Formal Approach for Generating Privacy Preserving User Requirements-Based Business Process Fragments

Mohamed Anis Zemni[1] **Amel Mammar**[2] **Nejib Ben Hadj-Alouane**[3]

[1] Ecole Nationale des Sciences de l'Informatique (ENSI), UR/OASIS
Campus Universitaire de la Manouba,
2010 Manouba, Tunisia,
Email: mohamedaniszemni@gmail.com

[2] Institut Mines-Telecom/Telecom SudParis, CNRS UMR 5157 SAMOVAR
9 Rue Charles Fourier,
91011 Evry, FRANCE,
Email: amel.mammar@telecom-sudparis.eu

[3] Ecole Nationale d'Ingenieurs de Tunis (ENIT), UR/OASIS
BP 37, Le Belvedere,
1002 Tunis, Tunisia,
Email: nejib_bha@yahoo.com

Abstract

A business process fragment is a portion of a business process, more commonly designed for reuse purposes. Fragments are intended to be declared as safe from a privacy perspective, when manipulated in an open context. Privacy is related to the authority to have a view on some sensitive information. A business process privacy-preserving fragmentation is the task of decomposing business processes into significant fragments, which can be reused in the future in order to build new business processes while preserving the sensitive information from leakage. This paper presents a design-time two-phases approach to decomposing existing business processes into significant fragments while preserving the integrity of data items that navigate within the process. The first phase is based on the so-called Formal Concept Analysis (FCA) technique handling semantic activity clustering according to designers requirements, while dealing with the privacy constraints. The second phase manipulates clusters of activities and generates ready-for-reuse fragments. Some experiments that demonstrate the feasibility of the proposed approach are also provided.

Keywords: Business Process, Fragmentation, Reuse, Privacy, Semantics, Requirements-Driven.

1 Introduction

In the industry of business process management (Ter Hofstede et al. (2003)), organizations and business entities are more and more focused on improving the quality of their services. At the same time, they experience a need to maintain a high degree of efficiency, with respect to delivery deadlines and productivity; all this, within the context of a continuously increasing competition. A key strategy consists in efficiently implementing and developing modern business processes relying on new Web technologies. A business process, as defined by Ouyang et al. (2007), consists of a set of operations, more commonly called activities, organized in a given manner so as to produce a specific service. Implementing new activities and business processes completely from scratch may be a very tedious and time consuming task. Reusing already existing business process *fragments* can reduce the business process development time and enhance its robustness (Schumm et al. (2010)). In fact, the reuse of business process fragments can be an important component of any flexible design strategy resulting in a reduction of process development periods (Markovic & Pereira (2008)). Schumm et al. (2010) cite two ways to design business process fragments: (1) from scratch (Eberle et al. (2009)) or (2) extracted from existing process models. In our work we focus on the second approach which is used to be done essentially manually. The work introduced by Schumm et al. (2011) demonstrates that, apart from improving the quality of the resulting business processes, reusing the fragments allows to avoid (i) the redesign of certain existing fragments, and (ii) the implementation and optimization of all business process artifacts from scratch.

Business process developers, however, need to pay special attention to the data privacy concerns. Indeed, nowadays, the individuals are becoming more and more concerned about the privacy of their personal data that may appear within the process boundaries. As defined by Sweeney (2002), privacy is related to the authority to receive some sensitive information, e.g., personal information. Such important information is called sensitive and should not be disclosed. Though sensitive information may occur in a business process for the purpose of performing its key functionalities, they are not intended for sharing or publishing. Consequently, the fragments that are to be reused, must be individually and conjointly safe. One has to make sure that (i) each fragment produced from a business process decomposition approach is safe, and (ii) two fragments from which sensitive information can be inferred should not be coupled together when a new business process is built.

In a previous work (Zemni et al. (2012*a*,*b*)), we have presented an initial and informal approach for business process decomposition while maintaining the privacy of sensitive information. The approach generates *any* fragment whose activities involve common functionalities. Fragments may contain superfluous activities which are not interesting for the designer. Moreover, the approach relies mainly on grouping semantically close activities while structural concerns still early stage. In this paper, we seek to improve on the fragmentation mechanism by attempting to provide requirements-driven algorithms, as well as formal definitions that make the approach formal. We also prove the correctness of the privacy-preserving mechanism. Here, a requirements-driven approach is presented in order to provide useful and reusable business process fragments. Indeed, approach relies on the fragmentation of existing business processes to retrieve fragments that may suit designers'requirements.

The structure of the paper is articulated as follows: in the next section, we give a motivating example with a detailed illustration of *reuse issues* that may occur during the fragmentation process (Section 2). Section 3 provides formal definitions for business processes and business process fragments along with *privacy* and *semantics* concerns that allow us to formalize and prove the decomposition approach. Section 4.1 introduces our requirements-driven clustering mechanism to generate *clusters* of *semantically* close activities, and deals with *privacy.* A detailed *proof* is also given in section 4.2 to show that the privacy is well respected. We then present an algorithm to generate *privacy-aware* and *reusable* fragments from the composition of the previous clusters 4.3. Section 5 evaluates the effectiveness of the proposed approach and gives the description of the most significant related statistics. We finally conclude with the related work and the results summary in sections 6 and 7, respectively.

2 Working Example

In this part, we specify in details the main issues related to the fragmentation through a practical case study. For this, we consider an inter-department collaboration scenario where a business process designer, from a given department, desires to provide relevant fragments of his own process to third designers, from another department, who needs to build a new process. Figure 1 depicts a "surgery performing" process case study.

Represented in Business Process Model and Notation (BPMN)[1], this process is roughly defined as a set of activities, represented with rounded boxes, a set of data items, represented with note boxes, a set of events, represented with circles, and, a set of gateways, represented with diamonds. Activities, gateways, and events are linked to each other with control flows, i.e., represented with solid arrows, and data items are linked to activities by means of data flows, i.e., represented with dashed arrows. Control flows depict the execution order of the elements they link, while dashed arrows represent the data items routed in between activities, i.e., as either inputs or outputs.

The process represented in figure 1 performs as follows: a surgery order triggers the process execution. Indeed, a message event, i.e., which is a start event, receives a '$surgery\ order\ number$'. Activity (a_1) uses the '$surgery\ order\ number$' to retrieve the $surgery\ information$ as well as the $patient's\ SSN$ (Social Security Number). Note that the $surgery\ information$ contains the type of surgery, whether it is urgent or not, etc. Activity (a_2) receives the $patient's\ SSN$ and retrieves the $patient's\ information.$ After that, two branches are called concurrently: (i) selecting a free surgeon (activity (a_5)) and selecting a free surgery room (activity (a_6)), and (ii) asking for patient's history (activity (a_3)) and compiling patient's record (activity (a_4)). Activities (activity (a_5)) and (activity (a_6)) use the $surgery\ information$ and check the surgeon and surgery room availabilities, while activities (activity (a_3)) and (activity (a_4)) check for surgery and history inconsistencies, e.g., whether any contraindication exists between the $surgery\ information$ and the $patient's\ record$, or if the patient take some drugs he must stop before the surgery. The process terminates immediately if any inconsistency is detected. Once both branches finish their execution, activity (a_7) confirms the patient for surgery. When the receive message event receives the $surgery\ report$, the patient's record (activity (a_8)) is updated (i.e., using the $surgery\ information$, the $patient's\ SSN$, and the $surgery\ report$), the $order\ patient\ follow\ up$ activity (activity (a_9)) is executed, and the process is ended.

