Towards an Automatic Identification of Microservices from Business Processes

Mohamed Daoud, Asmae El Mezouari, Noura Faci, Djamal Benslimane, Zakaria Maamar, Aziz El Fazziki

Towards an Automatic Identification of Microservices from Business Processes

Mohamed Daoud[1], Asmae El Mezouari[2], Noura Faci[1], Djamal Benslimane[1]
Zakaria Maamar[3] and Aziz El Fazziki[2]
[1]Université Lyon 1, Lyon, France – [2]Caddi Ayyad University, Marrakech, Morocco
[1]Zayed University, Dubai, UAE

***Abstract*—Microservices have emerged as an alternative solution to many existing technologies allowing to break monolithic applications into "small" fine-grained, highly-cohesive, and loosely-coupled units. However, identifying microservices remains a challenge that could undermine this migration success. This paper proposes an approach for microservices automatic-identification from a set of business processes (BP). The approach is multi-models combining different independent models that represent a BP's control dependencies, data dependencies, semantic dependencies, respectively. the approach is also based on collaborative clustering. A case study about renting bikes is adopted to illustrate and demonstrate the approach. In term of precision, the results show how BPs as inputs permit to generate better microservices compared to other approaches discussed in the paper, as well.**

***Index Terms*—Microservices, Semantics, Business Process, Disco, Data dependencies.**

I. Introduction

There is a consensus that existing solutions like Commercial-Of-The-Shelf and Component-Based Software Engineering have already shown their limitations to tackle the challenge of managing monolithic (legacy) systems. Monolithic means one block of strongly-coupled components that complexify the process of accommodating functional, non-functional, and structural changes that all organizations experience on a daily basis. Contrarily, microservices have emerged as an alternative solution allowing to break such systems into "small" units that are fine-grained, highly-cohesive, and loosely-coupled. However to sustain the benefits of microservices their successful discovery is a must. This paper presents a novel approach to discover microservices. First we use a set of Business Processes (BP) as an input for discovery while others use log files [6], source codes [7], UML class diagram [2], and legacy databases [3]. Second, we use clustering technique to capture the dependencies that characterize BPs' activities in terms of control, data, and functional.

Referred to as organizations' main assets, BPs could constitute an important source for identifying microservices. According to Weske, *"... A business process consists of a set of activities that are performed in coordination in an organizational and technical environment. These activities jointly realize a business goal. Each business process is enacted by a single organization, but it may interact with business processes performed by other organizations"* [14]. Despite being a rich reservoir of many details like who does what, when, where, and why, BPs seem overlooked during the exercise of identifying microservices. To the best of our knowledge, Amiri is the only one who adopted BPs in this exercise [1].

In this paper, we design and develop a collaborative clustering-based approach to automatically identify microservices. At the core of our approach is the idea of extracting microservices from BPs' activities by using first, a set of separate models that individually identify different kinds of information related to these activities and their dependencies and second, a collaborative clustering technique instead of a classical clustering technique to avoid aggregating extracted data that could lead to losing some implicit details. Our approach is multi-models in the sense that it combines independent models to represent a BP's structural dependencies, data dependencies, semantic dependencies, and so on. It is also based on collaborative clustering in the sense that all data that could be extracted from the different models are kept independent instead of aggregating them. Each data set is then handled by a separate clustering algorithm.

The rest of the paper is organized as follows. Section 2 describes related work. Section 3 presents a case study and gives an overview of our approach to automatically identify microservices from existing BPs. Section 4 formalizes the structural and semantic dependencies models. Section 5 details the collaborative clustering in this approach and then, the system implementing this approach along with some experiments' results. Section 6 concludes the paper.

II. Related work

Decomposing existing monolithic applications into a set of microservices is one of the main challenges in migrating such applications to a microservices-based architecture. In this section, we discuss different approaches that pursue such objective.

In‘ [3], a 3-step approach to break a business logic into potential microservice is presented. The first step manually refines a traditional Data-Flow Diagram (DFD) to shed light on data flows with respect to a business logic. During this step details about data stores and external entities are discarded. The next step uses some rules to abstract the refined DFD into a decomposable DFD allowing to improve the quality of the future microservices to obtain. Finally, the third step proceeds with identifying the microservices based on the operations of the DFD and their output data.

In [11], an approach based on functionality-oriented microservice extraction by clustering execution traces of programs collected at run-time is presented. These traces are

collected using program execution monitoring techniques and permit to examine the implicit and explicit program functional behavior. They also reveal which entities are used for which business logic. The approach clusters source code entities that are related to the same functionalities. Even if the work in [11] is interesting, it suffers from the strong dependence on the quality of the generated execution traces, and consequently on the quality of the test cases.

