Preprint of an article published in International Journal of Cooperative Information Systems (IJCIS), 24, 3, 2015, 1541003 Doi: 10.1142/S0218843015410038

http://www.worldscientific.com/ijcis

CHANGE PROPAGATION ANALYSIS AND PREDICTION IN PROCESS CHOREOGRAPHIES*

WALID FDHILA AND STEFANIE RINDERLE-MA AND CONRAD INDIONO

University of Vienna, Faculty of Computer Science
{*Walid.Fdhila, Stefanie.Rinderle-Ma, Conrad.Indiono*}*@univie.ac.at*

Business process collaborations among multiple partners require particular considerations regarding flexibility and change management. Indeed, each change or process redesign originated by a partner may cause ripple effects on other partners participating in the choreography. Consequently, a change request could spread over partners in an unexpected way with relevant costs due to its transitivity (e.g., in supply chains). In order to avoid costly negotiations or propagation failures, understanding this behavior becomes critical. This paper focuses on analyzing the behavior of change requests in process choreographies, i.e., the change propagation behavior. The input data might be available in two different formats, i.e., as change logs or change propagation logs. In order to understand the data and to explore potential analysis models and techniques, we employ exploratory data analysis as well analysis techniques from process mining and change management to simulation data. The results yield the requirements for designing a mining algorithm that derives the propagation behavior behind change logs. This algorithm is a memetic algorithm that is based on different heuristics. Its feasibility is shown based on a comparison with the other mining techniques.

Keywords: Collaborative Business Processes, Process Choreography, Change Prediction, Impact Analysis, Change Propagation, Change Minin, Memetic Mining

1. Introduction

Business process collaborations among multiple partners require particular considerations regarding flexibility and change management. According to Maier et al. [1], change propagation is the source of 50% of critical changes that endanger production in terms of cost and time, establishing the need to handle change propagation. Giffin et al. [2] define change propagation as a process where a *change to one part or element of an existing system configuration or design results in one or more additional changes to the system, when those changes would not have otherwise been required.*

*the work presented in this paper has been funded by the austrian science fund (fwf):i743.

1.1. *Problem Statement*

Most of the existing approaches have focused on flexibility in single service orchestrations so far, whereas only few tackle the process choreography case [3]. In business process choreographies, a change propagation could be defined as the changes required to other partners in order to ensure the consistency and compatibility of the collaboration after a particular partner process has changed [3]. Typical change patterns comprise, for example, adding or removing a set of activities from a process model or modifying the dependencies between the interactions of several business partners [4]. Change can also arise later in the life cycle, for instance, if process models are redesigned to reduce manufacturing costs. However, handling the effects of change propagation at the partners' sides can be neither enforced nor checked automatically as the information on the private partner processes is not available, change propagation might necessitate tenacious negotiations between partners [3]. A change propagation that even after many successful negotiations still fails in the end is costly and hampers the success of the collaboration. Such failures can become extremely costly as they are mostly accompanied by costly negotiations. Therefore, accurate assessments of change impact help avoiding such changes that do not provide any net benefit. With early consideration of derived costs and impacts, and by preventing change propagation, bears the potential to avoid and reduce both average and critical changes [5]. Hence, it is important that designers are able to accurately assess the scope and impact of each proposed change [6]. In this context, we transfer the prediction techniques used in other domains to process choreographies and distinguish between:

Priori Prediction The priori prediction assumes that no history of previous change propagations is available and hence, utilizes the choreography structure and graph theory to predict the propagation likelihood of a given change [7]. The priori prediction investigates two perspectives; i.e., static and dynamic. While the former estimates the propagation likelihood between the partners, the latter estimates the impacts of propagation on the running instances.

Posteriori Prediction The posteriori prediction assumes that a history of previous changes occurred in the choreography is available as a log file. This prediction utilizes mining techniques to build a model of the behavior of change propagation [8]. The latter can be used for further changes to estimate their impacts on the other partner processes. The analysis of the log files can also provide information about the partners and their behavior regarding changes (e.g. absorber, multiplier).

1.2. *Contribution*

This paper represents an extended and revised version of the work presented in [8]. Fdhila et al. [8] present a posteriori approach for predicting change propagation behavior in process choreographies. The posteriori prediction is based on a memetic mining algorithm coupled with the appropriate heuristics, that enables the mining

of prediction models on change event logs. The produced models help assessing the likelihood of change propagations, and the impacts on the collaborating partners.

Compared to [8], this paper provides significant extensions. They include an extensive discussion of the simulation environment and data set, and a discussion about change log aggregation approaches. Furthermore, this paper provides an analysis of simulation data using an open source tool for building and analyzing models of complex systems CAM (Cambridge Advanced Modeler) [9]. Doing so allows a classification of the partners according to their propagation behavior; e.g., multiplier, or absorber of changes. In addition, this paper features an exploratory data analysis of simulation data with respect to change propagation behavior, to analyze and understand the relationship between change request types; e.g. INSERT and DELETE. This also allows to better comprehend change transitivity and the factors behind a costly propagation. We also use the process mining tool ProM[a] on our simulation data in order to generate propagation models and give a comparison with our memetic mining algorithm. Finally, an extensive conclusion and thorough discussion about lessons learned is provided.

The remainder of the paper is organized as follows: Section 2 provides basic information and an illustrative example. Section 3 presents the simulation data and the change log formats, and discusses the log aggregation approaches. Section 4 follows up with application and analysis of simulation data with existing tools, and proposes an exploratory data analysis of simulation data. Section 5 presents the prediction models obtained through process mining tools and discusses a memetic mining algorithm for change logs. Section 6 discusses the related work and Section 7 provides a summary and an overview on lessons learned.

2. Motivating Example and Preliminaries

A process choreography is defined as a set of business partners collaborating together to achieve a common goal. Based on [10], we adopt a simplified definition of a process choreography $C := (\Pi, \mathcal{R})$ with $\Pi = \{\pi_i\}^{i \in \mathcal{P}}$ denoting the set of all processes distributed over a set of partners $\mathcal{P}$ and $\mathcal{R}$ as a binary function that returns the set of interactions between pairs of partner; e.g., in terms of message exchanges. Typical change operations comprise, for example, adding or removing a set of activities from a process model or modifying the interaction dependencies between a set of partners. A change operation is described by a tuple (δ, π) where $\delta \in \{$Insert, Delete, Replace$\}$ is the change operation to be performed on the partner process model π that transforms the original model π in a new model π' [10].

Consider the choreography process scenario as sketched in Figure 1 consisting of four partners `Acquirer`, `Airline`, `Traveler`, and `TravelAgency`. In this paper, we abstract from the notions private and public processes and assume that logs with change information on all partners exist (e.g. anonymized and collected). The

[a]http://www.promtools.org/prom6

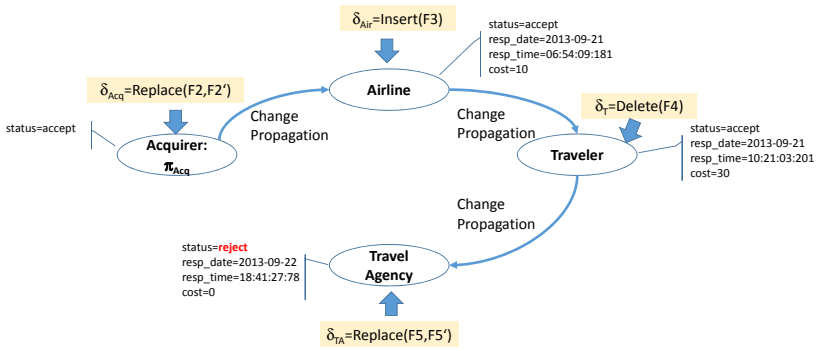

Fig. 1. Running Example: process choreography with Change Propagation

`Acquirer` initiates a change of its process (δ, Acq) that requires a propagation to the direct partner `Airline`. In order to keep the interaction between `Acquirer` and `Airline` correct and consistent, the `Airline` has to react on the change by inserting a new fragment F3 into its process. This insertion, in turn, necessitates a propagation to the `Traveler` that reacts by deleting process fragment F4. Finally, the change propagates to the `TravelAgency` that would have to replace fragment F5 by new fragment F5'. However, as the `TravelAgency` rejects the change, the entire change propagation fails. According to Fdhila et al.[10], such change propagation is defined as a function $\gamma : \{\text{Insert, Delete, Replace}\} \times \Pi \mapsto 2^{\{Insert, Delete, Replace\} \times \Pi}$ with $\gamma((\delta_i, \pi_i)) = \{(\delta_j, \pi_j)\}$. γ takes as input an initial change on a given process model and generates the ripple effects on the different partners affected by the change.

If we consider the same choreography process scenario but with additional data for predicting change behaviour then possibly the last rejection by the `TravelAgency` could have been avoided. Alas, past experience does not necessarily dictate future direction, but having a glimpse into the past might help avoiding costly actions.

3. Change Logs and Data Set

In this section, we briefly discuss various log aggregation approaches. We follow this discussion with a description of the concrete change log types we use in this work and conclude the section with the accumulation process for the data set and describing the data set itself.

For actually acting on past history by following the posteriori prediction route, recording the actual changes becomes an important issue; e.g., privacy. Several approaches have been proposed to deal with log aggregation, which range from using centralized coordinator to fully distributed aggregation. The first approach is simple to implement but raises new concerns related to availability and privacy. Privacy is lost as this coordinator has full visibility on participating partners (e.g. on the possibly confidential partnerships between partners). To overcome these lim-

itations, using a distributed aggregation seems more appropriate. Towards this end, aggregating the final historical change events should not reveal private information in any form. To fully cover the privacy aspect, the final change log needs to be anonymized[11]. Anonymization of the logs represents an important *privacy* step [11], which is a trivial operation in a non-distributed setting. In a distributed environment a consistent anonymization scheme needs to be employed, where for example *traveler* is consistently labeled as X. Anonymization is not in the focus of this paper, but techniques for collecting and anonymizing distributed data and logs can be found in [12,13,14].

