Self-learning Predictor Aggregation For the Evolution of People-driven Ad-hoc Processes

Christoph Dorn[1,3], César A. Marín[2], Nikolay Mehandjiev[2], and Schahram Dustdar[3]

[1] Institute for Software Research, University of California, Irvine, CA 92697-3455
cdorn@uci.edu
[2] Centre for Service Research, University of Manchester, Manchester M15 6PB, UK
firstname.lastname@manchester.ac.uk
[3] Distributed Systems Group, Vienna University of Technology, 1040 Vienna, Austria
lastname@infosys.tuwien.ac.at

Abstract. Contemporary organisational processes evolve with people's skills and changing business environments. For instance, process documents vary with respect to their structure and occurrence in the process. Supporting users in such settings requires sophisticated learning mechanisms using a range of inputs overlooked by current dynamic process systems. We argue that analysing a document's semantics is of uttermost importance to identify the most appropriate activity which should be carried out next. For a system to reliably recommend the next steps suitable for its user, it should consider both the process structure and the involved documents' semantics. Here we propose a self-learning mechanism which dynamically aggregates a process-based document prediction with a semantic analysis of documents. We present a set of experiments testing the prediction accuracy of the approaches individually then compare them with the aggregated mechanism showing a better accuracy. This supports our claim that our mechanism is reliable to support evolving people-driven ad-hoc processes.

Keywords: document analysis, process recommendation, people-driven ad-hoc processes, document and process evolution

1 Introduction

Changing business requirements, dynamic customer needs, and people's growing skills demand support for flexible business processes. Existing attempts to support such dynamic environments can be roughly categorised into ad-hoc processes [18], semi-structured or case-based processes [14], and well structured processes [13]. Their differences lie mostly in the flexibility at the process modelling level. Generally they allow the user to deviate from the prescribed course of action, adjusting the process according to the current needs. This flexibility comes at the price of users facing more complex situations while receiving little support on how to carry out their activities. Recent efforts partially address

this issue by recommending next steps and relevant documents [17, 4]. Yet the increase of information exchanged as documents in disparate forms adds to the cognitive load of the user.

People-driven ad-hoc processes prescribe the existence of a process model, which the user is completely free to deviate from using the presence of certain documents and their states to determine the next action to recommend to its user. Yet the prevailing assumption that process documents are always well structured, conveying a precise meaning and purpose is far from realistic. Achieving such a level of process flexibility — and subsequently providing the necessary user support — demands for an accurate recognition of imprecise documents, allowing the purpose of process activities to remain the same but the required input and output documents to vary.

Our approach applies the concept of process stability to determine whether the underlying process model or a document's semantic structure yield better prediction results. The analysis of prediction results feeds back in turn to automatically adjust the process stability and aggregation. To the best of our knowledge this dynamic predication combination is the first attempt to combine process information and document alignment to determine semantically the type of a given document. We present experimental results obtained through realistic simulations of changing processes and documents with varying semantic structure. Our results support our claim that a dynamic predictor aggregation outperforms individual predictors especially during evolving process phases.

In the remaining of this paper, Section 2 presents a reference scenario. Section 3 describes preliminaries and our approach. We outline the individual and aggregated predictors in Section 4. Experimental setup and results are discussed in Section 5. Related Work in Section 6 compares existing approaches to our contribution, followed by a conclusion and outlook on future work in Section 7.

2 Reference Scenario

The example in Figure 1 depicts a flexible people-driven order process. The individual work steps describe a general order of activities to successfully complete a process instance. The outlined flow, however, does not enforce the exact order of activities, which is up to the user, and covers no exceptions or process adaptations that might arise due to a specific customer request, incomplete information, or user specific expertise. Consequently, the listed types of documents that characterise the exchanged messages specify merely an initial set of expected documents. The process model in Figure 1 does not apply any particular modelling language but rather presents an intuitive view on the involved documents that represent input and output of activities. In this scenario, we encounter various forms of document-centric process evolution:

Missing Documents : When the *Replenish* step (C) is updated to make use of an automatic restocking system, only user confirmation is required and the *Quote* message (3) no longer occurs.

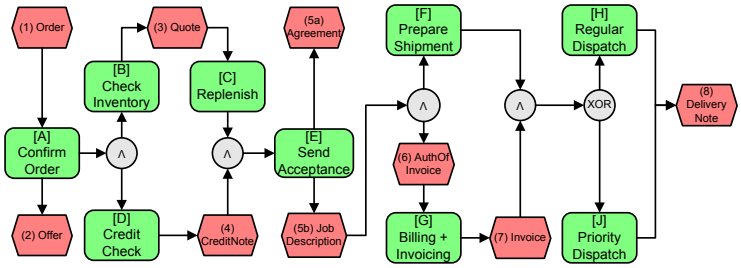

Fig. 1: Generic order process and associated types of documents.

Delayed Documents : A company potentially decides to delay the *Agreement* (5a) document until the *PrepareShipment* (F) activity signals the completion of the production and packaging sub process.

