Graph-Based Business Process Model Refactoring

María Fernández-Ropero, Ricardo Pérez-Castillo and Mario Piattini

Instituto de Tecnologías y Sistemas de la Información, University of Castilla-La Mancha
Paseo de la Universidad 4, 13071
Ciudad Real, Spain
+34 926295300 Ext. 96697

[MariaS.Fernandez | Ricardo.PdelCastillo |
Mario.Piattini]@uclm.es

Abstract. Companies are ever more interested in process-oriented organizational designs due to the competitive advantages they can achieve. Companies must therefore be able to manage their business process models and deal with quality assurance concerns, especially when business process models are mined by reverse engineering (e.g. from information systems) since it has harmful effects on the quality. For example, non-relevant and fine-grained elements or incomplete processes can be obtained. Refactoring techniques have become a widely-used method to mitigate those effects, changing the internal structure of business process models without altering its external behavior. Business process models can be transformed into graph structures since it has been proved as a more efficient way to manage information. This work presents IBUPROFEN, a set of graph-based refactoring algorithms to improve the quality of business process models. This paper demonstrates its feasibility by conducting a case study using a set of industrial business process models.

Keywords: Business process model, refactoring, graph-based, understandability, modifiability, case study.

1 INTRODUCTION

Business processes depict the set of coordinated activities that companies have to conduct to achieve their common business goals [1]. Business processes are often represented by graph models following standard notations such as BPMN (Business Process Modeling and Notation) [2]. In these graph representations, business activities or tasks are considered as nodes and sequence flows between these tasks as edges. These standard representations provides companies with a mean to manage their business processes [3], i.e., analyze, execute and adapt their business process in an effective way. In fact, the appropriate management of business processes led to competitive advantages [4].

Unfortunately, business process models (as abstract descriptions) become misaligned regarding business processes that are daily, actually executed. This is due to daily operations change faster than business process models. In turn, this is owing to the fact that IT technologies and enterprise information systems evolve over time by adding new functionalities and operations that are not updated in business process models [5]. As a consequence, organizations are increasingly interested in quality assurance of business process representations and models they own.

There are several quality assurance techniques to achieve business process models with the appropriate quality levels. Business process mining techniques [6] are employed to obtain business process models from execution logs. Similarly, business process archeology [7] analyzes existing artifacts such as source code or databases for discovering and retrieving business process models in line with actual operation. Repairing techniques are devoted to add missing parts and correct business process models to fit them to the reality [8]. A part from all these techniques, one of the most applied and well-proven technique is business process model refactoring [9], which change the internal structure of business process models without altering or modifying their external behavior, and therefore, improving the understandability and modifiability among other quality features.

Despite standard notations such as BPMN are graph-based, most business process model refactoring techniques [10-12], hardly ever are designed as algorithms that manage graphs. Instead, most refactoring techniques consider, for example, business processes as two isolated, linear sets of business tasks and sequence flows. This design decision probably is better for the effectiveness of the refactoring algorithms but has harmful effects in terms of efficiency. This means that non-graph-based algorithms could have time-consuming problems when face with large, complex business process models. In fact, the usage of graph in different contexts [13-15] proved to be much more efficient than any other data structure. Due to this fact, this paper proposes IBUPROFEN, a business process refactoring approach based on graphs. IBUPROFEN defines a set of algorithms that are grouped into three categories according to the quality assurance challenge that address: maximization of relevant elements, reduction of fine-grained granularity and completeness. This paper depicts how business process models are managed as graphs and how are refactored according to the set of graph-based algorithms proposed in IBUPROFEN. This paper illustrates the usage of IBUPROFEN by means of a case study involving business process models obtained from real-life information systems, some of which are around 255 nodes and 512 edges.

The remainder of this paper is organized as follows: Section 2 summarizes related work; Section 3 introduces IBUPROFEN, the business process refactoring approach. Hence, their graph-based refactoring algorithms are shown as well as their implementation; Section 4 presents the application of the proposal with real-life business process models. Finally, Section 5 discusses conclusions and future work.

2 RELATED WORKS

There are various approaches in literature which deal with business process model refactoring by using different data structures. *Dijkman et al.* [10] identify refactoring opportunities in process model repositories. In order to identify similar parts in two different process models, these authors decompose both process models into smaller parts. They use a Refined Process Structure Tree (RPST) where the smaller parts of the decomposition are connected sub-graphs, such that control enters the fragment via a single entry node and leaves the fragment via a single exit node. The RPST defines a unique decomposition, i.e., a hierarchy of fragments that can be computed in linear time. The main difference of this work with our approach is that they use hierarchical structures and graphs in combination.