Generally speaking, this process can be published in an open context to be reused as part of a new business process to perform more complex functionalities. For instance, the process may be part of a "Surgery Performing and Reimbursement" process to perform the costs reimbursement tasks by the social security in addition to the functionalities involved by the surgery performing process.

This process, however, is not necessarily *safe* while it may reveal some sensitive information compiled from data associations. For instance, the association between the data item $patient's\ information$ (activity a_2 output) and the data item $patient's\ record$ (activity a_3 output), is sensitive. Indeed, this association may lead to $patient's\ illnesses$ disclosure. In this paper, it is dealt with information, more specifically data items, that are routed between activities. Recall that information enclosed within the activities cannot be viewed, as activities are seen as black boxes and are then safe. More specifically, it is dealt with non-sensitive data items that are safe when they are considered separately, but turn out to be critical when they are associated together. Note that activities, also, are safe when they are taken separately, where for a given activity, inputs and outputs do not form sensitive associations. Indeed, it is the *association* between data items which may be *sensitive.* Consequently, when the process is coupled with malicious activities, the latter may probably make use of the data items and infer sensitive information. For example, some additional activities (or even a single one) may be added to the surgery performing process, using the $patient's\ information$ data item and the $patient's\ record$ data item manipulates them and generating the $patient's\ illnesses$.

Moreover, the process may be uninteresting for reuse, as a whole. For example, let the designer be interested only in preparing surgery and manipulating patient's record (i.e., history) within the desired "Surgery Performing and Reimbursement" process. The designer already has his own patient's follow up activities, and thus do not need them from the current process. Consequently, it would be wiser to catch only portions that semantically match the designer's requirements. Therefore, activities a_9, should not be kept for reuse. This solution, however, is intuitive, essentially manual, and needs business investigations, as for many existing decomposition approaches (as stated by Khalaf (2008)). This task also requires a

[1]http://www.omg.org/spec/BPMN/2.0/

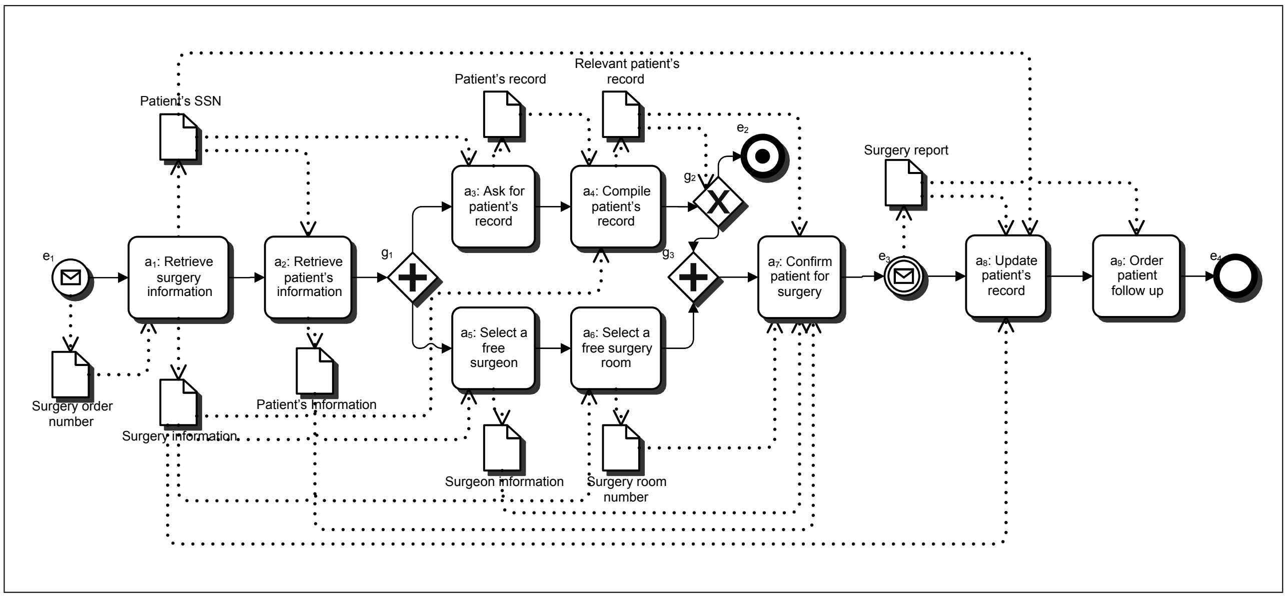

Patient's SSN
Patient's record
Relevant patient's record
e_2
g_2
Surgery report
e_1
a_1: Retrieve surgery information
a_2: Retrieve patient's information
g_1
a_3: Ask for patient's record
a_4: Compile patient's record
g_3
a_7: Confirm patient for surgery
e_3
a_8: Update patient's record
a_9: Order patient follow up
e_4
a_5: Select a free surgeon
a_6: Select a free surgery room
Surgery order number
Patient's information
Surgery information
Surgeon information
Surgery room number

Figure 1: An Example of Surgery Performing Process.

good understanding of the initial process to enable retrieving the interesting portion for the designer. This turns out to be a difficult task and thus needs automation especially to deal with big and complex processes.

3 Fragmentation Preliminaries

In this section, we present the necessary formal notions needed to understand and perform the fragmentation task, in order to generate well-formed and privacy-aware fragments whose functionalities are consistent with those required by the designer.

3.1 Business Process and Business Process Fragment Models

As stated by Mancioppi et al. (2011), a complete process model definition is an important artifact in order to enable performing the fragmentation task.

Ouyang et al. (2007) have introduced a business process (process for short) as a directed graph that is defined as a collection of activities, events, and gateways, linked with control flows. For our purposes, this definition is further refined, to clearly specify the various parts and aspects of a process, namely data elements, i.e., consisting of data items and data flows. We formally define a business process as follows:

Definition 1 (Business Process) *A Business Process is a tuple P=(O, A, G, C_f, D, D_f), where $O \subseteq (A \cup G)$ is a set of objects composed of a set of activities and events A, and a set of gateways, G. $C_f \subseteq (O \times O)$ is the control flow relation to link objects to each other, D is the set of data items handled by the activities, and $D_f \subseteq (D_{f_{in}} \cup D_{f_{out}})$ is the data flow relation to link objects to their corresponding input data items $D_{f_{in}} \subseteq (O \times D)$, and activities to their output data items $D_{f_{out}} \subseteq (A \times D)$.*

The relation $D_{f_{out}}$ does not involve gateways as they generate no data items. Note that events and activities are set in the same activity set A as they act alike. In the rest, we use "activities" instead of "activities and events".

