Inferring Missing Entity Identifiers from Context using Event Knowledge Graphs

Ava J.E. Swevels[1 [0000−0003−0992−208X]], Remco M. Dijkman[1 [0000−0003−4083−0036]], and Dirk Fahland[1 [0000−0002−1993−9363]]

Eindhoven University of Technology, Eindhoven, The Netherlands {a.j.e.swevels, r.m.dijkman, d.fahland}@tue.nl

Abstract. Complete event data is essential to perform rich analysis. However, real-life systems might fail in recording the (correct) case identifiers the system has operated on, resulting in incomplete event data. We aim to infer missing case identifiers of events by addressing the physical constraints of its process which previous work has failed to do. We modelled event data and its physical context in an Event Knowledge Graph (EKG) and formalized a definition for inference rules using EKGs. Five inference rules regarding physical objects are created to infer identifiers in a synthetic data set and a data set from the IC manufacturing industry. The approach is evaluated using conformance checking. Initially, none of the traces were complete. Using our method, we could infer a case identifier for 95% of the events resulting in 88% complete traces.

Keywords: Log repair · Event Knowledge Graph · Modeling · Inference Rule · Contextual Information · Physical Constraints

1 Introduction

Business Process Analytics (BPA) is an area of data analytics that facilitates rich analysis of the way in which business processes work, providing insight into how business processes can be improved. BPA techniques rely on event logs that record the events that happened in the business process, along with (the identifier of) the case to which they belong and the moment in time at which they happened, possibly extended with other information. However, in real-life systems the data in the event logs may be incomplete [18]. Consequently, before event data can be used for BPA, missing data must be added or incomplete events must be removed. If missing data can somehow be *inferred* from the context of the events, that is preferred, because it leads to more usable data to work with. This work focuses on inferring missing case identifiers, also known as the "event-case correlation" problem [5].

Several methods exist to infer missing case identifiers from context. The dominant context information that these methods use for inferring case identifiers is a process model [11,16]. This has some clear drawbacks. Firstly, a process model may not exist or may be hard to create, for example in multi-entity processes [9] or in case the data is at a different level of abstraction than the level at which

the process is understood. Secondly, inference for cyclic processes either requires further context information, such as constraints on time [4] or data [5], which may not be available either, or it may require time-consuming iterations [6] to do the inference. Thirdly, most existing methods only connect the incomplete information to the context information within the algorithm. Only [6] makes the connection between data and context available for further analysis.

To alleviate these problems, we aim to develop techniques for inferring missing identifiers *without* relying on a process model. This paper focuses on inferring missing case identifiers for *processes with batching* when context is available in the form of information about *physical objects*. Since these objects are bound by the laws of physics, information about them can be used in combination with simple physics rules to infer other information. For example, if a physical object is in one place in one event and in another place in another event, it must have been moved in between and a movement event must have involved this object.

Specifically, this paper proposes a technique to infer missing case identifiers when context information is available about locations at which activities are executed. We translate the incomplete event log into an (incomplete) *Event Knowledge Graph* (EKG) [9] and show how to extend this EKG with context knowledge about locations. An EKG models multi-dimensional event data considering multiple entity perspectives. Therefore, there are *no* cases in an EKG and hence the notion of a single case identifier does not exist. Instead the notion of multiple *entity identifiers* is used. Based on EKGs, containing data extended with context, we introduce pattern-based inference rules. The inference rules define how missing entity identifiers can be inferred from context information. The rules are implemented as queries in a graph database.

We evaluate the technique on an industrial case study with NXP Semiconductors by showing that it can be used to infer missing entity identifiers. Using our method on event data of 7250 events, where 86% of the events lacked an entitiy identifier, we could infer an identifier for 95% of the events within 30 seconds. Subsequent conformance checking against a normative process model validated the correctness of the inferred identifiers.

Against this background, the remainder of the paper is structured as follows. The related work is discussed in Sect. 2. The problem is elaborated on in Sect. 3 along a running example. Sect. 4 describes a method for inferring identifiers from context using an EKG. Sect. 5 shows the proposed method by creating inference rules for the running example and industrial use case and validates the inference using conformance checking. The findings are discussed in Sec. 6.

2 Related work

Data-driven process analysis defines minimal requirements for event logs [12]: each event contains at least an activity, timestamp and a case identifier. However, in practice, collected event data might not meet these requirements.

Event data quality issues are assessed in terms of missing or incorrectly recorded attribute values. These typically manifest themselves in systematic “im-

Table 1. Overview of types of (incomplete) data

Timestamp	Activity	Case Identifier	Context Knowledge used for Inference
-	✓	✓	timing information [17,10]
✓	-	✓	other attributes to activities [13,20,2]
✓	✓	-	acyclic process model [11,16]; process model + add. constraints [4,5]; process model + sim. annealing [6]; surrogate id [15]; activity properties **[this]**

perfection patterns" [18] due to imperfect data recording mechanism. Accordingly, specific techniques have been proposed for detecting if such data quality 'patterns' exist in an event log [1] and for repairing them if they exist, as we discuss below. Repairing the data enables us to apply techniques that require complete data such as rich analysis techniques and traceability.

Assuming correct activity and case attributes have been identified [3,14], we focus on *missing values* for the standard attributes of case/object identifier, activity, and timestamp. Existing literature infers missing values for one of the attributes based on information in the other attributes and *additional context knowledge* as summarized in Tab. 1.

Missing timestamps can be inferred by using knowledge of duration of process steps, both, for isolated cases [17] and cases processed via shared resources and queues [10]. *Missing activities* can be inferred by knowledge of how events with specific data attributes in a specific behavioral context relate to activities, e.g., by aggregating low-level observations to activities [13,20] or matching text attributes to activity descriptions [2].

Missing case identifiers are typically inferred by leveraging control-flow knowledge. Existing techniques use a given acyclic probabilistic Markov model [11] or first estimate a model of an acyclic process from an incomplete log and then infer case identifiers [16]. Inferring identifiers for cyclic processes requires, next to a process model, either additional timing constraints [4], data constraints [5], or multiple iterations of matching, e.g., through time-costly simulated annealing [6] that also generate rules for correlating events to cases based on the activity, event properties, and their immediate context. No process model is required when a surrogate identifier such as the user in a clickstream is present [15]. A common trait of all techniques is that they assume isolated cases (no batching) and the context knowledge is kept separate from the incomplete event data and reconciled within the algorithmic technique itself.

This paper addresses the problem of inferring missing entity identifiers in processes *with batching* but *without* leveraging a process model defining the control-flow or surrogate identifiers. Instead, we focus on processes handling *physical objects* and use knowledge on the activities themselves and their (abstract) locations to infer entity identifiers. Moreover, we show how to reconcile context information and incomplete event data within the same data model by leveraging *event knowledge graphs* [9] that naturally allow incorporating additional concepts, such as 'activity', 'location', and 'object' for representing event logs.

3 Missing Identifiers in Processes with Physical Objects

We illustrate the problem of identifying from incomplete data which physical objects in a process suffered from errors. We show by a simple example that existing inference techniques fail to reliably infer correct entity identifiers for processes with physical objects and batching. We illustrate how we can reliably infer entity identifiers when applying simple physical constraints and contextual knowledge about activities and their locations. Then, we state the specific problem together with the expected in- and output.

Notation on Event Data. We first recall some notation on event data over multiple identifiers, i.e., object-centric event data, based on [9]. We write Val for the universe of values, including disjoint sets of activity names and timestamps $Act, Time \subseteq Val$. $Time$ is totally ordered by $\leq$.