In [1], a clustering-based approach is discussed where structural and data dependencies between tasks are extracted from a given set of business processes. All these dependencies are merged into one dependency matrix. This latter is used by a classical clustering algorithm to identify candidate microservices.

In [9], a service-cutter system is presented allowing to decompose monolithic systems into "small" services. The decomposition uses the systems' functions and considers 3 criteria that are cohesion between entities at the property level, coherence between data of each microservice, and communication cost between services. In [3], an approach to identify microservices is proposed. It uses the semantic similarity of functionalities that are described in openAPI and a reference vocabulary.

III. Our approach for identifying microservices

This section consists of 4 parts. The first part presents a case study that refers to Barcelona's bike sharing system known as Bicing. The second and third parts discuss our approach's foundations in terms of dependencies between BPs' activities and collaborative clustering.

A. Bicing case-study

Bicing includes more than 400 bike anchor-stations spread across Barcelona and about 6000 bikes that users rent for a fee. Bicing's monolithic system is described in [8] along with the managerial and technical challenges that undermine its operations. For the needs of our work, we suggest in Fig. 1 a high-level representation of Bicing from a BP perspective. We resorted to the standard Business Process Model and Notation (BPMN) to illustrate this representation. We have identified different activities (a_1: request bike and a_9: dismantle bike), different dependencies (between a_1 and a_2), different logical operators (XOR between a_6 and a_{10} and OR between a_2 and a_3), and, finally, different artefacts (bike and user) and their respective attributes (e.g., ID and status). In the rest of this paper the discussion about Bicing is not restricted to renting bikes but includes other aspects like reporting and fixing bikes' defects and disposing bikes, if necessary.

It all starts when a user requests a bike (a_1) at a certain anchor station. After checking the user's credentials (a_2) and any late fee payment (a_3), the Bicing system updates the user's records (a_4) and then, approves the user's request (a_5). If it turns out that the bike is defective, the user puts it back (a_6) and eventually requests another one. Otherwise, the user starts his journey (a_{10}). Regularly all bikes are serviced (a_7) leading to either putting them back for rent (a_8) or disposing them (a_9). When the user arrives to destination, he returns the bike at a certain anchor point (a_{11}). Otherwise, the Bicing system blacklists the user (a_{14}) due to bike inappropriate return and geo-locates the bike (a_{12}) so that

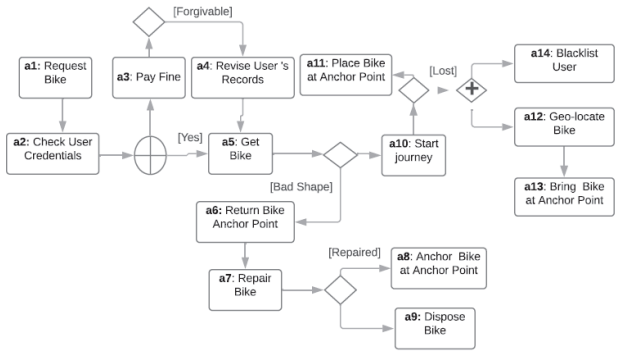


Fig. 1: An illustrative BPMN-based representation of the Bicing system

it is collected by the competent services and then made available to other users (a_{13}).

From a specification perspective, activities ($\{a_i\}$) may require inputs ($\{i_i\}$) and produce outputs ($\{o_i\}$) that both correspond to specific artefacts' attributes. An activity acts upon an attribute through 2 operations that are *read* (r) and/or *write* (w), which could lead to *updating* (u), *creating* (c), and/or *deleting* (d) artefacts.

B. Foundations

Simply put, a BP is a set of logically related activities that are performed to achieve goals [5]. "Logically related" means connecting activities using dependencies such as *control* (execution order), *semantics* (functional similarity), *data* (information sharing), and *organisational* (cross-functional operations).

- *Control dependency* refers to both the execution order (e.g., finish-to-start and start-to-start) between activities and the logical operators (e.g., XOR and AND) between activities as well. Should 2 activities be directly connected through a control dependency, then most probably they would form a highly-cohesive microservice to which they will belong. Contrarily, they would most probably be used to form separate microservices to which each will belong.
- *Semantic dependency* uses activities' names to establish their functional similarity in term of what they do. Activities' names refer to linguistic templates that must include a verb and object/result along with optional parameters like time and location. For instance, a_{11}'s name includes place (verb), bike (object), and anchor point (location). To assess activities' names similarity, we rely on either a reference vocabulary or an ontology. A highest/lowest similarity value would indicate strong/weak coupling between activities making them members of the same/different microservice/microservices.
- *Data dependency* is about data flows between activities' outputs/inputs that constitute artefacts' attributes that could be subject to *create* (c) and *write* (w) operations and could be labeled as either optional or mandatory for BP execution. We advocate that activities exchanging mandatory artefacts' attributes should be part of the same microservice allowing to comply with the strong coupling principle.