We define a path as a linear and connected portion of a business process describing a set of sequential activities, i.e., whose execution is sequential, starting from a given activity, traversing a set of sequential activities, and leading to another activity, and where each couple of consecutive activities are linked by means of a control flow. Let Paths be the set of all possible paths defined in a process such that $Paths = \{(act_1, ..., act_n) | \bigwedge_{i=1..n} a_i \in A \wedge \forall i.(1 \leq i \leq (n-1) => (act_i, act_{i+1}) \in C_f\}$. A process should contain no unconditional control flow cycles leading the process to run indefinitely. Consequently, there should not exist any path ending with an activity it has started by.

Let the following formally defined property that each business process has to respect:

Property 1 (Unconditional Cycle Free) $\forall(a_1, ..., a_n).((a_1, ..., a_n) \in Paths \Rightarrow a_1 \neq a_n)$.

A business process fragment (fragment for short) as defined by Schumm et al. (2010) is a *connected* subprocess designed for reuse purposes. It is composed of at least one activity, and of several edges, representing control and data flows. A fragment involves some functionalities and is intended to be composed with other ones to build new processes. A fragment can depict dangling control flows, i.e., with either no source or sink activities specified. More formally:

Definition 2 (Business Process Fragment) *A Business Process Fragment is a tuple f = (O, A, G, C_f, D, D_f, fn), with O, A, G, D, and D_f as in Definition 1. $C_f \subseteq (O \cup \{\perp\})^2 - \{(\perp, \perp)\}$ is the complete and/or dangling control flow relation to link objects to each other, and fn is a set of terms that describe the fragment's involved functionalities.*

Note that character $\perp$ is used to denote dangling control flows, i.e., representing missing start or end of a control flow.

As for the case of a complete process, a fragment should contain no unconditional cycles. This property is necessary to ensure that business processes that are made of fragments also respect property 1.

Let the following connectivity property that should be ensured when generating fragments:

Property 2 (Connectivity) *A fragment f is connected iff* $\forall(a_1, a_2).(a_1, a_2 \in f.A \Rightarrow (a_1, ..., a_2) \in Paths \vee (a_2, ..., a_1) \in Paths \vee \exists a_3.((a_3, ..., a_1) \in Paths \wedge (a_3, ..., a_2) \in Paths))$.

That is, a fragment is connected if and only if each couple of activities belong to a path when they are placed on the same branch or belong to two different paths with a common activity when they are placed on different branches. The connectivity condition is necessary to ensure that when a fragment is used in a new process, all activities can be executed.

Generally speaking, a fragment denotes a complete process when it encloses no dangling control flows, i.e., $C_f \subseteq O^2$.

3.2 Privacy-Preserving Mechanism

As we have already tackled in the motivating example, when publishing a fragment, the association between some non-sensitive data items, involved in this fragment, can disclose some sensitive information. For example, "*patient's illnesses*" can be inferred from the association between the data items "*patient's information*" and "*patient's record*". The association between such couples of data items are called *sensitive* and should never be published in an open context. Therefore, data items that are involved in a sensitive association are called *in conflict.* To get over such issues, we propose the following privacy constraint definition that should be taken into account when performing the fragmentation task.

Definition 3 (Privacy Constraints) *Privacy constraints, denoted* $C_N \subseteq D^2$*, is a set of data item couples that should not figure in the same fragment.*

Consequently to the definition, *activities that output conflicting data items should never figure in the same fragment*; they are called *in conflict* with each other. To assert that two activities are conflicting, only the data they produce are taken into account. Indeed, when an activity is selected for a fragment, its input data, i.e., that are in conflict with other ones in the fragment, are not yet received and then may be substituted with other ones.

For instance, C_N = {(*surgery information*, *patient's information*), (*patient's information*, *patient's record*), (*patient's information*, *relevant patient's record*)}, in Figure 1, depicts the privacy constraint set. Then, activities a_1 and a_2, activities a_2 and a_3, and activities a_2 and a_4 are pairwise in conflict.

The proposed privacy constraints leads to the following privacy-awareness property that we have to ensure on each fragment f:

Property 3 (Privacy-awareness) *A fragment f is safe iff* $\forall(a_1, a_2).(a_1 \in f.A \wedge a_2 \in f.A \Rightarrow ((f.D_{f_{out}}[\{a_1\}]^2 \times f.D_{f_{out}}[\{a_2\}]) \cup (f.D_{f_{out}}[\{a_2\}] \times f.D_{f_{out}}[\{a_1\}])) \cap C_N = \emptyset)$.

That is, each fragment should not contain any couple of activities that output conflicting data items.

3.3 Activities to Functional Requirements Similarity

In our work, we are interested in any fragment verifying Definition 2 and whose activities verify some given functional requirements, denoted Q, in addition to privacy concerns. To this aim, we focus on any activity textual description that gives the main functionalities involved by the activities. If no activity description is provided, then, activities' labels and data items' labels are retrieved. Generally speaking, activity descriptions are made of a set of terms that describe the functionalities involved by the activity.

Let $\hat{a}$ be the textual description of activity a, and $\hat{Q}$ be the textual description of requirements Q. In our work, activity descriptions, $\hat{a}$, as well as the functional requirements, $\hat{Q}$, are pretreated (e.g., by removing stopwords (Fox (1992)), by stemming terms (Porter (1997)), and by unifying synonyms), and weighted (e.g., using term weighting mechanisms (e.g., TF/IDF[3] (Salton & Buckley (1988)))). This respectively generates weighted description vectors $\overrightarrow{a}$ and $\overrightarrow{Q}$. Let $w(t, \overrightarrow{a})$ be a function that returns the weight of term t in the activity description $\hat{a}$. Note that function, w, returns 0 if term, t, does not belong to the activity description, $\hat{a}$. For example, $\hat{a_5}$ = {*'select', 'free', 'surgeri', 'surgeri', 'inform', 'surgeri', 'inform'*} is an activity description, and $\overrightarrow{a_5}$ = ((0.105, *select*), (0.105, *free*), (0.037, *surgeri*), (0.056, *inform*)) its corresponding vector, where $w($*surgeri*$, \overrightarrow{a_5}) = 0.037$.

To include or not an activity a in a fragment, we have to compute the similarity between the activity vector $\overrightarrow{a}$ and the requirements vector $\overrightarrow{Q}$. To this aim, we use Vector Space Model (VSM), introduced bySalton et al. (1975). Let α be a similarity threshold fixed by the designer, above which, the activity would be part of the fragment. Guidelines for fixing an appropriate threshold is out of the scope of this paper.

