Explainable Interpretation of Low-Level Process Traces via Abstract Argumentation*

(Discussion/Short Paper)

Bettina Fazzinga[1], Sergio Flesca[2], Filippo Furfaro[2] and Luigi Pontieri[3,*]

[1]DIMES - University of Calabria, Via P. Bucci, 87036 Rende, Italy
[2]DICES - University of Calabria, Via P. Bucci, 87036 Rende, Italy
[3]ICAR-CNR, Via P. Bucci, 87036 Rende, Italy

Abstract

Monitoring and analyzing process traces (i.e. event sequences) is an important task in modern companies and organizations. Focusing on scenarios where there is an abstraction gap between trace events and reference business activities, we here consider the *interpretation problem* of translating the current event generated by an ongoing trace into the step of the activity instance it corresponds to. We briefly discuss how this problem can be encoded into an *Abstract Argumentation Framework* (AAF), which allows for computing and exploring event interpretations by solving instances of the *AAF acceptance* problem.

Keywords

Business Process Intelligence, Log Abstraction, Abstract Argumentation

1. Introduction

Process Mining methods help better understand and enact business processes by supporting various analysis tasks over process *traces*, i.e. sequences of execution *events*, e.g. process-model discovery, log-to-model conformance analysis, runtime detection/prediction/recommendation. However, these methods usually assume each event to map to a well-understood "high-level" activity, which is not always the case in practice. As an instance, when the activities are performed in a lowly-structured way, trace events just represent low-level actions [2] with no clear reference/mapping to the activities. An example of this kind is presented below, for a toy hospital scenario where careflow process traces are gathered, which consist of events describing exams and checks performed by medical staff.

Example 1. *Suppose that a trace* $\Phi = e_1, e_2, e_3, e_4$ *is given, where* e_1, e_2, e_3 *and* e_4 *refer to events (precisely, event types)* Blood sample taken, Blood pressure measurement, Temperature measurement, *and* Cannula insertion, *respectively. Assume that it is known that the first three events can be performed during any of the high-level activities* A_1=pre-hospitalization, A_2=pre-surgery,

PMAI@IJCAI22: International IJCAI Workshop on Process Management in the AI era, July 23, 2022, Vienna, Austria
*Extended Abstract: the original work [1] was published on journal Information Systems in January 2022.
*Corresponding author.
bettina.fazzinga@unical.it (B. Fazzinga); flesca@dimes.unical.it (S. Flesca); furfaro@dimes.unical.it (F. Furfaro); luigi.pontieri@icar.cnr.it (L. Pontieri)

A_3 =post-surgery, *while* e_4 *can be performed only during activity* A_2. *To reconstruct/monitor patient histories, one needs to interpret such a trace in terms of activity executions. However, due to the many-to-many event-activity mapping, even if trace* Φ *described a completed process instance, multiple interpretations could exist for* Φ, *like:* $\boldsymbol{I_1} \equiv$ *"*e_1, e_2, e_3 is the sequence of steps produced by an instance of A_1 and e_4 is the unique step of an instance of A_2*"; and* $\boldsymbol{I_2} \equiv$ *"*e_1, e_2 and e_3, e_4 are the step sequences produced by instances of A_1 and of A_2, *respectively"*. □

Such an abstraction gap limits the usefulness of process discovery methods, and impedes using standard conformance-checking ones. Thus, *supervised log abstraction* methods were proposed recently [2], which try to *interpret* log traces automatically as executions of given high-level process activities, leveraging domain knowledge. However, these methods suffer from two limitations: (*i*) they map each (low-level) event *e* to just one activity, discarding alternative interpretations of *e*; (*ii*) they do not provide explanations for their results.

In [1], we defined a framework that supports interactive explorative analyses of any low-level trace Φ, which allow the user to obtain answers and explanations for *Interpretation Queries* over Φ. Technically, the underlying event-interpretation problem is encoded into an Abstract *Argumentation Framework* (*AAF*) (see Section 2), so that both interpretation queries and associated explanations are computed by solving instances of the AAF acceptance problem. This extended abstract summarizes some major features of the framework introduced in [1].