La Rosa et al. [12] provide a set of modularization of business process models based on graphs. This approach proposes, for example, horizontal modularization by obtaining sub-graphs of nearly equal size while keeping the number of cut edges low. However, not all the modularization and refactoring algorithms are based on graphs.

Similarly, the *Proviado* approach [16] applies a combination of graph reduction (Omission) and graph aggregation (Collapse) techniques to obtain customized process views based on a user query. The main difference of this work is that deal with visualization more than refactoring.

Weber et al. [11] enable designers to extract a process fragment into a sub-process to remove model redundancies, to foster reuse, and to reduce model size. Although this approach extracts fragments as sub-graphs, not all the refactoring algorithms proposed by these authors are based on graphs.

Finally, *Hauser et al.* [17] propose a process graph model to represent business process models as graphs and transform these graphs into executable code following the model-driven engineering principles. Unfortunately, the process graph model has not been used with refactoring purposes.

3 IBUPROFEN

IBUPROFEN (*Improvement and BUsiness Process Refactoring OF Embedded Noise*) addresses business process model refactoring. This technique has been especially designed for business process models represented according to the BPMN and mined by reverse engineering. IBUPROFEN allows applying different graph-based refactoring algorithms in order to address some of the challenges that involve this kind of business process models. For example, incompleteness is an important challenge to cope with since data can be distributed in several sources. Moreover, different types of granularity are a challenge to address because fine-grained granularity causes the quality degree is lower. Non-relevant information also causes a low degree quality since the model should not contain additional elements that do not perform any business logic in the organization.

With the aim to carry out the refactoring, business process models according BPMN are managed as graphs. Once business process models are represented as

graphs, a set of ten refactoring algorithms are performed. These ten refactoring algorithms supported by IBUPROFEN are divided into three categories regarding their purpose: maximization of relevant elements, fine-grained granularity reduction and completeness. For example, Fig. 1 shows one belonging to the first category, removing unnecessary elements. In that case, the gateway (that represents a decision node in BPMN) is removed since there are not nodes to choose, only one node (Task 2) can be executed after Task 1. Fig. 2 shows one belonging to the last category, completing the model. In that case, decision nodes (represented by gateways in BPMN) are added in incoming and outgoing branches. The diamond shape with the cross (exclusive gateway) represents that only one of the branches can be taken. The rest of refactoring algorithms can be consulted in [18].

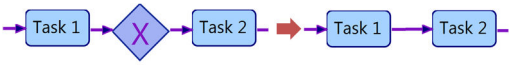

Fig. 1. Removing unnecessary nesting

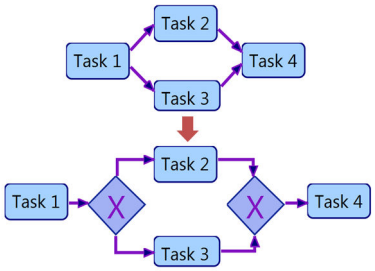

Fig. 2. Adding decision nodes between incoming and outgoing branches

IBUPROFEN is supported by a tool developed as an Eclipse™ plug-in (available in [19]). The supporting tool can therefore be used in combination with other Eclipse™ plug-ins aimed at obtaining business process models, e.g., from the source code of existing information systems. IBUPROFEN uses JGraphT [20] to manipulate graphs easily. JGraphT is a free Java graph library that provides mathematical graph-theory objects and algorithms. JGraphT supports various types of graphs such as, for example, directed graphs that are used in IBUPROFEN. Additionally, the BPMN file is read through the Jdom [21], library responsible for handling XML files via Java code.

The next sub-sections explain in detail the transformation between business process models according BPMN and graphs, as well as the implementation of each category of refactoring algorithms and the transformation from graphs to BPMN.

3.1 Transformation BPMN to Graphs

Each business process model is transformed into a directed weighted graph (`DefaultDirectedWeightedGraph<V,E>`), a non-simple directed graph in which

multiple edges between any two vertices are not permitted, but loops are. In our case, vertices (`V`) are modeled using the `BPElement` class while edges (`E`) are modeled using the `BPEdge` class. Both classes implement the `Cloneable` interface. `BPElement` saves the information about a BPMN element such as a task, data object, gateways and events. `BPEdge`, in turn, saves the information about an edge in the business process model such as a sequence flow and an association flow. The information that is saved by these classes is the name, the type, an identifier, among other. In case of events and gateways, nodes save the subtype of this type of node, i.e., start, intermediate and end for events and exclusive, parallel, inclusive and complex for gateways. Compounded tasks are, in turn, directed weighted graphs.

