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.

Keywords: Process Mining, Automated Process Discovery, Artifact-Centric Process, Synchronization Condition

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 designed to discover individual (monolithic) process models, as opposed to models structured in terms of subprocess models. A corollary of this limitation is that these methods 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 then waits for a subset of the spawned subprocesses to complete before proceeding. In the BPMN notation for example, this situation arises when a process containing a multi-instance activity, 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, 12] 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 [5, 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. Although the focus is on artifact-centric models, the principles of the proposed method can be used to discover unbounded synchronization conditions in other settings, such as discovering completion conditions of multi-instance activities in BPMN as exemplified above.

The presented methods are implemented as plug-ins for the ProM open-source process mining framework [17]. The implementation is part of the ArtifactModeling package available from www.processmining.org. The proposal has been 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 section 6 discusses related work. Next, a simple scenario that serves as a motivating example is presented in Section 3. Section 4 then presents the proposed method for discovering inter-artifact synchronization conditions. The validation on the grant assessment process is discussed in Section 5. Finally Section 7 concludes the paper with a discussion 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, 12]. 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.

Most existing work on business artifacts has focused on the use of life cycle models based on variants of finite state machines. Recently, a new approach was introduced - the Guard-Stage-Milestone (GSM) meta-model [5, 6] for artifact life cycles which is more declarative than the finite state machine variants and 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 activity preformed for, with or by an artifact instance intended to achieve one of the milestones belonging to the stage. Milestones correspond to business-relevant operational objectives, and are achieved (and possibly invalidated) based on triggering events and/or conditions over the information models of active artifact instances. Guards control when stages are activated, and, as with milestones, are based on triggering events and/or conditions. A stage can have one or more guards and one or more milestones. It becomes active (or open) when a guard becomes true and inactive (or closed) when a milestone becomes true.

Sentries are used in guards and milestones to control when stages open and when milestones are achieved or invalidated. Sentries contain a triggering event type and/or a condition. The events may be external or internal, and both the internal events and the conditions may refer to the artifact instance or to other artifact instances.

3 Motivating Example

As a motivating example we consider the following meeting planning process. A meeting assistant tries to organize a meeting between a group of people which in our example consists of 6 participants. The assistant proposes time and date for the meeting and communicates this to the participants. Each participant receives the proposal and considers their availability. If they are not available they reject the proposal. If they are available, they consider whether they are prepared to host the meeting and either accept the proposal (time is convenient but not prepared to host) or propose to host the meeting.

The meeting proposal is successful if at least three participants are available and at least one of them is prepared to host. If the proposal fails (more than three rejects or no proposal to host) the assistant proposes a new time and date for the meeting 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

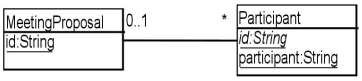

Fig. 1. Artifact types in the motivating scenario: Meeting Proposal and Participant.

(id, participant). Attribute id in Participant is a foreign key that shows how instances of one artifact type are related to instances of the other. For testing, the scenario was implemented in CPN Tools to generate event logs.

4 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 certain stage has been completed.

For our meeting planning example such condition can be defined, for instance, for the stages Meeting Successful and Meeting Failed of artifact Meeting. Meeting Successful can only open if not more than 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. Similarly, Meeting Failed can only open if at least 3 instances of artifact Participant have completed stage Reject Proposal or no instance has completed stage Host Meeting.

The previously presented methods in [14] provide us with the means to extract the knowledge of which artifacts have relationships with which other artifacts through primary-foreign key relations. By looking at the specific values of the primary and the foreign key attributes we can determine which specific instances of these artifacts 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 called *artifact synchronization logs* which will be defined in the next sub-section.

Using these logs, the process of discovering inter-artifact guard 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). For a given synchronization point, at the next step, we discover the best candidates for synchronization conditions. These two sub-problems are discussed in sub-sections 4.2 and 4.3 respectively.