- *Organizational dependency* refers to horizontal (cross) and vertical (silo) business operations. We advocate that activities that fall into horizontal interactions would more likely be part of the same microservices.

We focus on *control* and *semantic dependencies*, only. Upon establishing such dependencies, we quantify them using specific metrics that evaluate the cohesion and coupling among activities so they are gathered in either same or separate microservices. To evaluate a *control dependency* between 2 activities (a_i, a_j), we consider a_j's occurrence probability after executing a_i. This probability depends on the execution order and/or logical operators between a_i and a_j. Let us consider the *control dependency* between a_5 and a_{10} that is connected with a_6 through XOR (Fig. 1). After executing a_5, a_{10}'s occurrence probability depends on the decision made at XOR (i.e., either a_6 or a_{10}). We note that any activity's occurrence probability is continuously calculated using BP's execution logs.

To evaluate *semantic dependency* between 2 activities (a_i, a_j), we consider the distance between their respective semantic annotations. We identify 3 annotation strategies referred to as *word-driven* ($\mathcal{W}$), *concept-driven* ($\mathcal{C}$), and *fragment-driven* ($\mathcal{F}$). The first relies on a reference vocabulary to annotate activities with terms associated with their respective names. The second relies on some specific domain ontology to annotate activities with concepts related to their respective names. Finally, the third refines the second by annotating the activities with ontological fragments that would refer to functional domains.

Based on dependencies among activities, we gather activities into microservices by using clustering techniques. In the literature, clustering is either centralized or collaborative [10]. In the former, a single component manages the clustering by utilizing all individuals' features as inputs. In the latter, multiple components, each in charge of one type of features, exchange some details during clustering so that appropriate clusters are jointly built. Performance and appropriateness of clustering techniques are thoroughly discussed in the literature [4]. Many works like [4] and [10] advocate for collaborative clustering to identify microservices. It provides fine-grained and accurate results contrarily to centralized clustering where individuals' features need to be aggregated before initiating any clustering.

Fig 2 depicts our approach for microservices identification. It relies on the aforementioned dependencies and collaborative clustering to group activities that would form fine-grained, highly-cohesive, and loosely-coupled microservices. Our approach consists of the following steps. After examining activity dependencies, their details are stored in dedicated repositories. Then, these repositories' contents are submitted for collaborative clustering where different clustering techniques (one per dependency) are used to obtain consensual clustering solutions. More details are given in Section V-A.

IV. Analyzing dependencies

A. Control dependency analysis

Let CD(a_i, a_j[Operator $\{a_k\}$])$_{\text{ExecOrder}}$ be a direct *control dependency* referring to a certain execution order between a_i and a_j that is connected to other activities $\{a_k\}$ through a certain Operator. An execution order between 2 activities could be exemplified with either *finish-to-start* (our focus and denoted by SEQ)), *finish-to-finish*, *start-to-start*, or *start-to-finish*.

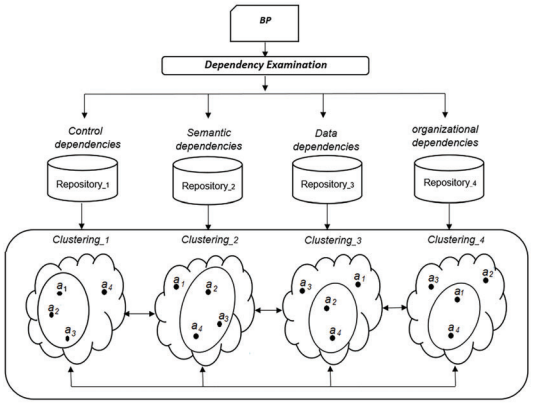


Fig. 2: General representation of our microservice identification approach

Let us start with the *control dependency* $CD(a_i, a_j)_{\text{SEQ}}$ (i.e., $\{a_k\} = \emptyset$)). Since SEQ between a_i and a_j means that a_j starts only after a_i has successfully completed, $CD(a_i, a_j)_{\text{SEQ}}$ denotes a_j's occurrence probability (p) after a_i's completion as per Equation 1:

$$CD(a_i, a_j)_{\text{SEQ}} = p \tag{1}$$

where $p \in]0, 1]$.

