Discovering Unbounded Synchronization Conditions in Artifact-Centric Process Models

Viara Popova and Marlon Dumas

University of Tartu, Estonia
{viara.popova,marlon.dumas}@ut.ee

Abstract. Automated process discovery methods aim at extracting business process models from execution logs of information systems. Existing methods in this space are designed to discover synchronization conditions over a set of events that is fixed in number, such as for example discovering that a task should wait for two other tasks to complete. However, they fail to discover synchronization conditions over a variable-sized set of events such as for example that a purchasing decision is made only if at least three out of an a priori undetermined set of quotes have been received. Such synchronization conditions arise in particular in the context of artifact-centric processes, which consist of collections of interacting artifacts, each with its own life cycle. In such processes, an artifact may reach a state in its life cycle where it has to wait for a variable-sized set of artifacts to reach certain states before proceeding. In this paper, we propose a method to automatically discover such synchronization conditions from event logs. The proposed method has been validated over actual event logs of a research grant assessment process.

Key words: Business Process Mining, Automated Process Discovery

1 Introduction

Process mining is concerned with the extraction of knowledge about business processes from execution logs of information systems [1]. Process mining encompasses a wide range of methods, including automated process discovery methods, which seek to extract business process models from event logs.

A common limitation of existing automated process discovery methods is that they are unable to discover synchronization conditions that involve one process waiting for a variable number of other processes to reach certain states. This situation arises for example when one process spawns a variable number of subprocesses and waits for a subset of them to complete before proceeding. In the BPMN notation, a process may contain a multi-instance activity which spawns a number of instances of a subprocess and waits for a subset of these instances to complete based on a so-called *completion condition*. A concrete example is the case where a procure-to-pay process spawns a number of subprocesses to retrieve quotes from multiple suppliers (determined at runtime) and then waits until a number of quotes have been obtained before proceeding with supplier selection. We hereby call such conditions *unbounded synchronization conditions*.

More general forms of unbounded synchronization conditions are found in artifact-centric process models. In artifact-centric modeling [3, 11] a process is decomposed into a collection of artifacts corresponding to business objects with their own life cycles and information models. For example, a conference reviewing process may be split into artifacts *Conference*, *Submission* and *Review*. In this setting, an unbounded synchronization condition is that "a submission can only be evaluated when at least three reviews are completed".

This paper addresses the problem of discovering unbounded synchronization conditions in artifact-centric process models. The contribution is framed in the context of artifact-centric processes represented using the Guard-Stage-Milestone (GSM) notation [6]. GSM divides the life cycle of an artifact into stages that open when their guard conditions become true and close when their milestone conditions become true. A guard condition of a stage of an artifact may refer to milestones of the artifact itself or attributes within the artifact's information model (intra-artifact condition) but it may also refer to the state of other artifacts (inter-artifact condition). The paper addresses the discovery of inter-artifact conditions where the number of artifact instances to be synchronized is not determined at design-time.

The presented methods are implemented as plug-ins in the ProM process mining framework [16] and validated using a real-life log of a research grant assessment process.

The paper is organized as follows. Section 2 gives a brief overview of artifact-centric process modeling using GSMs and a simple scenario that serves as a motivating example. Section 3 presents the proposed method for discovering inter-artifact synchronization conditions. The validation is discussed in Section 4. Sections 5 and 6 discuss related work and future research directions.

2 Background: Artifact-centric Modeling

Artifact-centric modeling is an approach for modeling business processes based on the identification of key objects (artifacts) that encapsulate process-related data and whose life cycles define the overall business process [3, 11]. An artifact type contains an information model with all data relevant for the artifacts of that type as well as a life cycle model specifying how an artifact responds to events and undergoes transformations from its creation until it is archived.

The Guard-Stage-Milestone (GSM) meta-model [6] can be used to represent the artifact life cycles which allows a natural way for representing hierarchy and parallelism within the same instance of an artifact and between instances of different artifacts. The key GSM elements for representing the artifact life cycle are *stages*, *guards* and *milestones*. Stages correspond to clusters of activities performed for, with or by an artifact instance intended to achieve one of the milestones belonging to the stage. A stage can have one or more guards and one or more milestones. A stage opens when a guard becomes true and closes when a milestone becomes true. Sentries are used in guards and milestones to

control when they become true. Sentries contain a triggering event type and/or a condition which may refer to the same or other artifact instances.