The set of business process models mined from existing information systems is transformed therefore to a list of graphs. Each graph represents one business process model and has several `BPElement` instances connected by means of several `BPEdge` instances. Thus, a business process model is represented as a graph through the notation G= (V, E), being v_1, $v_2 \epsilon$ V (nodes) and e ϵ E (edge), the edge between these nodes e = (v_1, v_2). For example, the connection between a task t1 and task t2 (sequence flow) is represented as follow: e = (v_1, v_2), e.type=SEQUENCE, e.name= "t1→t2", v_1.type=TASK, v_1.name=t1, v_2.type=TASK, v_2.name=t2.

3.2 Graph-based refactoring algorithms

IBUPROFEN provides a set of ten refactoring algorithms grouped into three categories: maximization of relevant elements, fine-grained granularity reduction and completeness [18]. The next paragraphs explain in detail the implementation of each refactoring algorithms belonging to each of the categories.

- **Maximization of relevant elements**.

This category groups the refactoring algorithms responsible for removing non-relevant elements found in business process models; these include isolated tasks, sheet tasks and inconsistencies. Moreover, nested gateways can bring about an increase in the complexity of business process models, so these are replaced by equivalent, light-weight structures. The refactoring algorithms are the following:

R1. Remove Isolated Nodes: This refactoring algorithm removes nodes (i.e., tasks, gateways or events) in the business process model that are not connected with any other node in that model, i.e., nodes without incoming and outgoing edges. Algorithm 1 illustrates this refactoring.

Algorithm 1: Remove Isolated Nodes.

Given G = (V, E)

$\forall$ a ϵ V

if $\neg\exists$ e ϵ E : (e = (b, a) $\vee$ e = (a, b), b ϵ V) then

V – {a}

R2. Remove Sheet Nodes: This refactoring algorithm removes elements in the business process model that are considered sheet nodes. These nodes can be gateways or intermediate events that have no successor nodes, i.e., nodes without outgoing edges in the graph. Algorithm 2 illustrates this refactoring.

Algorithm 2: Remove Sheet Nodes.

Given G = (V, E)

$\forall$ a ϵ V : a.type = INTERMEDIATE $\vee$ a.type = GATEWAY

if $\neg\exists$ e ϵ E : e = (a, b), b ϵ V then

V – {a}

R3. Merge nesting: This refactoring algorithm merges consecutive gateways of the same type (i.e., all gateways are exclusive or all are inclusive or parallel and so on). The merging is performed when the first gateway has only one output and the second has only one input. Algorithm 3 illustrates this refactoring algorithm.

Algorithm 3: Merge Nesting.

Given G = (V, E)

gatewaysToMerge $\leftarrow$ Ø

$\forall$ a ϵ V : a.type = GATEWAY

$\forall$ b ϵ V : □! e1 ϵ E : e_1 = (a, b)

if b.type = GATEWAY $\wedge$ □! e_2 ϵ E : v_2 = (b, a) $\wedge$ a.subType = b.subType then gatewaysToMerge $\leftarrow$ {(a,b)}

$\forall$ (g_1,g_2) ϵ gatewaysToMerge

$\forall$ e ϵ E : e = (g_2, a)

E $\leftarrow$ {e' = (g_1,a)}, E – {e}, V – {g_2}

R4. Remove Inconsistencies: This refactoring algorithm removes sequence flows in the business process model that are considered inconsistent. When two tasks are connected through a cut node, such as an intermediate event or a gateway, and through a direct sequence flow, this sequence flow is removed while the most restrictive path is maintained. Algorithm 4 illustrates this refactoring.

Algorithm 4: Remove Inconsistent Paths.

Given G = (V, E)

$\forall$ a ϵ V : a.type = GATEWAY $\vee$ a.type = INTERMEDIATE

$\forall$ e_s ϵ E : e_s = (a, a_s), a_s ϵ V

$\forall$ e_p ϵ E : e_p = (a_p, a), a_p ϵ V

if □ e ϵ E : e = (a_p, as) then

E – {e_s}, E – {e_p}, V – {a}

R5. Remove unnecessary nesting: This refactoring algorithm was shown in Fig. 1. It removes gateways that connect only two nodes, i.e. with one input and one output. This gateway is removed and a direct sequence flow is created between these nodes. Algorithm 5 illustrates this refactoring.