Premature Documents : For premium customers, shipping becomes independent of billing thus the *DeliveryNote* (8) potentially occurs before *Invoice* (7).

Shifted Documents : For regular customers, the company might decide to merge the *DeliveryNote* (8) into the *Invoice* (7) thereby having the invoice after the *PriorityDispatch* (H) and dropping the delivery note altogether.

3 Preliminaries

3.1 Document alignment

The alignment of documents consists of determining the best match between a document and a set of internal document representations [7], which include the expected document structure but also knowledge regarding what to do with this document. Therefore, this differs from the related activity of classifying documents. We call a business' internal representation of a document a *document type*. It is possible that more than one document type represents the same document, though with different degrees of similarity. However, the document alignment process provides the most appropriate one. In order to align a document to a document type, we need to define how documents are represented.

Representing and standardising the structural information of documents have been addressed by a number of efforts, cf.[12, 19]. In this paper we simply reuse the one explained in [11], where the hierarchical structure of the semantic concepts is represented by the path from the root concept to each of the leaf concepts. For example, consider the structure of the document in Listing 1.1 (upper part) and the representation of the concepts *Name* and *City* (lower part).

Notice that the two concepts *Name* belong to different entities and thus convey a different meaning. Also notice that even though the two concepts *City* possess the same meaning, they actually represent different concepts because of their position in the structure. We use this document representation for our

```
<Document>
   <ClientOrganisation>
      <Name>TUVUM Solutions</Name>
      <PhysicalAddress>
         <Street>Baker St</Street>
         <City>London</City>
      </PhysicalAddress>
      <PostalAddress>
         <POBox>1234</POBox>
         <City>London</City>
      </PostalAddress>
      <ContactPerson>
         <Name>Christoph Dorn</Name>
      </ContactPerson>
   </ClientOrganisation>
</Document>

Document->ClientOrganisation->Name
Document->ClientOrganisation->ContactPerson->Name
Document->ClientOrganisation->PhysicalAddress->City
Document->ClientOrganisation->PostalAddress->City
```

Listing 1.1: Document representation and semantic concepts.

experiments and leave the conversion of actual documents into XML out of the scope of this paper. We simply assume that others, cf.[9], have done it already.

3.2 People-driven Ad-hoc Processes

In [4] we introduced a modelling language for describing people-driven ad-hoc processes. Individual process steps are linked to connectors by means of transition arcs thereby creating a directed, a-cyclical graph. The main difference to previous works on flexible workflow support systems (e.g., [13, 2, 17]) is a supplementing sequence graph that describes the preferred order in which a user progresses through the activities in the various branches. An incoming transition details all document types that are required as inputs to an activity. An outgoing arc lists all document types that are produced by that activity — see Figure 2 for a process model excerpt describing part of the scenario. The process engine observes all messages as well as user actions to determine which process activities have been successfully completed, have been skipped, or are currently processed. Doing so, it keeps track of upcoming activities and expected incoming and outgoing documents. For each incoming document, the process recommendation algorithm then identifies which activity the user should carry out next, and what follow-up activities are suitable. This approach however, assumes that incoming documents are well-formed and thus are always mapped (i.e. aligned) to the correct document type.

3.3 A Framework for Self-learning Document Predictors

Our approach to self-learning predictor aggregation is visualised in Figure 3. The *Document-based Predictor* analyses the structure of every document occurring in

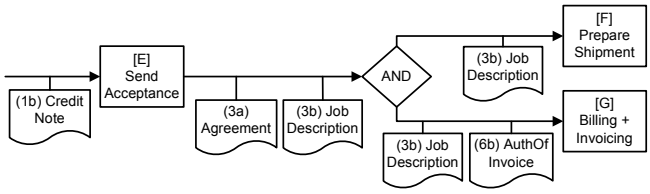

Fig. 2: Process model excerpt detailing document types on transition arcs between activities.

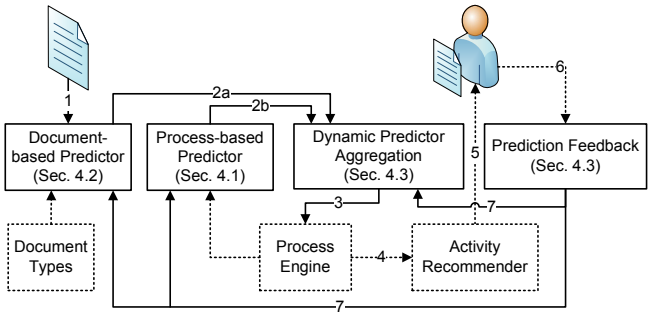

Fig. 3: Self-learning predictor aggregation through feedback analysis. Components point to their respective sections; remaining components are out of scope of the paper.

the process (1) and provides a ranking of likely types to the *Dynamic Predictor Aggregation* component (2a). Likewise, the *Process-based Predictor* provides expected document types (2b). The *Dynamic Predictor Aggregation* combines both rankings and provides the *Process Engine* (3) with the most likely document type, triggering (4) the *Activity Recommender* to suggest the next actions to the user (5). The *Prediction Feedback* component receives explicit user feedback in the form of document re-alignment, or implicit feedback by recommendation acceptance (6). This updates all three predictors to consider the prediction error and improve their individual performance (7). In the case of the dynamic aggregation, this includes an update of the process stability. Details on the process engine and activity recommender are out of scope of this paper as we focus on improving document type predictions, which consequently increase the precision of recommendations.