An *event table with entities* (i.e., object-centric log) $T = (E, Attr, Ent, \#)$ consists of events E, attributes $\{act, time\} \subseteq Attr$, entity type attributes $\emptyset \neq Ent \subseteq Attr$, and partial attribute value function $\# : E \times Attr \nrightarrow Val$ assigning $e \in E$ and $a \in Attr$ value $\#_a(e) = v$ ($\#_a(e) = \perp$ if a is undefined for e) with $\#_{time}(e) \in Time$ and $\#_{act}(e) \in Act$ are defined.

An event e may have a multi-valued attribute $\#_a(e) = \{v_1, \ldots, v_n\}$ (set) or $\#_a(e) = \langle v_1, \ldots, v_n \rangle$ (list) for example for events referring to multiple objects; and we write $v_i \in \#_a(e)$. For uniform notation, we also write $v \in \#_a(e)$ for single-valued $\#_a(e) = v$.

In the generalized setting of object-centric or multi-entity processes [9] a trace is defined in relation to an entity identifier. Let $ent \in Ent$ be an entity type. The entity identifiers of type $ent \in T$ are $ent(T) = \{n \mid n \in \#_{ent}(e), e \in E\} \setminus \{\perp\}$. Let $n \in ent(T)$ be an entity identifier. An *entity trace* of n is a sequence $\pi_n = \langle e_1 \ldots e_k \rangle$ of events $\{e_1 \ldots e_k\} = \{e \in E \mid n \in \#_{ent}(e)\}$ ordered by $\#_{time}(e_i) \leq \#_{time}(e_j)$ for $1 \leq i < j \leq k$. In the following, we specifically discuss (identifiers of) entities that are *physical objects*, e.g., a box.

Running example. Figure 1 shows a process, modeled as a proclet [10] (bottom), in the physical environment in which it is executed (top). There are two *Assembly Line*s at which *Box*es are filled and sealed. A *Tray* of *Box*es, i.e. a

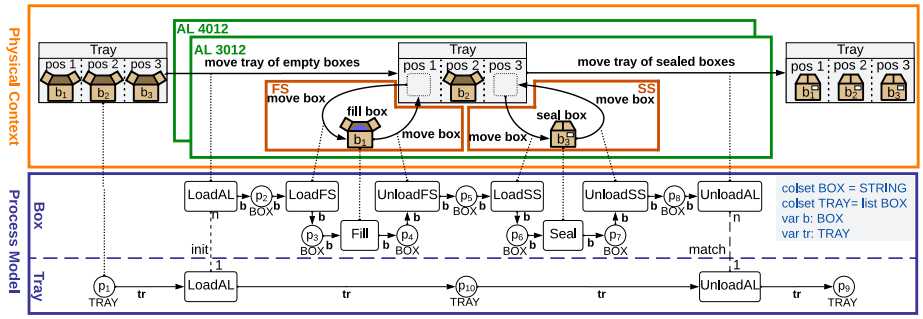

Fig. 1. A process model (proclets [10]) of assembly lines where boxes in a tray are filled, sealed, and labeled (bottom) together with the physical context (top).

LOG

	activity	time	b	p	equipment
e_1	LoadAL	t_1			3012
e_2	LoadFS	t_2		2	3012
e_3	Fill	t_3			3012
e_4	UnloadFS	t_4		2	3012
e_5	LoadFS	t_5		1	3012
e_6	Fill	t_6			3012
e_7	UnloadFS	t_7		1	3012
e_8	LoadFS	t_8		3	3012
e_9	LoadSS	t_9		1	3012
e_{10}	Seal	t_{10}	b_1		3012
e_{11}	Fill	t_{11}			3012
e_{12}	UnloadFS	t_{12}		3	3012
e_{13}	UnloadSS	t_{13}		1	3012
e_{14}	LoadSS	t_{14}		2	3012
e_{15}	Seal	t_{15}	b_2		3012
e_{16}	UnloadSS	t_{16}		2	3012
e_{17}	LoadSS	t_{17}		3	3012
e_{18}	Seal	t_{18}	b_3		3012
e_{19}	UnloadSS	t_{19}		3	3012
e_{20}	UnloadAL	t_{20}			3012

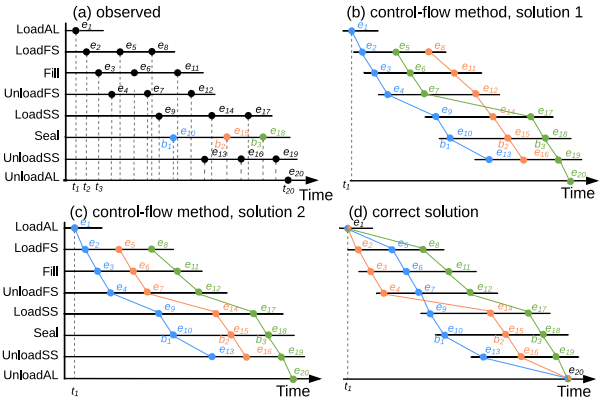

Fig. 2. A small log with missing identifiers: observed events with missing entity identifiers (a), behaviors found with a control-flow method (b, c) and correct behavior (d).

batch, is loaded into an *Assembly Line* if it is *Empty*, then each *Box* individually passes several stations and finally the *Tray* is unloaded from the *Assembly Line*. The first station is the *Fill Station*, at which a *Box* is loaded, filled and unloaded. The second station is the *Seal Station* at which a *Box* is loaded, sealed and unloaded. *Loading* and *Unloading* of the stations is done by a robot arm that needs to be aware of the *Pos*ition of the *Box* in the *Tray*.

Only when *Seal*ing, each *Box* is labeled making it uniquely identifiable. Hence only *Seal* events record an entity identifier for the box. Further, *Load* and *Unload* events register per station the *Pos*ition of a *Box* in the *Tray*. For instance, processing three boxes results in the *incomplete* log shown in Fig. 2 where attribute b records the box identifier and p the position in the tray. No event records, both, b and p. Fig. 2(a) visualizes this incomplete log as a *Performance Spectrum* [7]. Note that no complete traces of boxes can be constructed.

Suppose a *Fill*ing error occurred at e_3. We cannot determine which box was affected by the error on the incomplete log. As a result, an operator has to inspect and possibly even discard the entire *Tray*. To mitigate this problem, we have to obtain complete traces by inferring the likely values for b.

Using control-flow models as context knowledge, e.g., [11,16] allows to infer unknown value $\#_b(e_i)$ from known value $\#_b(e_j)$ when e_j directly precedes or succeeds e_i; e.g., infer $\#_b(e_9) = b_1$ from $\#_b(e_{10}) = b_1$. Existing technique fail to address the physical constraints and batching (boxes in trays) in our example. The method of [16] generates multiple possible solutions of box id values and traces (depending on parameters), two are shown in Fig 2(b) and (c): besides the analyst having to pick a solution, the method wrongly relates e_1 and e_{20} to only a single box and wrongly claims b_1 went through filling and sealing first, while the log shows that the box at $p = 2$ was filled first and sealed second.

Physical constraints and context. In contrast, Fig. 2(d) visualizes the complete traces of the three boxes in line with the (physical) constraints and locations of the process. We note three basic (physical) principles that hold for this pro-

cess: (P1) for a physical object to be present at/removed from a location, the physical load/unload activity of that location must have involved this physical object; (P2) activities are performed at physical locations (stations) and the physical object they operate on must be at that location; (P3) physical objects that are consistently kept at the same (relative) spatial location can consistently be distinguished from each other (e.g. positions in the tray/on a conveyor belt).