3.1. *Change Log Types*

In this section, we introduce the two different change log types. *Change logs* are a common way to record information on change operations applied during process design and runtime for several reasons such as recovery and compact process instance representation [15]. For process orchestrations, change logs have been also used as basis for change mining in business processes in order to support users in defining future change operations [16].

Change logs can be also used for process choreographies. Here, every change log contains all individual change requests performed by every partner, where no propagation information are described in the log. At a certain time, all the public parts of the change logs owned by the partners participating in the collaboration are anonymized [11], normalized, collected and put in one file to be mined. In the following, we refer to this type of log as CEL (Change Event Log).

In practice, it is also possible to have a change propagation log CPL (i.e. containing the change requests, their impacts and the propagation information as well). However, since the processes are distributed, it is not always possible for a partner to track the complete propagation results of his change requests (due to transitivity and privacy). To be more generic, we adopt change logs that contain solely change events CEL (without information about propagations) to be mined. However, in order to validate our mining approach, and assess the quality of the mined model from the CEL, we also maintain a log of the actual propagations CPL. The results of the predicted model from the CEL are compared and replayed on the CPL propagation events.

Table 1 describes a sample of a change record. Each record includes information about the partner that implemented the change (anonymized), the change ID and type, the timestamps and the magnitude of the change. The latter is calculated using the number of affected nodes (in the process model), the costs (generated randomly), and the response time. Other costs can be added as needed. Table 2 describes a propagation record, with more propagation information.

Attribute	Value
Initial Change ID	15d6b27b
Request time	2014–08–03T00:41:15
Change type	Insert
Partner	TravelAgency
Magnitude	0.6
Status	completed
Response time	2014–08–05T12:32

Table 1. Change Event Record

Attribute	Value
Initial Change ID	15d6b27b
Request time	2014–08–03T00:41:15
Change type	Insert
Partner	TravelAgency
Partner target	Airline
Derived change ID	c25b8c67a
Derived change type	Insert
Magnitude	0.6
Status	completed
Response time	2014–08–05T12:32

Table 2. Change Propagation Record

3.2. *Data Set*

The data used in this paper is obtained through our C^3Pro change propagation prototype[b] [3]. The process of accumulating the data set is visualized in Figure 2. Based on [10], the simulation follows the change propagation flow for process choreographies and employs the respective change propagation algorithms in the inner loop. For this purpose, a collaboration scenario has been implemented. The choreography model as well as the process models of the different partners have been created through the Signavio Process Editor and imported to our C^3Pro prototype. Note that it was ensured that all models are structurally and behaviorally sound. The C^3Pro prototype is a framework for propagating changes between process partners. The framework takes as input a process partner change (δ, π) and calculates the propagation effects on the public processes of the different partners $\gamma(\delta, \pi)$. For more information about the C^3Pro prototype, the reader can refer to [3]. Even though the C^3Pro Editor is satisfactory for performing change scenarios on the graph manually, we applied the following methodology to generate random change scenarios automatically and extract the most important metrics to gather statistics: for a given choreography model as input we iterate over all associated public models and generate a set of random structural change operations on those public models: DELETE, INSERT, and REPLACE. The propagation effects of each of these change operations is then calculated using the actual propagation algorithms provided by the C^3Pro prototype. The derived changes concern their public models and are calculated automatically. The impacts of the public changes on the private processes are supposed to be calculated by the corresponding partners (and are not in the focus of this paper). The randomly generated change request as well as their actual derived changes (not random) are used to build the two previously classified log types; i.e. CEL and CPL. In this work, the CPL is used for performing various change propagation analysis tasks to (a) observe the behavior of the implemented

[b]http://www.wst.univie.ac.at/communities/c3pro/index.php?t=downloads

change propagation algorithms and (b) to classify the participating partners in terms of how they respond to incoming change requests. Finally, the CEL is used as input for the memetic change mining algorithm to non-deterministically derive a prediction model, which we elaborate on starting from section 5.2.

With respect to data quality, although the initial change requests are synthetic and generated randomly (random structural modifications of the process models), the subsequent propagation impacts on the other partners, of each initial change request, are calculated according to the change propagation algorithms defined in a previous work [3]. These algorithms take into consideration the model dependencies, the propagation soundness, and model correctness after changes. Therefore the dependencies between initial change requests and their propagations are not random and follow the aforementioned model restrictions. This ensures the quality of the simulation data and the quality of the generated analysis and mining results.

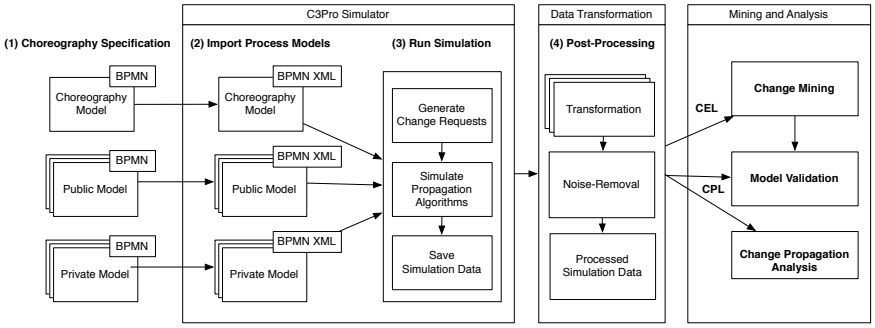

Fig. 2. Overall Simulation and Data Set Gathering Process (for CEL and CPL)

For the simulation, we designed a collaboration consisting of 25 partners, with a total number of 422 nodes and an average of 16.8 nodes per partner. For each process model, a set of change requests of type REPLACE, INSERT and DELETE were generated randomly. 17209 change requests were created with an average of 688.3 requests per partner; i.e., 236.2 of type Replace, 435.3 of type Insert, and 16.8 of type Delete, resulting in 50953 change propagation records in the CPL with an average of 2.9 derived propagations per initiated change request. In total 66822 change event records were generated and logged in the CEL. The logged data are in CSV format.

4. Change Propagation Analysis

We start this section by analyzing the CPL data set in terms of the change propagation behavior of the participating partners within the process choreography. We conduct this analysis by (1) employing an existing change analysis toolset: CAM and

(2) by deriving Partner-CPI. We conclude the section by exploratively analyzing the data set in terms of the behavior of the change propagation algorithms.

4.1. *Change Analysis with CAM*

We used the CPM[c] plugin of the CAM[9] tool to analyze and visualise the change propagation behavior in complex product structures. The main idea is to identify significant statistical observations about the likelihood and impacts of propagation. For this purpose, we first mined the initial change log containing all the change requests mentioned in section 3.2, and for each couple of partners (i, j) we calculated the occurrence frequency as well as the average impact of propagation from i to j. Then, we transformed the results into a Data Structure Matrix DSM[17]. A DSM is a matrix indicating the relationships between the partners in terms of propagation and impacts. The DSM is generated automatically from the CSV file through the CAM tool.

The results of this analysis help in classifying the business partners according to their change propagation behavior, and assessing the magnitude of a change request based on several criteria. We employ the classification picked up in [7]:

- *Constants* unaffected by changes, hence not included in the propagation model: $\sum_{ij} P(\pi_i/\pi_j) + \sum_{ij} P(\pi_j/\pi_i) = 0$.
- *Absorbers* propagate less changes than they are affected by. This is reflected in the propagation model as follows: $\sum_{ij} P(\pi_i/\pi_j) - \sum_{ij} P(\pi_j/\pi_i) < 0$.
- *Carriers* carry on the amount of changes they receive resulting in $\sum_{ij} P(\pi_i/\pi_j) - \sum_{ij} P(\pi_j/\pi_i) = 0$.
- *Multipliers* extend the number of changes they are impacted by, resulting in $\sum_{ij} P(\pi_i/\pi_j) - \sum_{ij} P(\pi_j/\pi_i) > 0$.

Figures 3, 4 and 5 present the analysis results generated by the CAM tool. The Figures 3 and 4 give the likelihood of propagation as well as the impacts on the other partners from the perspective of one partner; e.g., `TravelAgency`. In particular, Figure 3 considers a change on the `TravelAgency` and analyzes the outgoing effects on the other partners. For instance, the probability of propagating the change from the `TravelAgency` to `partner14` is very low ($\simeq 0.05$) , but the impact of a propagation is very high ($\simeq 0.8$). On the other hand, the propagation probability to `Traveler` is very high ($\simeq 0.95$) while the impact on it is very low ($\simeq 0$).

Figure 4 describes the ingoing likelihood and impacts of change propagation from all the other partners to the `TravelAgency`. Particularly, `partner3`, `partner7` and `partner9` have the highest probability of propagation to the `TravelAgency` with very high impacts ($\simeq 1$). Figure 5 give a global overview on all partners behaviors regarding change propagation and impacts. For example the `Traveler`

[c]http://www-edc.eng.cam.ac.uk/projects/cpmincam/

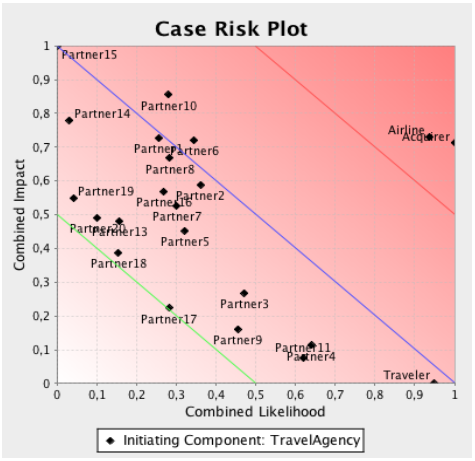

Fig. 3. Outgoing Impact

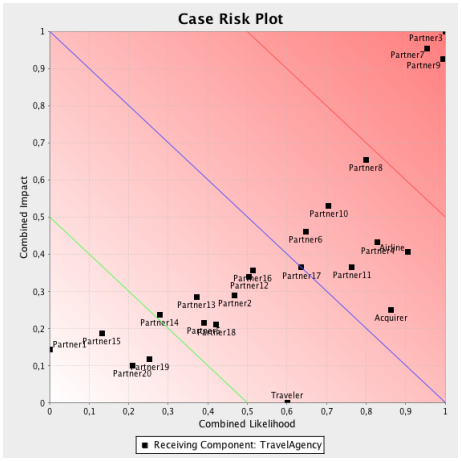

Fig. 4. Ingoing Impact

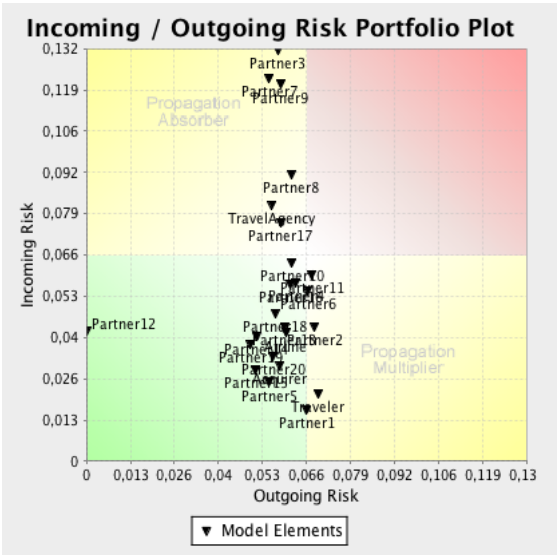

Fig. 5. Classification

is considered as a multiplier with a relatively high outgoing risk and low incoming risk, while `partner3` is seen as a propagation absorber with a high incoming risk and relatively low outgoing risk.