Algorithm 5: Remove unnecessary nesting.

Given G = (V, E)

$\forall$ a ϵ V : a.type = GATEWAY

if $\exists!e_s \epsilon E : e_s = (a, a_s) \wedge \exists!e_p \epsilon E : e_p = (a_p, a), a_s, a_p \epsilon V$ then

$E - \{e_s\}, E - \{ep\}, V - \{a\}, E \leftarrow \{e' = (a_p, a_s)\}$

- **Fine-grained granularity reduction**

The different granularity of business tasks and callable units in existing information systems constitutes another important challenge [22]. According to the approach proposed by [23], each callable unit in an information systems is seen as a candidate business task. However, existing systems typically contain thousands of callable units, some of which are large ones supporting the main business functionalities of the system, while many are very small and do not support any business activity directly. In other situations, a set of small callable units together support a business activity. This means that this category provides two refactoring algorithms to deal with large sets of fine-grained business tasks and data objects:

R6. Create compound tasks: This refactoring algorithm transforms each task into a compound task when this task has several subsequent tasks, which are in turn connected by a round-trip sequence flow to the task. This scenario comes about because each callable unit is transformed into a task during the reverse engineering stage when a given callable unit can invoke another callable unit, returning a value to the first one. In this case, a compound task is created with a start and end event connected with each subsequent task through the respective split and join exclusive gateways. Algorithm 6 illustrates this refactoring algorithm.

Algorithm 6: Create compound tasks.

Given G = (V, E)

$\forall$ a ϵ V : a.type = TASK

children $\leftarrow \emptyset$

$\forall$ b ϵ V : b.type = TASK

if $\exists$ $e_1 \epsilon E : e_1 = (a, b) \wedge \exists! e_2 \epsilon E : e_2 = (b, a)$ then

children $\leftarrow$ b

$V - \{b\}, E - \{ e_1\}, E - \{ e_2\},$

G' = (V', E')

V' $\leftarrow \{a_1, a_2, a_3, a_4\}$, a_1.type = START, a_2.type = END, a_3.type = COMPLEX, a_4.type = COMPLEX

E’ ← {m_1,m_2,m_3,m_4}, m_1.type = SEQUENCE, m_2.type = SEQUENCE, m_3.type = SEQUENCE, m_4.type = SEQUENCE, m_1 = (a_1,a_3), m_2 = (a_4, a_2)

∀ c ϵ children

E’← {m’,m’’}, m’.type = SEQUENCE, m’ = (a_3,c), m’’.type = SEQUENCE, m’’ = (c, a_4)

a.type = COMPOUND, a.subGraph = G’

R7. Combine data objects: This refactoring algorithm combines data objects that are input and/or output of a task. The combination is possible when those data objects are used exclusively (written or read) for that task. The combination is done when the number of data objects is above a threshold. To mitigate the collateral semantic loss, all the names of the grouped data objects are saved in the documentation attribute provided by the BPMN standard. Algorithm 7 illustrates this refactoring algorithm.

Algorithm 7: Combine data objects.

Given G = (V, E)

∀ a ϵ V : a.type = TASK

dataWrite ← Ø , dataRead ← Ø

∀ e ϵ E : e = (a, b), b ϵ V, b.type = DATA

if □! e ϵ E: e = (c, b) ∧ c=a then

dataWrite ← {b}

V – {b}, E – {e}

∀ e ϵ E : e = (b, a), b ϵ V, b.type = DATA

if □! e ϵ E: e = (b, c) ∧ c=a then

dataRead ← {b}

V – {b}, E – {e}

V ←{d_w}, d_w.type = DATA

∀ d_1 ϵ dataWrite

d_w.additionalInfo ← d_1.name

E ← {e_w}, e_w.type = SEQUENCE, e_w = (a, d_w)

V ←{d_r}, d_r.type = DATA

∀ d_2 ϵ dataRead

d_r.additionalInfo ←{d_2.name}

E ← {e_r = (d_r, a)}, e_r.type = SEQUENCE

- **Completeness**

Any reverse engineering technique implies an increase in the degree of abstraction, and therefore there is a semantic loss. This category is provided for that reason, to deal with semantic loss by means of the incorporation of additional elements that are not been retrieved in the reverse engineering phase. The refactoring algorithms are the following:

R8. Join Start and End events: This refactoring algorithm joins the start and end event to the starting and ending tasks, respectively. These events are created whether or not they were created before. When there are several starting tasks, the algorithm adds a split complex gateway between the start event and starting tasks. Similarly, if there are several ending tasks, it adds a joining complex gateway between ending tasks and the end event [24]. Algorithm 8 illustrates this refactoring.