Generally speaking, an activity may correspond to all the query terms or *only part of them.* The latter occur when the query terms involve complex functionalities which cannot all be met by a single activity. Consequently, it would be wiser considering only terms of the activity descriptions for the similarity computation instead of the union of both query and activity description terms, as set in the literature. This enables to focus on the functionalities in Q that are involved in a given activity. The similarity function is formally defined as follows.

Definition 4 (Similarity Function) *Given a functional requirement vector* $\overrightarrow{Q}$ *and an activity vector* $\overrightarrow{a}$*, the similarity between* $\overrightarrow{Q}$ *and* $\overrightarrow{a}$ *is defined by:*

$$sim(\overrightarrow{Q}, \overrightarrow{a}) = \frac{\sum_{j \in dist(\hat{a})} w(j, \overrightarrow{Q}) \times w(j, \overrightarrow{a})}{\sqrt{\sum_{j \in dist(\hat{a})} w(j, \overrightarrow{Q})^2} \times \sqrt{\sum_{j \in dist(\hat{a})} w(j, \overrightarrow{a})^2}}$$

with $dist(\hat{a})$ *is a function which returns distinct terms in* $\hat{a}$.

For instance, $sim(\overrightarrow{Q}, \overrightarrow{a_5}) =$

$$\frac{0 * 0.105 + 0 * 0.105 + 2 * 0.037 + 0 * 0.056}{\sqrt{2^2} * \sqrt{0.105^2 + 0.105^2 + 0.037^2 + 0.056^2}} = 0.36$$

where $\overrightarrow{Q}$ = ((2, *patient*), (2, *surgeri*), (2, *record*), (2, *reimburs*)) is the requirements vector. Note that the requirement terms are all weighted 2 meaning that they are as important as each other.

[2] Given the relation $R \subseteq X \times Y$, then, $R[X_1] = \{y | y \in Y \wedge \exists x.(x \in X_1 \wedge (x, y) \in R)\}$, and $R^{-1}[Y_1] = \{x | x \in X \wedge \exists y.(y \in Y_1 \wedge (x, y) \in R)\}$.

[3] TF/IDF is a term weighting method which reflects the importance of a term for a document among a corpus of documents. Higher weights are assigned to terms occurring frequently in a particular document, but rarely on the remainder of the document collection. In our work, documents represent activities.

4 Privacy-Aware Business Process Fragmentation

In this section, we propose a two-phases requirement-driven fragmentation. The first phase involves an algorithm in which we incorporate privacy constraints and semantics to generate semantically close and privacy-aware clusters of activities. A cluster of activities is a group of coupled activities that cooperate together closely to achieve the same goal. We also provide a proof demonstrating the correctness of the approach w.r.t. Property 3. The second phase involves an algorithm that derives, from clusters, reusable fragments. Indeed, clusters of activities are garnished with structural concerns (gateways, flows, etc...) so as to draw connected and reusable fragments.

4.1 Clustering Algorithm

The process clustering consists in selecting activities that are semantically close to the functional requirements and classifying them w.r.t. the functionalities they are involving, and this, while maintaining the sensitive information privacy. The clustering is based on the so-called Formal Concept Analysis (FCA) (Ganter & Wille (1999)). The latter is a data analysis technique, used for classifying similar objects within object collections, w.r.t. their common attributes. Our work adapts the FCA to the process activity clustering. That is, activities are mapped onto objects, and activity descriptions are mapped onto attributes. We extend this technique so as to compute the similarity between the activities and the required functionalities. This enables forming clusters consisting of activities that cooperate to involve the same functionalities. We also constrain the technique with the privacy constraints.

The following are the extended FCA element definitions that the clustering algorithm, Algorithm 1, relies on.

Definition 5 (Formal Context) *A formal context is a triplet, $C = (A, T, w)$ involving a set of process activities, A, a set of terms, T, that has been retrieved from the process activity descriptions, and a weight function, $w : T \times A \rightarrow \mathbb{R}$, that returns the weight of a term, $t \in T$, for an activity $a \in A$.*

Definition 6 (Clustering System) *$S = (C, D_{f_{out}}, C_N)$ is a clustering system where C is the formal context, $D_{f_{out}}$ is the output data flow relation, and C_N are the privacy constraints.*

Definition 7 (Galois Correspondence) *A Galois correspondence involves two functions, Θ and Δ, for a clustering system $S = (C, D_{f_{out}}, C_N)$ and functional requirements Q.*

$\Theta : \mathcal{P}(A) \rightarrow \mathcal{P}(T \cap Q)$ is defined over the power set $\mathcal{P}(A)$, and returns the maximal set of terms, among Q, that are shared by all the activities, where Q is the set of requirements' terms. That is, for a given $A_i \in \mathcal{P}(A)$, where $\forall(a_1, a_2).(a_1 \in A_i \land a_2 \in A_i \Rightarrow ((D_{f_{out}}[\{a_1\}] \times D_{f_{out}}[\{a_2\}]) \cup (D_{f_{out}}[\{a_2\}] \times D_{f_{out}}[\{a_1\}])) \cap C_N = \emptyset))$, then, $\Theta(A_i) = \{t \in (T \cap Q) | w(t, \overrightarrow{a}) \neq 0, \forall a \in A_i\}$.

$\Delta : \mathcal{P}(T \cap Q) \rightarrow \mathcal{P}(A)$ is defined over the power set $\mathcal{P}(T \cap Q)$, and returns relevant-enough activities (that are maximal), according to a fixed threshold α, that share all the terms in $(T \cap Q)$. That is, for a given $T_j \in \mathcal{P}(T \cap Q)$, $\Delta(T_j) = \{a \in A | sim(\overrightarrow{Q}, \overrightarrow{a}) \geq \alpha\}$.

Definition 8 (Formal Concept/Cluster) *Given a clustering system $S = (C, D_{f_{out}}, C_N)$, a formal concept is a tuple $Con = (T_j, A_i)$, where $\Theta(A_i) = T_j$, and, $\Delta(T_j) = A_i$. A_i is called a cluster made of relevant-enough activities collaborating to process the functionalities represented with T_j.*

Algorithm 1 depicts the requirement-driven privacy-aware clustering process. It takes as input parameters a clustering system S (Definition 6), and the functional requirements Q. The algorithm returns the set of formal concepts *conSet* from which clusters are derived.

Algorithm 1 Requirement-driven Privacy-aware clustering

```
1: function CLUSTERING(FormalContext $C$, Output $D_{f_{out}}$, Privacy $C_N$, Requirements $Q$):FormalConceptSet
2:     declare FormalConceptSet $conSet \leftarrow new$ FormalConceptSet()
3:     begin
4:     for all $A_i \in \mathcal{P}(C.A)$ do
5:         if $notConflicting(A_i, C_N, D_{f_{out}})$ then
6:             TermSet $T_i \leftarrow \Theta(A_i)$
7:             if $\Delta(T_i) = A_i$ then
8:                 $conSet \leftarrow conSet \cup \{new$ FormalConcept $(T_i, A_i)\}$
9:             end if
10:        end if
11:    end for
12:    return $conSet$
13:    end
14: end function
```

The formal context, depicted in definition 5, represents the clustering basis. It involves the activities to classify, the classification criteria, i.e, the terms, and the weights of terms for a given activity. Note that we have extended the relation between terms and activities from binary relations to values ones, i.e, given by the function w. The clustering system involves the process corresponding formal context, the privacy constraints it is subject to, as well as output data flows to decouple conflicting activities during the clustering task.