This classification helps to avoid propagating changes to *multipliers* by restructuring the initial change request and prioritize propagation to carriers or absorbers. Additionally, the prediction models allow measuring the magnitude of a change request by following the possible propagation paths scenarios and considering the predicted impacts on the different partners. Having the classification of partners helps with future decisions in that sense. In the following sections we will classify partners with an alternative approach and finally look deeper into the data set by

considering metrics specific to change propagation in process choreographies.

4.2. *Change Propagation Analysis with Partner-CPI*

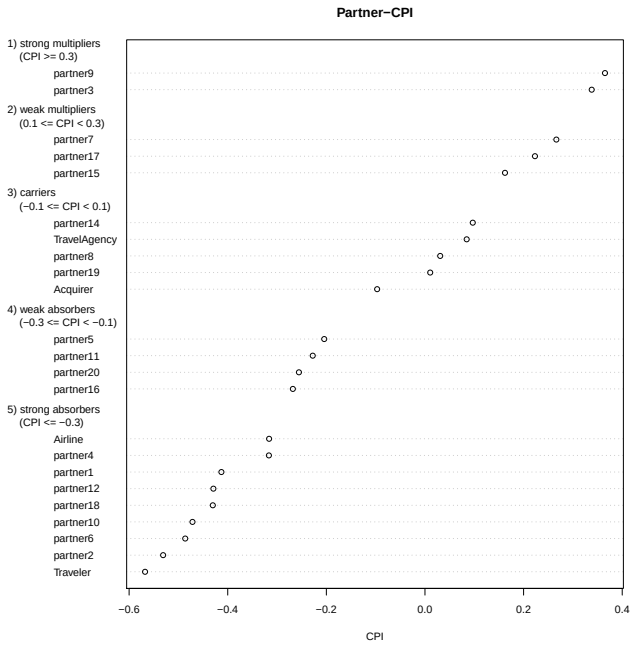

Fig. 6. Partner-CPI plot

First defined by Eckert et al.[18] and subsequently refined by Giffin et al. [2], the *change propagation index* (*CPI*) is a normalized value that classifies the change propagation behavior of a component, meaning any subpart of a product that can receive and itself emit changes to other parts. In [19], the *CPI* has been adapted to the perspective of an engineer -- called *Engineer-CPI* -- in order to study the change propagation behavior of each engineer. In this work we want to observe the change propagation behavior of each partner in the collaboration. Thus we define *Partner-CPI* as follows:

$$\textit{Partner-CPI}(i) = \frac{P_{\text{out}}(i) - P_{\text{in}}(i)}{P_{\text{out}}(i) + P_{\text{in}}(i)} \tag{1}$$

Equation 1 defines the *CPI* from the perspective of the partners participating in the collaboration. $P_{\text{in}}(i)$ specifies the sum of changes that target the given partner i (the *in-degree*), and $P_{\text{out}}(i)$ represents the *out-degree* or the sum of all changes that are propagated from partner i to other relevant partners throughout

the simulation. Using *Partner-CPI*'s value range of $[-1.0, 1.0]$ allows us to determine the propagation behavior of each individual partner. Eckert et al. [18] define the different change propagation behavior a component can take. *Multipliers* are those that emit more changes than they receive. Specifically, a *Partner-CPI* value above 0.1 are considered *weak multipliers*; those above 0.3 are considered *strong multipliers*. *Carriers* are those that emit a similar amount of changes they receive ($-0.1 >= Partner\text{-}CPI <= 0.1$). *Absorbers* propagate changes less than they receive. We distinguish between *weak absorbers* ($Partner\text{-}CPI < -0.1$) and *strong absorbers* ($Partner\text{-}CPI < -0.3$).

As shown in Figure 6 we have found two partners classified as *strong multipliers*. These are partners 3 and 9 and unsurprisingly show up prominently in the top 10 most frequent change request in Table 3. Interesting to note is despite partner 7's prominent position in Table 3, partner 3 is considered a stronger *multiplier* than partner 7. The *weak multipliers* are filled with three members. *Strong absorbers* are considered the largest group with nine members, followed by *carriers* with five and finally *weak absorbers* with four partners. In this particular collaboration the *multipliers* are outnumbered by *carriers* and *absorbers* by nearly $1:5$, making it easy to identify potential large change propagations. The partner with the highest *Partner-CPI* is partner 9 with 0.365 and is considered as a *strong multiplier*.

4.3. *Explorative Data Analysis of the Data Set*

In this section, we analyze and observe the behavior of the employed change propagation algorithms in an explorative way[d]. In the following, we define an *initial change request* to be a change that is applied to any partner π_1. Since changes can affect other partners that are associated with π_1, the change propagation algorithms transform the *initial change request* into one or several *derived change requests*. Change requests affect the fragments of a partner. We call *source nodes* those of the fragment that were touched by the *initial change request* and *target nodes* to be those fragment nodes that were affected by the *derived change requests*.

Table 3. Top 10 Change Requests sorted by frequency (N=50953)

	From	To	Frequency	%
1	partner7	partner10	2428	4.77%
2	partner9	partner7	2343	4.60%
3	partner7	partner9	1936	3.80%
4	partner9	partner10	1726	3.39%
5	partner9	partner8	1550	3.04%
6	partner9	partner2	1311	2.57%
7	partner7	partner4	1303	2.56%
8	partner7	partner8	1215	2.38%
9	partner3	partner4	1033	2.03%
10	partner9	TravelAgency	859	1.69%

[d]Explorative data analysis was firstly advocated by Tukey et al. [20].

4.3.1. *Descriptive Statistics of the Simulation Result*

Table 4. Descriptive Statistics of Simulation Result Data

	INSERT			DELETE			REPLACE		
	min	avg	max	min	avg	max	min	avg	max
Source									
All nodes	3.00	6.75	38.00	1.00	5.22	38.00	6.00	15.47	76.00
Activity nodes	1.00	2.75	22.00	1.00	2.56	22.00	2.00	6.68	44.00
AND Gateways	0.00	0.38	6.00	0.00	0.41	6.00	0.00	1.32	12.00
XOR Gateways	0.00	1.63	14.00	0.00	2.13	14.00	0.00	5.32	28.00
Target									
All nodes	3.00	18.50	208.00	1.00	3.40	30.00	0.00	22.10	198.00
Activity nodes	1.00	6.87	90.00	1.00	2.60	2.60	0.00	10.90	80.00
AND Gateways	0.00	0.62	12.00	0.00	0.30	4.00	0.00	1.18	16.00
XOR Gateways	0.00	6.80	100.00	0.00	0.49	6.00	0.00	8.44	96.00
	INSERT			DELETE			REPLACE		
	min	avg	max	min	avg	max	min	avg	max
Generated Ops.									
INSERT	1.00	2.10	10.00	0.00	0.00	0.00	0.00	0.00	0.00
DELETE	0.00	0.00	0.00	1.00	1.98	10.00	0.00	0.00	0.00
REPLACE	0.00	0.78	9.00	0.00	1.82	9.00	0.00	1.14	10.00

We start by illustrating some basic descriptive statistics on the resulting CPL. The number of *initial change* requests that were generated during the lifetime of the simulation is 17209. The total number of changes being propagated is 50953. This means the change propagation algorithms have derived 33744 changes in total that were suggested to corresponding partners. Table 3 shows the top 10 change requests sorted by frequency. In that table we see three dominant change initiators: partners 9,7 and 3.

Table 4 summarizes the change propagation simulation in terms of (a) *source nodes*, (b) *target nodes* and (c) the generated operations types in the *derived change requests*. It shows that in average the *initial change* requests of the operation type REPLACE contain higher amount of *source nodes* than the other operation types combined. For the *target nodes* REPLACE operations affect the most nodes in average, closely followed by INSERT operations. The maximum amount of *target nodes* registered during this simulation was with a change request type of INSERT (208 nodes). The generated operations help us come to obvious conclusions: INSERT operations solely cause *derived changes* of the same type. For each INSERT operation, 2.1 INSERT operations are derived and propagated in average. Similar observations can be made for the DELETE operation. Interestingly, a REPLACE change request -- as a composite change operation -- generates 0.78 INSERT operations, 1.82 DELETE operations, and 1.14 REPLACE operations in average.