At the last step, the generated conditions are evaluated and scored which allows to attach a confidence score to them and, if needed, to filter out the ones with lowest confidence. The scoring process is discussed at the end of this section in subsection 4.5. Finally, implementational details are given in subsection 4.6.

4.1 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 $\{A_1, A_2, \ldots, A_n\}$ 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 A_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 $v_i = (A_i, d_i)$ where A_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 will denote the event type of event e by $a(e)$ and the timestamp of e by $\tau(e)$.

Definition 2 (Raw log). *A* raw log *L is a finite sequence of events $L = e_1 e_2 \ldots e_n$. L induces the total order $<$ on its events with $e_i < e_j$ iff $i < j$.*

The order of events in L respects the temporal order of their timestamps, i.e. if event e precedes event e' temporally then $e < e'$.

The general definition for a log does not put any restriction on the types of events to be included in it, if and how they are related to each other and if and how they are grouped within the log. In practice, based on the log's collection method and purpose, logs can differ significantly.

A raw log imposes no additional internal structure of the events and assumes that all events in the log represent the behavior of a single system. This system can execute a single process or, for a multi-artifact system, can contain events reflecting the behavior of instances of multiple artifacts running in parallel.

Artifact-centric log consists of the events belonging to a single artifact and can represent the behavior of multiple instances of this artifact existing in parallel. The events of one instance are grouped in a single trace, therefore, the log consists of multiple traces of ordered events.

Artifact instances in a multi-artifact system can be related to each other in various ways, through instance creation, communication and so on. Our previously developed methods methods (see [14]) can be used to discover which artifact instances are related using, among other, methods for discovering functional and inclusion dependencies. These methods cannot provide reliable information about the nature of the interaction however they allow to discover foreign-primary key relationships between artifacts and the types of these relationships (1:n, m:n). Through these relationships, it is visible which instances of one artifact are related to which instances of another artifact.

For the purposes of this paper, such information is assumed to be accessible either using the above-mentioned approach or additional domain knowledge.

This allows us to define and generate so-called artifact synchronization logs which will then be used as input for the methods presented in this paper.

Let $\mathcal{A} = \{Art_1, \ldots, Art_n\}$ be an artifact system consisting of 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. In the following we denote by ${I_i}^j \mapsto Art_i$ the fact that instance ${I_i}^j$ is an instance of artifact Art_i. Also, ${T_i}^j \rightarrow {I_i}^j$ denotes that trace ${T_i}^j$ is an artifact log trace for instance ${I_i}^j$ and consists of all events of that instance.

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 part of a trace in an artifact synchronization log from a simulation of the Meeting Planning example. For this log the primary artifact is Meeting Proposal and the secondary artifact - Participant. Here only the most relevant attributes are included and the event types are printed in bold face.

1970-01-07T03:59:00+02:00	**InitiateMeetingPlanning**	id=769
1970-01-07T04:02:00+02:00	**ProposeDateTime**	id=769
1970-01-07T04:06:00+02:00	**ReceiveProposal**	id=769 participant=5
1970-01-07T04:06:00+02:00	**ReceiveProposal**	id=769 participant6
1970-01-07T04:06:00+02:00	**ReceiveProposal**	id=769 participant4
1970-01-07T04:06:00+02:00	**ReceiveProposal**	id=769 participant2
1970-01-07T04:06:00+02:00	**ReceiveProposal**	id=769 participant1
1970-01-07T04:06:00+02:00	**ReceiveProposal**	id=769 participant3
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 a simulation of the Meeting Planning process

4.2 Discovery of 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 therefore they need to be discovered.

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.

The "waiting" here is not measured based on time as the timestamps might or might not be reliable and the waiting time can be domain-specific and depend highly on the type of activity, delays in recording, implementation details, etc. Instead we consider the number of events occurring in the "waiting window". More precise definition will be given in the following paragraphs.

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. Based on the quality and the completeness of the available logs, this definition can be instantiated in different ways.

Let us assume that sufficiently complete logs are available, including all relevant data values as well as starting and completion points of activities. Based on such logs, life cycle models can be generated for each artifact. Alternatively such models might already be available. For these models, intra-artifact data-dependent conditions can also be generated.