4 Document Predictors

A predictor P provides (for a particular process instance and given time) a probability measurement $prob(dt)$ in the interval $[0, 1]$ for each document type $dt \in DT$. The resulting prediction set R is normalised so that that $max(prob(dt) \in R) = 1$. Note that multiple document types within a set R can yield the maximum probability of 1. For example, upon completion of activity (E) producing

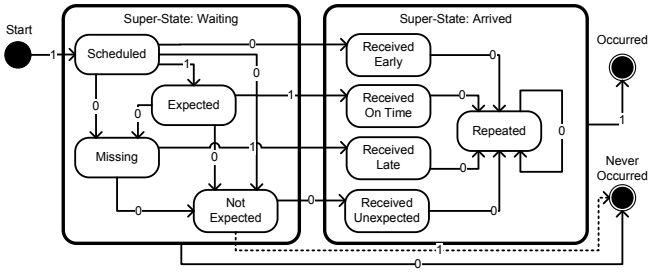

Fig. 4: State transition model for a document type. Each model instance provides the transition probabilities for particular combinations of document type and process model.

two documents (3a, 3b), the process-based predictor would potentially give them equal probability of 1.

4.1 Predictor based on Process Progress

Process-based prediction relies on a combination of the previously introduced process model with document type annotations (see Section 3.2) and a probabilistic document state model. A detailed discussion of the state model including the mechanisms for ageing and co-evolution with process changes is provided in [5]. We include a brief description of the state model here for completeness.

The document state model (Figure 4) describes the likelihood for transitions between various document states (such as scheduled, expected, arrived) during the process. We track only documents that are defined in the process model, each in a separate state model instance that is valid only for that particular pair of process type and document type. Based on incoming and outgoing documents as well as completed process steps, the process engine triggers the transition events in one or more message state models. In the scenario, for example, the *Agreement* and *JobDescription* documents remain in state *Scheduled* until a *CreditNote* triggers the activation of the *SendAcceptance* activity and thus sets both documents to *Expected*. In case the *PrepareShipment* activity is completed without a detected *JobDescription*, that document is set to *Missing*.

Ultimately, the process-based prediction leverages the transition labels between document states. In any of the *Waiting* super-states the probability is determined based on the transition to the respective received state. From any of the *Arrived* super-states the probability is determined by the transition weight to the *Repeated* state. For prediction, all documents are ranked according to their probability to occur in their current state. The top ranked documents constitute the prediction of newly intercepted documents.

The process-based predictor's main disadvantage is the lack of knowledge about the document's internal structure. Thus the predictor cannot discriminate between multiple types whenever a process step requires more than one input or

output documents, or when the process has arrived at a split where the branches are triggered by different document types.

4.2 Predictor based on Document Alignment

As mentioned in Section 3.1, documents are represented by the concepts contained within which indicate the semantic structure by representing the paths from the root concept to each of the leaf concepts. Therefore the meaning of the document consists of the set of concepts contained within it. Yet because in reality documents vary, aligning a document to a document type requires an approximation of one to the other. We perform this by calculating the observed "average" document type. We call this calculation the Centre of Mass.

We consider DT to be a set of document types already defined and which we want to align a document to. A document type $dt \in DT$ is composed by a set of concept paths $cp_{dt} \subseteq CP$ where CP represents the universe of known concept paths. Likewise, a document d is composed by a set of concept paths $cp_d \subseteq CP$. We say a document d is aligned to a document type dt if the condition $cp_d \cap cp_{dt} \neq \emptyset$ is valid. Normally, the greater the concept paths shared between d and dt the more similar the two are. A document is considered aligned to a document type when the two share most of their concept paths. We use the Centre of Mass cm as the approximate representative of a document type dt and is defined as follows

$$cm_{dt} = \prod_{\bigcup_{\forall d \mapsto dt}} \frac{|cp|}{|\{\forall cp \in \bigcup\}|} \tag{1}$$

where cm is a vector where each element contains the proportion [3] of each concept path cp with respect to all the concept paths included in the union of all documents d already aligned to a document type dt. Such a proportion can be also seen as the concept path weight to represent a document type. Once the Centre of Mass is calculated, we simply add up the weights of all the concept paths contained in the document we want to align. Obviously, concept paths in the document but not contained in the Centre of Mass do not convey any weight. The result is a value within the range $[0, 1]$, the higher the value the most similar it is to the approximation of that document type. This differs from the Centroid document classifier [6] where the cosine between two vectors (documents) is calculated to determine their similarity.