Given that the clustering task is activity-centric, the algorithm applies on every element of the power set $\mathcal{P}(C.A)$[4]. The algorithm uses the Galois correspondence functions (Definition 7), Θ and Δ (*lines* 6,7), that we have extended with semantic concerns. Function Θ returns a set of terms that represent the functionalities that may probably be involved by the activities (e.g., $\Theta(\{e_1, a_1, a_5, a_6\}) = \{surgeri\}$). The Θ parameter is privacy-aware as the privacy is checked when selecting the subset A_i by means of function *notConflicting* (*line* 5) which returns *true* if the activities in A_i are not conflicting given C_N. Function Δ applies over $(T \cap Q)$. This permits to focus the similarity computation on the terms in Q that may be involved by the process activities. Δ returns the activities that are close to the functional requirements Q, w.r.t. threshold α. For instance, $\Delta(\{surgeri\}) = \{e_1, a_1, a_5, a_6\}$, where activities a_4, a_7, e_2, a_9, a_8 are not returned, i.e., even if they involve the term *surgeri* in their description, as they are not relevant enough according to threshold $\alpha = 0.35$ (e.g., $sim(\overrightarrow{Q}, \overrightarrow{a_9}) = 0.114 < \alpha$). Note

[4]$C.A$ is the activity set, A, for a formal context, C. The same thing applies to similar notations.

that Δ, as it exists in the literature, would return $\{e_1, a_1, a_5, a_6, a_4, a_7, e_2, a_9, a_8\}$ for $\Delta(\{surgeri\})$.

Using functions Θ and Δ, we generate the formal concepts (Definition 8). A formal concept in our work represents a cluster of close activities, that share the same functionalities. Indeed, for a given formal concept, *all* activities involve the functionalities described by the terms in Q or part of them, and the terms are significant enough for *only* those activities according to the fixed threshold α. For instance, $(\{patient, sergeri\}, \{a_1, a_7, a_8\})$ is a formal concept, but $(\{patient, sergeri\}, \{a_1, a_4, a_7, a_8, a_9\})$[5] is not a formal concept as $\Delta(\{patient, sergeri\}) = \{a_1, a_7, a_8\}$ does not contain a_4 nor a_9.

Note that, *without* considering the privacy constraints, the clustering task would return the cluster $\{a_2, a_3, a_4, a_7, a_8\}$, i.e., involving patient's manipulation, while a_2 should not be put in the same cluster as a_3 and a_4, as explained in section 3.2.

In the following subsection, we demonstrate that Algorithm 1 verifies property 3.

4.2 Correctness of the Clustering Algorithm

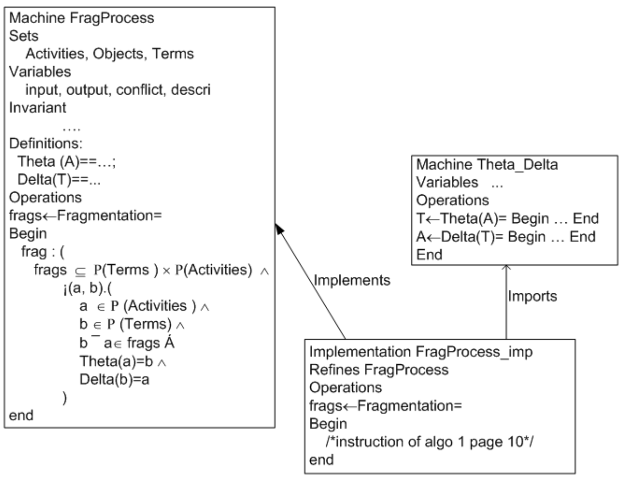
In order to validate the proposed approach, we have verified the correctness of Algorithm 1 using the B formal method and its refinement concept. Before describing how we used this method to prove the correctness of the algorithm, we give a brief introduction of the B method.

Introduced by Abrial (1996), B is a formal method for safe project development. B specifications are organized into abstract machines that encapsulate state variables on which operations are expressed. The set of the possible states of the system are described using an invariant which is a predicate in a simplified version of the ZF-set theory, enriched with many relational operators. Refinement is the process of transforming a specification into a less abstract one. In B, we distinguish behavioral and data refinement. The behavioral refinement, used in this paper, includes weakening of preconditions, the replacement of parallel substitution with a sequence one, etc. To ensure the correctness of a B specification, a set of proof obligations is generated for each B component. At the abstract level, these proofs aim at verifying that the invariant of the system is satisfied after the execution of each operation. Refinement proofs permit us to check the correctness of each concrete operation with respect to its abstraction. We assume that readers are familiar with B method and more details can be found in Abrial (1996).

To establish the correctness of Algorithm 1, we have adopted the B architecture depicted in Figure 2:

- At the abstract level, we build a B machine *FragProcess* that defines a set of types which correspond to *Activities, Objects (Data items)* and *Terms*. The process is described through 3 variables defined as follows:
    1. *output*: this function gives the data items used by an activity as output.
    2. *desc*: this function gives the set of terms corresponding to each activity (it corresponds to $\hat{a}$).
    3. *conflict*: this relation stores the sets of the data items couples that are in conflict (privacy constraints C_N).

[5]This formal concept is generated using the FCA technique as it exists in the literature



```
Machine FragProcess
Sets
   Activities, Objects, Terms
Variables
   input, output, conflict, descri
Invariant
          ....
Definitions:
  Theta (A)==...;
  Delta(T)==...
Operations
frags←Fragmentation=
Begin
   frag : (
      frags ⊆ P(Terms ) × P(Activities) ∧
         ¡(a, b).(
            a ∈ P (Activities ) ∧
            b ∈ P (Terms) ∧
            b ¯ a∈ frags Á
            Theta(a)=b ∧
            Delta(b)=a
         )
end
```

```
Machine Theta_Delta
Variables ...
Operations
T←Theta(A)= Begin … End
A←Delta(T)= Begin … End
End
```

```
Implementation FragProcess_imp
Refines FragProcess
Operations
frags←Fragmentation=
Begin
   /*instruction of algo 1 page 10*/
end
```

Figure 2: B Architecture for Algorithm 1 and its Correctness.

Finally, a B operation is specified in order to perform the fragmentation of the process. This operation is expressed in an abstract manner by giving the properties that the returned fragments should verify. Functions, Θ and Δ are declared as definitions

- According to the B refinement technique, proving that algorithm 1 is correct comes down to establish that it is a possible implementation of the previous B machine. To do that, we create a B implementation component of the previous machine in which operation $Fragmentation$, verifying the privacy awareness and maximality properties, is implemented by the instructions of algorithm 1. This implementation component imports a B machine in which function Θ and Δ are also described as operations in order to be called from the implementation component.