Combining P1-P3 with context knowledge about which activities move or handle physical objects at which locations (c.f. Fig. 1) allows to infer the missing identifiers as follows: Activities *Fill* and *Seal* process boxes at distinct locations (the *Fill* and *Seal Station*), while *physical (Un)LoadSS* and *(Un)LoadFS* activities *move* a box into/out of these locations. Physical activities *LoadAL* and *UnloadAL* move a *Tray* of boxes into/out of the *AssemblyLine containing* the locations of the *Seal* and *Fill* station. (1) From $\#_b(e_{10}) = b_1$ we know that b_1 is at the *Seal Station*; by P1, box b_1 must have been moved into/out of the *Seal Station*: $\#_b(e_9) = \#_b(e_{13}) = b_1$. (2) From $\#_b(e_{10}) = b_1, \#_b(e_{15}) = b_2, \#_b(e_{18}) = b_3$ we know that b_1, b_2, b_3 are at the *Seal Station* and thus at the *AssemblyLine*; by P1 they must have been moved into/out of the *AssemblyLine* resulting in multi-valued $\#_b(e_1) = \#_b(e_{20}) = \{b_1, b_2, b_3\}$. (3) Box $\#_b(e_9) = b_1$ is at location $\#_p(e_9) = 1$ in the *Tray*; from $b_1 \in \#_b(e_1) = \#_b(e_{20})$ follows that b_1 at $p = 1$ is at the *AssemblyLine* from time t_1 to t_{20}. By P2, P3 and $\#_p(e_5) = \#_p(e_7) = 1$ we know that e_5 and e_7 must have operated on the box with $p = 1$ at the fill station, resulting in $\#_b(e_7) = b_1$. (4) From $\#_b(e_5) = \#_b(e_7) = b_1$ we know that b_1 is present at the *Fill Station* from t_5 to t_7. From P2 follows that e_6 must operate on a box present at the *Fill Station*, thus $\#_b(e_6) = b_1$. Applying this reasoning consistently infers all identifiers shown in Fig. 2(d).

The applied contextual knowledge is *not* a process model, but knowledge of the physical constraints of a process: (C1) which activities process which kind of physical objects at which locations, (C2) which physical activities move which kinds of physical objects into/out of locations, (C3) relations between locations and (C4) relations between individual physical objects and batches.

Problem statement. The problem we address in this paper is how to encode such contextual knowledge of the physical constraints of a process to infer missing entity identifiers to obtain complete traces as they likely have happened in reality. We assume as input: (I1) an event table T where each event has a timestamp, activity and the top-level location (equipment), and each entity $n \in ent(T)$ has at least one event $e' \in E, \#_{ent}(e) \neq \perp$, and (I2) contextual information about the physical constraints of a process, i.e., C1-C4 stated above. Given I1 and I2, we want to (O1) infer for each event the (likely) entities involved such that (O2) the resulting entity traces describe a consistent execution of the process matching the (physical) context. The latter can be validated by replaying the log on a process model, though the model itself is not a required input. Next, we introduce a model to encode context knowledge of physical constraints (i.e. I2) and how to use it for inference (i.e., O1,O2).

4 Inferring Entity Identifiers from Context

We now describe a new method for inferring identifiers from context. First, we illustrate the basic idea on how to define and use context information for inference. Then we model event data and its context in an *event knowledge graph* (EKG) by showing how it can be refined to reflect a use case and extended to include contextual information. Finally, we formally define an *inference rule* (IR) along with an example and the semantics of rule application on EKGs.

4.1 Basic Idea

Here we illustrate how to encode and use context information to infer missing entity identifiers on a part of the log repeated in Fig. 3.

The log has event e_{10} with activity *Seal* for box b_1 in equipment 3012. The physical context described in Fig. 1 shows that *Seal* occurs at the *Seal Station* within equipment *3012*. The log contains events e_9 and e_{13} with activities *LoadSS* and *UnloadSS* which are are not correlated to a box, i.e. $\#_b(e_9) = \#_b(e_{13}) = \perp$. Context (c.f. Fig. 1) shows that (1) activities *LoadSS* and *UnloadSS* operate on a box, thus e_9 and e_{13} have incomplete information, and (2) these activities occur as the same location as e_{10} (the *Seal Station* of equipment *3012*). From principle (P1), b_1 must have been loaded into the *Seal Station* before t_{10} and unloaded from *Seal Station* after t_{10}, e_9 and e_{13} are the first preceding load event and succeeding unload event, hence $\#_b(e_9) = \#_b(e_{13}) = b_1$. Even though events e_{11} and e_{12} occur in between e_{10} and e_{13}, they are not considered simply because they occur at a different location.

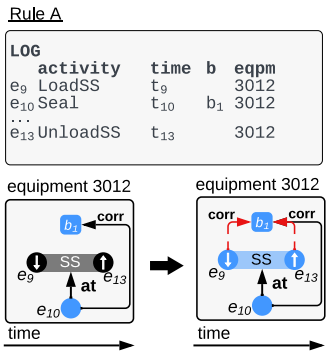

Fig. 3. Example on how to infer missing entity identifiers using context information.

Fig. 3 (bottom) schematically visualizes this reasoning over the context of the events as an inference rule. In the following subsections, we discuss how to precisely define and apply such inference rules on event data.

4.2 Modeling Context in Property Graphs

The rule and its application we illustrated in Fig. 3 reasoned over events correlated to multiple entities, and their context of activities (with properties) and locations (see C1-C4 in Sect. 3). We now show how to formally model event data with such contextual information using *event knowledge graphs* (EKGs).

We first recall the underlying data model of *labeled property graphs* (LPG), the specific model of EKGs, and how the incomplete log of Fig. 3 is modeled as an EKG. Afterwards, we extend the meta-model of EKGs with contextual information and define rules over these extended EKGs.

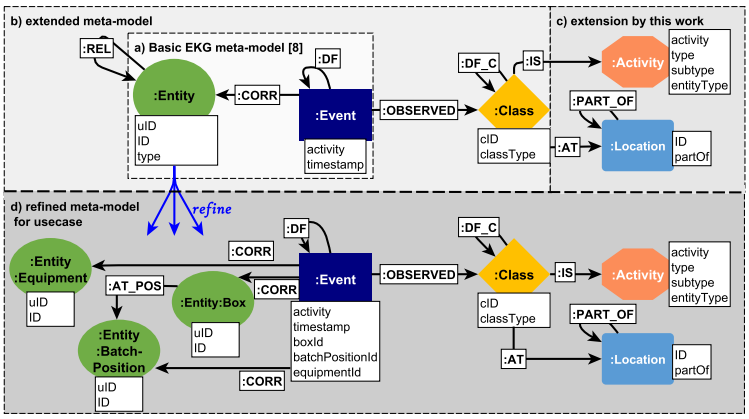

Fig. 4. Schematic meta-model; (a) basic EKG meta-model [8]; (b) extended meta-model; (c) extension by this work; (d) refined meta-model for use case.

Existing Models. An LPG $G = (N, R, Attr, \#, Lab, \lambda)$ has Nodes N, relationships R, attributes $Attr$ and a partial attribute value function $\# : (N \cup R) \times Attr \nrightarrow Val$, where $r \in R$ defines the edge $\overrightarrow{r} = (n_{src}, n_{tgt}) \in N \times N$. Each node $n \in N$ has labels $\lambda(n) \in 2^{Lab}$; each relationship $r \in R$ has a single label $\lambda(r) \in Lab$. We write $n \in \ell$ if $n \in N, \ell \in \lambda(n)$ and $(n, n') \in \ell$ or $n\ \ell\ n'$ if $r \in R$, $\overrightarrow{r} = (n, n')$, $\lambda(r) = \ell$.

An Event Knowledge Graph (EKG) is an LPG G with node labels Event, Entity and relationship labels corr ("event correlated to entity"), and df ("event directly followed by event") so that (1) Event and Entity nodes are disjoint, (2) each $e \in$ Event has $\#_{time}(e) \in Time$ and $\#_{act}(e) \in Act$ defined and (3) each df-relationship $r \in$ df, $\overrightarrow{r} = (e, e')$ defines that e' directly follows e from the perspective of entity $r.ent = n \in$ Entity.[1] Fig. 4a shows this basic meta-model.