Algorithm 8: Join start and end events.

Given G = (V, E)
∀ a ϵ V : a.type = TASK
    start ← Ø , end ← Ø
    if !□e ϵ E: e = (b, a), b ϵ V then
        start ←{a}
    if !□e ϵ E: e = (a, c), c ϵ V then
        end ←{a}
    V ← {v_1,v_2,v_3,v_4}, v_1.type = START, v_2.type = END, v_3 = COMPLEX, v_4 = COMPLEX
    E ←{e_1,e_2}, e1 = (v_1, v_3), e_2 = (v_4,v_2)
    ∀ v ϵ start
        E ←{e' = (v_3, v)}
    ∀ v ϵ end
        E ←{e' = (v, v_4)}

R9. Add gateways in incoming and outgoing branches: It is possible to obtain business process models by revere engineering that do not follow some of the good modeling practices that would be in accord with the BPMN standard as regards the usage of gateways [24]. In this case, this algorithm adds a split and join exclusive gateway when a certain task has several precursor or subsequent tasks, respectively (see Fig. 2). Algorithm 9 illustrates this refactoring.

Algorithm 9: Add gateways in branches.

Given G = (V, E)
∀ a ϵ V : a.type = TASK ∨ a.type = EVENT
    successor ← Ø, predecessor ← Ø
    ∀ b ϵ V :□e ϵ E: e = (a, b)
        if b.type = TASK ∨ b.type = EVENT ∨ b.type = GATEWAY
            successor ← {b}
            E – {e}
    ∀ b ϵ V :□e ϵ E: e = (b, a)
        if b.type = TASK ∨ b.type = EVENT ∨ b.type = GATEWAY
            predecessor ←{b}
            E – {e}

if |successor|>1 then

V ← {v_1}, v_1.type = EXCLUSIVE

∀ s ϵ successor

E ←{e_1,e_2}, e_1 = (a, v_1), e_2 = (v_1,s)

if |predecessor|>1 then

V ← {v_2}, v_2.type = EXCLUSIVE

∀ p ϵ predecessor

E ←{e_1,e_2}, e_1 = (p, v_1), e_2 = (v_1,a)

R10. Refine names: This refactoring algorithm implements a heuristic to improve labels of business tasks that were obtained almost directly from methods or functions of legacy source code through reverse engineering. These labels usually follow naming conventions present in most programming approaches such as the concatenation of various capitalized words. This refactoring algorithm split these labels into ones containing various words. This algorithm is not necessary to be shown due to the easy implementation using graphs.

3.3 Graph to BPMN

After applying refactoring algorithms, each graph is transformed into a business process model and each graph element is transformed into a BPMN element. Thus, each `BPElement` is transformed into a task, data object, gateway or event according to its type while each `BPEdge` is transformed into a sequence flow or association flow according to its type.

4 CASE STUDY

This section provides a case study with a real-life information system. The object of this case study is IBUPROFEN and the purpose of this case study is to evaluate how each refactoring algorithm affects to the understandability and modifiability of the business process model.

Despite the understandability depends on the people in charge of use, management or evaluation such business process models, i.e., it is subjective, understandability can be assess through several quality measures such as the number of nodes in the business process models, the number of nesting branches, the connectivity between elements, the density of elements, among others. For this reason, this paper considers as *dependent variables* the size, density and separability of a business process model in order to assess understandability and modifiability of a business process model.

- **Size** is the number of nodes in a business process model. This measure affects negatively to the understandability, i.e. a higher size difficult the understandability of a certain business process model [24].

- **Density** is the ratio between the total number of edges in a business process model and the theoretical maximum number of possible edges regarding the number of nodes. It affects the understandability and modifiability negatively, i.e., lower density values lead to business process models with a lower level of intricacy.
- **Separability** represents the ratio between the number of cut-vertices in a business process model (i.e. nodes that serve as bridges between otherwise strongly-connected components) and the total number of nodes. Separability affects the modifiability negatively, since higher separability implies hard and error-prone modifications of business process models.