To validate these B components, we have generated and proved a set of proof obligations. Not surprising, components *FragProcess* and *Theta_Delta* do not generate any proof obligation because their operations do not modify their variable. Consequently, the invariant remains true. Component *FragProcess_Imp* generates 61 proof obligations : 53 have been discharged automatically where the others require the human intervention in order to help the prover find the right deduction rules to establish them. Recall that the proof obligations of this component aims at proving that the implementation of operation $Fragmentation$ is correct with respect to its abstract specification. In this way, we prove the correctness of algorithm 1.

4.3 Fragments' Structure Building Algorithm

Although we ensured the relevance and privacy-awareness of the derived clusters, the clustering phase lefts the structural concerns away. Indeed, some clusters depict disconnected structures, when reconstructed into fragments. This is not conform to property 2. For instance, the cluster $\{e_1, a_1, a_5, a_6\}$ is disconnected and would not lead to a correct fragment. Moreover, the result may further be improved by attempting to merge fragments, i.e., that do not reveal sensitive information, into coarser-grained ones. This permits to involve more complex functionalities. For instance, the cluster $\{e_1, a_1, a_5, a_6\}$, the cluster

Algorithm 2 Fragment Building

```
1:  function BUILDFRAG(FormalConceptSet $conSet$, Process $P$, Privacy $C_N$):FragmentSet
2:      declare FragmentSet $F \leftarrow new$ FragmentSet()
3:      begin
4:      for all $(superT, superCl) \in \{(T,A)|\exists((T_1,A_1),(T_2,A_2)).(\{(T_1,A_1),(T_2,A_2)\} \subseteq conSet \wedge (\forall(a_1,a_2).(a_1 \in A_1 \wedge a_2 \in A_2 \Rightarrow notConflicting(\{a_1,a_2\},C_N))) \wedge (A_1 \cup A_2) \subseteq A \wedge (T_1 \cup T_2) \subseteq T)\}$ do
5:          Fragment $f \leftarrow new$ Fragment()
            //Insert activities and data elements
6:          $f.A \leftarrow superCl$
7:          $f.D \leftarrow P.D_{f_{in}}[superCl] \cup P.D_{f_{out}}[superCl]$
8:          $f.D_{f_{in}} \leftarrow superCl \lhd P.D_{f_{in}}$
9:          $f.D_{f_{out}} \leftarrow superCl \lhd P.D_{f_{out}}$
            //Insert gateways and the corresponding input data elements
10:         $f.G \leftarrow ran((superCl \lhd C_f) \rhd G) \cup dom((G \lhd C_f) \rhd superCl)$[6]
11:         $f.D \leftarrow f.D \cup P.D_{f_{in}}[f.G]$
12:         $f.D_{f_{in}} \leftarrow f.D_{f_{in}} \cup f.G \lhd P.D_{f_{in}}$
13:         $f.O \leftarrow f.A \cup f.G$
14:         $f.D_f \leftarrow f.D_{f_{in}} \cup f.D_{f_{out}}$
15:         $f.C_f \leftarrow f.O \lhd (P.C_f \rhd f.O)$//Insert complete control flows
16:         $f.C_f \leftarrow (f.C_f^{-1}[\emptyset] * \bot) \cup (\bot * f.C_f[\emptyset])$//Insert dangling control flows
17:         $f.fn \leftarrow superT$//Insert fragment's functionalities
18:         $F \leftarrow F \cup splitIntoConnected(f)$
19:     end for
20:     return $F$
21:     end
22: end function
```

$\{a_1, a_7, a_8\}$, and the cluster $\{a_3, a_4, a_7, a_8\}$, that were generated from the clustering phase, can be melted together in a single cluster to involve both surgery preparation and patient's record manipulation.

Algorithm 2, *buildFrag*, takes as parameters the input process, P, the formal concepts, *conSet*, that are generated during phase 1, and the privacy-constraints C_N. The algorithm returns a set of connected and privacy-aware fragments.

The algorithm applies on each super-formal concept that is made of the union of formal concepts whose cluster activities are not conflicting (*line* 4). For example, given two formal concepts (T_1, A_1) and (T_2, A_2), where activities from A_1 are not conflicting with activities from A_2, their super-formal concept is $(T_1 \cup T_2, A_1 \cup A_2)$. The merge aims at assembling clusters involving different functionalities in order to provide coarser-grained fragments making easier their integration into new processes. A super-cluster corresponds to a connected fragment when all its activities can be connected. It corresponds to multiple connected fragments otherwise. For instance, $\{e_1, a_1, a_3, a_4, a_5, a_6, a_7, a_8\}$ is the super-cluster made from clusters $\{e_1, a_1, a_5, a_6\}$, $\{a_1, a_7, a_8\}$, and $\{a_3, a_4, a_7, a_8\}$.

Given a super-cluster, the algorithm builds a new fragment consisting of the super-cluster activities. The latter are garnished with the corresponding data elements, i.e., namely data items (either inputs or outputs), and data flows (*lines* 5-9).

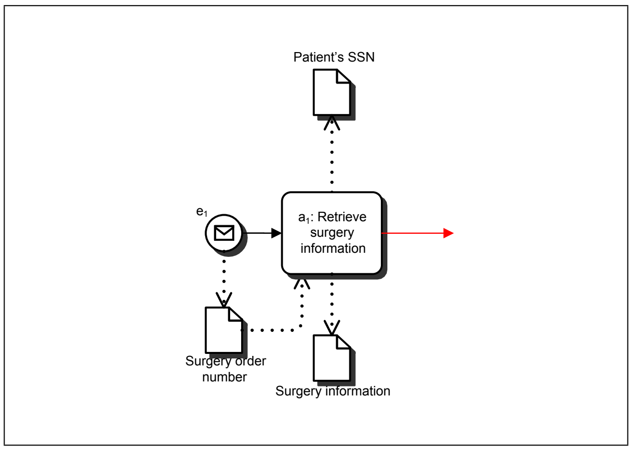


Figure 3: Example 1 of a Connected Fragment Generated from the Super-Cluster $\{e_1, a_1, a_3, a_4, a_5, a_6, a_7, a_8\}$.

Next, the algorithm inserts the gateways that are linked to the selected activities, i.e., in the initial process. Gateways' data elements are also picked and reported in the new fragment (*lines* 10-12). After that, the algorithm draws the control flows. Indeed, complete control flows are imported from the original process P, and this, when both source objects and sink objects belong to the fragment's objects, $f.O$ (*line* 15). Control flows, that were broken during the clustering phase, are replaced with dangling ones (*line* 16). Dangling control flows represent gluing points regarding the new process.