4.3.2. *Observing the effect Source Nodes size has on Target Nodes*

In this subsection we observe the behavior of the change propagation algorithms in regards to the *target node* count when increasing the *source node* counts in each

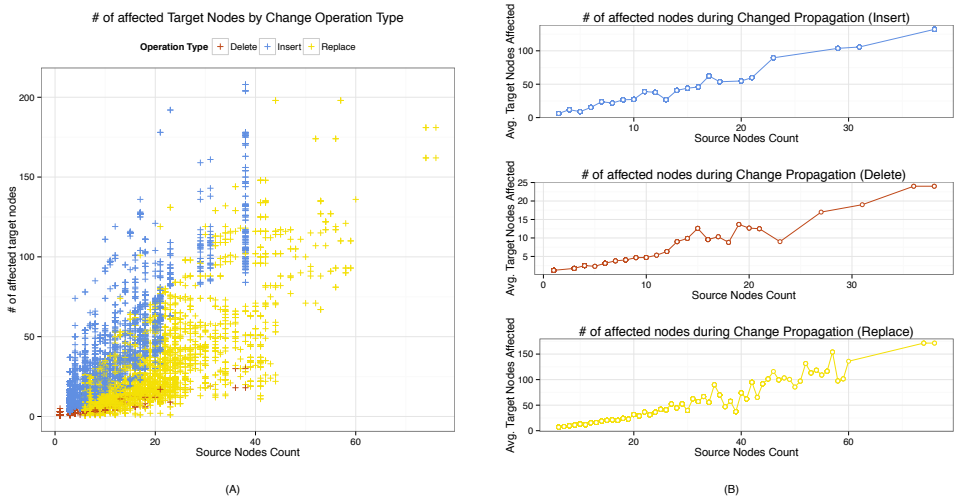

Fig. 7. Change Impact Plot: Source vs Target Nodes

change request. The first chart (Fig. 7A) shows a scatter plot of all change requests grouped by the change operation type. It shows in three distinct colors the relevant change operations being performed. In this case the *source nodes* are measured against the affected *target nodes*. Noticeable is the relative infrequent occurrence of the DELETE operation compared to the other two operations. The flat progression of the DELETE line might point out the near-linear mapping between the *source nodes* and the affected *target nodes*. In contrast, the INSERT and REPLACE operations seem to show an exponential increase. The INSERT change operation seems to be the one that has the most effects on partner nodes. For all values of *source nodes*, the INSERT operation shows the highest amount of affected *target nodes*. The REPLACE being in the middle seems to be obvious, as embedded into the REPLACE algorithm is the INSERT algorithm for specific cases.

In Figure 7(B) we split the different operation types and view them individually to identify the correct *target node* value ranges. The plots confirm what we observed in the previous paragraph: The DELETE operation behaves linearly. The number of affected *target nodes* increases with the *source node* count equally for each operation time. The most expensive operation in terms of affected *target nodes* seem to be the REPLACE operation followed by the INSERT operation.

4.3.3. *Observing the effect Source Nodes size has on Execution Time*

One important aspect for the change propagation algorithms is how much time it takes to calculate the *derived changes*: execution time. In this simulation, we measured a total of 33744 *derived changes*. These took exactly 2.039787 days to calculate, of which 78% is spent handling change requests of the type INSERT

(aprox. 1.59 days). 21.8% was spent handling REPLACE operations (aprox. 10.7 hours). Handling DELETE operations is the fastest with 15.26 seconds spent in total (aprox. 0.008%). The average execution time for INSERT takes 12.5 seconds. In average the REPLACE operation takes half of that time: 6.6 seconds and DELETE taking nearly no time at all: 35.5ms.

Further, we take the execution time and compare the numbers against increasing number of *source nodes* in the *initial change requests*. Figure 8(A) shows the raw as well as the average execution time values of such a comparison. Each point in the plot represents a single change propagation, grouped by the operation types represented by their distinct colors. We can generally observe that the INSERT change operations show the sharpest increase with increasing number of *source nodes*, followed by the REPLACE operation. The DELETE operation seems to run in constant time, regardless of the *source node* amount in the change request.

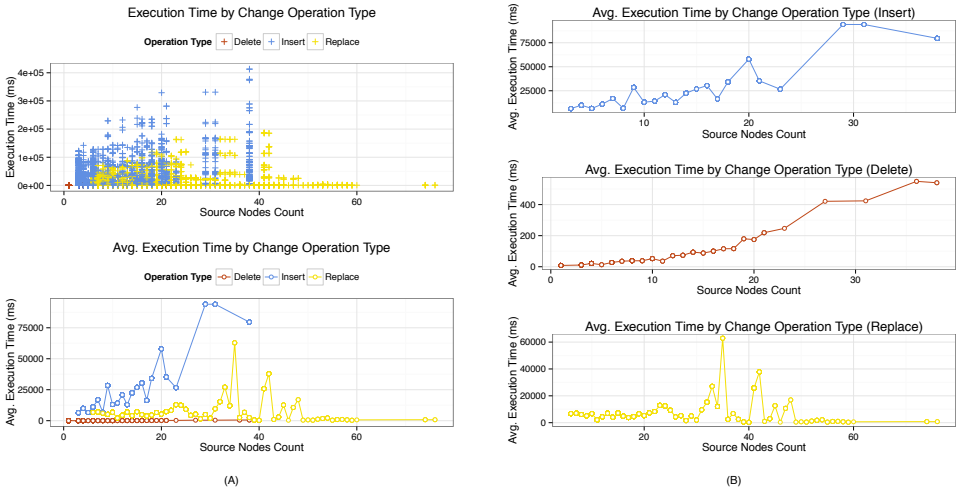

Fig. 8. Source vs Execution Time

Figure 8(B) takes the averaged execution time values and further displays each operation type in a separate graph. This allows us to observe the linearity of the DELETE operation that we could not detect in Figure 8(A). The REPLACE operation behaves differently than the other two operations. From the raw data we know that only one-third of the time is spent executing the pure REPLACE logic (i.e. replacing a fragment with another). The rest of the time is spent either executing the INSERT or DELETE logic, which explains the non-linear, near-erratic behavior.

5. Change Mining in Process Choreographies

In the previous sections we have classified the participating partners by their change propagation behavior, as well as exploratively analyzing the CPL. In the next section we will proceed with deriving a prediction model from the CPL using an existing mining tool: ProM. In the subsequent sections we argue that existing tools lack some capabilities specifically for dealing with change propagation in the choreography context and thus identify several heuristics to cover those shortcomings. Finally, we introduce a memetic change mining algorithm that integrates these heuristics and evaluate its results.

5.1. *Change Mining with ProM*

In this section we simulate several tests on the process mining tool ProM[e] using a set of different available plug ins. As input we use the CPL. In order to use ProM, it is necessary to format and transform the data initially logged as CSV format into either MXML or XES format. In general, tools such as ProMimport[f], Disco[g] and Nitro[h] can be used. ProM provides several mining techniques to analyze and monitor business processes from different perspectives; e.g., control-flow mining or organizational perspective (social network). It also supports analysis of performance and conformance checking of the mined models. Change mining is also available as a plugin in ProM, but from the perspective of one process and produces as a result the relationships between the change operations. This is different from our work, since it concerns collaborative processes and mines the propagation patterns between the different partners and using different metrics.

Process Mining Techniques: We opted for using Genetic Mining (GM) [21] and Heuristic Mining (HM) [22]. HM is based on frequencies whereas GM applies evolutionary optimization. In order to validate and evaluate the quality of the propagation model issued from either HM or GM, we make use of the log file. In particular, we utilize the cross validation technique where the log file is split into two parts; a training log and test log. The training log is used to learn the propagation model whereas the test log is used to evaluate this model based on unseen cases (cf. [22], p. 151). A prediction model is considered good if it fits with most of the change requests in the test log.

Figure 9 shows the prediction model issued from the GM plug in of ProM. The mining was applied to a slice of 1000 change events of type `Replace` taken from the initial log; i.e. training log. The generated model is a directed graph with partners as nodes, and edges as the propagation dependencies between the partners. The edges are weighted with frequency of propagation.

[e] http://www.promtools.org/prom6
[f] http://www.promtools.org/promimport/
[g] http://www.fluxicon.com/disco/
[h] http://www.fluxicon.com/nitro/

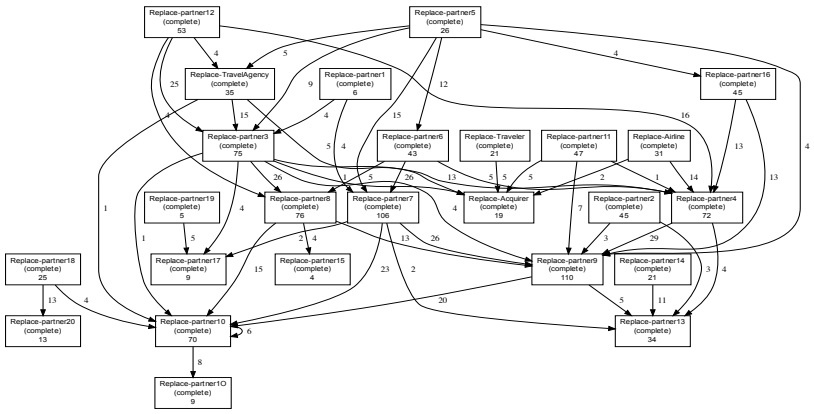

Fig. 9. Replace-Change Propagation Model Resulting from GM in ProM

5.2. *Heuristics*

Feeding the ProM toolkit with a CEL results to inadequate results. The reasons are mainly incomplete change propagation information due to the lack of it in the CEL. Having the ability to derive a prediction model from a CEL is a worthy goal as the privacy aspect is thusly maintained. Towards that goal, in this section we infer four groups of heuristics that can be exploited for mining change events in process choreographies.

5.2.1. *Time Related Heuristic (TRH):*

In connection with process mining [22,23,24], if two activities a and b whose most occurrences in the log are such as the completion time of a always precedes the start time for the execution of b, then we conclude that a precedes b in the process model. If there exist cases where the execution start time of b occurs before the completion of a, then we can say that a and b could be in parallel. In change mining, a partner π_2 that always performs changes directly after a partner π_1 has changed its process, may lead to the conclusion that the changes on π_2 are the consequences of the changes of π_1. However, this does not always hold true. Indeed, the change events are collected and merged from different sources, and several independent change requests can be implemented by different partners at the same time.