We now examine the *control dependency* $CD(a_i, a_j$ Operator $\{a_k\})_{\text{SEQ}}$ (i.e., $\{a_k\} \neq \emptyset$)). According to Operator's semantics, we assume that some r activities in $\{a_k\} \cup a_j$ will be selected for execution. Equation 2 defines the number of activities that will be selected for execution as a combination $C(n, r)$ where n corresponds to $card(\{a_k\} \cup a_j)$.

$$C(n, r) = \frac{n!}{r! \times (n-r)!} \tag{2}$$

Depending on the semantics of Operator whether AND, XOR, or OR, $CD(a_i, a_j$ Operator $\{a_k\})_{\text{SEQ}}$ is calculated as follows:

1) $CD(a_i, a_j$ AND $\{a_k\})_{\text{SEQ}}$. This dependency means that a_j will start only after a_i has successfully completed regardless of $\{a_k\}$. Formally, Equation 3 computes $CD(a_i, a_j$ AND $\{a_k\})_{\text{SEQ}}$ as follows:

$$CD(a_i, a_j \text{ AND } \{a_k\})_{\text{SEQ}} = C(n, n) * CD(a_i, a_j)_{\text{SEQ}} \tag{3}$$

where $p \in]0, 1]$ & $C(n, n) = 1$, as per Equation 2.

2) $CD(a_i, a_j$ XOR $\{a_k\})_{\text{SEQ}}$. This dependency means that one activity from $\{a_k\} \cup a_j$ will be selected after a_i has successfully completed. Formally, Equation 4 computes $CD(a_i, a_j$ XOR $\{a_k\})_{\text{SEQ}}$ as follows:

$$CD(a_i, a_j \text{ XOR } \{a_k\})_{\text{SEQ}} = \frac{1}{C(n,1)} * CD(a_i, a_j)_{\text{SEQ}} \tag{4}$$

where $C(n, 1)$ is the number of possibilities to select one activity from $\{a_k\} \cup a_j$. As per Equation 2, $C(n, 1)$ is equal to n.

3) $CD(a_i, a_j \text{ OR } \{a_k\})_{\text{SEQ}}$. This dependency means that a set of r activities from $2^{\{a_k\} \cup a_j}$ (i.e., all possible multiple choices) will be selected after a_i has successfully completed. For the sake of simplicity, we assume that any activity in $\{a_k\} \cup a_j$ has the same occurrence probability over $2^{\{a_k\} \cup a_j}$, that is equal to $\frac{r}{n}$ where r varies from 1 to n. Formally, Equation 5 computes $CD(a_i, a_j \text{ OR } \{a_k\})_{\text{SEQ}}$ as follows.

$$CD(a_i, a_j \text{ OR } \{a_k\})_{\text{SEQ}} = \frac{\sum_{r=1,n}(\frac{r}{n} \times C(n,r))}{\sum_{r=1,n} C(n,r)} * CD(a_i, a_j)_{\text{SEQ}} \quad (5)$$

where

- $\sum_{r=1,n}(\frac{r}{n} \times C(n,r))$ represents the number of a_j's occurrences among possible combinations of activities[1].
- $\sum_{r=1,n} C(n,r)$ corresponds to the total number of possible combinations of activities[2].

We now look into indirect *control dependency* between a_i and a_j where there is a set of other peers connected with operators. This dependency, denoted as $CD(a_i, a_j)_{path^1_{i,j}}$, refers to a certain execution path ($path^1_{i,j}$) and is computed as per Equation 6:

$$CD(a_i, a_j)_{path^1_{i,j}} = \prod_{a_l, a_m \in path^1_{i,j}} CD(a_l, a_m \text{ Operator } \{a_{k_m}\})_{\text{SEQ}} \quad (6)$$

When multiple execution paths exist between a_i and a_j, we refer to this *control dependency* as $CD(a_i, a_j)_{paths_{i,j}}$ and is computed as per Equation 7:

$$CD(a_i, a_j)_{paths_{i,j}} = \max_{q=1,...}(CD(a_i, a_j)_{path^q_{i,j}}) \quad (7)$$

where $path^q_{i,j}$ represents the q^{th} possible execution path between a_i and a_j. Table I depicts an excerpt of *control dependencies* in the Bicing system.

TABLE I: Control dependencies with $p = 0.5$

Activity	a_1	a_2	a_3	a_4	a_5
a_1	-	1/2	5/6	11/12	17/12
a_2	1/2	-	1/3	7/12	11 /12
a_3	5/6	1/3	-	1/4	3/4
a_4	11/12	7/12	1/4	-	1/2
a_5	7/12	11/12	3/4	1/2	-

B. Semantic dependency analysis

Let $\mathcal{X}$ be the annotation strategy and $SD^{\mathcal{X}}(a_i, a_j)$ be a *semantic dependency* between a_i and a_j using $\mathcal{X}$. Prior to formalizing $SD^{\mathcal{X}}$, we first describe our 3 proposed annotation strategies.

Word-driven strategy ($\mathcal{W}$). Algorithm 1 outlines how to annotate an activity a_i using a set of the n most similar words ($\mathcal{W}_{a_i}$). To develop $\mathcal{W}_{a_i}$, we adopt No et al.'s solution, $DISCO$ [13], that assumes that words with similar meaning (co-)occur in similar bag-of-words contexts. Thus, $\mathcal{W}_{a_i}$ represents the set of a_i's most distributionally similar words (w_k) along with their respective similarity degrees ($sd_{i,k}$). It is worth mentioning that any other semantic textual similarity solution can be used.