A basic extension to the EKG (see Fig. 4b) adds relationships n_1 rel n_2 between $n_1, n_2 \in$ Entity and relationships e observed c describing that $e \in Event$ belongs to event class $c \in$ Class, e.g. a particular type of activity [8].

The standard EKG construction from an event table with entities creates Event, Entity and Class nodes and corr, df and observes relationships [8]. The procedure is as follows: (1) each event record is translated into an Event node; (2) infer entity node n with $\#_{type}(n) = ET$ if there is an Event node e with $\#_{ET}(e) = n$ and add e corr n (e is correlated to n); (3) infer df-relationships between events correlated to the same entity node n; (4) infer Class nodes c with $\#_{id}(c) = a$ if there is an Event node e with $\#_{act}(e) = a$ and add e observed c.

In case of incomplete information in the event table, step (2) results in incomplete correlation (corr) relationships and step (3) results in incomplete and false df-relationships.

[1] Formally, let $e, e' \in$ Event be correlated to the same entity $n \in$ Entity, $(e, n), (e', n) \in$ corr: e df e' holds iff $\#_{time}(e) < \#_{time}(e')$ and there is no other event $e'' \in$ Event, $(e'', n) \in$ corr between e and e', i.e. $\#_{time}(e) < \#_{time}(e'') < \#_{time}(e')$.

Refining the meta-model to reflect the use case. For a concrete application, the generic schema of Fig. 4a+b is refined by distinguishing different types of entities and their relationships through dedicated labels. Fig. 4d (ignoring Activity and Location nodes) shows the refined schema for our running example showing 3 types of entity nodes for Equipment, Box and BatchPosition. Storing semantic information in labels simplifies specifying the context of events compared to the generic schema of [8,9] where entity and relationship semantics were expressed as node and relationship properties only.

To reflect the refinement in the EKG, the procedure to infer Entity nodes (step 2) is changed. We infer a node n with labels $\lambda(n) = \{\mathsf{entity}, \mathsf{ET}\}$ if there is an Event node e with $\#_{ET}(e) = n$; then relationship e corr n is added as usual. The automatic construction of the log of Fig. 3 yields the EKG of Fig. 5a+b.

Adding Context. While an EKG models entity context of an event, it does not model the *activity and location context*. We therefore extend the basic schema of EKGs (Fig 4a+b) to model the Activity and Location of events as shown in Fig. 4c: we introduce node labels Activity and Location and relationship labels at, is and part_of. Each Event observes a Class that is an Activity at a specific Location. A Location can be directly part_of another Location[2].

We now discuss how to extend such an EKG by Activity and Location nodes. This requires domain knowledge about activities and locations which are assumed to be specified as records in tabular form similar to event tables (see Tab. 2). Given such records, the EKG is extended by adapting the procedure of [8] as follows: for each activity record a we treat the attribute $\#_{activity}(a)$ as a unique identifier and create a node $act \in$ Activity and set $\#_{prop}(act) = \#_{prop}(a)$ for each $prop \in Attr$ in the activity record. Location nodes l are imported similarly. We then add the relationship c is a for $c \in$ Class, $a \in$ Activity iff $\#_{id}(c) = \#_{activity}(a)$ and add the relationship l part_of l' for $l, l' \in$ Location iff $\#_{id}(l') = \#_{part\_of}(l)$. To add the at relationship between Class and Location nodes, we require a third table specifying for each activity the location at which it happens (see Table 2). For each record the corresponding Class node c and Location node l is deter-

[2] Formally, two locations $l, k \in Location$ have l part_of k iff the location represented by l is physically part of the location represented by k and there is no other $m \in Location$ such that l part_of m part_of k

Table 2. Records containing contextual data.

Activity Records			
activity	type	subtype	entity type
LoadAL	physical	load	box
UnloadAL	physical	unload	box
LoadFS	physical	load	box
UnloadFS	physical	unload	box
Fill	admin	-	box
..	..	..	..

Location Records	
id	part of
AL	-
FS	AL
SS	AL

Relation Records	
activity	at
LoadAL	AL
UnloadAL	AL
LoadFS	FS
UnloadFS	FS
Fill	FS
..	..

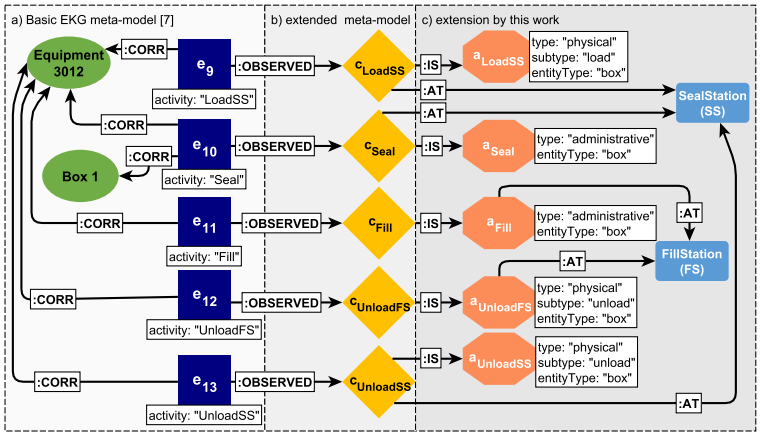

Fig. 5. Partial instance of the EKG

mined, then the relationship c at l is added. Fig. 5 shows a (partial) instance of the EKG obtained from the incomplete log (Fig. 2) extended with Activity and Location nodes based on the records and relations in Tab. 2.

4.3 Inference Rules

We now have an EKG G extended with context information where corr relationships are incomplete due to missing information in the underlying event table. As df-relationships are unreliable due to missing corr relationships, we remove all df-relationships from G. We infer the missing corr relationships based on which we can compute reliable df-relationships. Modeling event context in an EKG enables us to infer missing corr relationships by defining local rules describing contextual patterns in EKGs. We first define the rules in general, then present a first example and then define the semantics of rule application on EKGs.

We define an inference rule as a simple graph-transformation pattern in an EKG G that defines for nodes $n_1, \ldots, n_r$ a "left-hand-side" (LHS) graph pattern specifying the context from which missing relationships can be inferred. The "right-hand side" (RHS) of the rule are then relationships from $n_1, \ldots, n_r$ to other nodes in the LHS; e.g. adding corr relationships between events and entities provides the missing identifiers.

Formally, an *inference rule* $IR = (G, N_{incomplete}, R_{inferred}, c, m)$ has an EKG $G = (N, R, Attr, \#, Lab, \lambda)$ according to a refined meta-model (e.g. Fig. 4c). The nodes $N_{incomplete} \subseteq N$ are the nodes for which relations shall be inferred. The RHS of IR is defined by the relations $R_{inferred} \subseteq R$ with $\#_{RHS}(r) = True$, $\forall r \in R_{inferred}$ so that for each $r \in R_{inferred}$ exists $n \in N_{incomplete}$ and $m \in N$ with $\overrightarrow{r} = (n, m)$. The LHS of IR is G without $R_{inferred}$, i.e., $LHS(IR) = (N, R \setminus R_{inferred}, Attr, \#, Lab)$. Further, IR defines an *ordering condition* c and a *minimization condition* m over the properties of the nodes N in G to limit the matches of the LHS.

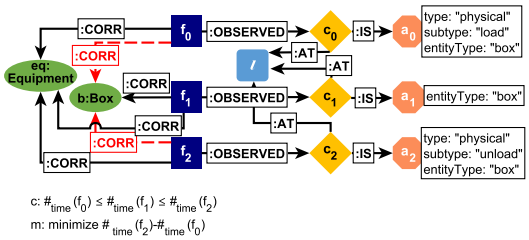

Fig. 6. Inference rule detailing the short-hand notation of Fig. 3.