As a motivating example we consider the following meeting planning process. An assistant tries to organize a meeting between a group which in our example consists of 6 participants. The assistant proposes time and date. Each participant receives the proposal and considers their availability. If they are not available, they reject the proposal. If available, they consider whether they are prepared to host the meeting and accept the proposal (it is convenient but not prepared to host) or propose to host the meeting. The proposal is successful if at least three participants are available and at least one of them is prepared to host. If it fails, the assistant proposes a new time and date and the process continues until a proposal is successful.

Adopting an artifact-centric modeling approach for this scenario, we can consider two artifact types: Meeting Proposal and Participant (see Figure 1). One instance of Meeting Proposal reflects the efforts of the assistant to organize a single meeting for specific time and date. One instance of the Participant artifact reflects the activities of one participant for responding to a meeting proposal. Instances of the Meeting Proposal artifact type are identified by attribute id while instances of the Participant artifact type are identified by attribute pair (id, participant). For testing, the scenario was implemented in CPN Tools to generate event logs.

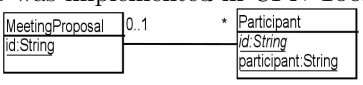

Fig. 1. Artifact types in the motivating scenario: Meeting Proposal and Participant.

3 Discovery of Synchronization Conditions

The aim of this research is to propose a method for discovering inter-artifact synchronization conditions which can then become part of the guard of the corresponding stage in the GSM model. These conditions reflect the knowledge that the stage can only open when a certain number of instances of another artifact reach a certain state. This state will more specifically be represented by the fact that a milestone of another stage has been reached.

For our meeting planning example such condition can be defined, for instance, for the stage Meeting Successful of artifact Meeting Proposal. Meeting Successful can only open if at most 3 instances of artifact Participant have completed stage Reject Proposal, at least 2 instances have completed stage Accept Proposal and at least one instance has completed stage Host Meeting.

The previously presented methods in [13] provide us with the means to extract the knowledge of which specific instances of an artifact are related to which instances of other artifacts. This information is used when discovering inter-artifact guard conditions. To take advantage of it, a new format for logs is used - *artifact synchronization logs*.

Event logs consist of events which represent executions of activities in the system. Events contain attributes which determine the activity, the time at which the event happened and what business-relevant data values are associated to it.

Definition 1 (Event). *Let $\{N_1, N_2, \ldots, N_n\}$ be a set of attribute names and $\{D_1, D_2, \ldots, D_n\}$ - a set of attribute domains where D_i is the set of possible values of N_i for $1 \leq i \leq n$. Let $\Sigma = \{A_1, A_2, \ldots, A_m\}$ be a set of event types. An event e is a tuple $e = (A, \tau, v_1, v_2, \ldots, v_k)$ where*

1. *$A \in \Sigma$ is the* event type *to which e belongs,*
2. *$\tau \in \Omega$ is the timestamp of the event where Ω is the set of all timestamps,*
3. *for all $1 \leq i \leq k$ v_i is an attribute-value pair $v_i = (N_i, d_i)$ where N_i is an attribute name and $d_i \in D_i$ is an attribute value.*

All events of an event type A are called event instances of A.

We denote the event type of event e by $A(e)$ and the timestamp of e by $\tau(e)$.

Definition 2 (Trace, log). *A* trace *L is a sequence of events ordered according to a total order $\leq$ induced by the event timestamps. A set of traces is called a* log*. An artifact-centric log consists of the traces of its artifact instances. The trace of an artifact instance contains all events affecting the instance in question.*

Let $\mathcal{A} = \{Art_1, \ldots, Art_n\}$ be an artifact system with artifacts Art_i, $1 \leq i \leq n$. Let $\mathcal{L} = \{L_1, \ldots, L_n\}$ be a set of artifact-centric logs of the behavior of the system where log L_i describes the behavior of artifact Art_i and its instances $\{{I_i}^1, \ldots, {I_i}^m\}$. For each instance ${I_i}^j$, trace ${T_i}^j$ from log L_i contains all events for this instance. We denote by ${I_i}^j \mapsto Art_i$ the fact that instance ${I_i}^j$ is an instance of artifact Art_i. ${T_i}^j \rightarrow {I_i}^j$ denotes that trace ${T_i}^j$ describes instance ${I_i}^j$.