2. Background and problem statement

AAF basics An AAF can be viewed as a graph where nodes and edges model *arguments* and *attacks* from an argument to another, respectively. Given argument set S and argument α, we say that "S *attacks* α" (resp., α attacks S) if there is an argument β in S such that β attacks α (resp., α attacks β). Given arguments α, β such that α attacks β, an argument γ is said to "*defend* β" if γ attacks α. Argument set S is said to "*defend* β" if S contains some argument defending β. Argument α *is acceptable w.r.t.* S if every argument attacking α is attacked by S, while S is *conflict-free* if there is no attack between its arguments. Different semantics have been proposed to identify "reasonable" sets of arguments, called *extensions* [3], in an AAF. In particular, a set $S \subseteq A$ is said to be an *admissible extension* iff S is conflict-free and all its arguments are acceptable w.r.t. S. An admissible extension that is maximal (w.r.t. $\subseteq$) is a *preferred extension*. Acceptance of an argument α in an AAF F can be stated under both skeptical and credulous perspectives. In particular, adopting the preferred semantics (as in [1]), α is *credulously accepted* (resp., *skeptically accepted*), if it belongs to at least one (resp., every) preferred extension of F.

The interpretation problem Given an ongoing trace Φ, as soon as a new event e_{curr} appears in Φ, we want to interactively support the analyst in obtaining (*a*) answers to *Interpretation Queries* on the alternative *valid* interpretations of e_{curr}, and (*b*) *explanations* for certain interpretations considered *not valid*, where interpretation validity descends from well-structuredness properties and given domain knowledge (described later on). Roughly speaking, basic interpretation queries allow for asking whether e_{curr} can be viewed as a step of an instance of a given activity A, possibly stating conditions on the activity lifecycle (e.g.,"*is* e_{curr} *the initial step of an*

instance of A?" or on the activity occurrence (e.g., "*is* e_{curr} *a step of the* 2*nd instance of A?*"). Any such a query can be evaluated under *skeptical semantics*, in order to know whether the proposed interpretation is the *unique* valid one for e_{curr}. Moreover, *enumeration queries* ask for all valid interpretations of e_{curr} satisfying the conditions required.

Two kinds of domain knowledge are considered in the above-described problem: (*i*) the list of high-level activities in terms of which events must be interpreted, along with the candidate type-level mappings between these activities and log events; (*ii*) a declarative process model consisting of activity constraints, including temporal rules of the forms $MC_j : A \Rightarrow_T B_1|\dots|B_k$ (*must-constraint*) and $NC_i : A \Rightarrow_T \neg B$ (*not-constraint*), for any given activities $A, B, B_1, \dots, B_k$ and time period $T \in [1..\infty]$, and their respective time-symmetric versions (i.e., positive and negative "precedence" constraints). MC_j (resp., NC_i) means that an instance A must (resp., cannot) be followed by an instance of one of the activities $B_1, \dots, B_k$ (resp., of activity B) in T steps. In Example 1, one could use such a constraint to model known behaviors like that A_2 is always preceded by A_1 immediately; if also knowing that A_1 must start with *Blood sample taken* and end with *Blood pressure measurement*, this allows for discarding interpretations like I_1.

3. Overview of the Solution Approach

The core problem of computing valid (w.r.t. domain knowledge) interpretations of events in a trace Φ is modelled as a dispute via an ad hoc AAF $F(\Phi) = \langle\ Arg,\ Att\ \rangle$, on top of which interpretation queries and explanation requests can be answered. $F(\Phi)$ is built incrementally by adding arguments of the following kinds (and attacks between them) as a new step of Φ arrives:
1) *Interpretation arguments* of the form $\langle e_i, A, X, j\rangle$, where A is an activity label, j is a positive integer and $X \in \{$ *first*, *intermediate*, *last*, *first&last*$\}$ indicates a life-cycle execution stage for A. Such an argument encodes the interpretation of e_i as a step of type X of the j-th execution instance of activity A. Interpretation I_1 in Example 1 can be encoded via interpretation arguments $\langle A_1, first, 1\rangle\langle A_1, intermediate, 1\rangle\langle A_1, last, 1\rangle\langle A_2, first\&last, 1\rangle$.
2) *Undermining arguments* of five different forms: (*i*) $NotInterpreted_i$, meaning that no interpretation has been chosen for step $\Phi[i]$; (*ii*) $NotEnoughExecutions_\alpha$, where $\alpha = \langle e_i, A, X, j\rangle$ with $X \in \{$*first*, *first&last*$\}$, meaning that α's interpretation of e_i as the start of the j-th instance of A contrasts with interpreting no previous step as the start of the $(j-1)$-th instance of A; (*iii*) $NotStarted_\alpha$, where $\alpha = \langle e_i, A, X, j\rangle$ with $X \in \{$*intermediate*, *last*$\}$, meaning that α's interpretation of e_i as a non-initial step of the j-th instance of A contrasts with interpreting no previous step as the start of the j-th instance of A; (*iv*) $NotEnded(j, A)$, where j is an instance counter and A an activity, meaning that the trace has terminated and some event was interpreted as the initial step of the j-th instance of A, but no event has been interpreted as the last step of this instance; (*v*) MC^i_q, descending from some must-constraint $MC_q : A \Rightarrow_T B_1|\dots|B_k$, and meaning that the i-th event of Φ cannot be interpreted as the final step of an instance of A, for no subsequent event (in T steps) has been interpreted as the start of an instance of some B_i.