In addition, in process execution logs, each trace represents a sequence of events that refer to an execution case (i.e. an instance of the process execution). The precedence relationships between events of a same trace are explicit. In change logs CEL, each trace represents solely one change event. Even if the timestamps give an explicit precedence relationship between different traces, it is not possible to directly conclude that they are correlated. For example, we assume that the actual

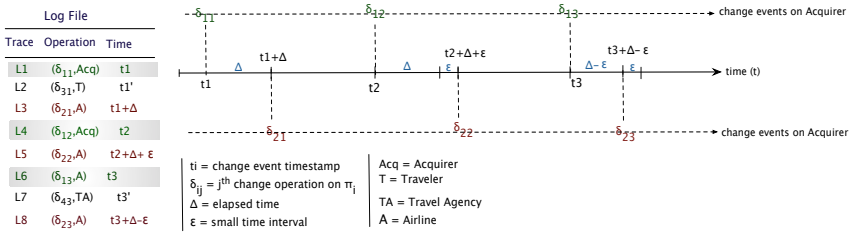

Fig. 10. Change Event Log (CEL): Representation over Time

propagation of two initial change requests $(REPLACE, Acq)$ and $(DELETE, A)$ on two different partners `Acquirer` (Acq) and `Airline` (A) are such that:

- the actual propagation of $(Replace, Acq)$ results in $(Insert, T)$.
- the actual propagation of $(Delete, A)$ results in $(Delete, TA)$.

Where T refers to `Traveler` and TA to `TravelAgency`. Since, in this paper, we consider that we do not have the information about the propagations, and that each of these change events is logged separately by the partner which implemented it, then, according to the timestamps, the merging and ordering of these change events in the CEL may lead to the following sequence: $[(Replace, Acq), (Delete, A), (Insert, T), (Delete, TA)]$. According to this ordering, $(Delete, A)$ may be considered as a consequence of $(Replace, Acq)$ and $(Delete, TA)$ as a consequence of $(Replace, Acq)$. In order to avoid such erroneous interpretations of the log, we need to enhance the heuristic with new elements.

Figure 10 illustrates an example of a sample CEL and its representation over time. The Figure shows a log file containing traces of 8 change events occurred on process partners `Acquirer` (Acq), `Traveler` (T), `Airline` (A) and `TravelAgency` (TA) at different times. We assume that the log is chronologically ordered. According to the timestamps, there is a strict precedence relationship between the change occurrences on Acq and the change occurrences on T. However, we can not directly deduce that the latter are the effects of the changes on Acq. Therefore, it is necessary to find another correlation between the timestamps that is more relevant regarding the identification of the propagation patterns. In this sense, we can remark that each time a change operation δ_i occurs on Acq at time t, there is a change operation δ_j that occurs on T at $t + \Delta$ with a variance of $\pm\epsilon$. This deduction holds true when the number of change occurrences on T in the interval $[t + \Delta - \epsilon, t + \Delta + \epsilon]$ becomes high, and when Δ corresponds to the average latency between partners Acq and T. The identification of Δ and ϵ are calculated empirically and should consider the noise in the event logs (e.g. rare and infrequent behavior).

5.2.2. *Window Related Heuristic (WRH):*

Figure 11 presents another example of change events extracted from a CEL. For instance, we consider a Replace on partner `Acquirer` (A) followed by an Insert on `Airline` (B), which in turn, followed by a Delete on the `TravelAgency` (C). We also consider Γ_i as the response time of partner i (the **average** time required by partner

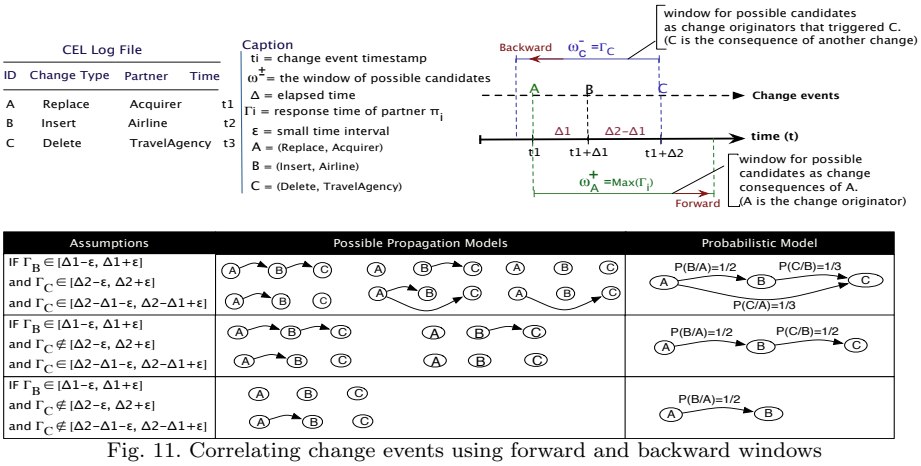

Fig. 11. Correlating change events using forward and backward windows

i to implement a change). When mining this log, the first challenge is to know if a change event is the originator or a consequence of another change. For this purpose, we define two types of windows; i.e., backward and forward. Given a change event, the forward window represents all the following change events that can be considered as effects of its propagation. In contrast, a backward window includes all previous change events that can be considered as the originators of the change event in question. For instance, in Figure 11, the forward window of A (i.e., ω_A^+), is defined by the maximum of the response times of all change events $(i.e., Max_{i\in\mathcal{P}}(\Gamma_i))$. Indeed, Γ_i is the time required by a partner to react and implement a change. So, with respect to A, if a following change event B was implemented at a time t_B such as $t_B - t_A > \Gamma_B$, then B cannot be considered as consequence of A. In turn, if $t_B - t_A < \Gamma_B$, then B might be a consequence of A (but not necessarily).

For a given change A, since we know that only change events that respect this constraint can be considered, then the possible candidate events as consequences of A should be within this window $\omega_A^+ = Max_{i\in\mathcal{P}}(\Gamma_i)$. According to this approach, in Figure 11, the possible change events that can be considered as effects of A are B and C.

This forward window allows to avoid parsing all change events that come after A to the end of the log CEL. However, a change event within this window does not necessarily imply that it is a consequence of A. for example, C is within the ω_A^+ window, but Γ_C **can be** such as $\Gamma_C < t_C - t_A < \omega_A^+$ or $t_C - t_A < \Gamma_C < \omega_A^+$. For instance, if we assume that in time scale, $t_A = 3$, $\omega_A^+ = 10$, $t_C = 9$ and $\Gamma_c = 4$, then $\Gamma_C = 4 < t_C - t_A = 6 < \omega_A^+ = 10$. In this case, C is within the window of A, but did not occur within its response time $t_A + \Gamma_C \pm \epsilon$ (ϵ is a variance value).

On the other hand, for a given change event C, the backward window ω_C^- includes all possible change events that can be considered as the originators that triggered C. In this sense, if C is a consequence of another change A, then $t_C - t_A$ should be approximately equal to Γ_C, and therefore $\omega_C^- = \Gamma_C$. However, a change event A that occurred at $t_C - \Gamma_C \pm \epsilon$ does not necessarily mean that C is consequence of A.

Indeed, both events can be independent.

Back to Figure 11, the table shows the possible propagation models that can be generated according to the assumptions based on backward and forward windows. In the first assumption, we assume that the response time of B matches the time of its occurrence after A, and the time of its occurrence with respect to B. Therefore, C can be seen as a possible consequence of either A or B. In the same time the occurrence of B after A falls within its response time Γ_B, and therefore B can be possibly a consequence of A. As aforementioned, matching the timestamps of the change events do not necessarily mean they are correlated, and then we have to consider the possibility that the events might be independent. The possible propagation models are then depicted in the second column, which, merged together, give the probabilistic model in column 3. In the second assumption, we assume that C can not be candidate for A (according to its response time), and therefore the number of possible propagation models is reduced to only 4. In the last assumption, C can not be considered as consequence of both A and B, and then the number of models is reduced to 2.

To conclude, the forward and backward windows can be very useful in reducing the search space and highlighting the more probable propagation paths between the change events. In addition, the example of Figure 11 considers only a small window of events, where each event type occurred only once. In a bigger log, a same change event (e.g. a Replace on a partner) can occur several times, which may improve the precision of the probabilistic models.

5.2.3. *Change Related Heuristics (CRH):*

The calculation of the prediction model could benefit from the relationships between change operation types. Indeed, from our experience [10], and considering solely the structural and public impacts, an initial change request of type Insert always generates insertions on the affected partners, and the same holds for Delete which generates only deletions. However, the Replace could generate all (three) types of changes. From this we deduce, that we can not have propagation of the type $(Insert, \pi_i) \rightarrow (Delete \vee Replace, \pi_j)$ or $(Delete, \pi_i) \rightarrow (Insert \vee Replace, \pi_j)$. Using these as punishments, the mining techniques could reduce the search space and therefore avoid incorrect behavior.

5.2.4. *Choreography Model Heuristics (CMH):*

Another improvement consists in using the choreography model. The latter sketches all the interactions between the partners and gives a global overview on the collaboration structure. In this sense, we can use the dependencies between the partner interactions as a heuristic to identify transitive propagations (e.g. centrality). More details about heuristics that stem from the choreography structure can be found in [7]. These heuristics can be used to improve the mining results. For example, an identified **direct** propagation link between two partners through mining could be invalidated if the partners have no direct interactions together in the choreography

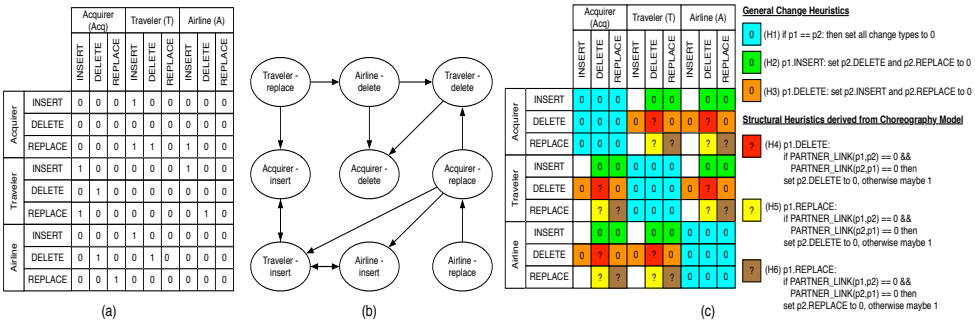

Fig. 12. (a) Genetic Encoding of a candidate solution (b) Candidate solution in graph form (c) Visualization of heuristics affecting a candidate solution

model, and the change type is Delete. Because, unlike the Replace and Insert, the Delete does not result in new dependencies between the partners.