The case under study is XCare information system. XCare is a mobile application of 9.9 thousands of lines of code. This application is intended for diabetes patients, which analyzes blood (through an external device) and suggests diet plans. Hence, the *independent variables* of this case study are each business process model obtained from XCare through reverse engineering.

The case study procedure consists of a set of semiautomatic steps that are executed in a computer with a 2.66 GHz dual processor and 4.0 GB RAM. The steps are as follows:

1. First of all, the extraction of business process model from XCare is performed by using MARBLE [25]. MARBLE is a tool used to recover business process models from existing Java code. This tool was selected because is released as an Eclipse plug-in and it therefore can be easily integrated with the IBUPROFEN tool.
2. Fig. 3 gives an example of a business process model obtained by MARBLE from XCare. This business process model contains 255 nodes and 512 edges, being the largest mined from XCare. The smallest model obtained has around 7 nodes and 6 edges. The sample can be visualized entirely and perfectly online [26]. Thus, a sample of 25 business process models is obtained from the source code from XCare.
3. The whole set of IBUPROFEN refactoring algorithms, that was mentioned in Section 3.2, are applied in each business process model retrieved in the above step. Refactoring algorithms are applied in isolation.
4. The dependent variables (size, density and separability) are recorded before and after applying each refactoring algorithm in order to be analyzed later.

Table 1 collects the value of the size, density and separability mean after applying each refactoring algorithm, as well as the gain obtained with respect to the original value. The gain is defined as the ratio between the difference of measure values and the original measure value. Thus, a positive gain means that the refactoring affects the measure positively while a negative gain means that the refactoring affects the measure negatively. A zero gain means that the value for a certain measure did not change after refactoring.

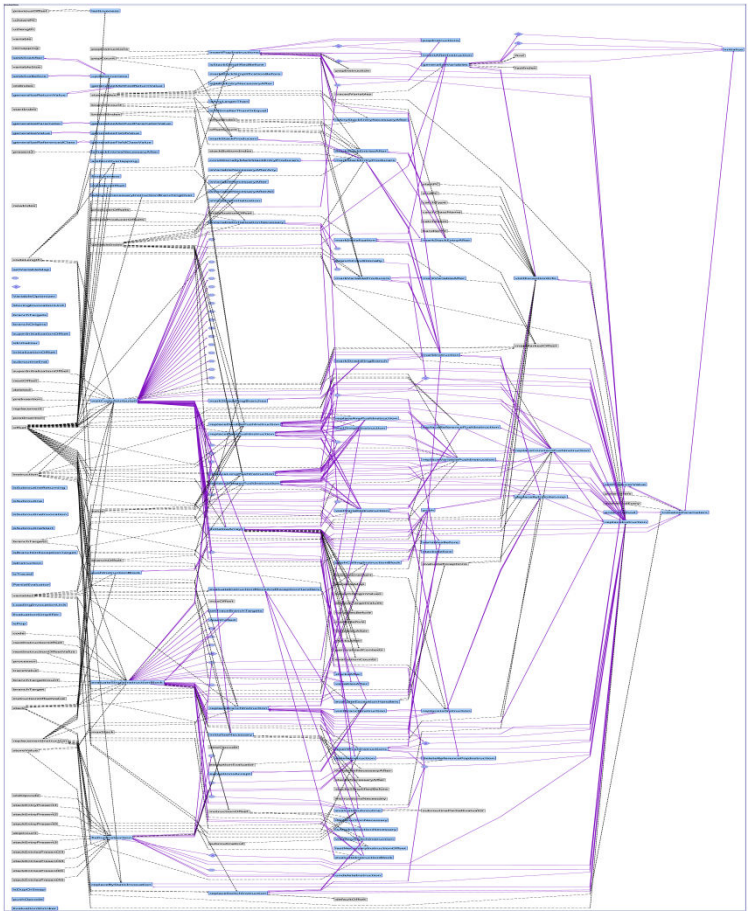

Fig. 3. Example of business process model managed by IBUPROFEN

Table 1. Effect of each refactoring algorithms on the size, density and separability