The Centre of Mass learns by aligning a document to a document type. Whether the algorithm is supervised or not, the Centre of Mass re-calculates the approximation of a document type and considers the newly aligned document. In order to accommodate evolving document structures, a learning window is implemented by including only the X most recent documents per document type to compose its Centre of Mass. That is, we limit the number of documents to include to the most recently aligned documents. This way the Centre of Mass will take less time to converge to the new "average" document type.

4.3 Self-adjusting Predictor Aggregation

The self-adjusting aggregation of multiple predictors consists of two phases. Through-out the process lifetime, the individual document-based and process-based document type ranks are aggregated and subsequently applied in user recommendation on which process step to execute next. Upon process termination, the second phase utilizes explicit user feedback (i.e., correction of misclassified documents) and implicit ranking analysis to improve the classification mechanism. Our mechanism dynamically adjusts the weight of process-based rankings (i.e., when messages occur according to the process model) and correspondingly the weight of the structural similarity (i.e., when the documents no longer follow the process) according to the underlying *Process Stability* ($stability = [0, 1]$).

Dynamic Aggregation The first step in the predictor aggregation algorithm (Algorithm 1) is the intersection of process-based (R_P) and document-based (R_S) prediction sets (line 2). The algorithm deals with strongly conflicting predictions in lines 3 to 10, and otherwise focuses on optimal aggregation in lines 11 to 22. Note that prior to aggregation, we remove any document type dt from R_P and R_S that is not expected, i.e. $prob(dt) = 0$.

An empty intersection (R_{aggr}) indicates a strong disagreement and hints at a deviation from the process or a very distorted document. Hence, we derive all document types that are still expected to occur — or that are likely to occur again — and intersect them with the document-based prediction result. If this yields an empty set again, we apply the stability value to decide whether classification R_P or R_S is a more viable predictor.

An intersection R_{aggr} with two or more elements needs closer analysis to decide upon the most suitable aggregation preference $pref = [0, 1]$. Here, process stability is not the sole factor for determining the significance of process-based and document-based predictors. Both predictors potentially encounter situations when they cannot clearly distinguish between multiple document types and thus have to give roughly equal probability to most of them. We apply the widely-known Shannon's entropy to determine whether a ranking result yields crisp or indecisive probability values. The normalized entropy $h(R)$ for a document type ranking set is calculated as:

$$h(R) = \frac{\sum_{i=1}^{|R|} \frac{score(type_i)}{\sum score(type)} * log(\frac{score(type_i)}{\sum score(type)})}{log(|R|)} \tag{2}$$

The normalized entropy yields values close to 1 for rankings of (almost) equal score, and values close to 0 for crisp scores regardless of ranking size. Subsequently we apply the following rules to determine the impact of process-based and document-based rankings: If both are indecisive, we completely rely on the prediction that the process stability currently tends towards. That is, for process $stability > 0.5$, we apply only process-based rankings, otherwise we apply only document-based rankings. If both are decisive, stability serves directly as the

Algorithm 1 Predictor Aggregation Algorithm $\mathcal{A}(R_P, R_S, stability)$.

```
1: function AGGREGATE(R_P, R_S, stability)
2:     R_aggr ← R_P ∩ R_S
3:     if |R_aggr| = 0 then
4:         E ← getAllExpectedMsgTypes(R_P)
5:         R_aggr ← R_C ∪ E                    ▷ retain only expected message types
6:         if |R_C| = 0 then
7:             if stability > 0.5 then
8:                 R_aggr ← R_P
9:             else
10:                R_aggr ← R_S
11:    else
12:        if |R_aggr| > 1 then
13:            R'_P ← R_P ∩ R_aggr
14:            procEntr = calcNormalizedEntropy(R'_P)
15:            R'_S ← R_S ∩ R_aggr
16:            contentEntr = calcNormalizedEntropy(R'_S)
17:            pref ← 0.5
18:            if procEntr > indecThreshold ∧ contentEntr > indecThreshold then
19:                pref ← round(stability)
20:            if procEntr < indecThreshold ∧ contentEntr < indecThreshold then
21:                pref ← stability
22:            R_aggr ← aggregate(R'_P, R'_S, pref)
23:    normalize(R_aggr) return R_aggr
```

trade-off factor. Otherwise — when only one predictor is decisive — we apply no preference at all ($pref = 0.5$).

An intersection R_{aggr} with a single element represents a good agreement of both predictors and does not need any further processing. Before returning the aggregated set of document type probabilities, the probability values are normalized such that $\sum_i prob(d_i) = 1 \forall d_i \in R$. This allows us to calculate the prediction error if the top ranked document type is incorrect.