Let $\mathcal{R}$ be a set of relationships R_{ij} between artifacts of $\mathcal{A}$ in $\mathcal{L}$ such that R_{ij} defines which instances of artifact Art_i are related to which instances of artifact Art_j, $R_{ij} = \{({I_i}^t, {I_j}^s) : {I_i}^t \mapsto Art_i, {I_j}^s \mapsto Art_j\}$.

Definition 3 (Artifact synchronization log). *We define an artifact synchronization log L for artifact Art_i with respect to artifact Art_j as the set of traces $\{ST_s\}$ such that ST_s consists of the events of instance I_i^s of artifact Art_i and the events of all instances of artifact Art_j related to that instance: $ST_s = {T_i}^s \cup \mathcal{T}$ where ${T_i}^s \rightarrow {I_i}^s$ and $\mathcal{T} = \{{T_j}^p : {T_j}^p \rightarrow {I_j}^p, {I_j}^p \mapsto Art_j, ({I_i}^s, {I_j}^p) \in R_{ij}\}$. Artifact Art_i is called primary or main artifact for L and Art_j - secondary artifact.*

Figure 2 shows a trace in an artifact synchronization log from the example for primary artifact - Meeting Proposal and the secondary artifact - Participant. Note that an artifact synchronization log has the usual structure of a log and can be represented using existing log formats such as XES and MXML.

Using the artifact synchronization logs, the process of discovering inter-artifact conditions (also called *synchronization conditions*) consists of three steps. The first step is to discover which stages should contain such conditions (i.e., are synchronization points). In some cases, the synchronization points of an artifact might be known in advance. For completeness, however, we assume that this is not the case and thus they need to be discovered. For a given synchronization point, at the next step, we discover the best candidates for synchronization conditions. These two sub-problems are discussed in Section 3.1. The generated conditions are then assigned a confidence score, allowing the user to rank them and focus on the most likely ones. This is discussed in Section 3.2.

1970-01-07T03:59:00+02:00	**InitiateMeetingPlanning**	id=769
1970-01-07T04:02:00+02:00	**ProposeDateTime**	id=769
1970-01-07T04:03:00+02:00	**ReceiveProposal**	id=769 participant=5
1970-01-07T04:04:00+02:00	**ReceiveProposal**	id=769 participant=6
1970-01-07T04:05:00+02:00	**ReceiveProposal**	id=769 participant=4
1970-01-07T04:06:00+02:00	**ReceiveProposal**	id=769 participant=2
1970-01-07T04:07:00+02:00	**ReceiveProposal**	id=769 participant=1
1970-01-07T04:08:00+02:00	**ReceiveProposal**	id=769 participant=3
1970-01-07T04:16:00+02:00	**AnswerACCEPT**	id=769 participant=6
1970-01-07T04:16:00+02:00	**AnswerACCEPT**	id=769 participant=4
1970-01-07T04:17:00+02:00	**AnswerHOST**	id=769 participant=1
1970-01-07T04:22:00+02:00	**AnswerACCEPT**	id=769 participant=2
1970-01-07T04:24:00+02:00	**AnswerACCEPT**	id=769 participant=5
1970-01-07T04:26:00+02:00	**AnswerHOST**	id=769 participant=3
1970-01-07T04:39:00+02:00	**ProposalSuccessful**	id=769
1970-01-07T04:44:00+02:00	**ConfirmMeeting**	id=769

Fig. 2. Partial artifact synchronization trace from the Meeting Planning process

3.1 Discovery of Synchronization Points and Conditions

Here we define a simple heuristic which allows us to filter out the points that are most probably not synchronization points. The remaining points are the points for which conditions will be generated. The intuition behind the proposed heuristic is that a synchronization point will sometimes be forced to wait for the synchronization condition to become true (the instances of the other artifact to reach the desired states) even though the instance might be ready to open the stage based on the state of its life cycle alone.