	Size		Density		Separability	
	Mean	Gain	Mean	Gain	Mean	Gain
Original	70.760	0.000	0.086	0.000	47.88	0.000
R1	46.440	0.366	0.150	-5.903	23.56	0.450
R2	67.120	0.030	0.087	-0.061	44.24	0.040
R3	70.760	0.000	0.086	0.000	47.88	0.000
R4	70.760	0.000	0.086	0.007	47.88	0.000
R5	70.440	0.003	0.086	-0.002	47.88	0.000
R6	62.560	0.114	0.067	0.093	48.76	-0.015
R7	63.040	0.098	0.093	-0.068	41.96	0.120
R8	74.360	-0.201	0.114	-0.511	51.48	-0.249
R9	90.160	-0.229	0.064	0.111	48.08	-0.002
R10	70.760	0.000	0.086	0.000	47.88	0.000

Table 1 reveals that removing isolated nodes decreases the size and separability while the density is increased. Despite the density is higher after R1, the relevance of the model has been increased since non-relevant elements have been removed. Similarly, R2 causes an increase of density when the size is decreased. Separability is decreased slightly. However, R3 has not impact on these measures due to business process models under study do not have nesting gateways. Removing inconsistencies (R4) maintains the same size and separability while the density is decreased because the number of edges is lower. Unnecessary gateways are removed (R5) and therefore the size is decreased while the density is increased owing to the number of nodes is lower. Separability after R5 is exacerbated slightly. R6 creates compound tasks in several business process models. This fact makes the size and density decrease in the most of cases. The same happens with R7, the number of nodes is lower but the number of edges is lower to and therefore, the density may increase while separability increases. R8 adds new missing elements in the model as start and end event as well as complex gateways. This makes that the size, separability and density are higher. In the same way, adding gateways in incoming and outgoing branches causes higher size. Nevertheless, the density after R9 tends to decrease due to there is more nodes in the model. Separability increases slightly since elements are more connected. In contrast, R10 does not have affect in any measures but the refinement of names implies an enhancement of the understandability.

5 CONCLUSIONS AND FUTURE WORK

Refactoring techniques has proved to be a good choice for improving business process models in terms, for example, of their understandability and modifiability levels. While graph-based algorithms have been successfully employed in different contexts, most business process model refactoring techniques often use alternative data structures [10-12] which leads to inefficient results. For this reason, this paper presents IBUPROFEN as a technique for refactoring business process models following a graph-based approach. Thus, the business process model is managed by means of a graph, changing its internal structure while its semantic is preserved. IBUPROFEN proposes ten refactoring algorithm divided into three groups in order to address the common problems that organizations have to deal when they retrieved such business process models by reverse engineering.

In order to demonstrate the feasibility of this approach, IBUPROFEN has been firstly implemented as an open source tool, and secondly, a case study with industrial business process models has been conducted. The case study reveals that the application the proposed graph-based refactoring algorithms improve the size, separability and density of business process models in the most of cases by removing non-relevant and fine-grained elements as well as by completing the models. The main limitation of this study is that results show size and density have an inverse relationship, i.e., when one increase the other decrease.

The second limitation lies in the empirical study analyzes the application of each refactoring algorithm in isolation. However, studies reveals that the order of applica-

tion of various refactoring algorithms in sequence could have an effect on the obtained results [18].

In line with the mentioned limitations, the future work will focus on the replication of the case study by analyzing alternative measures as well as the effect of different application orders. Furthermore, an algorithm improvement endeavor will be made to conciliate the size and density gain at the same time.

Acknowledgments

This work was supported by the FPU Spanish Program and the R&D projects MAGO /PEGASO (Ministerio de Ciencia e Innovación [TIN2009-13718-C02-01]) and GEODAS-BC (Ministerio de Economía y Competitividad & Fondos FEDER [TIN2012-37493-C03-01]).