[1]Let n be 3, $\sum_{r=1,n}(\frac{r}{n} \times C(n,r))$ has the following value: $(\frac{1}{3} \times 3 + \frac{2}{3} \times 3 + \frac{3}{3} \times 1$=4).

[2]Let n be 3, $\sum_{r=1,n} C(n,r)$ has the following value: (3+ 3+ 1)=7.

Algorithm 1: *Word-driven* annotation strategy

Input: a_i, n
Output: $\mathcal{W}_{a_i}$
1 **begin**
2 $\quad \mathcal{W}_{a_i} \leftarrow DISCO(a_i, n)$
3 $\quad$ **return** $\mathcal{W}_{a_i}$

Concept-driven strategy ($\mathcal{C}$). Algorithm 2 outlines how to annotate an activity a_i using a set of the most similar concepts ($\mathcal{C}_{a_i}$) that would belong to the BP's domain ontology ($\mathcal{O}_{BP}$). For each concept c_j, the algorithm uses No et al.'s similarity measure between a_i and c_j, namely $DISCO_2$. In line 3, max_s function develops $\mathcal{C}_{a_i}$ by considering the concepts with the highest similarity values with respect to a certain precision σ. For instance, if the highest value is 0.5 and σ is set to 0.1, then max_s also includes all the concepts with a similarity degree between 0.4 and 0.5.

Algorithm 2: *Concept-driven* annotation strategy

Input: a_i, $\mathcal{O}_{BP}$, σ
Output: $\mathcal{C}_{a_i}$
1 **begin**
2 $\quad$ **foreach** $c_j \in \mathcal{O}_{BP}$ **do**
3 $\quad\quad \mathcal{D}[i,j] \leftarrow DISCO_2(a_i, c_j)$
4 $\quad \mathcal{C}_{a_i} \leftarrow max_s(\{\mathcal{D}[i,j]\}, \sigma)$
5 $\quad$ **return** $\mathcal{C}_{a_i}$

Fragment-driven strategy ($\mathcal{F}$). Algorithm 3 outlines how to annotate an activity a_i using a set of the most similar (eventually overlapped) fragments ($\mathcal{F}_{a_i}$) that each encompasses concepts belonging to $\mathcal{O}_{BP}$. Lines 2-3 compute a_i's membership degrees ($\mathcal{M}[i]$) to each fragment $\mathcal{F}_k \in \mathcal{O}_{BP}$ based on the set of common concepts that belong to $\mathcal{F}_k$ and $\mathcal{C}_{a_i}$ along with $sd_{i,j}$ associated with each c_j. In Line 4, max_m function develops $\mathcal{F}_{a_i}$ by considering the fragments with the highest similarity degrees with respect to a certain precision σ.

Algorithm 3: *Fragment-driven* annotation strategy

Input: $\mathcal{C}_{a_i}$, $\mathcal{O}_{BP}$, σ
Output: $\mathcal{F}_{a_i}$
1 **begin**
2 $\quad$ **foreach** $\mathcal{F}_k \in \mathcal{O}_{BP}$ **do**
3 $\quad\quad \mathcal{M}[i,k] \leftarrow \sum_{c_j \in \mathcal{F}_k \cap \mathcal{C}_{a_i}} \mathcal{D}[i,j]$
4 $\quad \mathcal{F}_{a_i} \leftarrow max_m(\mathcal{M}[i], \sigma)$
5 $\quad$ **return** $\mathcal{F}_{a_i}$

Formally, Equation 8 computes the Semantic Dependency (SD) between a_i and a_j.

$$SD(a_i, a_j) = 1 - d_{\mathcal{X}}(\mathcal{X}_{a_i}, \mathcal{X}_{a_j}) \quad (8)$$

where $\mathcal{X}_{a_i}$ reflects the annotation strategy's outcome (i.e., either $\mathcal{W}_{a_i}$, $\mathcal{C}_{a_i}$, or $\mathcal{F}_{a_i}$) and $d_{\mathcal{X}}$ represents the distance between $\mathcal{X}_{a_i}$ and $\mathcal{X}_{a_j}$. We hereafter define each $d_{\mathcal{X}}$.

