DF {EntityType:"resource"}]->(:TaskInstance {task:h2.task})
5     AND size(h2.path) > 1 AND h2.task=h1.task
6   MATCH p=(h1)-[:DF*3..]->(h2)
7   WHERE all(r IN relationships(p) WHERE (r.EntityType = "resource"))
8     AND all(n IN nodes(p) WHERE n.task = h1.task) AND all(idx IN
9     range(0, size(nodes(p)) - 2) WHERE datetime((nodes(p)[idx]).end_time) >
10    (datetime((nodes(p)[idx+1]).start_time) - duration("PT30M")))
11  RETURN p
```

In lines 1-2, we query the start of the path `h1` that does not have an incoming resource DF-edge from a TI node describing the same task execution and has a path length > 1 (i.e., is a P4 instance). In lines 3-5 we query the end of the path `h2` using the inverse of the previous query and we additionally constrain `h2` to be the same task execution as `h1`. Lines 6-10 then retrieve all possible paths from h1 to h2 of at least 3 resource DF-edges containing only TIs of the same task execution timestamped at most 30 minutes apart, which are returned in line 11.

C Legend for action name abbreviations

The following legend maps the abbreviations used in Figures 5 to 7 to their full written action names.

A_0	: A_Accepted+COMPLETE	C_1	: W_Call after offers+SCHEDULE
A_1	: A_Cancelled+COMPLETE	C_2	: W_Call after offers+START
A_2	: A_Complete+COMPLETE	C_4	: W_Call after offers+ATE_ABORT
A_3	: A_Concept+COMPLETE	I_1	: W_Call incomplete files+SCHEDULE
A_4	: A_Create Application+COMPLETE	I_2	: W_Call incomplete files+START
A_5	: A_Denied+COMPLETE	I_4	: W_Call incomplete files+ATE_ABORT
A_6	: A_Incomplete+COMPLETE	W_1	: W_Complete application+SCHEDULE
A_7	: A_Pending+COMPLETE	W_2	: W_Complete application+START
A_8	: A_Submitted+COMPLETE	W_4	: W_Complete application+ATE_ABORT
A_9	: A_Validating+COMPLETE	W_5	: W_Complete application+COMPLETE
O_1	: O_Cancelled+COMPLETE	H_1	: W_Handle leads+SCHEDULE
O_2	: O_Create Offer+COMPLETE	H_3	: W_Handle leads+WITHDRAW
O_3	: O_Created+COMPLETE	V_1	: W_Validate application+SCHEDULE
O_4	: O_Refused+COMPLETE	V_2	: W_Validate application+START
O_5	: O_Returned+COMPLETE	V_4	: W_Validate application+ATE_ABORT
O_6	: O_Sent (mail and online)+COMPLETE	V_5	: W_Validate application+COMPLETE