The implementation and the evaluation of the proposed heuristics within the memetic mining is described in the following sections.

5.3. *Memetic Change Mining*

In this section we outline the memetic algorithm for mining change event logs used to build change propagation models. This core algorithm is enriched with an implementation of the heuristics sketched out in the previous Section 5.2. Employing a change propagation model, predicting the behaviour of change requests in the choreography becomes possible. Memetic algorithms follow the basic flow of genetic algorithms (GAs) [25], which are stochastic optimization methods based on the principle of evolution via natural selection. They employ a population of individuals that undergo selection in the presence of variation-inducing operators, such as mutation and crossover. For evaluating individuals, a fitness function is used, which affects reproductive success. The procedure starts with an initial population and iteratively generates new offspring via selection, mutation and crossover operators. Memetic Algorithms are in their core GAs adding an inner local optimization loop with the goal of maintaining a pool of locally optimized candidate solutions in each generation [26].

5.3.1. *Genetic Encoding*

The genetic encoding is the most critical decision about how best to represent a candidate solution for change propagation, as it affects other parts of the genetic algorithm design. In this paper, we represent an individual as a matrix $\mathcal{D}$ that states the change relationships between the partners (the genotype). Each cell d_{ij} in the matrix has a boolean value equal to 1 only if a change on π_i is propagated to π_j, and zero otherwise. The matrix is non symmetric and the corresponding graph for change propagation is directed. This means that the probabilities of propagating changes from π_i to π_j and from π_j to π_i are not equal. This is due to the fact that

the log file may contain more change propagations from π_i to π_j than from π_j to π_i. Figure 12(a) shows the representation of a candidate solution. Internally, the table rows are collapsed resulting in a *bitstring* of length $(m \times n)^2$ where n is the number of partners and m is the number of change operation types (e.g., $(3 \times 3)^2$ in the Figure 12(a)). Figure 12(b) represents the corresponding propagation model graph. Figure 12(c) shows the importance of the heuristics in reducing the search space and their effects on candidate solutions.

5.3.2. *Initial Population Generation*

Two approaches are applicable for generating an initial population: (i) starting with random change propagation models by defining random paths for propagating change requests in each of the partners. This generated model may not respect the dependencies defined by the choreography model. Also considering the complexity of the problem caused by several constraints, the obtained results prove to be not sufficient. (ii) starting with an initial good solution using a part of the propagation dependencies existing in the log file. This solution could represent an incomplete behavior since some propagation paths of the log are not incorporated into the model. For the implementation we have chosen approach (ii) as it allows us to start with an approximate solution using that as the basis for search space exploration.

5.3.3. *Heuristics*

Here we briefly outline the implemented heuristics extracted from Section 5.2.

- **H1 (CMH)** - A change to a partner's process (π_i) never results in a propagation to itself (e.g. $\pi_i = \pi_j$). We can avoid solutions with this property in the search space by applying this heuristic.
- **H2 (CMH)** - Through our extensive simulations we can rule out the case where the originating change is of the type Insert and where the propagated change type is anything other than Insert (e.g. Delete and Replace).
- **H3 (CMH)** - Similarly to H2 we can rule out the cases where the originating change type is Delete, and the propagated change type is anything other than Delete (e.g. Insert and Replace).
- **H4 (CRH)** - Rule out propagated changes of type Delete $\iff$ the originating change is of type Delete, and there is no interaction between the two partners (i.e. $\mathcal{R}(\pi_i, \pi_j) \cap \mathcal{R}(\pi_j, \pi_i) = \{\emptyset\}$, $\mathcal{R}$ is defined in Section 2).
- **H5 (CRH)** - Rule out propagated changes of type Delete $\iff$ the originating change is of type Replace, and there is no interaction between the two partners (i.e. $\mathcal{R}(\pi_i, \pi_j) \cap \mathcal{R}(\pi_j, \pi_i) = \{\emptyset\}$).
- **H6 (CRH)** - Rule out propagated changes of type Replace $\iff$ the originating change is of type Replace, and there is no interaction between the two partners (i.e. $\mathcal{R}(\pi_i, \pi_j) \cap \mathcal{R}(\pi_j, \pi_i) = \{\emptyset\}$).
- **H7** - The mutation as well as the crossover operation change a solution candidate in random ways. We can limit candidates of lower quality by

taking into consideration only those events where both the partner and the change operation type occur (in pairs) in the change event log.

- **H8 (TRH/WRH)** - Both the timestamp and the window related heuristics are implemented as H8. The goal of both is to probabilistically find the correct (i) affected events given an originating event and (ii) originator given an affected event. This is accomplished via the forward (i.e. ω_i^+) as well as the backward window (i.e. ω_i^-) concept to limit the filtering process for the most probable candidate events. Both windows are determined by $Max_{i \in \mathcal{P}}(\Gamma_i)$, i.e. the maximum average response times over all partner response times. For change event candidate selection inside the window, the individual timestamps are used to determine Δ. For the actual selection, the variance value ϵ can be determined empirically. We have opted to base this value on the candidate partner's average response time.

5.3.4. *Fitness Function*

The fitness function measures the quality of a solution in terms of change propagation according to the change event log CEL. The fitness score is a major component in (i) parent selection, determining each individual's eligibility for generating offspring and (ii) survival selection, determining which individuals survive into the next generation to pass on their genes. The following scoring logic is implemented as the fitness function as follows.

$$fitness = w_1 \times completeness + w_2 \times precision \tag{2}$$

Where w_1 and w_2 are weights, the completeness privileges individuals that are more complete according to the CEL and the precision penalizes propagation models that have extra behavior according to the CEL. We define ξ_{ij} as the probability that a change event type (i.e., Replace, Insert or Delete) on partner j is a consequence of a change event type on i. This probability is calculated based on the backward and forward windows, weighted by the number of occurrences of the same sequence of changes in the CEL. As illustrated in Figure 11, the individuals are probabilistic models that represent all or a subset of the change events of the CEL. A node in the propagation model σ is represented by a tuple (δ, π) containing a change type and a partner. The propagation probability between nodes i and j in σ is equal to ξ_{ij}. Then the completeness is given by the following weighted equation:

$$completeness = w_{11} \times \frac{\sum_{i \in \sigma} (\delta_i, \pi_i)}{\sum_{i \in CEL} (\delta_i, \pi_i)} + w_{12} \times \sum_{i,j=1..n} \xi_{ij} \times \phi_{ij}$$
$$with\ \phi_{ij} = \begin{cases} 1 & if (i,j) \in \sigma \\ 0 & otherwise \end{cases} \tag{3}$$

The first term concerns the percentage of traces of the CEL that are represented by the predicted model, while the second term calculates the percentage of the

identified correlations between change events in the CEL that are considered by the propagation model. The attributes w_{11} and w_{12} are the weights given to each term of the equation.

$$precision = \sum_{i,j=1..n} (k \times \xi_{ij} - 1) \times \phi_{ij} \quad with\ \phi_{ij} = \begin{cases} 1 & if(i,j) \in \sigma \\ -1 & otherwise \end{cases} \tag{4}$$

The precision penalizes individuals with extra behavior (noise), by accumulating appropriate negative scores for propagation paths with very low propagation probabilities. The variable k is used to define when an event is considered to be noise; e.g., k=5, means propagation edges with probabilities less then 1/5=0.2 are considered as noise. Therefore, models containing several propagation paths with probabilities lower than 0.2 are classified as bad models. In this equation, we used a simple linear function to penalize noise $k \times \xi_{ij} - 1$, but can be changed to a more complex function (e.g. logarithmic). k is determined empirically.

5.3.5. *Benchmark Results*

	P=3			P=9			P=15		
	G=1	G=10	G=20	G=1	G=10	G=20	G=1	G=10	G=20
Heuristics									
None	-48.74	-48.26	-27.80	-268.75	-226.71	-186.64	-553.90	-502.61	-476.96
H1	-38.08	-30.03	-22.96	-241.34	-197.63	-168.61	-520.53	-482.92	-449.32
H1-H2	-25.94	-20.24	-17.17	-138.19	-106.18	-88.86	-337.45	-289.14	-250.25
H1-H3	**0.73**	**0.78**	**0.80**	0.50	0.55	0.56	0.54	0.55	0.55
H1-H4	0.72	0.76	0.75	0.53	0.58	0.55	0.57	0.55	0.61
H1-H5	0.72	0.77	0.75	0.56	0.50	0.64	0.60	0.62	0.56
H1-H6	0.72	0.77	**0.80**	0.50	0.63	0.63	0.62	0.65	0.67
H1-H7	0.72	**0.78**	0.77	**0.65**	0.67	0.56	**0.71**	**0.73**	0.72
H1-H8	0.71	0.77	0.75	**0.65**	**0.68**	**0.70**	**0.71**	**0.73**	**0.75**

Table 5. Benchmark Results: Memetic Mining of Change Logs. P=Partner Size. G=Generation. Score values are in range $[-\infty, 1]$ (inclusive), where 1 represents the best possible validation score.