- $d_{\mathcal{W}}$ compares all the words in $\mathcal{W}_{a_i}$ to those in $\mathcal{W}_{a_j}$. Formally, Equation 9 computes $d_{\mathcal{W}}$ as follows.

$$d_{\mathcal{W}}(\mathcal{W}_{a_i}, \mathcal{W}_{a_j}) = \sum_{w_k \in \mathcal{W}^p_{a_i}} sd_{i,k} + \sum_{w_k \in \mathcal{W}^p_{a_j}} sd_{j,k} \quad (9)$$

where $\mathcal{W}^p_{a_i|a_j}$ represents $\mathcal{W}_{a_i|a_j}$'s privative set of words (i.e., $\mathcal{W}_{a_i|a_j} - \mathcal{W}_{a_i} \cap \mathcal{W}_{a_j}$).

Equation 10 computes the normalized $d_{\mathcal{W}}$ as follows.

$$d_{\mathcal{W}}^{norm}(\mathcal{W}_{a_i}, \mathcal{W}_{a_j}) = \frac{d_{\mathcal{W}}(\mathcal{W}_{a_i}, \mathcal{W}_{a_j})}{|\mathcal{W}_{a_i}^p| + |\mathcal{W}_{a_j}^p|} \quad (10)$$

- $d_{\mathcal{C}}$ compares all the concepts in $\mathcal{C}_{a_i}$ with those in $\mathcal{C}_{a_j}$. Formally, Equation 11 computes $d_{\mathcal{C}}$ as follows.

$$d_{\mathcal{C}}(\mathcal{C}_{a_i}, \mathcal{C}_{a_j}) = \sum_{c_k \in \mathcal{C}_{a_i}} (1 - \mathcal{D}[i,k]) * \sum_{c_l \in \mathcal{C}_{a_j}} (1 - (\mathcal{D}[j,l] * WU(c_k, c_l)) \quad (11)$$

Equation 12 computes the normalized $d_{\mathcal{C}}$ as follows.

$$d_{\mathcal{C}}^{norm}(\mathcal{C}_{a_i}, \mathcal{C}_{a_j}) = \frac{d_{\mathcal{C}}(\mathcal{C}_{a_i}, \mathcal{C}_{a_j})}{|\mathcal{C}_{a_i}| * |\mathcal{C}_{a_j}|} \quad (12)$$

- $d_{\mathcal{F}}$ compares all the fragments in $\mathcal{F}_{a_i}$ with those in $\mathcal{F}_{a_j}$. Formally, Equation 13 computes $d_{\mathcal{C}}$ as follows.

$$d_{\mathcal{F}}(\mathcal{F}_{a_i}, \mathcal{F}_{a_j}) = \prod_{\mathcal{F}_k \in \mathcal{O}_{BP}} (1 - |\mathcal{M}[i,k] - \mathcal{M}[j,k]|) \quad (13)$$

Let $\mathcal{P}_2(BP)$ be the set of all distinct activity pairwises built from BP (i.e., $a_k \neq a_l$). Formally, Equation 14 computes the normalized $d_{\mathcal{F}}$ as follows.

$$d_{\mathcal{F}}^{norm}(\mathcal{F}_{a_i}, \mathcal{F}_{a_j}) = \frac{d_{\mathcal{F}}(\mathcal{F}_{a_i}, \mathcal{F}_{a_j})}{d_{\mathcal{F}}^{max}} \quad (14)$$

where

- $d_{\mathcal{F}}^{max} = max(\{d_{\mathcal{F}}(\mathcal{F}_{a_k}, \mathcal{F}_{a_l})\}_{<a_k,a_l> \in \mathcal{P}_2(BP)})$

V. Collaborative Clustering Algorithm and Experimental Evaluation

In this section, we first describe our proposed collaborative clustering of BPs' activities and then discuss the experiments and evaluation.

A. Collaborative clustering

We designed a collaborative clustering algorithm (cHAC) that extends the classical Hierarchical agglomerative algorithm (HAC) [12]. cHAC is performed by N homogeneous clustering nodes $(CN_1, CN_2, \ldots, CN_n)$ executing the same program. However they differ with respect to their inputs. Each CN node handles one and only one dependency matrix.

cHAC algorithm fosters collaboration between CN since each CN has its own dependency matrix along with "keeping an eye" on what other CNs are doing by sharing some dependencies scores about activities, if deemed necessary. Thus, prior to each new HAC clustering iteration, a CN uses both a Local Score Matrix (LSM) that stores dependency scores between couple of activities and a Shared Score Matrix (SSM) that stores a global dependency score between each couple of activities.

Our cHAC algorithm goes over 3 phases:

- Initialization phase (Algorithm 4). All CN nodes are launched with their respective number of clusters k, and their respective local dependency score matrices. An empty shared dependency matrix is also created to store the shared dependency score of activities. Each activity constitutes a cluster. A shared variable is also introduced to synchronize the iteration of the nodes. A new iteration is launched at a given node if-and-only-if the other nodes have already finished their current iterations.