For each occurrence of a candidate synchronization point, we define a window starting at the point in time when it is known that the activity is ready to be executed, based on the state of the instance, until the point in time when it is known that the activity has started.

Definition 4 (Window). *Let L be an artifact synchronization log with primary artifact Art_i and secondary artifact Art_j. Let $T = e_1 e_2 \dots e_n$ be a trace in L and event type A is a candidate synchronization point for Art_i. Let e_k be an event instance of A in T with timestamp $\tau(e_k)$. For the event instance e_k, a window w is $w = (\tau(e_m), \tau(e_k))$ with starting point $\tau(e_m)$ and end point $\tau(e_k)$, $\tau(e_m) < \tau(e_k)$, $\tau(e_m)$ is the timestamp of event e_m of the primary artifact Art_i in T and there exists no other event e of Art_i in T for which $e_m < e < e_k$.*

Definition 5 (Window activity level). *The activity level in window $w = (\tau(e_m), \tau(e_k))$ of event e_k in trace T is the number of events of the secondary artifact Art_j occurring in the window interval in T.*

Finally, we define the activity level for each event type considered as a candidate synchronization point:

Definition 6 (Activity level). *The activity level $AL(A)$ of a candidate synchronization point A in a log L is the average window activity level for the occurrences of A in L.*

Activity levels smaller than δ for a sufficiently small δ indicate that the candidate is most probably not a synchronization point. In the experiments $\delta = 1$ was used. This proved suitable in experiments with two real-life and one syntheric log. If longer delays are expected between the time point when the task is ready to be executed and the actual execution, higher values for δ might be more appropriate. For our example, Meeting Proposal has two synchronization points: ProposalSuccessful and ProposalFailed; Participant has one synchronization point: ReceiveProposal. Event type ProposalFailed, on the other hand, has activity level zero and is not considered to be synchronization points.

We now discuss the proposed method for discovering conditions for a given synchronization point. The general form of the discovered conditions is as follows:

Definition 7 (Synchronization condition). *A synchronization condition $C(S)$ of a synchronization point S is the disjunction $\bigvee_{i=1..n} C_i$ such that $C_i = \bigwedge_{j=1..m}(a_j \; op \; v_j)$ where a_j is a variable that corresponds to an event type A_j in the secondary artifact, $v_j \in \mathbb{N}$ and $op \in \{\leq, >\}$.*

An elementary condition $a_j \; op \; v_j$ is interpreted as: the number of instances of the secondary artifact where an event instance of A_j was the most recently executed event needs to be at most/greater than v_j for the stage S to open.

In order to discover such conditions, we represent the relevant data as feature vectors which form one dataset for each candidate synchronization point w.r.t. a specific secondary artifact. Let S be a candidate synchronization point in primary artifact Art_1 w.r.t. secondary artifact Art_2. For each event type A_i in Art_2 we construct an integer feature F_i such that, for a specific execution (event e) of S, $F_i(e) = v$ where v is the number of related instances of Art_2 for which the last occurred event at the time of occurrence of e was of event type A_i. Intuitively, v instances were in state "A_i executed" when activity S was executed. We refer to these features as *synchronization features*. More precisely, for the set of instances $\{I_1, \ldots, I_m\}$ of Art_2 appearing in the same synchronization trace as e:

$$F_k(e) = |\{I_i : \exists e_1 \in I_i, A(e_1) = A_k, e_1 < e \wedge (\forall e_2 \in I_i : e_2 < e_1 \vee e_2 > e)\}|.$$

Applying this approach to the artifact synchronization log for Art_1 w.r.t. Art_2, we generate a set of positive examples in the dataset for S.