The algorithm, then, tags each fragment with the terms, *superT*, describing the fragment involved functionalities (*line* 17).

Finally, the algorithm uses function *splitIntoConnected*, according to property 2, in order to split the fragment f into multiple connected fragments when activities cannot be connected to each other. It keeps the fragment's structure as it is otherwise. Function *splitIntoConnected* is not detailed within this paper. Indeed, it is straightforward to check whether a cluster is connected or not respectively to property 2, e.g., using existing tree navigation algorithms (Nuutila & Soisalon-Soininen (1994)).

Note that the generated fragments also fulfill the property 1. Indeed, given that processes are assumed respecting such property and fragments are portions of those processes then fragments respect property 1, too.

Figure 3 and figure 4 illustrate generated fragment examples from the super-cluster $\{e_1, a_1, a_3, a_4, a_5, a_6, a_7, a_8\}$. According to algorithm 2, the fragments deal with both surgery preparation and patient's record manipulation. Note that the fragment in figure 3 involves, in fact, only surgery preparation. Assigning the specific functionalities to the fragments will be handled in future works. The privacy-preserving is well preserved, w.r.t property 3 as clusters whose output activities may form sensitive associations are dissociated. The activities of both fragment examples are semantically relevant enough for parts of the functional requirements Q. Note that the *reimburs* term, i.e., part of the requirements Q, are left away as there are no activities involving such functionality. Moreover, the fragment structure is correct respectively to definition 2 and dangling

[6]Given the relation $R \subseteq X \times X'$, $dom(R) = \{x | x \in X \wedge \exists x'.(x' \in X' \wedge x \mapsto x' \in R)\}$ and $ran(R) = \{x' | x' \in X' \wedge \exists x.(x \in X \wedge x \mapsto x' \in R)\}$

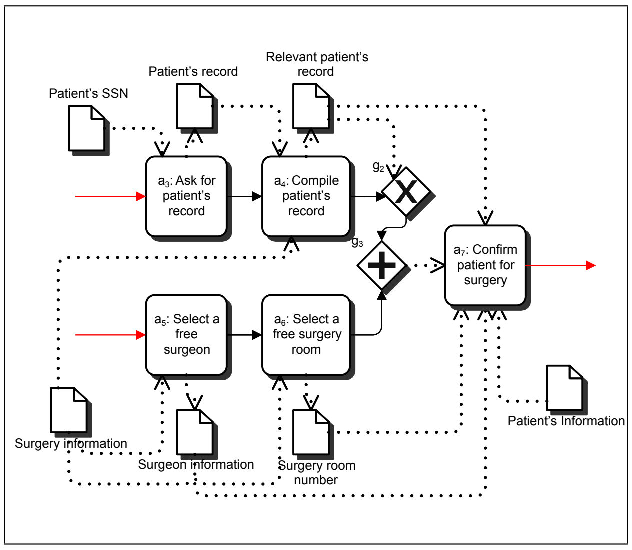

Patient's SSN
Patient's record
Relevant patient's record
a_3: Ask for patient's record
a_4: Compile patient's record
g_2
g_3
a_7: Confirm patient for surgery
a_5: Select a free surgeon
a_6: Select a free surgery room
Surgery information
Surgeon information
Surgery room number
Patient's Information

Figure 4: Example 2 of a Connected Fragment Generated from the Super-Cluster $\{e_1, a_1, a_3, a_4, a_5, a_6, a_7, a_8\}$.

control flows play gluing points in the new process. Therefore, the generated fragments can be reused in building new business processes.

5 Implementation and Results

We have implemented a tool that corresponds to the proposed algorithms. Our approach plays on existing FCA mechanisms adaptability. Indeed, a Colibri-java library[7], for classical Formal Concept Analysis, has been extended with a prototypical coding of the semantics and privacy extensions. The tool receives as input an XML document containing the activity descriptions of each selected process, an XML document containing the process organization (as presented in definition 1), and the set of privacy constraints. It produces a set of privacy-aware fragments as presented in definition 2. We have implemented activity description preparation classes based on Fox (1992) and Porter (1997), respectively for removing stopwords and stemming terms, as well as a weighting class implementing TF/IDF Salton & Buckley (1988) weighting mechanism.

To evaluate our decomposition algorithm, we have directed the tests to check (i) the granularity of the resulted fragments and (ii) the scalability of the algorithm. To evaluate the granularity of the resulted fragments, we ran the algorithm over a set of 5 processes from SAP library (Industry Specific Business) while varying the value of the similarity threshold α and counting the average number of activities within the resulted fragments. We took a set of 3 terms, having the same weight, for the functional requirements $\overrightarrow{Q}$. The decomposition of each process leads to 2 fragments, on average. The statistical results are depicted in table 1.

Note that more the similarity threshold increases more the granularity decreases. This means that some activities were removed as their similarity were below α. The similarity may be further improved when activities have rich description. The granularity is not high compared to the size of the processes which reflects the weak activities interdependency. This may

[7]http://code.google.com/p/colibri-java/source/browse/#svn%2Ftrunk%2Fcolibri

	size	0	0.1	0.2	0.3	0.4	0.5
P_1	10	3.5	2.5	1.5	1	1	0
P_2	20	3.25	3.25	3	2.66	2	1.5
P_3	9	3.5	3.5	2.25	1.75	1.33	1
P_4	5	1.5	1.5	1.5	1.5	1	1
P_5	7	2.33	2.33	1.75	2	1	0

	size	0.6	0.7	0.8	0.9	1
P_1	10	0	0	0	0	0
P_2	20	1.5	1	0	0	0
P_3	9	0	0	0	0	0
P_4	5	1	1	0	0	0
P_5	7	0	0	0	0	0

Table 1: Fragments Granularity.

be improved by considering some terms similar, e.g., drug, medicine.

We finally evaluated the scalability of our decomposition algorithm. The tests were conducted on a laptop with Core Intel i5 processor, (2.27GHz*2), 4 GB memory, running Microsoft Seven. We have tested the execution on 3 processes: the first one containing 270 activities performs in 1.605 seconds, the second process containing 90 activities performs in 0.875 seconds, and the third process containing 3 activities performs in 0.531 seconds. Thus, the algorithm can be performed with big processes with a fair execution time.

6 Related Work

Several approaches have been proposed in the area of reusing business process fragments. Nevertheless, it is not well explored when it is about automatically retrieving reusable fragments and even less when it comes to ensure the privacy of sensitive information that may be inferred intentionally or not. In the following, we present some existing approaches related to business process decomposition and privacy-preserving in business process reuse.

Huang et al. (2010) propose a workflow decomposition mechanism for reuse purposes. Their technique aims to provide reusable fragments in a bottom-up fashion. Following Huang et al. (2010), processes are organized into a hierarchy of reusable fragments. The approach then computes the interdependence between each fragment activities. This approach, however, lacks semantics where it manipulates the co-occurrence of activity pairs enclosed within each fragment.