Prediction Feedback for Stability Adjustment Upon each successful process termination, the corresponding stability measure is adjusted. The amount and direction of adjustment depends on explicit user feedback (i.e., the overall prediction was incorrect) and implicit feedback (i.e., only one predictor was wrong). Algorithm 2 retrieves the process-based and document-based document type set for each document in the given process (lines 5 and 6) and calculates prediction error rates for overall and individual predictors (lines 7 to 14). The prediction error function (lines 31 to 37) checks whether the true document type resides on the first position of the sorted document type set (i.e., no error). Otherwise, the function retrieves the position of the right type and calculates the error as the difference between probability and 1. For two or more document types that yield probabilities $prob(dt) = 1$ (i.e., the predictor was indifferent),

Algorithm 2 Stability Adjustment Algorithm $\mathcal{A}(WF, oldStability)$.

```
1: function ADJUST(WF, oldStability)
2:     stabChange ← 0
3:     for Message m ∈ WF do
4:         R_aggr ← getPrediction(WF, m)
5:         R_P ← getProcPrediction(WF, m)
6:         R_S ← getStructPrediction(WF, m)
7:         if hasUserCorrection(WF, m) then
8:             MessageType trueType ← getTrueType(WF, m)
9:             error ← calcPredictionError(R_aggr, trueType)
10:        else
11:            MessageType trueType ← getMessageType(R_aggr[0])
12:            error ← 0
13:        procError ← calcPredictionError(R_P, trueType)
14:        structError ← calcPredictionError(R_S, trueType)
15:        if error > 0 then
16:            if procError > 0 ∧ structError = 0 then
17:                stabChange ← stabChange − 1
18:            if procError = 0 ∧ structError > 0 then
19:                stabChange ← stabChange + 1
20:        else
21:            if procError > 0 ∧ structError = 0 then
22:                stabChange ← stabChange − procError
23:            if procError = 0 ∧ structError > 0 then
24:                stabChange ← stabChange + structError
25:    if stabChange < 0 then
26:        delta ← max(0, oldStability + stabChange)
27:    if stabChange > 0 then
28:        delta ← min(1, oldStability + stabChange)
29:    newStability ← γ * delta + (1 − γ) * oldStability
30:    return newStability
31: function CALCPREDICTIONERROR(R, trueType)
32:    PredictionResult cr ← R[0]
33:    if cr == trueType then return 0
34:    else
35:        for PredictionResult cr ∈ R do
36:            if getMessageType(cr) == trueType then return 1 − getProb(cr)
37:        return 1
```

the error rate hence will be zero. Note that for the normalized overall prediction result, any misclassification will result in an error rate greater than zero.

Next, we determine the process wide stability change (lines 15 to 24). We increase the change for every document-based error and process-based correct classification and vice versa — independent of overall prediction success. There is no impact when both predictors are wrong. Any prediction error, however, weighs in heavier when the user corrects the combined prediction result (i.e.,

error > 0). Thus, as long as the overall prediction is correct, an individual error's impact is proportional to the error amount. Thus the effect on stability change remains potentially small.

Once we determine the overall stability change we apply the exponentially weighted moving average (EWMA) as the ageing function to update the previous process stability factor (lines 25 to 29). When the stability change is greater than zero, the new stability value will move towards 1. For a stability change value below zero, the new stability is closer to 0.

The bootstrapping of the process stability value is straight forward. *Empty* process models that have no document information (yet) — respectively only a list of associated types but no mapping to transitions — start with $stability = 0.3$. We rely on the document-based predictor in the beginning as the process-based predictor still needs to learn the correct mappings. For new process models with detailed document information, we initiate $stability = 0.7$ as we assume that a new process is stable and not subject to immediate evolution.

5 Evaluation

We evaluate our approach based on the reference scenario in Section 2. Specifically, we are interested in (i) the prediction error rate and how well the dynamically aggregated predictor performs against the individual predictors as well as a naive fixed aggregation; (ii) how robust the dynamically aggregated predictor is in the present of increasingly noisy (varying) documents; and (iii) how accurately the process stability metric reflects the actual process evolution. Furthermore, we distinguish between two cases: the evolution of a stable process and the evolution of an empty process.

5.1 Experimental Setup

We simulate a user behaviour by playing prefabricated log sequences against the prediction system. Each log sequence represents one process execution and is denoted a *round*. Multiple, sequential rounds make up one experiment run. Within a run, any changes to the process model at the end of a round are made available to the subsequent round — thereby enabling process evolution. Process evolution itself is simulated by gradually switching from process-coherent log sequences to log sequences that follow a different process model. Note that we keep the structure of process steps stable as we focus on evaluating the document prediction aspect only. Consequently, the evolution affects only the type, location, and timing of documents within a process but not the sequence of activities.

Two log sequences ($L1, L2$) are sufficient to cover all independent paths in the scenario process model. Two additional log sequences ($L3, L4$) describe an evolved model. Here, the *Quote* (3) document is no longer used, the *Agreement* (5a) is delayed until completion of the *PrepareShipment* (F) activity which also triggers the *Authorization Of Invoice* document (6). The *Invoice* (7) is produced

when executing *Priority Dispatch* (J). The *Delivery Note* (8) only applies to *Regular Dispatch* (H). In all experiments the EWMA coefficient (for process stability ageing) α is set to 0.3 which corresponds to a trade-off between rapid uptake of evolving process patterns and robustness against one-time process deviations. The used logs contain following sequences of process steps and documents:

$L1$: 1, A, 2, B, C, 3, D, 4, E, 5a, 5b, F, 6, G, 7, J, 8

$L2$: 1, A, 2, D, 4, B, C, 3, E, 5a, 5b, F, 6, G, 7, H, 8

$L3$: 1, A, 2, B, C, D, 4, E, 5b, F, 6, 5a, G, J, 7

$L4$: 1, A, 2, D, 4, B, C, E, 5b, F, 6, 5a, G, H, 8

We use a pool of valid documents to generate documents for our experiments. Initially we gathered the documents mentioned in the scenario (Section 2) from companies in our project consortium: 6 Orders, 4 Offers, 5 Quotes, 5 Credit Notes, 5 Agreement, 7 Job Descriptions, 5 Authorisation of Invoicing, 8 Invoices, and 3 Delivery Notes. Even when they represent the same document type, they contain slightly different information from each other. Then we manually extracted the concepts paths using the Core Components standard [19] as a reference for matching concepts. We randomly selected one document from each type to be considered as the document type specification throughout the experimentation. The rest of the documents then formed the pool of valid documents to choose from. Additionally, we developed a document generator which took a document of a specified type from the pool and introduced noise to it in order to take our experiments closer to practical reality. Such noise consists of randomly introducing, deleting, or replacing concept paths.

We conduct each experiment with four different noise levels in the documents, namely 5%, 10%, 15%, and 20% noise in each generated document. In addition, every combination of experiment run and document noise is carried out 10x with differently initialised random number generators. In total every experiment run is thus executed 40x. Within these 40 instances we obtain the prediction error for individual predictors, fixed aggregation, and dynamic aggregation, as well as the process stability metric in each round.

We evaluate the responsiveness and adaptivity of our aggregation mechanism to process evolution. As success criteria, we measure the absolute prediction error of a predictor each round by simply counting how often it wrongly predicts a document. Thus an *average absolute prediction error* of 1 signifies that the predictor makes one prediction error per process instance. We compare process-based, document-based, fixed aggregation, and dynamic aggregation predictors. The fixed aggregation predictor simply merges process- and document-based ranks ($R_{fixed} \leftarrow R_P \cup R_S$).

5.2 Experiments and Results

In experiment 1, we first run 16 process instances (determined by alternating $L1$ and $L2$) which represent normal process behaviour (Phase 1). Then we evolve the process by having an interleaving of $L1$ to $L4$ for 12 rounds (Phase 2), and then continue for another 16 rounds of stable evolved behaviour — alternating $L3$ and $L4$ (Phase 3). The initial stability value is 0.7 as we assume a stable

process. Figure 5a displays the average absolute prediction error for each of the four predictors with document noise level 20%. The three phases are clearly visible — with low(er) error rates during stable periods (round 1 to 16 and 29 to 44).

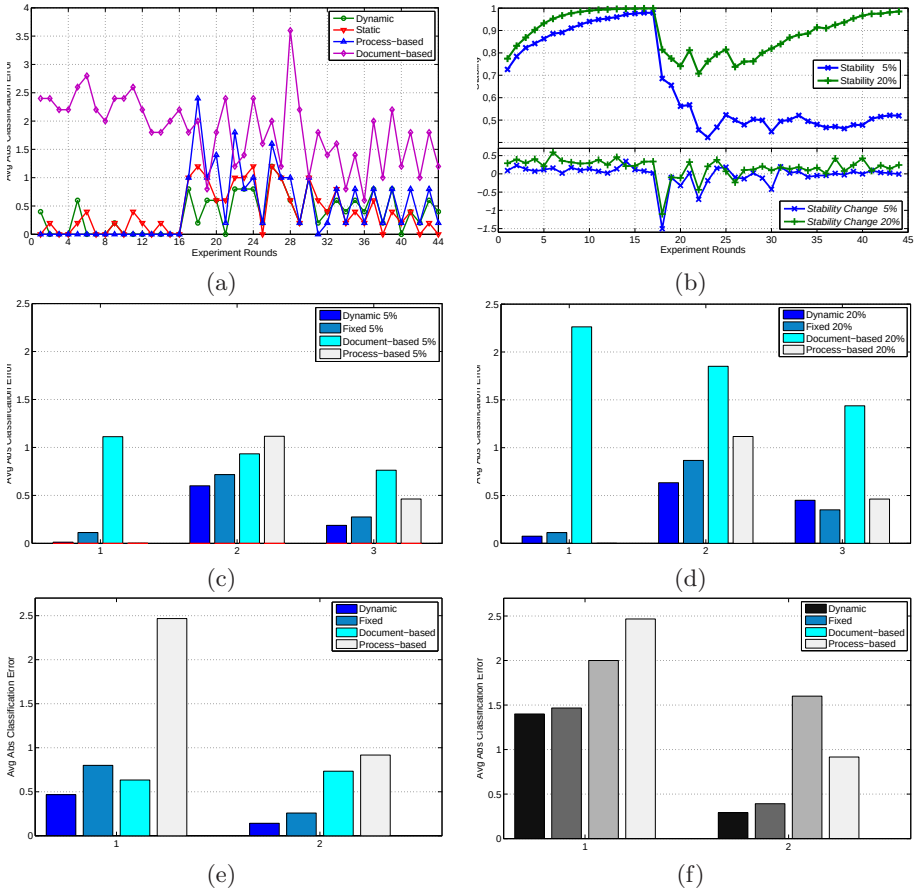

Fig. 5: Experiment 1: Comparison of document predictors (a), process stability/change (b) across 44 rounds, and error rates for 5% (c) and 20% noise (d) averaged for stable (1), evolving (2), and evolved (3) process phases. Experiment 2: Learning documents in an empty process model: comparison of predictor error rates for 5% (e) and 20% noise (f) averaged for learning (1) and stabilized (2) process phases.