Assuming, for example, a Petri Net model is available for the artifact, it is then possible to replay each artifact trace in the model and find out at which point in the trace the transition representing the candidate synchronization point is enabled (tokens are present in all its pre-places). This marks the beginning of the window and the end of the window is when the activity starts its execution.

Unfortunately in real-life situations the quality of the available information and logs is usually far from ideal. A more realistic scenario would be such that the logs only contain the activity completion event but not the starting event. It might also be possible that a good quality model with good conformance to the logs is not available. In this case, we approximate the window based only on logs in the following way.

Definition 4 (Window (log-based)). *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 $e_m < e < e_k$.*

Irrespective of whether we use a log-based or model-based definition for the windows, we can calculate the activity level for each window as follows:

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. Experiments show that $\delta = 1$ is a good assignment.

For our example scenario, artifact type Meeting Proposal has two synchronization points: ProposalSuccessful and ProposalFailed. Artifact type Participant has one synchronization point: ReceiveProposal. Event types such as ProposalFailed and ProposeDateTime, on the other hand, have activity level zero and are therefore not considered to be synchronization points (see Figure 2).

4.3 Discovery of Conditions for a Synchronization Point

In this sub-section we 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 in state where an event instance of A_j was the most recently executed event needs to be smaller or equal to/greater than v_j in order for the stage S to open.

Note that it is possible to adapt and reuse the definition and the developed methods to discover conditions containing fractions rather than number of instances in the conditions (relative rather than absolute conditions). This will result in conditions such as "Half of the participants accept the proposal" or "30% of the reviews are completed". In the rest of the paper we focus on absolute conditions but the discussion is equally applicable for relative conditions.

In order to discover this type of conditions, we represent the relevant data as feature vectors which form one dataset for each candidate synchronization point with respect to a specific secondary artifact.

Let S be a candidate synchronization point in primary artifact Art_1 and with respect to secondary artifact Art_2. Let the set of event types in Art_2 be

$A = \{A_1, \ldots, A_n\}$. For each event type A_i in A we construct a feature F_i of integer type 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, this can be interpreted as: v instances were in state "A_i executed" when activity S was executed. In the following we will 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 whole artifact synchronization log for Art_1 (primary artifact) with respect to Art_2 (secondary artifact), we generate a set of feature vectors which reflect all observed configurations of Art_2 at the times when activity S was executed. These form the 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 at the time ProposalSuccessful was executed, none were in state AnswerREJECT executed, four were in state AnswerACCEPT executed and two in state AnswerHOST executed.

In a similar way a set of negative examples is constructed. The difference however is that each feature does not correspond to an execution of activity S but to the execution of any activity of the secondary artifact. The intuition here is that these are configurations of the secondary artifact's instances which did not trigger the execution of S. Note that we do not need to redefine the synchronization features as they refer any event e.

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 with respect to 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 the secondary artifact Art_j we construct one negative example.*

The resulting data set can be used to generate a classifier [21] 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.

The generated decision tree can be transformed into rules in a straightforward way, we can then select only the rules predicting a positive result (activity S executed), therefore 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.

Before applying the classification algorithm, we use several methods to refine the dataset in order to improve the quality of the data. These methods are described in the next sub-section.

4.4 Data Set Refinement

If we use the above described approach for generating negative examples directly and without modification, this can sometimes result in identical feature vectors being classified both as positive and negative examples.

In order to avoid this inconsistency, for the dataset of synchronization point S, 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 the one in the corresponding positive example (the execution of S).

For the trace in Figure 2, let us assume that InitiateMeetingPlanning is a synchronization point for which we are generating a dataset. ReceiveProposal was executed immediately after InitiateMeetingPlanning in an instance of the secondary artifact. At that time the configuration of the secondary artifact's instances would be the same as for the execution of InitiateMeetingPlanning, therefore a positive and a negative example with the same values would be included. To avoid this, the negative example is not included in the dataset.