Definition 8 (Positive example). *Let S be a synchronization point in artifact Art_i for which we are generating synchronization conditions. Let e be an event of event type S in artifact synchronization log L with main artifact Art_i and secondary artifact Art_j. Let $\{F_1, \ldots, F_n\}$ be the synchronization features where F_k is the feature for event type A_k of Art_j, $1 \leq k \leq n$. The feature vector $(F_1(e), \ldots, F_n(e))$ for event e is called a positive example.*

A positive example for synchronization point ProposalSuccessful from the trace in Figure 2 is the tuple (ReceiveProposal:0, AnswerREJECT:0, AnswerACCEPT:4, AnswerHOST:2) meaning that no related instances of the secondary artifact were in state ReceiveProposal executed or AnswerREJECT executed, four were in state AnswerACCEPT executed and two in state AnswerHOST executed.

Similarly, a set of negative examples is constructed. The difference is that the features do not correspond to executions S but to executions of any activity of the secondary artifact. Intuitively, these are configurations of the secondary artifact's instances which did not trigger the execution of S.

Definition 9 (Negative example). *Let S be a synchronization point in artifact Art_i for which we are generating synchronization conditions. Let e be an event in artifact Art_j in artifact synchronization log L with main artifact Art_i and secondary artifact Art_j. Let $\{F_1, \ldots, F_n\}$ be the synchronization features where F_k is the feature for event type A_k of Art_j, $1 \leq k \leq n$. The feature vector $(F_1(e), \ldots, F_n(e))$ for event e is called a negative example.*

A negative example for synchronization point ProposalSuccessful from the trace in Figure 2 is the tuple (ReceiveProposal:1, AnswerREJECT:0, AnswerACCEPT:4, AnswerHOST:1) recorded for the event AnswerHOST for participant 3. This means that at some point in time one instance of the secondary artifact was in state ReceiveProposal executed, none in state AnswerREJECT executed, four in state AnswerACCEPT and one in state AnswerHOST and this configuration did not trigger the execution of ProposalSuccessful.

Definition 10 (Dataset). *Let S be a synchronization point in Art_i w.r.t. Art_j and let L be the synchronization log with primary artifact Art_i and secondary artifact Art_j. The dataset for S in L contains one synchronization feature for each event type of Art_j and consists of the following feature vectors:*

- *For each execution of S in L we construct one positive example.*
- *For each execution of event from Art_j we construct one negative example.*

The resulting data set can be used to generate a classifier [19] that distinguishes between the positive and the negative examples. Since we are looking for an explicit representation of the discovered conditions, a natural choice for a classification algorithms is a decision tree algorithm as the tree can be transformed into rules in a straightforward way. We select the rules predicting a positive result (activity S executed) and the conditions (antecedents) of these rules form the synchronization condition as part of the sentry of the guard for stage S.

For example, the synchronization condition discovered for ProposalSuccessful is: "ReceiveProposal ≤ 0 and AnswerREJECT ≤ 2 and AnswerHOST > 0", which is interpreted as: no instances are in state ReceiveProposal executed, at most 2 instances are in state AnswerREJECT executed and at least one instance is in state AnswerHOST executed.

Using this approach can sometimes result in identical feature vectors being both positive and negative examples. In order to avoid this inconsistency, we remove from the set of negative examples any feature vector that corresponds to an event occurring immediately after an execution of S. These record the same configuration as for the corresponding positive example (the execution of S).

3.2 Confidence Score of Synchronization Conditions

After discovering the condition for every candidate synchronization point, we apply additional analysis in order to assign a confidence score to each of them. We consider three factors as part of the confidence score.

First, we consider the quality of the generated decision tree. The intuition here is that the better the classification given by the tree, the higher the confidence score should be. We use the well-known and often-used F-measure [2] to assess the quality of classification given by the generated decision tree. The F-measure combines the precision and recall of the model, defined as follows. Here tp is the number of true positive examples (the positive examples correctly classified by the model as positive), tn is the number of true negative examples (the positive examples correctly classified by the model as negative), fp is the number of false positive examples (negative examples incorrectly classified by the model as positive and fn is the number of false negative examples (the positive examples incorrectly classified by the model as negative).

Definition 11 (Precision). *The precision $P(M)$ of a classification model M is the percentage of true positive examples out of all examples classified as positive by the model:* $P(M) = \mathsf{tp}/(\mathsf{tp} + \mathsf{fp})$.