Fig. 1 sketches an excerpt of the AAF resulting from processing a partial trace $\Phi = e_1, \dots, e_6$. Clearly, the AAF includes several conflicting event interpretations, encoded by interpretation arguments with mutual attacks, e.g.: α' and α'' (representing alternative interpretations of the same event), α' and β'' (interpreting distinct events as first steps of the first instance of A), and

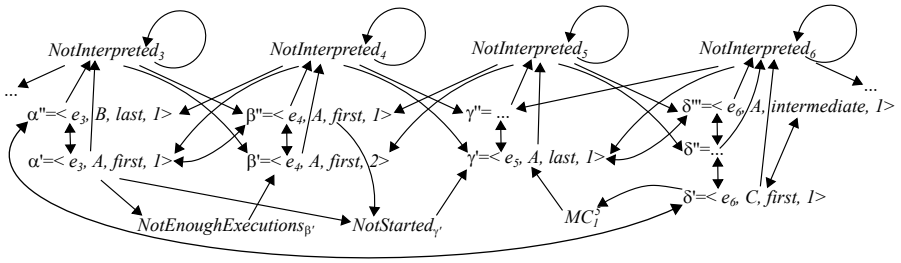

Figure 1: A portion of $F(\Phi)$ (edges with two arrows represent pairs of mutual attacks)

γ' and δ''' (where δ''' interprets e_6 as a step of an instance interpreted as closed by γ).

As an example of how the AAF ensures activity counters to be consistent, consider *NotEnoughExecutions*$_{\beta'}$, which attacks β' and it is counter-attacked by α'.

Importantly, the AAF enforces the choice of an interpretation for each trace step. In particular, it contains exactly one argument *NotInterpreted*$_x$ for each step e_x, that is attacked by the interpretation arguments over the same step e_x and that attacks the interpretation arguments over the immediately preceding and following steps e_{x-1} and e_{x+1}.

The AAF also ensures that activity instances can be prolonged only if previously started. For example, in Fig. 1, γ' interprets e_5 as a non initial step of the first instance of A and is attacked by *NotStarted*$'_\gamma$, and is defended from this attack by α' and β'', that interpret some previous steps as the starting steps of the same instance of A.

All the constraints in the given declarative process model are enforced as well. For instance, in Fig. 1, the must-constraint $MC_1 : A \Rightarrow_1 C$ (triggered by argument γ' over step 5) is encoded by the undermining argument MC_1^5 and the attacks (MC_1^5, γ') and (δ', MC_1^5).

Further attacks are included to guarantee that $\emptyset$ is the only admissible extension of $F(\Phi)$ iff there is no valid interpretation of Φ (i.e. trace behavior deviates from the process model). This is the case of self-attacks on all *NotInterpreted*$_i$ and the special attack from *NotInterpreted*$_1$ towards each argument of type *NotEnoughExecutions*, *NotStarted*, *NotEnded*.

It was proven in [1] that there is a biunivocal correspondence between the set of valid interpretations of Φ and the set of preferred extensions of $F(\Phi)$, so one can reason on the interpretations of the current event e_{curr} by deciding the acceptance of interpretation arguments over e_{curr}. Details on the computation of the AAF $F(\Phi)$ and of answers to interpretation queries and associated explanation requests can be found in [1].

References

[1] B. Fazzinga, S. Flesca, F. Furfaro, L. Pontieri, Process mining meets argumentation: Explainable interpretations of low-level event logs via abstract argumentation, Information Systems 107 (2022). doi:https://doi.org/10.1016/j.is.2022.101987.

[2] S. J. van Zelst, F. Mannhardt, M. de Leoni, A. Koschmider, Event abstraction in process mining: literature review and taxonomy, Granular Computing 6 (2021) 719–736.

[3] P. M. Dung, On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games, Art. Int. 77 (1995) 321–358.