For example Fig. 6 describes an inference rule based on principle (P1) explained in Sect. 4.1. It infers the unknown entity identifiers of load and unload events f_0 and f_2 from event f_1 occurring in between f_0 and f_2 at the same location.

Fig. 6 depicts the LHS of the rule by solid edges. It defines three events ($f_0, f_1, f_2 \in$ Event). The activity and location of each event f_i is modeled through the event class f_i observed $c_i, c_i \in$ Class where each class is related to an activity c_i is $a_i, a_i \in$ Activity. Specifically, by the properties specified for a_0, a_1, a_2 event f_0 and f_2 observe a physical loading and unloading activity for a box while f_1 observes any activity for a box. The events are observed at the same location ℓ by the relationships f_i observed c_i at ℓ, $\ell \in$ Location, $i = 1, 2, 3$. Furthermore all events are correlated to the same equipment f_i corr $eq \in Equipment, i = 1, 2, 3$, and only f_1 is correlated to box b, i.e., f_1 corr $b \in Box$ while f_0 and f_2 are not correlated to a box, i.e., $N_{incomplete} = \{f_0, f_2\}$.

Ordering condition c: $\#_{time}(f_0) \leq \#_{time}(f_1) \leq \#_{time}(f_2)$ restricts the LHS to events f_0,f_1,f_2 that follow each other in time. Minimization condition m: minimize $\#_{time}(f_2) - \#_{time}(f_0)$ restricts the LHS to only those f_0, f_2 such that no other (un)load events happen in between f_0 and f_2. m also implies that f_0, f_2 are the first preceding load event and first succeeding unload event w.r.t. f_1. Note that the LHS cannot rely on df-relationships to express ordering of the events as df-relationships are incomplete due to incomplete corr relationships.

The RHS of the rule is $R_{inferred} = \{r_0, r_2\}$ with $\overrightarrow{r_0} = (f_0, b)$ and $\overrightarrow{r_2} = (f_2, b)$ (indicated by red dashed edges in Fig. 6, and formally $\#_{RHS}(r_0) = \#_{RHS}(r_2) = True$). Subsequently, we use the notation shown in Fig. 3 as short-hand for inference rules, i.e., Fig. 3 denotes the rule of Fig. 6.

An inference rule $IR = (G, N_{incomplete}, R_{inferred}, cond)$ is *applied* on an (incomplete) EKG G' as follows. An *instance* of $LHS(IR)$ in G' is a sub-graph of G that is injectively homomorphic[3] to LHS, i.e., a mapping $\beta : N \mapsto N'$ from nodes of $LHS(IR)$ to nodes of G' (called *binding*) that respect the labels, properties and relationships between the nodes in N so that the ordering condition $c[N/\beta(N)]$ (replacing the variables (nodes of $LHS(IR)$) in c with the nodes of G) evaluates to true. The empty condition c always evaluates to true. Then from all instances of $LHS(IR)$, pick the instances that minimize all conditions of m. In case m is empty, all instances of $LHS(IR)$ are picked. For each minimal instance $LHS(IR)$ in G' with binding β, apply IR by extending G' by the RHS of IR,

[3] Formally, we define a mapping $f : G \mapsto G'$ for $G = (N, R, Attr, \#, Lab, \lambda)$ and $G' = (N', R', Attr, \#, Lab, \lambda)$ by $f : N \mapsto N'$ where for any $u, v \in N$ holds: $f(u) = f(v) \Rightarrow u = v$ (injective), and $(u, v) \in R \Rightarrow (f(u), f(v)) \in R'$ (homomorphic).

i.e., add a new relationship r^* to G' for each $r \in R_{inferred}$, $\overrightarrow{r} = (e, n)$ so that $\lambda'(r^*) = \lambda(r), \overrightarrow{r^*} = (\beta(e), \beta(n))$.

For example, LHS of Fig. 6 has an instance in G of Fig. 5 by binding $\beta : f_0 \mapsto e_9, f_1 \mapsto e_{10}, f_2 \mapsto e_{13}, b \mapsto Box1, eq \mapsto Equipment3012, c_0 \mapsto c_{LoadSS}$, $c_1 \mapsto c_{Seal}, c_2 \mapsto c_{UnloadSS}, \ell \mapsto SealStation, a_0 \mapsto a_{LoadSS}, a_1 \mapsto a_{Seal}$ and $a_2 \mapsto a_{UnloadSS}$. Note that β respects LHS as all labels, properties and relationships specified in LHS also hold in G. Further, $c[LHS/\beta(LHS)] = \#_{time}(e_9) \leq \#_{time}(e_{10}) \leq \#_{time}(e_{13})$ holds and $\#_{time}(e_13) - \#_{time}(e_9)$ minimizes. Applying the RHS adds corr relationships $\overrightarrow{r_9} = (e_9, b_1)$ and $\overrightarrow{r_{13}} = (e_{13}, b_1)$

The complete inference procedure using a set of rules $\{IR_1, \ldots, IR_k\}$ on an EKG G (with added context) is: (1) remove all df-relationships from G, (2) repeatedly apply each IR_i until no more relationships are added to G, (3) infer the df-relationships (now based on complete corr-relationships).

5 Application and Evaluation

We now show how to translate the three basic principles (P1-P3) for physical objects stated in Sect. 3 into inference rules. The rules are implemented as queries over the Neo4j graph DB. We report on their use in an industrial use case.

5.1 Inference Rules

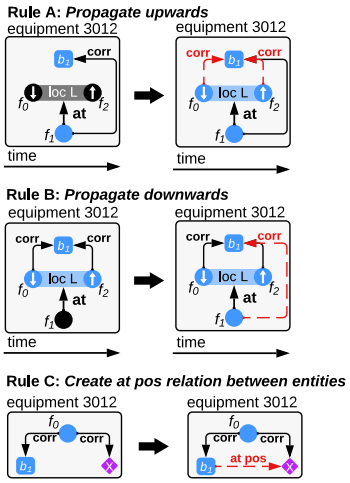

Fig. 7. Rules A, B, and C

We provide five inference rules defined in the model of Sect. 4.3 that we derived from the principles (P1-P3) of Sect. 3.

Inference for one level Fig. 7 shows two rules to infer missing entity identifiers for an object at a location L using the short-hand notation introduced in Sect. 4.1, see App. B for EKG notation. Rule A is explained in Sect. 4.1; from an event f_1 with known identifier, we can infer the identifiers for the load and unload events f_0 and f_2 at the same location L. Rule B is simply the reverse of Rule A; based on the load and unload events of a location, we infer the missing identifiers of the events happening at that location by propagating downwards. Note that any rule can correlate any number of entities to an event. For Rule A, this is desired behavior: all physical objects present at location L need to be loaded and unloaded from L. For Rule B, it depends on the context whether this is desired behavior. If f_1 should only be correlated to one of the objects loaded/unloaded during f_0 and f_2, more information is required which will be explained in Rule E.

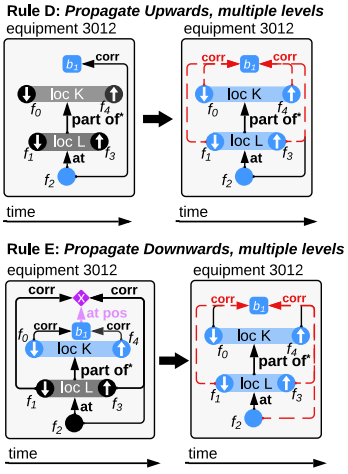

Fig. 8. Rules D and E

Inference between entities Rule C (Fig. 7) is derived from principle (P3); from an event f_0 associated both to a box b_1 and a batch position x, we can infer that box b_1 is at pos x in the tray.

Inference for multiple levels Rules D and E of Fig. 8 use the same principles as Rule A and B respectively for locations containing other locations. As the part of relation is transitive, principles (P1) and (P2) can be also used to infer missing entity identifiers on higher/lower location levels via part of (see Fig. 4) or part of* [4] Rule D infers missing identifiers through multiple levels by propagating upwards and Rule E downwards through multiple levels.