Definition 12 (Recall). *The recall $R(M)$ of a classification model M is the percentage of true positive examples out of all positive examples:* $R(M) = \mathsf{tp}/(\mathsf{tp} + \mathsf{fn})$.

Definition 13 (F-measure). *The F-measure $F(M)$ for a classification model M is defined as:* $F(M) = 2P(M)R(M)/(P(M) + R(M))$.

The second factor we consider in the confidence score is the size of the tree which we denote by $S(M)$. The intuition behind it is that the conditions used in practice are simple and a larger tree will most probably be a sign of overfitting the data to discover patterns that are not actually present. A decision tree can be built from any data set even a random one. Such a tree will be large to describe every data point. The tree size factor aims at filtering out such cases.

To include in the confidence score, we measure the number of leaves in the tree and then normalize so that the lowest possible size (i.e. 2 leaves) becomes 1 and the highest observed size among all generated trees becomes 0.

Finally, we also use the previously-defined activity level $A(S)$ in the confidence score which is also normalized so that the lowest observed activity level becomes 0 and the highest observed activity level becomes 1.

Definition 14 (Confidence score). *The overall confidence score $C(S)$ for the conditions discovered for model M generated for candidate synchronization point S is defined as:* $C(S) = (F(M) + S(M) + A(S))/3$.

The confidence score calculated for the discovered synchronization rule for ProposalSuccessful was 0.9866.

Note that the confidence score is an arithmetic average, however a weighted average could be used instead. This latter option pre-supposes however that the user has a reason to weigh one factor higher than others.

4 Validation

All methods presented in this section have been implemented within the ProM framework as part of the Artifact Mining package. For decision tree generation, the WEKA suite [5] was used and, more specifically, WEKA's J48 implementation of the C4.5 algorithm [14]. In our implementation, C4.5 was used with 10-fold cross-validation, no pruning and the minimal number of points in a leaf was set to 1. Similar results were obtained for the minimal number of points set to 2. Higher values can result in stopping the splitting prematurely and higher misclassification rate.

In this section we describe the data used for validation and the results received from applying the implemented tools.

The TBM data describes the application and funding process of a funding programme on applied biomedical research - TBM (Toegepast Biomedisch Onderzoek) managed by the IWT agency in Belgium (Flemish region). The data covers project application receipt, evaluation, reviewing, acceptance or rejection, contract signing and payments processing, for the period 2009-2012.

The data was collected from the funding agency's database in the form of three spreadsheets each describing one part of the process: project proposals, reviews and payments. It includes timestamps of events in the life cycles of proposals, reviews and payments as well as relevant data attributes such as project id, reviewer id, reviewer role, partner id, payment number and so on. The data was transformed into a raw log format containing 1472 events and the whole tool chain of methods presented in [13] and this paper was applied. As expected, we discovered three artifact types: Project (121 instances), Review (777 instances) and Payment (50 instances) (Fig. 3).

The Project artifact instances are identified by the projectID and their life cycles cover the process of proposal submission, initial evaluation for adherence to the formal requirements, if approved, then the final decision is taken (based on the received reviews) and, if accepted, the contract is signed. This includes the following event types: ProjectReceived, ProjectAccepted, ProjectRejected, ProjectDecided, ContractIn, ContractOut.

The Review artifact instances are identified by attribute pair (projectID, reviewerID) and each instance describes the life cycle of a review for a project proposal by a reviewer. Due to missing data, only one event type belonging to the Review artifact was present in the log - review completion (ReviewIn).

Finally, the Payment artifact instances are identified by the attribute pair (betaling, partnerID) where betaling refers to the number of the payment for this particular project partner. The life cycle includes the event types ApprovalCO, ApprovalWA, SentToBank, StartApprovalPayment, Signing, Payed.

In the model in Fig. 3, the attribute projectID is a foreign key in the Payment and Review entities which establishes the relationship with the Project entity. One project can have multiple reviews and multiple payments. Each review and each payment are for a single project.