In the same thinking, Rosa et al. (2010) have proposed an approach for merging a set of processes in a pairwise fashion. The merge consists in capturing the common connected activity regions of the initial processes and adds independent parts. This provides a unique version of multiple processes that share some elements. Common regions may be retrieved and reused as part of new processes. While regions structurally fit the fragment's definition, they may however enclose irrelevant elements for the designer. Business process patterns are addressed by La Rosa et al. (2011) to reduce the complexity of business process structures. Four out of twelve proposed patterns foster the reusability of the reduced portions. This approach is also essentially structure-centric and does not provide semantics grouping concerns.

Smirnov et al. (2011) provides a semantic approach to abstract business process models into high level views. This consists in generating coarse-grained activities, a.k.a. clusters, sharing the same property values over several property types. The proposed approach is based on a binary VSM handling property values, i.e., only property types are weighted. Our approach permits such manipulations. Moreover, our approach assigns a weight for each property value making the similarity computation more signification for activities.

Furthermore, Huang et al. (2010), Rosa et al. (2010), La Rosa et al. (2011), Smirnov et al. (2011) do not address any privacy-preserving mechanism. Indeed, sensitive information may freely be inferred by malicious activities that fragments are linked to.

Khalaf & Leymann (2012) and Khalaf & Leymann (2006) handle business process partitioning in order to assign some given functionalities to outside partners. The partitioning is made in such a way to maintain the behavior of the original process. Nevertheless, the partitioning is mainly performed manually. Indeed, the partitioning task follows fixing the corresponding activities for each partner.

The work presented by Ivanovic et al. (2010), proposes a fragment identification approach for outsourcing portions of a business process, while dealing with predefined privacy policies. The latter are used to restrict access to certain information to third parties. The decomposition, while it ensures the privacy of sensitive information, focuses only on the information routed between activities. The generated fragments do not involve semantics features to enable reusing them later.

7 Conclusion

We have presented a novel approach in order to provide useful, privacy-aware and reusable fragments. This is ensured by the proposed process decomposition mechanism that (i) performs according to the designer needs (requirement-driven), and (ii) takes into account privacy constraints avoiding sensitive information inferences. Furthermore, fragments are reusable and may easily be integrated into new processes as they depict a connected structure. The integration is enabled as fragments have gluing points consisting of dangling control flows.

Further improvements can be added on similarity computation. This can be achieved using ontologies. Our future work is directed towards this axis.

References

Abrial, J. R. (1996), *The B-Book: Assigning Programs to Meanings*, Cambridge University Press.

Eberle, H., Unger, T. & Leymann, F. (2009), Process fragments, *in* 'On the Move to Meaningful Internet Systems: OTM 2009', Vol. 5870 of *Lecture Notes in Computer Science*, Springer, pp. 398–405.

Fox, C. (1992), 'Lexical analysis and stoplists', *Information Retrieval: Data Structures & Algorithms* pp. 102–130.

Ganter, B. & Wille, R. (1999), *Formal concept analysis - mathematical foundations*, Springer.

Huang, Z., Huai, J., Liu, X. & Zhu, J. (2010), Business process decomposition based on service relevance mining, *in* 'Web Intelligence', pp. 573–580.

Ivanovic, D., Carro, M. & Hermenegildo, M. V. (2010), Automatic fragment identification in workflows based on sharing analysis, *in* 'ICSOC', Vol. 6470 of *Lecture Notes in Computer Science*, pp. 350–364.

Khalaf, R. & Leymann, F. (2006), Role-based decomposition of business processes using bpel, *in* 'ICWS', pp. 770–780.

Khalaf, R. & Leymann, F. (2012), 'Coordination for fragmented loops and scopes in a distributed business process', *Information Systems* pp. 593–610.

Khalaf, R. Y. (2008), Supporting business process fragmentation while maintaining operational semantics: a BPEL perspective, PhD thesis, Institute of Architecture of Application Systems, University of Stuttgart.

La Rosa, M., Wohed, P., Mendling, J., ter Hofstede, A. H., Reijers, H. A. & van der Aalst, W. M. (2011), 'Managing process model complexity via abstract syntax modifications', *Industrial Informatics, IEEE Transactions on* **7**(4), 614–629.

Mancioppi, M., Danylevych, O., Karastoyanova, D. & Leymann, F. (2011), Towards classification criteria for process fragmentation techniques, *in* 'Business Process Management Workshops', pp. 1–12.

Markovic, I. & Pereira, A. (2008), Towards a formal framework for reuse in business process modeling, *in* 'Business Process Management Workshops', Springer, pp. 484–495.

Nuutila, E. & Soisalon-Soininen, E. (1994), 'On finding the strongly connected components in a directed graph', *Information Processing Letters* **49**(1), 9–14.

Ouyang, C., Dumas, M., Ter Hofstede, A. & Van Der Aalst, W. (2007), 'Pattern-based translation of bpmn process models to bpel web services', *International Journal of Web Services Research (JWSR)* **5**(1), 42–62.

Porter, M. F. (1997), An algorithm for suffix stripping, *in* K. Sparck Jones & P. Willett, eds, 'Readings in information retrieval', Morgan Kaufmann Publishers Inc., pp. 313–316.

Rosa, M. L., Dumas, M., Uba, R. & Dijkman, R. M. (2010), Merging business process models., *in* 'OTM Conferences (1)', pp. 96–113.

Salton, G. & Buckley, C. (1988), 'Term-weighting approaches in automatic text retrieval', *Information processing & management* **24**(5), 513–523.

Salton, G., Wong, A. & Yang, C. (1975), 'A vector space model for automatic indexing', *Communications of the ACM* **18**(11), 613–620.

Schumm, D., Karastoyanova, D., Kopp, O., Leymann, F., Sonntag, M. & Strauch, S. (2011), 'Process fragment libraries for easier and faster development of process-based applications', *Journal of Systems Integration* **2**(1), 39–55.

Schumm, D., Leymann, F., Ma, Z., Scheibler, T. & Strauch, S. (2010), 'Integrating compliance into business processes: Process fragments as reusable compliance controls', *Proceedings of the Multikonferenz Wirtschaftsinformatik (MKWI10)* .

Smirnov, S., Reijers, H. A. & Weske, M. (2011), A semantic approach for business process model abstraction, *in* 'CAiSE', pp. 497–511.

Sweeney, L. (2002), 'k-anonymity: A model for protecting privacy', *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* **10**(5), 557–570.

Ter Hofstede, A., van der Aalst, W. & Weske, M. (2003), 'Business process management: A survey', *Business Process Management* **2678**, 1019–1019.

Zemni, M. A., Hadj-Alouane, N. B. & Yeddes, M. (2012*a*), An approach for producing privacy-aware reusable business process fragments, *in* 'ICWS', pp. 659–661.

Zemni, M. A., Hadj-Alouane, N. B. & Yeddes, M. (2012*b*), A semantics-based privacy-aware approach for fragmenting business processes, *in* 'MVDA'.