In experiment 2, we evaluate the learning behaviour for an initially empty process (i.e., the process model is unaware of any document types). A run consists of 30x alternating sequences of $L3$ and $L4$. The initial stability value is set to 0.3. Here we split the 30 rounds into two phases — learning (1) during round 1 to 6 (Figure 5e) and stabilized (2) during rounds 7 to 30 (Figure 5f).

In general, we find the dynamic aggregation predictor outperforming the document-based predictor considerably in all situations (absolute error reduction between 0.57 and 2.19). In experiment 1 (Figure 5c+d), the dynamic aggregation predictors fares as good as the fixed aggregation predictor as well as against the process based predictor for stable process phases (fixed: [-0.10, 0.10] and proc: [-0.08, 0.28]). However, it outperforms both predictors during process evolution (fixed: [0.12, 0.23] and proc: [0.48, 0.52]). In experiment 2 (Figure 5e+f), the dynamic aggregation predictor always outperforms all other predictors — the absolute error difference for fixed: [0.07, 0.33], doc: [0.17, 1.31], and proc: [0.63, 2.0].

Document noise has little effect on the dynamic aggregation in experiment 1. The increase in document-based predictor error is between 0.67 and 1.15 when moving form 5% to 20%. At the same time, the increase in dynamic predictor error is only between 0.06 and 0.26. In experiment 2, the effect of noise is more visible as the dynamic predictor relies more on the document-based predictor during the initial learning phase (doc error diff: 1.37, dynamic error diff: 0.93). Once stabilized, the noise has little impact (doc error diff: 0.87, dynamic error diff: 0.15).

Finally, as shown in Figure 5b, the feedback mechanism successfully determines stability values that accurately reflect the process dynamics. Process stability drops precisely whenever evolution begins (start of phase 2 in experiment 1 and start of phase 1 in experiment 2) and stabilizes once evolution terminates. Stability values for experiment 2 and additional analysis is available as supporting online material (SOM)[4].

6 Discussion and Related Work

Multiple research efforts have focused on supporting ad-hoc workflows (e.g., [18]), semi-structured processes/case-based systems (e.g., [14, 2]), and well-structured processes (e.g., [13, 20, 1]) over the past decade. The increase in flexibility, however, comes at the cost of lacking user guidance. Recently this shortcoming has been addressed by introducing recommender systems [17, 4, 5]. All these approaches are focused on user activities and assume that the process documents are either rigorously defined (explicitly or implicitly) or that the documents have no impact on adaptation and thus recommendation.

This contrasts with existing work in the area of document-driven processes, where, for example, [8] presents an approach for dynamic workflows supported by software agents. Documents are in XML and trigger events in the system such as document arrival, document updating, or document rejection. A set of rules are also defined to represent the business logic (workflow process). Agents then exchange documents between each other according to the business logic thus managing the document life cycle. However, even when this approach considers documents within the processes, it does not semantically analyse the contents thus rendering this approach different from ours.

[4] `http://www.infosys.tuwien.ac.at/staff/dorn/bpm2011SOM.pdf`

Another approach presented by [21] shows a framework for document-driven workflows where documents are modelled with their meta-information rather than their internal semantic structure as opposing to our approach. Moreover, our document type state transition model (see Figure 4) could be seen as an alternative document meta-model, yet with our approach processes are allowed to evolve due to the probabilities the transition model manages whereas the other approach does not prove how their document meta-model is better supported.

Additionally, [10] aims at detecting which document content has an effect on the process outcome, ultimately to provide user recommendations. The process itself however remains rigid and the document structure is pre-defined.

An E-mail can be seen as a form of document being exchanged especially between small companies. [15] presents Semanta, an speech act theory approach to analyse E-mails in terms of the verbs contained, mapping between collections of verbs and activities in ad-hoc workflows. In [16] Semanta is presented as an E-mail based system for workflow support, yet the E-mail (document) analysis is exclusively based on the verbs found in contrast to our semantic analysis.

7 Conclusions

In this paper we presented a self-learning mechanism for determining document types in people-driven ad-hoc processes. The mechanism dynamically decides whether the prediction needs to rely more on process structure information or rather more on document semantics. We introduce a process stability metric which describes whether a process is currently evolving. Implicit and explicit user feedback keep the metric always up-to-date. Simulations derived from real-world documents and processes demonstrate that our mechanism yields the lowest prediction error rates during evolving and stable process phases.