Furthermore, we remove the duplications to find out how many unique examples are present. It will often be the case that the negative examples will outnumber the positive examples which is known to pose difficulties to the classification algorithms. For instance, predicting the majority class on all examples will generate a classifier with relatively low misclassification rate, yet, it does not correctly differentiate between the two classes.

One straightforward way to address the problem is to use re-sampling, i.e. changing artificially the class distribution. We can, for example, reduce randomly the set of negative examples (under-sampling). This however results in loss of data which might be important in correctly separating the positive and the negative examples by the classification algorithm.

Instead, we choose to duplicate the positive examples (over-sampling) until we end up with a balanced data set. In [7] over-sampling was shown to outperform under-sampling for the classification of imbalanced data and both produce better results than the original data set. Other, more sophisticated, approaches have also been proposed in the literature (see [8, 18] for an overview) for addressing the problem of imbalanced data and some of them could be used here to improve the results further.

The resulting data set is used as an input for the decision tree algorithm.

4.5 Scoring 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 one of them. This can be used to rank the results.

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. The quality of classification models is usually not measured on the data used to build the model as the model might overfit the data and give unrealistically high results.

As the data sets generated using the proposed approach are often relatively small, splitting the data into a training and a validation sets is usually not a good option. Instead, we use 10-fold cross-validation which iteratively splits the data in training and validation sets and averages the results.

We use the well-known and often-used F-measure [2] from the area of Information Retrieval to represent 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:*

$$P(M) = \mathsf{tp}/(\mathsf{tp} + \mathsf{fn}).$$

Definition 13 (F-measure). *The F-measure $F(M)$ for a classification model M is defined as:*

$$F(M) = 2P(M)R(M)/(P(M) + R(M))$$

where $P(M)$ is the precision of the model and $R(M)$ is the recall of the model.

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 in the data.

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.

4.6 Implementation

All methods presented in this section have been implemented as two plug-ins within the ProM [17] framework as part of the Artifact Mining package.

The first plug-in implements the generation of artifact synchronization logs. The second plug-in takes as input one synchronization log, discovers the best candidates for synchronization points (if such exist) and generates a synchronization condition and its score for any discovered synchronization point.

For decision tree generation, the WEKA suite [4] was used and, more specifically, WEKA's J48 implementation of the C4.5 algorithm [15]. In our implementation, C4.5 was used with 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 sometimes resulted in stopping the splitting prematurely and higher misclassification rate.

5 Validation

In this section we describe the data used for validating the developed methods and show the results received from applying the implemented tools on this data.

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 and the whole tool chain of methods presented in [14] and this paper was applied as described below.

The first step is to discover the underlying artifact structure. As expected, we discovered three artifact types: Project, Review and Payment (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 the attribute pair (projectID, reviewerID) and each instance describes the life cycle of a review for a specific project proposal by a specific reviewer. Normally it would include assignment of review, reviewer's confirmation and review completion, however, 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 morel

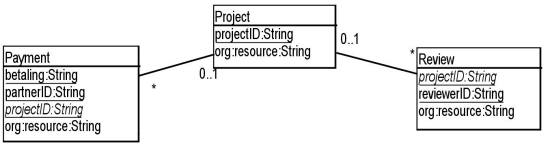

Fig. 3. The discovered ER model for the TBM data.

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 as an input, the artifact logs generation tool generates three logs - one for each artifact. From each of these logs, artifact life cycle models can be generated.

Using the raw log and ER model we can also generate the artifact synchronization logs. The tool produces a separate log for each combination of primary and secondary artifact which in this case is six logs - each of the three artifacts is taken once as primary and once as a secondary artifact.

For each of the generated logs, we can 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 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 with respect to each other artifact).

Primary artifact	Secondary artifact	Synchronization point	Condition	Confidence score
Payment	Project	startApprovalPayment	contractIn > 0	0.97
Reviewer	Project	reviewIN	ProjectAccepted > 0	0.97
Project	Reviewer	ProjectDecided	reviewIN > 4	0.87
Payment	Project	approvalIWA	contractIn > 0	0.64

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

6 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 [19], the ILP miner [20], 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.