Using the raw log and the discovered ER model we generated the artifact synchronization logs - a separate log for each combination of primary and sec-

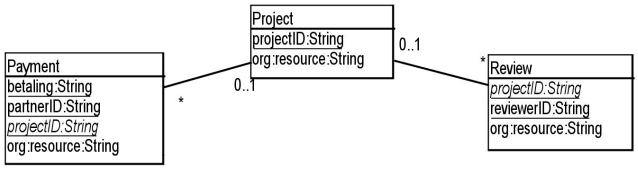

Fig. 3. The discovered ER model for the TBM data.

ondary artifact. For each log, we apply the tool for discovering inter-artifact synchronization conditions. Not all logs contain such conditions. For example the log with Project as a primary artifact and Payment as a secondary artifact generates an empty set of synchronization conditions since no activity in the Project artifact is waiting for any activity in the Payment artifact.

Table 4 shows all synchronization conditions found for the TBM data. The last column gives the number of times the activity was executed in the logs (each execution will generate a positive example in the data set). The features of the data set are equal to the number of event types in the secondary artifact. The first condition says that the approval of payment can only start after the contract has been signed. The second condition says that the review process can only be started after the project has been administratively accepted (i.e. conforms to the formal criteria and is accepted for further evaluation). The third condition says that a final decision on the project can only be taken if at least 5 reviews were completed. The last condition says that an additional approval process for the payment can only be performed if the contract has been signed and is the only condition with lower confidence score. This is due to lower F-measure for the decision tree indicating higher number of exceptions where the condition is not satisfied. Such rules can either be excluded or presented to the user for confirmation. The tool manages to filter out 22 of the candidates which are not real synchronization points (we consider each event type in an artifact as a candidate for a synchronization point w.r.t. each other artifact).

Primary art.	Secondary art.	Synchronization point	Condition	Conf. score	Occurrences
Payment	Project	startApprovalPayment	contractIn > 0	0.97	53
Reviewer	Project	reviewIN	ProjectAccepted > 0	0.97	777
Project	Reviewer	ProjectDecided	reviewIN > 4	0.87	112
Payment	Project	approvalIWA	contractIn > 0	0.64	53

Fig. 4. The conditions discovered for the TBM data.

5 Related Work

Process mining [1] methods have been developed in many areas, e.g. process discovery from logs, conformance checking, performance analysis and so on. A number of process discovery methods exist including the heuristics miner [17], the ILP miner [18], etc. Most generate a single flat model, usually a Petri Net and thus cannot represent synchronization based on unbounded number of events as

in the case when a process spawns a variable number of subprocesses. A few methods generate hierarchical models (e.g. [4, 10, 20]) but still do not allow one-to-many relationships between the main process and its subprocess, and thus are not concerned with discovering unbounded synchronization conditions.

In [12] a method was presented for mining artifact-centric models by discovering the life cycles of the separate artifacts and translating them into GSM notation. This method does not consider the question of how artifact instances synchronize their behavior.

The discovery of branching conditions in a single flat model has been addressed in [15, 9]. Such conditions determine which branch of the model to choose based on the current values of relevant variables at execution time and are also not applicable for synchronization with unbounded number of processes or sub-processes. The method is based on decision tree mining which is also the approach taken in this paper.

In the area of specification mining, methods exist for mining specifications which carry parameters that are instantiated to concrete values at runtime [8]. Most, however, do not tackle the problem of process synchronization. A method for mining guards for events based on messages received was presented in [7]. It allows to discover some types of guard conditions but is not able to discover conditions of the type: at least n messages of a certain type are received.

To the best of our knowledge, despite the large body of work in the field of process mining, the problem of discovering unbounded synchronization conditions is open and is addressed in this paper.

6 Conclusion and Future Work

The method presented in this paper can be extended in a number of ways. It is possible to consider synchronization with multiple artifacts as well as relative rather than absolute conditions, e.g. "Half of the participants accept the proposal" or "30% of the reviews are completed". We can also consider the information models of the secondary artifact instances and generate conditions based also on the data rather than life cycle only, e.g. "At least three participants located in USA and at least one in UK have accepted the invitation and at least one of them has chosen to be a host".