Rule E also deals with events that should only be correlated to one of the objects loaded/unloaded during f_0 and f_4. As stated before, additional information is required for correct inferences such as the relative spatial position of an object (P3). Hence, events happening at a lower-level location L correlated to position x can be related to entity b_1 at pos x that is loaded into K.

These rules allow correlating multiple entities to the same events, which allows to infer, e.g., batching, when physically possible, or reveal ambiguous context information when physically impossible.

5.2 Implementation and Demonstration

We implemented the approach of Sect. 4 and rules A-E of Sect. 5.1 as Cypher queries over the Neo4j graph database; see App. C and https://github.com/Ava-S/EKG_Inference. Applying the implementation on an incomplete event table of our running example infers the correlation and traces shown in Fig. 9 as follows: (1) applying Rule D propagates b_1, b_2 and b_3 from *Seal* events upwards to *LoadSS*, *UnloadSS*, *LoadAL* and *UnloadAL* events; (2) applying Rule C creates the at_pos relation between Box and BatchPosition nodes using *LoadSS* events which are now correlated to both these entities; (3) applying Rule E propagates entity identifiers from *(Un)LoadAL* events downwards to *LoadFS* and *UnloadFS* events; (4) applying Rule B propagates identifiers from *(Un)LoadFS* events to *Fill* events. The corresponding EKG after inference of the event table of Fig. 9 is shown in App. A.

Recall that any rule can correlate any number of entities to an event. Rule D assigns multiple boxes (b_1, b_2 and b_3) to the *LoadAL* event e_1 and *UnloadAL* event e_{20}. Even though Rule D is unaware of batching events, it is still able to

[4] Formally, l partof$*$ k for $l, k \in$ *Location* iff $l = k$, or l partof k, or there exists $m_2, .., m_i \in$ *Location*, for $i \in \mathbb{N}$, $i \geq 2$ such that l part of m_2 partof .. partof m_i partof k.

infer multiple identifiers to batching events. The resulting traces align with the process in Fig. 1 and the physical constraints.

5.3 Industrial Use Case: NXP's Sawing Process

The proposed method was applied on an industrial use case; the sawing process at NXP Semiconductors (NXP). NXP is a globally operating company that designs, develops, and manufactures Integrated Circuit (IC) chips. We focus on the dicing step in which wafers are sawn into dies (unpackaged chips). The sawing equipment has several sensors installed to record the different events operating on the wafers. The collected data covered a week of manufacturing. Data inspection revealed that some entity identifiers were recorded incorrectly. These identifiers were dropped during data cleaning and needed to be inferred.

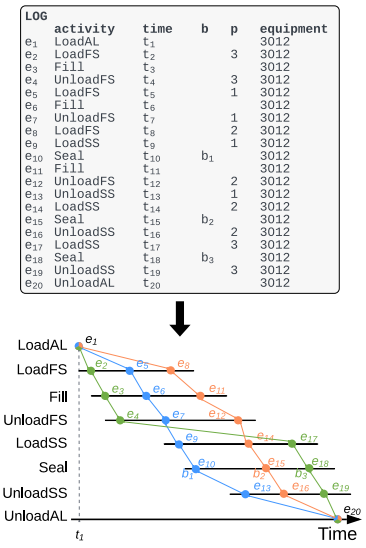

Fig. 9. An incomplete event table of running example.

Process Description Multiple wafers are loaded into the sawing equipment together in a rack for wafers. All wafers are then handled in parallel as follows: the wafers are aligned and cut at the cutting station, and then cleaned at the cleaning station. Finally the rack with all cut wafers is unloaded. A rack has multiple slots containing one wafer each; wafers in the same rack are distinguished by their slot position in the rack.

Activities and events Table 3 gives an overview of the different activity types in NXP's sawing process, how many activities of this type the process has and their frequency. Only 3/24 activities recorded wafer identifiers, resulting in $\approx$ 6250/7250 events without wafer identifier.

Inferring missing information The event data and contextual data was modelled with an adapted meta-model of Fig. 4c to reflect the use case of NXP. We applied Rules A-E (adapted to the NXP meta-model) to infer the missing wafer identifiers. Construction of the complete EKG and inference required less than 30 seconds, resulting in 7225/7250 events correlated to at least one wafer.

Table 3. Overview of the activity types and whether the wafer identifiers are present.

type	subtype	entity	wafer identifier(s)	# activities	# events
administrative	-	wafer	✓	3	$\approx$ 1000
administrative	-	wafer	-	10	$\approx$ 3100
physical	load	(batch of) wafers	-	1	$\approx$ 50
physical	load	wafer	-	5	$\approx$ 1550
physical	unload	wafer	-	5	$\approx$ 1550
Total				24	$\approx$ 7250

Validation We validated the correctness of the inferred identifiers through alignment-based conformance checking. A proclet reference model of the multi-entity process was created and validated with NXP [19]; the procles describes for each entity type (e.g. wafer, rack) a life-cycle model as state machine without concurrency. The proclet model was added to the EKG by modeling df_c relationships between Class nodes (see Fig. 4 and [9]), where c df_c c' describes that activity c can be directly followed by c' for a particular entity, e.g. wafer or a rack. This allowed us to measure whether all df-relationships of a wafer w form a complete trace according to the wafer life-cycle using the technique in [9, Sect 6.4]: we checked whether each df-relationship (e, e') of w has a corresponding df_c-relationship (c, c') for wafers with e observes c and e' observes c'. While for the extracted data, 0 wafers had a complete trace, after inference 88% had complete traces, the remaining 12% of incomplete traces were attributed to non-recorded or double-recorded events.

6 Conclusion

In this paper, we studied the problem of inferring missing entity identifiers in the generalized setting of multi-entity processes (e.g., with batching) that are executed in a physical environment. We introduced a new method for extending incomplete event data with context information about the physical environment (locations of activities) using event knowledge graphs (EKGs) [9]. We showed that inference rules to infer missing identifiers from the context can be directly modeled in terms of EKGs based on simple principles of physical processes. An industrial case study proved that our technique is applicable to industrial processes both in terms of quality of inference and performance.

Our work shows that missing identifiers can be inferred efficiently with simple rules even when no normative control-flow model is available or applicable (i.e., multi-entity processes). Further, we demonstrated that event knowledge graphs are a versatile data model to integrate observational data and structural and contextual knowledge to solve industrially relevant problems. These insights suggest further viable research for integrating event data and process knowledge for inference in processes, not just limited to missing identifiers. Our study only explored inference rules for multi-entity interactions in the form of batching 1:n synchronization and hierarchical locations. While the model of EKGs and the definition of inference rules are not limited to such processes, further research is needed to explore inference for processes with other forms of synchronization and other forms of physical context knowledge.

Acknowledgement. The research underlying this paper was partially supported by NXP Semiconductors and by AutoTwin EU GA n. 101092021.