Algorithm 4: cHAC - Initialization

Input: N - number of clustering node CN, K_i $i = 1..N$ - targeted number of clusters of each node CN_i, $DM_i[M,M]$ $i = 1..N$ - DM_i Dependency Matrix of CN_i, $SDSM[M,M]$ - Shared dependency score matrix;
Output: final cluster result fc;

```
1 begin
2     SIC ← 0 ;        // shared counter between CN nodes
3     cSET[i] ←cHACn(k_i, DM_i, @SDSM, @SIC), i = 1..M ;
        // parralel execution of CN nodes
4     ;
5     fc ← chooseFinalClusterResult(cSET);
6     return fc;
```

- Collaborative Iteration phase. A classical HAC clustering is extended to make it collaborative as per Algorithm 5. The nearest pair of clusters C_u and C_v is computed by using both LSM and SSM based on calculating the distance using the formula:

$$SM_p\,(C_u, C_v) = \sum_{(i,j)=(1,1)}^{|C_u|*|C_v|} SM_{p-1}\,(a_i, a_j)\,/\,(|C_u| * |C_v|)$$

where the score matrix SM_p designates either LSM or SSM matrix at the p^{th} iteration.

Clusters C_u and C_v are merged if and only if:[3]

$$[distance(C_u, C_v)]_p^{LSM} >= [distance(C_u, C_v)]_{p-1}^{SSM}$$

Then, the node updates The Local Score Matrix by calculating the new scores of activities using the formula:

$$SM_p\,(a_i, a_j) = \sum_{j=1}^{|C_v|} SM_p\_1\,(a_i, a_j)\,/|C_v|$$

To foster similarities between couples of activities (a_i, a_j), the shared score matrix is also updated as follows:

$[SSM(a_i, a_j)]_p = Max([LDSM(a_i, a_j)]_p, [SSM(a_i, a_j)]_{p-1})$

- Selection phase. Once the different clustering results are produced by the different CNs, the distance metrics are applied to them to choose the best one that fosters both cohesion and loose-coupling of groups.

B. Experiments

We implemented the collaborative clustering approach in Java. To this end, different modules have been developed. Some of these modules permitted to extract control dependencies and data dependencies from BPs and to run the collaborative clustering algorithm that took the number of clusters and dependency matrices as inputs. Bicing system's 14 activities were initially used to test the algorithm. Then, more activities were randomly generated to capture the complexity of real BPs.

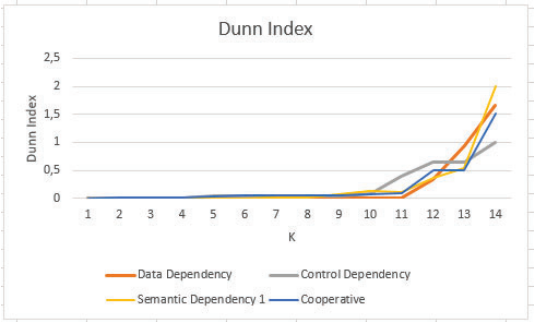
For evaluation needs we considered the internal validation metric Dunn index that measures the quality of clustered results by identifying the clusters that are compact (minor

[3] $[distance(C_u, C_v)]_p^{SC}$ computes any distance between 2 clusters C_u and C_v by using the score Matrix SM of the iteration p.

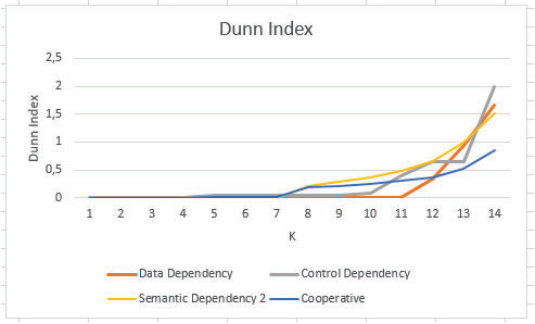
Algorithm 5: cHACn - collaborative clustering

Input: K - targeted number of clusters, N: number of clustering node CN, DM: Dependency matrix, $@SDSM$: Shared dependency score matrix;
Output: $\mathbb{C}$ - clustering set

begin
  C_u, C_v, C_p - cluster variables
  $\mathbb{C} \leftarrow \{\{a_1\}, \{a_2\}, ..., \{a_M\}\}$; // each activity is a cluster
  while $|\mathbb{C}| > k$ **do**
    $(C_u, C_v) \leftarrow NearestPeerCluster(\mathrm{C})$;
    if $[distance(C_u; C_v)]^{DM} \geqslant [distance(C_u; C_v)]^{SDSM}$ **then**
      $C_p \leftarrow fusion(C_u, C_v)$;
      $\mathbb{C} \leftarrow$C-(C$_u$ $\cup$C$_v$);
      $\mathbb{C} \leftarrow$C$\cup$C$_p$;
      updateDM();
      $@SIC \leftarrow @SIC + 1 \mod N$; // The node indicate to other nodes that its current clustering iteration is done
    WAIT ($@SIC = 0$) ; // the new iteration will be started only when all oher CN nodes have finished their current clustering iteration
    updateSDSM();
  return $\mathbb{C}$;



(a) Dunn Index corresponding to the control, data and Semantic (1st strategy) models



(b) Dunn Index corresponding to the control, data and Semantic (2nd strategy) models

Fig. 3: Dunn Index of clustering results

variance between activities of the same cluster) and separate (clusters are enough far apart). A higher Dunn index indicates better clustering.