6 REFERENCES

1. Weske, M., *Business Process Management: Concepts, Languages, Architectures*2007, Leipzig, Germany: Springer-Verlag Berlin Heidelberg. 368.
2. OMG. *Business Process Modeling Notation Specification 2.0*. 2011; Available from: http://www.omg.org/spec/BPMN/2.0/PDF/.
3. Jeston, J., J. Nelis, and T. Davenport, *Business Process Management: Practical Guidelines to Successful Implementations*. 2nd ed2008, NV, USA: Butterworth-Heinemann (Elsevier Ltd.). 469.
4. Davenport, T.H., *Need radical innovation and continuous improvement? Integrate process reengineering and TQM.* Strategy & Leadership Journal, 1993. **21**(3): p. 6-12.
5. Heuvel, W.-J.v.d., *Aligning Modern Business Processes and Legacy Systems: A Component-Based Perspective (Cooperative Information Systems)*2006: The MIT Press.
6. van der Aalst, W., *Process Mining: Overview and Opportunities.* ACM Transactions on Management Information Systems (TMIS), 2012. **3**(2): p. 7.
7. Pérez-Castillo, R., I. García-Rodríguez de Guzmán, and M. Piattini, *Business Process Archeology using MARBLE.* Information and Software Technology, 2011.
8. Fahland, D. and W.M.P.v.d. Aalst, *Repairing Process Models to Reflect Reality.* 2012.
9. Fernández-Ropero, M., R. Pérez-Castillo, and M. Piattini, *Refactoring Business Process Models: A Systematic Review*, in *7th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE)*, J. Filipe and L. Maciaszek, Editors. 2012, SciTePress: Wrocław, Poland. p. 140-145.
10. Dijkman, R., et al., *Identifying refactoring opportunities in process model repositories.* Inf. Softw. Technol., 2011. **53**(9): p. 937-948.
11. Weber, B., et al., *Refactoring large process model repositories.* Computers in Industry, 2011. **62**(5): p. 467-486.
12. La Rosa, M., et al., *Managing process model complexity via abstract syntax modifications.* Industrial Informatics, IEEE Transactions on, 2011. **7**(4): p. 614-629.
13. Pham, M.-D., P. Boncz, and O. Erling, *S3G2: A Scalable Structure-Correlated Social Graph Generator*, in *Selected Topics in Performance Evaluation and Benchmarking*, R. Nambiar and M. Poess, Editors. 2013, Springer Berlin Heidelberg. p. 156-172.

14. Gubichev, A. and T. Neumann, *Fast approximation of steiner trees in large graphs*, in *Proceedings of the 21st ACM international conference on Information and knowledge management*2012, ACM: Maui, Hawaii, USA. p. 1497-1501.
15. Mens, T., G. Taentzer, and O. Runge, *Analysing refactoring dependencies using graph transformation*. Software & Systems Modeling, 2007. **6**(3): p. 269-285.
16. Bobrik, R., M. Reichert, and T. Bauer, *View-Based Process Visualization*, in *Business Process Management*, G. Alonso, P. Dadam, and M. Rosemann, Editors. 2007, Springer Berlin Heidelberg. p. 88-95.
17. Hauser, R. and J. Koehler, *Compiling Process Graphs into Executable Code*, in *Generative Programming and Component Engineering*, G. Karsai and E. Visser, Editors. 2004, Springer Berlin Heidelberg. p. 317-336.
18. Fernández-Ropero, M., et al., *Assessing the Best-Order for Business Process Model Refactoring*, in *28th Symposium On Applied Computing (SAC)*2013: Coimbra, Portugal. p. 1400-1406.
19. Fernández-Ropero, M., R. Pérez-Castillo, and M. Piattini. *IBUPROFEN*. 2012; Available from: http://marketplace.eclipse.org/content/IBUPROFEN.
20. Naveh, B. and Contributors. *JGraphT*. 2011; Available from: http://jgrapht.org/.
21. Hunter, J., *JDOM*, 2009.
22. Pérez-Castillo, R., et al., *Generating Event Logs from Non-Process-Aware Systems Enabling Business Process Mining*. Enterprise Information System Journal, 2011. **5**(3): p. 301–335.
23. Zou, Y. and M. Hung, *An Approach for Extracting Workflows from E-Commerce Applications*, in *Proceedings of the Fourteenth International Conference on Program Comprehension*2006, IEEE Computer Society. p. 127-136.
24. Mendling, J., H.A. Reijers, and W.M.P. van der Aalst, *Seven process modeling guidelines (7PMG)*. Information and Software Technology, 2010. **52**(2): p. 127-136.
25. Pérez-Castillo, R., et al., *MARBLE. A Business Process Archeology Tool*, in *27th IEEE International Conference on Software Maintenance (ICSM'11)*2011, IEEE Computer Society: Williamsburg, Virginia, USA. p. 578-581.
26. Fernández-Ropero, M., R. Pérez-Castillo, and M. Piattini. *Extra material of Graph-Based Business Process Model Refactoring* 2013; Available from: http://alarcos.esi.uclm.es/per/mfernandez/material3.html.