Another avenue for future work is to adapt and test the proposed method to discover completion conditions of multi-instance activities in BPMN process models. Until now, little attention has been given in the field of automated process discovery to the identification of advanced process modeling constructs such as multi-instance activities.

Finally, additional testing on real-life event logs would be beneficial in order to better test the performance of the developed methods and gain insights into how to tune the δ parameter and how to assign weights for the three components of the confidence score, which are currently left unweighted.

Acknowledgment

This research is supported by the EU's FP7 Program (ACSI Project). Thanks to IWT and Pieter De Leenheer for facilitating the logs used in this research.

References

1. van der Aalst, W.M.P. Process Mining: Discovery, Conformance and Enhancement of Business Processes. Springer (2011).
2. Baeza-Yates, R., Ribeiro-Neto, B.: Modern Information Retrieval. Addison-Wesley (1999).
3. Cohn, D., Hull, R.: Business artifacts: A data-centric approach to modeling business operations and processes. IEEE Data Eng. Bull., 32, 3-9 (2009).
4. Greco, G., Guzzo, A., Pontieri, L.: Mining Hierarchies of Models: From Abstract Views to Concrete Specifications. BPM 2005, LNCS 3649, 32–47 (2005).
5. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.: The WEKA data mining software: An update. SIGKDD Explorations, 11 (2009).
6. Hull, R. et al. Introducing the guard-stage-milestone approach for specifying business entity lifecycles. In: Proc. of 7th Intl. Workshop on Web Services and Formal Methods (WS-FM 2010), LNCS 6551. Springer-Verlag (2010).
7. Kumar, S., Khoo, S., Roychoudhury, A., Lo, D.: Inferring class level specifications for distributed systems. ICSE 2012: 914–924 (2012).
8. Lee, C., Chen, F, Rosu, G.: Mining Parametric Specifications, ICSE'11, ACM, 591–600 (2011).
9. de Leoni, M., Dumas, M., Garca-Bauelos, L.: Discovering Branching Conditions from Business Process Execution Logs. FASE 2013, Springer LNCS, 114–129 (2013).
10. Li, J., Bose, R.P.J.C., van der Aalst, W.M.P.: Mining Context-Dependent and Interactive Business Process Maps Using Execution Patterns. In: BPM 2010 Workshops, LNBIP 66, 109-121, 2011. Springer-Verlag (2011).
11. Nigam, A., Caswell, N. S.: Business artifacts: An approach to operational specification. IBM Systems Journal, 42(3), 428-445 (2003).
12. Popova, V., Dumas, M.: From Petri Nets to Guard-Stage-Milestone Models, In: Proc. of BPM 2012 Workshops, Springer LNBIP 132, 340–351 (2013).
13. Popova, V., Fahland, D., Dumas, M.: Artifact Lifecycle Discovery, arXiv:1303.2554 [cs.SE] (2013).
14. Quinlan, J.: C4.5: Programs for Machine Learning. Morgan Kaufmann (1993).
15. Rozinat, A., Van der Aalst, W.M.P.: Decision Mining in ProM. In: Proc. of BPM 2006, LNCS 4102, 420-425. Springer-Verlag (2006).
16. Verbeek, H., Buijs, J. C., van Dongen, B. F., van der Aalst, W. M. P.: ProM: The process mining toolkit. In: Proc. of BPM 2010 Demonstration Track, CEUR Workshop Proceedings, vol. 615, 2010.
17. Weijters, A.J.M.M., Ribeiro, J.T.S.: Flexible Heuristics Miner, In: Proc. of the IEEE Symposium on Computational Intelligence and Data Mining, 310–317 (2011).
18. van der Werf, J.M.E.M., van Dongen, B.F., Hurkens, C.A.J., Serebrenik, A.: Process discovery using integer linear programming. In: PETRI NETS, LNCS 5062, 368-387 (2008).
19. Witten, I., Frank, E., Mark, A.: Data Mining: Practical Machine Learning Tools and Techniques. Elsevier, 3rd ed. edition (2011).
20. Yzquierdo-Herrera, R., Silverio-Castro, R., Lazo-Cortes, M.: Sub-process Discovery: Opportunities for Process Diagnostics. In: LNBIP 139, 48-57 (2013).