In the future we plan to refine the dynamic aggregation mechanism by capturing process stability on a more fine-grained level through detection of activity hot spots. Additional analysis of the prediction success rates for individual document types and the similarity in-between document types seems promising to reduce error rates even further.

References

1. Adams, M., Edmond, D., ter Hofstede, A.H.M.: The application of activity theory to dynamic workflow adaptation issues. In: 7th Pacific Asia Conference on Information Systems. pp. 1836–1852 (2003), http://eprints.qut.edu.au/archive/00007983/
2. Adams, M., Hofstede, A., Edmond, D., van der Aalst, W.: Facilitating flexibility and dynamic exception handling in workflows through worklets. In: Proceedings of the CAiSE?05 Forum, FEUP. pp. 45–50 (2005)
3. Berry, M.W., Drmac, Z., Jessup, E.R.: Matrices, vector spaces, and information retrieval. Society for Industrial and Applied Mathematics Review 41(2) (1999)
4. Dorn, C., Burkhart, T., Werth, D., Dustdar, S.: Self-adjusting recommendations for people-driven ad-hoc processes. In: Proceedings of International Conference on Business Process Modelling. Springer (September 2010)

5. Dorn, C., Dustdar, S.: Supporting dynamic, people-driven processes through self-learning of message flows. In: Proceedings of 23rd International Conference on Advanced Information Systems Engineering (CAiSE'11). Springer (June 2011)
6. Han, E.H., Karypis, G.: Centroid-based document classification: Analysis and experimental results. In: Zighed, D., Komorowski, J., Zytkow, J. (eds.) Principles of Data Mining and Knowledge Discovery, Lecture Notes in Computer Science (LNCS), vol. 1910, pp. 116–123. Springer-Verlag, Heidelberg (2000)
7. Joseph, D., Marín, C.A.: A study on aligning documents using the circle of interest technique. In: 5th International Conference on Software and Data Technologies. pp. 374–383. SciTePress (2010)
8. Kuo, J.Y.: A document-driven agent-based approach for business processes management. Information and Software Technology 46(6), 373–382 (2004)
9. Laclavík, M., Dlugolinký, S., Šeleng, M., Kvassay, M., Hluchý, L.: Email analysis and information extraction for enterprise benefit. Computing and Informatics 30(1) (2011)
10. Lakshmanan, G.T., Duan, S., Keyser, P.T., Khalaf, R., Curbera, F.: A heuristic approach for making predictions for semi- structured case oriented business processes. In: Proceedings of First Workshop on Traceability and Compliance of Semi-structured Processes @BPM2010. Springer (September 2010)
11. Marín, C.A., Carpenter, M., Wajid, U., Mehandjiev, N.: Devolved ontology in practice for a seamless semantic alignment within dynamic collaboration networks of smes. Computing and Informatics 30(1) (2011)
12. OASIS: ebXML technical architecture specification. Tech. rep., ebXML (2001), technical report
13. Reichert, M., Rinderle, S., Dadam, P.: Adept workflow management system: flexible support for enterprise-wide business processes. In: Proceedings of the 2003 international conference on Business process management. pp. 370–379. BPM'03, Springer-Verlag, Berlin, Heidelberg (2003)
14. Reijers, H., Rigter, J., Aalst, W.V.D.: The case handling case. International Journal of Cooperative Information Systems 12, 365–391 (2003)
15. Scerri, S., Davis, B., Handschuh, S.: Improving Email Conversation Efficiency through Semantically Enhanced Email. In: Proceedings of the 18th International Conference on Database and Expert Systems Applications. pp. 490–494. IEEE Computer Society, Washington (2007)
16. Scerri, S., Davis, B., Handschuh, S.: Semanta Supporting E-mail Workflows in Business Processes. In: Proceedings of the 2009 IEEE Conference on Commerce and Enterprise Computing. pp. 483–484. IEEE Computer Society (2009)
17. Schonenberg, H., Weber, B., Dongen, B., van der Aalst, W.: Supporting flexible processes through recommendations based on history. In: BPM '08: Proceedings of the 6th International Conference on Business Process Management. pp. 51–66. Springer-Verlag, Berlin, Heidelberg (2008)
18. Stoitsev, T., Scheidl, S., Spahn, M.: A framework for light-weight composition and management of ad-hoc business processes. In: TAMODIA. pp. 213–226 (2007)
19. UN/CEFACT: Core components technical specification – part 8 of the ebXML framework. Tech. rep., UN/CEFACT (2003), technical report
20. Vanderfeesten, I.T.P., Reijers, H.A., van der Aalst, W.M.P.: Product based workflow support: Dynamic workflow execution. In: CAiSE. pp. 571–574 (2008)
21. Wang, J., Kumar, A.: A framework for document-driven workflow systems. In: Business Process Management, BPM 2005, Lecture Notes in Computer Science (LNCS), vol. 3649, pp. 285–301. Springer-Verlag, Heidelberg (2005)