References

1. Andrews, R., Emamjome, F., ter Hofstede, A.H.M., Reijers, H.A.: An expert lens on data quality in process mining. In: ICPM 2020. pp. 49–56 (2020)

2. Baier, T., Di Ciccio, C., Mendling, J., Weske, M.: Matching events and activities by integrating behavioral aspects and label analysis. Softw. Syst. Model. **17**(2), 573–598 (2018)
3. Bala, S., Mendling, J., Schimak, M., Queteschiner, P.: Case and activity identification for mining process models from middleware. LNBIP p. 86–102 (2018)
4. Bayomie, D., Awad, A., Ezat, E.: Correlating unlabeled events from cyclic business processes execution. In: CAiSE 2016. LNCS, vol. 9694, pp. 274–289. Springer (2016)
5. Bayomie, D., Di Ciccio, C., Mendling, J.: Event-case correlation for process mining using probabilistic optimization. Inf. Syst. **114**, 102167 (2023)
6. Bayomie, D., Revoredo, K., Di Ciccio, C., Mendling, J.: Improving accuracy and explainability in event-case correlation via rule mining. In: ICPM 2022. pp. 24–31. IEEE (2022)
7. Denisov, V., Fahland, D., van der Aalst, W.M.P.: Unbiased, fine-grained description of processes performance from event data. LNCS p. 139–157 (2018)
8. Esser, S., Fahland, D.: Multi-dimensional event data in graph databases. Journal on Data Semantics (2021)
9. Fahland, D.: Process mining over multiple behavioral dimensions with event knowledge graphs. Lecture Notes in Business Information Processing p. 274–319 (2022)
10. Fahland, D., Denisov, V., van der Aalst, W.M.P.: Inferring unobserved events in systems with shared resources and queues. Fundam. Informaticae **183**(3-4), 203–242 (2021)
11. Ferreira, D.R., Gillblad, D.: Discovering process models from unlabelled event logs. LNCS p. 143–158 (2009)
12. Jagadeesh Chandra Bose, R.P., Mans, R.S., van der Aalst, W.M.P.: Wanna improve process mining results?: it's high time we consider data quality issues seriously. In: Proceedings of the IEEE CIDM. pp. 127–134. IEEE (2013)
13. Mannhardt, F., de Leoni, M., Reijers, H.A., van der Aalst, W.M.P., Toussaint, P.J.: From low-level events to activities - a pattern-based approach. LNCS p. 125–141 (2016)
14. de Murillas, E.G.L., Reijers, H.A., van der Aalst, W.M.P.: Case notion discovery and recommendation: automated event log building on databases. Knowl. Inf. Syst. **62**(7), 2539–2575 (2020)
15. Pegoraro, M., Uysal, M.S., Hülsmann, T., van der Aalst, W.M.P.: Uncertain case identifiers in process mining: A user study of the event-case correlation problem on click data. In: BPMDS and EMMSAD 2022. LNBIP, vol. 450, pp. 173–187. Springer (2022)
16. Pourmirza, S., Dijkman, R.M., Grefen, P.W.: Correlation miner: Mining business process models and event correlations without case identifiers. Int. J. Cooperative Inf. Syst. **26**(2), 1742002:1–1742002:32 (2017)
17. Rogge-Solti, A., Mans, R.S., van der Aalst, W.M.P., Weske, M.: Repairing event logs using timed process models. In: OTM 2013 Workshops. pp. 705–708. Springer (2013)
18. Suriadi, S., Andrews, R., ter Hofstede, A.H.M., Wynn, M.T.: Event log imperfection patterns for process mining: Towards a systematic approach to cleaning event logs. Information Systems **64**, 132–150 (2017)
19. Swevels, A.J.E.: Creating a Digital Shadow of a Manufacturing Process with Inferred Missing Information using an Event Knowledge Graph. Master's thesis, Eindhoven University of Technology (2022)
20. Tax, N., Sidorova, N., Haakma, R., van der Aalst, W.M.P.: Mining process model descriptions of daily life through event abstraction. Studies in Computational Intelligence p. 83–104 (2017)

A Complete EKG of Running Example

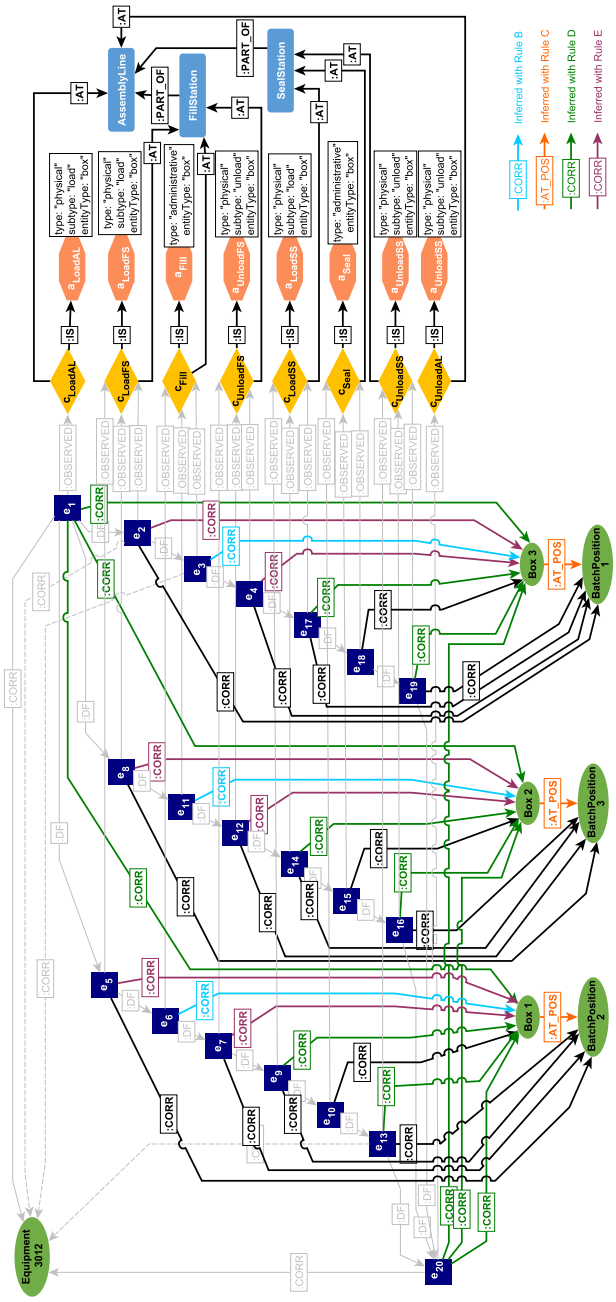

Fig. 10. Complete instance of Event Knowledge Graph after inference for the Event Table in 9(top)

B Extended Inference Rules

B.1 Rule A: propagate upwards, one level

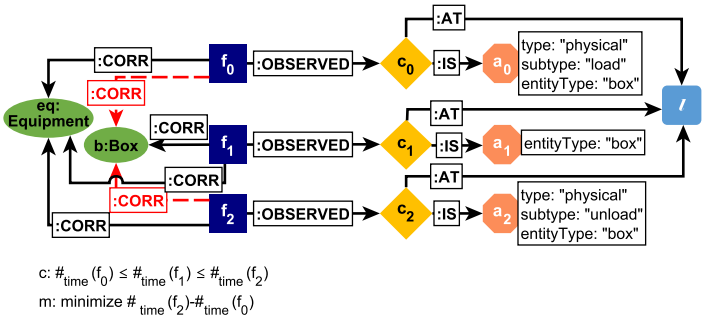

Fig. 11. Inference Rule detailing the short-hand notation of Fig. 7 Rule A.

B.2 Rule B: propagate downwards, one level

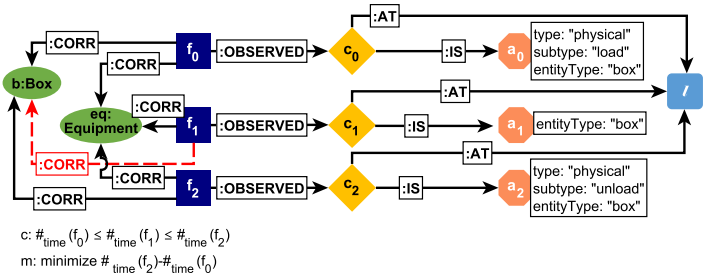

Fig. 12. Inference Rule detailing the short-hand notation of Fig. 7 Rule B.