In the first experiment, we measured the Dunn index of the clustering algorithm by using Control, Data and Semantic dependencies matrices. Fig. 3a and Fig. 3b illustrate the obtained results clearly showing that the Dunn index that results from the Control dependencies node is almost always better than the Dunn index that results from both Data and Semantic dependencies nodes. This means that for a given BP, the control dependency model is richer and more informative than the other models. We also computed the Dunn index of the clustering result by aggregating the three control, data and semantic dependencies in one matrix.

The obtained results show that the clustering quality of the Control dependency node is often better than the one obtained by merging control, data and semantic dependencies. It confirms that aggregating different dependency matrices degrades the quality of the final microservice generation and the collaboration between nodes is the appropriate option.

VI. Conclusion

Automatic identification of microservices is one step allowing to migrate from monolithic systems to microservices-based systems. This paper proposed a multi-model based-approach to support this migration using BPs as an input to this identification. Each model represents dependencies between BPs' activities from one perspective (control, semantics, data, etc.). The generated data dependencies are then used by a collaborative clustering algorithm to identify groups of cohesive activities that could become potential microservices. The technical results are promising showing how our collaborative clustering algorithm outperforms the cooperative clustering algorithm in term of precision.

References

[1] Mohammad Javad Amiri. Object-aware identification of microservices. In 2018 IEEE International Conference on Services Computing (SCC), pages 253–256. IEEE, 2018.

[2] Luciano Baresi, Martin Garriga, and Alan De Renzis. Microservices identification through interface analysis. In European Conference on Service-Oriented and Cloud Computing, pages 19–33. Springer, 2017.

[3] R. Chen, S. Li, and Z. Li. From monolith to microservices: A dataflow-driven approach. In 2017 24th Asia-Pacific Software Engineering Conference (APSEC), pages 466–475. IEEE, Dec 2017.

[4] Antoine Cornuéjols, Cédric Wemmert, Pierre Gançarski, and Younès Bennani. Collaborative clustering: Why, when, what and how. Information Fusion, 39:81–95, 2018.

[5] T.H. Davenport and J.E. Short. The new industrial engineering: Information technology and business process redesign. Sloan Management Review, 1990.

[6] Ervin Djogic, Samir Ribic, and Dzenana Donko. Monolithic to microservices redesign of event driven integration platform. In 41st International Convention on Information and Communication Technology, Electronics and Microelectronics, MIPRO 2018, Opatija, Croatia, May 21-25, 2018, pages 1411–1414, 2018.

[7] Daniel Escobar, Diana Cardenas, Rolando Amarillo, Eddie Castro, Kelly Garcés, Carlos Parra, and Rubby Casallas. Towards the understanding and evolution of monolithic applications as microservices. In XLII Latin American Computing Conference, CLEI 2016, Valparaíso, Chile, October 10-14, 2016, pages 1–11, 2016.

[8] M. Estañol. Artefact-centric Business Process Models in UML: Specification and Reasoning. PhD thesis, Universitat Politècnica de Catalunya, 2016.

[9] Michael Gysel, Lukas Kölbener, Wolfgang Giersche, and Olaf Zimmermann. Service cutter: A systematic approach to service decomposition. In European Conference on Service-Oriented and Cloud Computing, pages 185–200. Springer, 2016.

[10] K.M. Hammouda and M.S. Kamel. Collaborative document clustering. In SIAM International Conference on Data Mining, 2006.

[11] Wuxia Jin, Ting Liu, Qinghua Zheng, Di Cui, and Yuanfang Cai. Functionality-oriented microservice extraction based on execution trace clustering. In 2018 IEEE International Conference on Web Services, ICWS 2018, San Francisco, CA, USA, July 2-7, 2018, pages 211–218, 2018.

[12] Fionn Murtagh and Pierre Legendre. Ward's hierarchical clustering method: Clustering criterion and agglomerative algorithm. CoRR, abs/1111.6285, 2011.

[13] N.P.A Vo, F. Guillot, and C. Privault. Disco: A system leveraging semantic search in document review. In COLING, 2016.

[14] Mathias Weske. Business Process Management: Concepts, Languages, Architectures. Springer, 2019.