For the benchmark, the goal was on one hand to observe the effects of the applied heuristics on reducing the search space and on the other hand to validate the resulting mined model. Towards the former, each inclusion of a new heuristic should increase the maximal achievable score, as it takes less time to find an improved candidate solution. The benchmark was conducted in the following manner: (1) We start with creating distinct CEL slices with differing partner sizes of the range $[3, 16]$. $\Lambda = \{\lambda_i\}^{i \in [3,16]}$ (2) Then we define the heuristic sets to benchmark. The heuristic set *None* means we do not apply any heuristics, which practically reduces the memetic algorithm into a genetic algorithm. A heuristic set *H1-H3* means we apply heuristics *H1*, *H2* and *H3* within the local optimization loop of the memetic algorithm. We have several such heuristic sets as can be observed by the rows in Table 5. (3) Each heuristic set is loaded into the memetic algorithm and executed on the change logs in Λ for up to 20 generations in turn. (4) For validating the mined

model, we derive a propagation model from the CPL (i.e. σ_{CPL}), and compare it to the mined model (i.e. σ_{CEL}) by applying the following scoring function:

$$fitness_{validation} = completeness \times precision - penalties \tag{5}$$

$$precision = \frac{Nb\_extra\_edges}{Nb\_total\_edges} \tag{6}$$

$$completeness = \frac{Nb\_valid\_edges}{Nb\_total\_edges} \tag{7}$$

In equation 6, relationships between change events (edges) that exist in the CEL mined model and not in the CPL model are considered as noise and penalize the generated CEL model. Equation 7 penalizes CEL models that do not include change relationships (edges), which exist in the CPL model; i.e., missing edges. Valid edges are edges from the CPL that exist in the CEL. The total number of edges correspond to the number of edges in the CPL model. The term Penalties is calculated using the heuristics. For example by counting the number of times a CEL model violates the rule of H3: a change operation of type DELETE generates a change operation of type INSERT.

This validation function of Equation 5 returns score values in the range $[-\infty, 1]$, where 1 represents the best possible validation score, meaning the two models are identical. We repeat this process ten times, storing the average score into the respective cells in Table 5. Underlined values are the best scores identified for each column.

We can generally observe the following: regardless of the employed heuristics, with each increasing partner size we obtain a lower quality candidate, except in the case where H8 is introduced. This behavior signals the positive effects of time related heuristics (TRH) as well as window related heuristics (WRH). Similarly, with each increasing generation, the best candidate solution score increases. This holds true, except in cases where the survival selection routine (tournament selection) misses the current best candidate solution, resulting in a lower fitness score. Finally, we can indeed conclude that the proposed heuristics reduce the search space, as the quality of the best candidate solutions increase as more heuristics are added to the memetic change mining algorithm.

In terms of validation, Figure 13(b) shows the mined change propagation model using the described memetic change mining algorithm (on the CEL) with parameters: $partners = 3$, heuristics H1-H6 applied, and $generation = 20$. In contrast, Figure 13(a) represents the change propagation extracted from the CPL. According to Table 5, the average validation score of these two models are 0.72. The differences between these two models are visually illustrated in the annotations in Figure 13(b). As can be seen in that figure, the memetic change mining algorithm could find a good approximation for the prediction model, showing only three extraneous edges (i.e. $(Replace, \pi_8) \rightarrow (Delete, \pi_9)$, $(Replace, \pi_8) \rightarrow (Delete, \pi_7)$ and $(Delete, \pi_7) \rightarrow (Delete, \pi_9)$). Our proposed memetic change mining algorithm fared

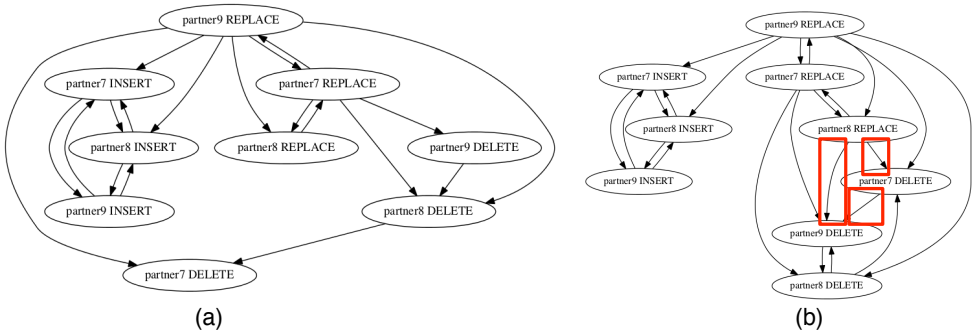

Fig. 13. Mined change propagation models (a) via CPL and (b) via CEL

well in this instance. As more partners are added, more candidates are included as potential consequences, leading to bigger models with more extraneous edges.

6. Related Work

In this section, we discuss the state of the art of change propagation in process choreographies and related fields. The discussion also includes the researches focusing on change prediction and analysis.

6.1. *Change propagation in process choreographies*

Change propagation has been an active research area in several fields. These range from handling change and evolution in service-oriented computing (SOC) [27,28,29], product development [30,31,32,33], as well as large complex software systems [34,35,2,17]. From the business process management field, we have tangential work [36,37], as well as change management approaches that mainly tackle the non-collaborative case [38,39,40].

Only few approaches have been proposed to compute the changes and their propagation in collaborative process settings [41,42,43,44,45,10,3]. Most of these approaches use either the public parts of the partner processes or the choreography/collaboration model; i.e., the global view on all interactions, to calculate the derived changes. They mainly calculate the public parts to be changed, but cannot anticipate the impacts on the private parts, which in turn, could engage knock-on effects on other partners. Besides, in some collaboration scenarios, a partner may have access to only a subset of the partner processes, and consequently could not estimate the transitive effects of the change propagation. Most of the mentioned approaches provide algorithms that calculate the actual propagation of changes in process choreographies. In comparison with these approaches, this paper, estimates the likelihood of propagation and the impacts before the actual propagation. This helps avoiding costly negotiations and propagation failures. A more detailed and thorough comparison between the approaches for actual change propagation can be found in [3].

6.2. *Change Prediction and analysis in Process Choreographies*

Change impact analysis has been an active research area in the context of large complex systems and software engineering [46,2,17,47,6,48]. Giffin et al. [2] perform a comprehensive change propagation analysis on a product that has a history of 41,500 change requests over a period of eight years. In order to analyze the change networks, Design Structure Matrix DSM was used to represent the change propagation structure, and three relationship types between the changes requests (Motifs) were identified. Further, three indices for quantifying each area in the system in terms of its propensity for accepting, reflecting or propagating changes were defined. [48] investigaes a literature review of 150 studies about impact analysis in software systems and provides a classification based on the evaluation of the taxonomy and its criteria.

Eckert et al. [17,6] use mathematical models to analyze the change behavior of GKN Westland Helicopters. Their priori approach predicts the change propagation likelihood and impact on an existing product. The prediction is specific to large complex systems or software systems.

The respective approaches cannot be directly transferred to process choreographies, which entail several particularities [3]; e.g., the distributed model structure, partly unavailable information about partner processes, dynamic aspects, and specific requirements such as compliance, privacy and security. Moreover, the use of the structured change propagation logs combined with memetic as well as genetic mining has not been employed before in these fields. This work analyzes the applicability and transferability of these techniques to process choreographies.

In the context of web service choreographies, only few papers tackled the impact analysis of changes propagation. Oliva et al. [49] compare different choreography adaptation strategies and identify aspects that were neglected: namely *scalability* and *implementation*. Adaptations are mainly reconfigurations of service endpoints. The proposed approach aims at supporting dynamic reconfiguration of collaborative BPEL processes, and assessing the impact of changing services. The approach utilizes a use case defined in BPEL rather than a generic model for business process choreographies. Wang et al. [50,28] define dependency and entropy based impact analysis model to identify impact patterns, and use them to automatically derive changes to the connected web services, reconfiguring them. Change propagation happens after these connected web services are changed. Subsequently, other parts of the business process associated with those web services might change in response. The approach combines dependency analysis to measure the importance of a given service in the collaboration, and information entropy to allow quantification measures of the system. Both approaches consider only static impacts of changes and do not assess the dynamic effects on running instances. Similarly, Dahman et al. [29] avoids duplicate work by mapping the business process model and the service component configurations. In their work, reconfiguration of web service endpoints happen automatically as changes to the business processes are applied. Our previ-

ous work [7] provides a priori approach to analyze dynamic and static change impacts in collaborative process scenarios. The method is based on an analysis of the choreography structure and the different accompanying metrics.

In comparison with this work, the respective approaches use priori analysis techniques and they do not consider previous change propagation experience to enhance the prediction models. These approaches can be complementary to our work as heuristics for the memetic mining .

7. Summary, Lessons Learned and Future Work

Generally, process choreographies realize service-based applications. According to Di Nitto et al. [51], service-based applications undergo three activities during their life cycle ranging from analysis, design, and development over quality definition, negotiation, and assurance to runtime adaptation. Putting the prediction for change propagation into this life cycle, it will be placed in the first phase of analysis, design, and development. Note that at the point a chance propagation becomes necessary, the process choreography consisting of service compositions is already defined and possibly in execution. Change propagation then might require the redesign of service compositions and the prediction techniques presented in this paper analyze and support planning and conducting this redesign in an efficient way.

Being able to predict the change propagation behavior in collaborative process scenarios thus can contribute to time as well as cost reductions, which can determine the overall success of the cooperative process execution. This is particularly crucial in case of choreographies with several partners and complex partner processes. In the simulation, 25 partners with 422 nodes were assumed. In practice, several examples exist with a comparable number of partners as used in the simulation. Zimmermann et al. [52] describe a telecommunication scenario with > 150 partners. In the manufacturing project ADVENTURE, more precisely in a supply chain scenario, around 120 members of the Welsh automotive forum were participating in a process choreography [53]. Generally, manufacturing is a domain where large choreographies with respect to the number of partners are present, specifically, in the context of large and complex product structures [17]. The 422 nodes of the simulation scenario correspond to the number of tasks and connectors in the partner processes which amounts to around 17 tasks and connectors per partner. This is roughly comparable to the 300 activities mentioned in [52].

In this paper, we have shown that there can be two types of change log types that can be utilized for prediction in process choreographies: change propagation logs (CPL) and change event logs (CEL). The former can form the basis for various change propagation analysis approaches, one of them being the Cambridge Advanced Modeler (CAM) with its CPM plugin. With its help, participating partners of the process choreography are classified by their change propagation behaviour. These classifications inform the design of *initial change requests*. We have seen that existing process mining tools, such as ProM, allow us to visualize the prediction

model without the change propagation behaviour classification.