B.3 Rule C: Create at pos relation between entities

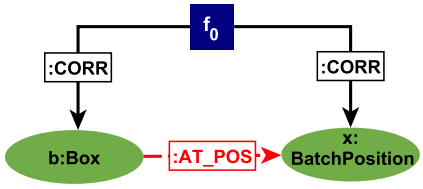

Fig. 13. Inference Rule detailing the short-hand notation of Fig. 7 Rule C.

B.4 Rule D: propagate upwards, multiple levels

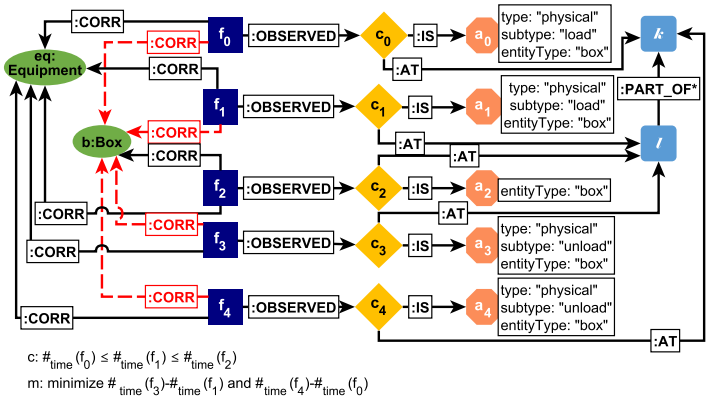

Fig. 14. Inference Rule detailing the short-hand notation of Fig. 8 Rule D.

B.5 Rule E: propagate downwards, multiple levels

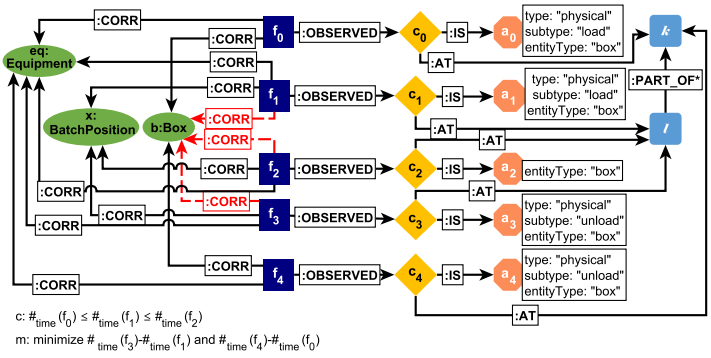

Fig. 15. Inference Rule detailing the short-hand notation of Fig. 8 Rule E.

C Queries

Rules A, B, D and E determine the load and unload events at the different location levels such that the time difference is minimized. Given that all events are recorded correctly, the constraint that the administrative event should be in between a pair of load and unload events can be relaxed by using the first preceding load event (or first succeeding unload event) w.r.t. the administrative event. With this relaxation, the inference rules are also applicable when there are

only *load* events (no *unload* events). The shown queries determine the inference using the first preceding load events. The queries can be adapted without loss of generality to use the first succeeding unload events.

C.1 Rule A; propagate upwards, one level

```
MATCH (f1: Event) - [:CORR] -> (b: Box)
MATCH (f1) - [:CORR] ->  (equipment: Equipment)
MATCH (f1) - [:OBSERVED] -> (c1: Class) - [:AT] -> (l: Location)
WITH f1, l, equipment, b
CALL {WITH f1, l, equipment
    // ensure f0 is a load event operating on a box
    MATCH (f0: Event) - [:OBSERVED] -> (c0: Class)
        - [:IS] -> (a0: Activity {type: 'physical', subtype: 'load', entity: 'Box'})
    MATCH (c0) - [:AT] -> (l)
    MATCH (f0) - [:CORR] ->  (equipment)
    WHERE f0.timestamp <= f1.timestamp
    RETURN f0 as f0_first_preceding // find the first preceding f0
    ORDER BY f0.timestamp DESC LIMIT 1}
MERGE (f0_first_preceding) - [:CORR] -> (b)
```

C.2 Rule B; propagate downwards, one level

```
MATCH (f1: Event) - [:CORR] -> (equipment: Equipment)
MATCH (f1) - [:OBSERVED] -> (c1: Class) -[:AT]-> (l: Location)
MATCH (c1) - [:IS] -> (a1 :Activity {entity: 'Box'}) // ensure f1 should operate on a box
WITH f1, equipment, l
CALL {WITH f1, equipment, l
    // ensure f0 is a load event operating on a box
    MATCH (f0: Event)-[:OBSERVED]->(c0: Class)
        - [:IS] -> (a0: Activity {type:'physical', subtype: 'load', entity:'Box'})
    MATCH (c0) - [:AT] -> (l)
    MATCH (f0)-[:CORR]->(equipment)
    WHERE f0.timestamp <= f1.timestamp
    RETURN f0 as f0_first_prec // find the first preceding f0
    ORDER BY f0.timestamp DESC LIMIT 1
}
// only merge when f0_first_prec is actually related to a Box
WITH f1, [(f0_first_prec)-[:CORR]->(b: Box) | b] as related_b
FOREACH (b in related_b |
    MERGE (f1) - [:CORR] -> (b)
)
```

C.3 Rule C: create at pos relation between entities

```
    MATCH (e: Event) - [:CORR] -> (b: Box)
    MATCH (e) - [:CORR] -> (bp: BatchPosition)
    MERGE (b) - [:AT_POS] -> (bp)
```

C.4 Rule D: propagate upwards, multiple levels

```
MATCH (f2: Event) - [:CORR] -> (b: Box)
MATCH (f2) - [:CORR] ->  (equipment: Equipment)
MATCH (f2) - [:OBSERVED] -> (c2: Class)
   - [:AT] -> (l: Location) - [:PART_OF*0..] -> (k: Location)
WITH f2, k, equipment, b
CALL {WITH f2, k, equipment
   // ensure f0 is a load event operating on a box
   MATCH (f0: Event) - [:OBSERVED] -> (c0: Class)
      -  [:IS] -> (a0: Activity {type: 'physical', subtype: 'load', entity: 'Box'})
   MATCH (c0) - [:AT] -> (k)
   MATCH (f0) - [:CORR] ->  (equipment)
   WHERE f0.timestamp <= f1.timestamp
   // find the first preceding f0
   RETURN f0 as f0_first_preceding
   ORDER BY f0.timestamp DESC LIMIT 1}
MERGE (f0_first_preceding) - [:CORR] -> (b)
```

Note that Rule A is also applied when Rule D is applied, then there are zero part of relationships and $l = k$.

C.5 Rule E: propagate downwards multiple levels

```
MATCH (f2: Event) - [:CORR] -> (bp: BatchPosition)
MATCH (f2) -[:CORR] -> (equipment: Equipment)
MATCH (f2) -[:OBSERVED] -> (c2: Class)
   -[:AT]-> (l: Location) -[:PART_OF*0..] ->(k: Location)
// ensure f2 should operate on a box
MATCH (c2) -[:IS] -> (a2: Activity {entity: 'Box'})
WITH f2, equipment, l, bp
   CALL {WITH f2, equipment ,l
      // ensure f0 is a load event operating on a box
      MATCH (f0: Event)-[:OBSERVED]->(c0: Class) -[:IS]
      -> (a0: Activity {type:'physical', subtype: 'load', entity:'Box'})
      MATCH (c0) -[:AT] -> (k)
      MATCH (f0)-[:CORR]->(equipment)
      WHERE f0.timestamp <= f2.timestamp
      // find the first preceding f0
      RETURN f0 as f0_first_prec
      ORDER BY f0.timestamp DESC LIMIT 1
      }
// only merge when f0_first_prec is actually related to a Box
WITH f2, [(f0_first_prec) -[:CORR]-> (b: Box) -[:AT_POS] -> (bp) | b] as related_b
FOREACH (b in related_b |
   MERGE (f2) -[:CORR] -> (b)
)
```