In [13] 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 [16, 11]. 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 [10]. Most, however, do not tackle the problem of process synchronization. A method for mining guards for events based on messages received was presented in [9]. 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 will be addressed in this paper.

7 Conclusions and Future Work

The method presented in this paper can be extended in a number of ways. As mentioned earlier, it is possible to consider fractions rather than number of instances in the conditions. This will result in conditions such as "Half of the participants accept the proposal" or "30% of the reviews are completed". Another direction for future research is to also consider the information models of the secondary artifact instances and generate conditions based also on the data rather than life cycle only. This could generate conditions such as: "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".

This raises the more general question of how to choose between alternative candidate conditions for the same synchronization point. Such conditions can be of different formats and contain different types of information in the terms. The proposed confidence score is not directly applicable for such comparison - for example the simplicity component of the score, for which we used the size of the decision tree, needs to be replaced with a more general measure able to provide meaningful comparison.

Finally, additional testing on real-life event logs would be beneficial in order to gain more insight into the performance of the developed methods.

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. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.: The WEKA data mining software: An update. SIGKDD Explorations, 11 (2009).
5. 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).
6. Hull, R. et al: Business Artifacts with Guard-Stage-Milestone Lifecycles: Managing Artifact Interactions with Conditions and Events. In: DEBS 2011, 51–62 (2011).
7. Japkowicz, N., Stephen, S.: The class imbalance problem: a systematic study. Intelligent Data Analysis, 6(5), 429–450 (2002).
8. Kotsiantis, S., Kanellopoulos, D., Pintelas, P.: Handling imbalanced datasets: A review, GESTS International Transactions on Computer Science and Engineering, 30(1), 25–36 (2006).
9. Kumar, S., Khoo, S., Roychoudhury,A, Lo, D.: Inferring class level specifications for distributed systems. ICSE 2012: 914–924 (2012).
10. Lee, C., Chen, F, Rosu, G.: Mining Parametric Specifications, ICSE'11, ACM, 591–600 (2011).
11. de Leoni, M., Dumas, M., Garca-Bauelos, L.: Discovering Branching Conditions from Business Process Execution Logs, FASE 2013, Springer LNCS, 114–129 (2013).
12. Nigam, A., Caswell, N. S.: Business artifacts: An approach to operational specification. IBM Systems Journal, 42(3), 428-445 (2003).
13. Popova, V., Dumas, M.: From Petri Nets to Guard-Stage-Milestone Models, In: Proc. of BPM 2012 Workshops, Springer LNBIP 132, 340–351 (2013).
14. Popova, V., Fahland, D., Dumas, M.: Artifact Lifecycle Discovery, arXiv:1303.2554 [cs.SE] (2013).
15. Quinlan, J.: C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers (1993).
16. Rozinat, A., Van der Aalst, W.M.P.: Decision Mining in ProM. In: Proc. of BPM 2006, LNCS 4102, 420-425. Springer-Verlag (2006).
17. 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.
18. Weiss, G.M.: Mining with rarity: a unifying framework. SIGKDD Explor. Newsl., 6(1), 7–19 (2004).
19. Weijters, A.J.M.M., Ribeiro, J.T.S.: Flexible Heuristics Miner (FHM), In: Proceedings of the IEEE Symposium on Computational Intelligence and Data Mining, CIDM 2011, 310–317 (2011).
20. van der Werf, J.M.E.M., van Dongen, B.F., Hurkens, C.A.J., Serebrenik, A.: Process discovery using integer linear programming. In: van Hee, K.M., Valk, R. (eds.) PETRI NETS 2008. LNCS 5062, pp. 368-387 (2008).
21. Witten, I., Frank, E., Mark, A.: Data Mining: Practical Machine Learning Tools and Techniques. Elsevier, 3rd ed. edition (2011).