The CPL is lacking in one aspect: when it comes to preserving privacy-sensitive data. In contrast, the CEL still maintains this important issue, which have the advantage of not revealing possibly confidential partnerships especially in combination with anonymization. Unfortunately, existing tools do not handle CEL well for extracting an actionable prediction model, since the core data that handle change propagation are not available. Towards this end we have shown a memetic change mining approach for building a posteriori prediction models based on change event logs (CEL). This approach helps in cases where change propagation logs (CPL) (i.e. those logs which include complete propagation information) are lacking. In addition to the CEL as input, we have proposed a set of heuristics embedded in the memetic change algorithm to guide the candidate selection process towards higher quality ones. The conducted benchmarks and validation of the mined models (see Table 5) show the positive effects of the defined heuristics for reducing the search space, thus reducing the exploration time for finding accurate prediction models. Future work aims at mining change propagation logs (CPL), and analyzing dynamic impacts of process choreography changes.

References

1. Maier, A., Langer, S.: Engineering change management report 2011: Survey results on causes and effects, current practice, problems, and strategies in denmark. Technical report, Denmark Technical University - Management Engineering (2011)
2. Giffin, M., de Weck, O., Bounova, G., Keller, R., Eckert, C., Clarkson, P.J.: Change propagation analysis in complex technical systems. Journal of Mechanical Design **131**(8) (2009)
3. Fdhila, W., Indiono, C., Rinderle-Ma, S., Reichert, M.: Dealing with change in process choreographies: Design and implementation of propagation algorithms. Inf. Syst. **49** (2015) 1–24
4. Weber, B., Reichert, M., Rinderle-Ma, S.: Change patterns and change support features - enhancing flexibility in process-aware information systems. Data and Knowledge Engineering **66**(3) (2008) 438–466
5. Maier, A., Langer, S.: Engineering change management report 2011. Technical University of Denmark, DTU (2011)
6. Clarkson, P.J., Simons, C., Eckert, C.: Predicting Change Propagation in Complex Design. Journal of Mechanical Design **126**(5) (2004) 788–797
7. Fdhila, W., Rinderle-Ma, S.: Predicting change propagation impacts in collaborative business processes. In: Symposium of Applied Computing. (2014) 1378–1385
8. Fdhila, W., Rinderle-Ma, S., Indiono, C.: Memetic algorithms for mining change logs in process choreographies. In: Int'l Conf. Service-Oriented Computing. (2014) 47–62
9. Wynn, D.C., Wyatt, D.F., Nair, S., Clarkson, P.J.: An introduction to the cambridge advanced modeller. Modelling and Management of Engineering Processes (2010)
10. Fdhila, W., Rinderle-Ma, S., Reichert, M.: Change propagation in collaborative processes scenarios. In: CollaborateCom, IEEE (2012) 452–461
11. Dustdar, S., Hoffmann, T., van der Aalst, W.M.P.: Mining of ad-hoc business processes with teamlog. Data Knowl. Eng. **55**(2) (2005) 129–158
12. Jurczyk, P., Xiong, L.: Distributed anonymization: Achieving privacy for both data

subjects and data providers. In: Data and Applications Security. Volume 5645. (2009) 191–207
13. Feild, H.A., Allan, J., Glatt, J.: Crowdlogging: Distributed, private, and anonymous search logging. In: Conference on Research and Development in Information Retrieval. SIGIR '11, New York, NY, USA (2011) 375–384
14. Kohlmayer, F., Prasser, F., Eckert, C., Kuhn, K.A.: A flexible approach to distributed data anonymization. Journal of Biomedical Informatics **50**(0) (2014) 62 – 76
15. Rinderle, S., Jurisch, M., Reichert, M.: On deriving net change information from change logs: the Deltalayer-Algorithm. In: BTW. (2007) 364–381
16. Günther, C., Rinderle-Ma, S., Reichert, M., van Der Aalst, W., Recker, J.: Using process mining to learn from process changes in evolutionary systems. International Journal of Business Process Integration and Management **3**(1) (2008) 61–78
17. Eckert, C.M., Keller, R., Earl, C., Clarkson, P.J.: Supporting change processes in design: Complexity, prediction and reliability. Reliability Engineering and System Safety **91**(12) (2006) 1521–1534
18. Eckert, C., Clarkson, P.J., Zanker, W.: Change and customisation in complex engineering domains. Research in Engineering Design **15**(1) (March 2004) 1–21
19. Pasqual, M.C., Weck, O.L.d.: Multilayer network model for analysis and management of change propagation. Research in Engineering Design **23**(4) (October 2012) 305–328
20. Tukey, J.: Exploratory data analysis. Addision-Wesley (1977)
21. De Medeiros, A., Weijters, A., van der Aalst, W.: Genetic process mining: an experimental evaluation. Data Mining and Knowledge Discovery **14**(2) (2007) 245–304
22. Van der Aalst, W.M.P.: Process Mining: Discovery, Conformance and Enhancement of Business Processes. 1st edn. Springer (2011)
23. Buijs, J.C.A.M., van Dongen, B.F., van der Aalst, W.M.P.: Mining configurable process models from collections of event logs. In: BPM. (2013) 33–48
24. Gaaloul, W., Gaaloul, K., Bhiri, S., Haller, A., Hauswirth, M.: Log-based transactional workflow mining. Distributed and Parallel Databases **25**(3) (2009) 193–240
25. Goldberg, D.: Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley Longman Publishing Co. (1989)
26. Eiben, A., Smith, J.: Introduction to Evolutionary Computing. Natural Computing. Springer-Verlag, Berlin (2007)
27. Charfi, A., Mezini, M.: Aspect-oriented web service composition with AO4BPEL. In Zhang, L.J.L., Jeckle, M., eds.: Web Services. Number 3250 in Lecture Notes in Computer Science. Springer Berlin Heidelberg (January 2004) 168–182
28. Wang, Y., Yang, J., Zhao, W., Su, J.: Change impact analysis in service-based business processes. Service Oriented Computing and Applications **6**(2) (June 2012) 131–149
29. Dahman, K., Charoy, F., Godart, C.: Alignment and change propagation between business processes and service-oriented architectures. In: 2013 IEEE International Conference on Services Computing (SCC). (June 2013) 168–175
30. Wright, I.C.: A review of research into engineering change management: implications for product design. Design Studies **18**(1) (January 1997) 33–42
31. Eger, T., Eckert, C.M., Clarkson, P.J.: Engineering change analysis during ongoing product development. (2007)
32. Jarratt, T.a.W., Eckert, C.M., Caldwell, N.H.M., Clarkson, P.J.: Engineering change: an overview and perspective on the literature. Research in Engineering Design **22**(2) (April 2011) 103–124
33. Ahmad, N., Wynn, D.C., Clarkson, P.J.: Change impact on a product and its redesign process: a tool for knowledge capture and reuse. Research in Engineering Design **24**(3) (July 2013) 219–244

34. Ajila, S.: Software maintenance: An approach to impact analysis of objects change. Software - Practice and Experience **25** (1995) 11551181
35. Malik, H., Hassan, A.: Supporting software evolution using adaptive change propagation heuristics. In: IEEE International Conference on Software Maintenance, 2008. ICSM 2008. (September 2008) 177–186
36. Desai, N., Chopra, A.K., Singh, M.P.: Amoeba: A Methodology for Modeling and Evolving Cross-Organizational Business Processes. (2008)
37. Mahfouz, A., Barroca, L., Laney, R., Nuseibeh, B.: Requirements-driven collaborative choreography customization. In Baresi, L., Chi, C.H., Suzuki, J., eds.: Service-Oriented Computing. Number 5900 in Lecture Notes in Computer Science. Springer Berlin Heidelberg (January 2009) 144–158
38. Reichert, M., Dadam, P.: ADEPT flex - supporting dynamic changes of workflows without loosing control. Journal of Intelligent Information Systems **10** (1998) 93–129
39. Casati, F., Ceri, S., Pernici, B., Pozzi, G.: Workflow evolution. In Thalheim, B., ed.: Conceptual Modeling ¡E2¿¡80¿¡94¿ ER '96. Number 1157 in Lecture Notes in Computer Science. Springer Berlin Heidelberg (January 1996) 438–455
40. Weidlich, M., Mendling, J., Weske, M.: Propagating changes between aligned process models. Journal of Systems and Software **85**(8) (August 2012)
41. van der Aalst, W.M.P., Basten, T.: Inheritance of workflows: an approach to tackling problems related to change. Theor. Comput. Sci. **270**(1-2) (2002) 125–203
42. Atluri, V., Chun, S.A.: Handling dynamic changes in decentralized workflow execution environments. In: DEXA'03. (2003) 813–825
43. Rinderle, S., Wombacher, A., Reichert, M.: Evolution of process choreographies in DYCHOR. In: CoopIS. LNCS (2006) 273–290
44. Fdhila, W., Rinderle-Ma, S., Baouab, A., Perrin, O., Godart, C.: On evolving partitioned web service orchestrations. In: SOCA. (2012) 1–6
45. Wang, M., Cui, L.: An impact analysis model for distributed web service process. In: Computer Supported Cooperative Work in Design (CSCWD). (2010) 351–355
46. Bohner, S.A., Arnold, R.S.: Software change impact analysis. IEEE Computer Society (1996)
47. Eckert, C., Zanker, W., Clarkson, P.J.: Aspects of a better understanding of changes. In: ICED. Volume 1. (2001)
48. Steffen, L.: A review of software change impact analysis. Technical report, Universittsbibliothek Ilmenau (2011)
49. Oliva, G.A., de Maio Nogueira, G., Leite, L.F., Gerosa, M.A.: Choreography Dynamic Adaptation Prototype. Technical report, Universidade de So Paulo (2012)
50. Wang, S., Capretz, M.: Dependency and entropy based impact analysis for service-oriented system evolution. In: Web Intelligence. (2011) 412–417
51. Di Nitto, E., Ghezzi, C., Metzger, A., Papazoglou, M., Pohl, K.: A journey to highly dynamic, self-adaptive service-based applications. Automated Software Engineering **15**(3-4) (2008) 313–341
52. Zimmermann, O., Doubrovski, V., Grundler, J., Hogg, K.: Service-oriented architecture and business process choreography in an order management scenario: rationale, concepts, lessons learned. In: Companion to the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications, ACM (2005) 301–312
53. AZEV, INESC, T.: Adventure: D7.2 use case implementation report (2014)