An Approach to the Decomposition of Business Processes for Execution in the Cloud

Lucas Venezian Povoa[1,3], Wanderley Lopes de Souza[1], Luís Ferreira Pires[2], Antonio Francisco do Prado[1]
[1]Department of Computer Science (DC), Federal University of São Carlos (UFSCar)
São Carlos, Brazil
[2]Faculty of Electrical Engineering, Mathematics and Computer Science (EEMCS), University of Twente (UT)
Enschede, the Netherlands
[3]São Paulo Federal Institute of Education, Science and Technology (IFSP)
Caraguatatuba, Brazil
venezian@ifsp.edu.br, desouza@dc.ufscar.br, l.ferreirapires@utwente.nl, , prado@dc.ufscar.br

***Abstract*—Although Business Process Management has emerged as a means to manage and improve business processes, it may require high costs due to the need for software, hardware and technical support. Cloud Computing can help achieve efficient business processes with lower costs, since it provides a fast and cheap way to acquire computing resources in a pay-per-use manner. However, due to safety requirements, certain data or activities of a business process should be kept within the user premises, while others can be allocated to a cloud. This paper presents an approach to decomposition of business processes, which preserves the data constraints, and demonstrates its use through a case study in the healthcare domain.**

Keywords—Graph-based Model, BPM, BPMN, WS-BPEL

I. INTRODUCTION

In the last decades, workflow-based applications have been successfully applied to solve both scientific and business problems [1]. Business Process Management (BPM) is often considered as an extension of the classical approaches to workflows, and it has been used to design, enact, manage and analyze business processes [2]. Service-Oriented Architecture (SOA) has facilitated the use of Business Processes Management Systems (BPMSs) to help companies reach their business goals by using business processes to orchestrate business services. However, the development of scalable architectures for business processes is still a challenge, since the use of a single process engine by BPMSs leads to an excessive centralization of the processes coordination. Moreover, it is quite expensive to extend this architecture to cope with scalability issues, due the costs of software, hardware, and technical support, which can be a prohibitive factor for medium and small companies.

Cloud computing deals with the scalability and cost issues, providing a seemingly unlimited set of computational resources, such as hardware, and software, in a pay-per-use basis [3]. However, although Cloud computing is turning Computing into the fifth utility [4], similar to water, electricity, gas, and telephony, it has trust issues that can limit its application in certain domains such as in healthcare, where data confidentiality is regulated by laws. This problem arises mainly because cloud providers cannot guarantee the required levels of confidentiality, since they normally do not reveal how data are handled inside the cloud environments.

This paper proposes an approach to the decomposition of a monolithic business process into multiple sub-processes to be deployed on premise or in a cloud, taking into account costs, performance, and data safety restrictions. The general applicability of our approach is demonstrated with a case study of a business process of the healthcare domain. We presented the initial ideas of our approach earlier in [5], illustrated with the case study using WS-BPEL. In this paper we improve the approach, and illustrate it with a model of our case study process in Business Processes Model and Notation (BPMN).

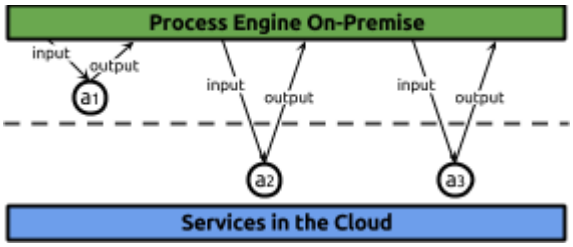
The remainder of this paper is organized as follows: Section II gives an overview of our approach; Section III presents the intermediate model; Section IV describes the lifting and grounding transformations; Section V discusses the calculation of an optimal distribution for activities and associated data; Section VI deals with the decomposition of business processes; Section VII presents a case study; Section VIII discusses related work; and Section IX presents our conclusions and some directions for future work.

II. APPROACH OVERVIEW

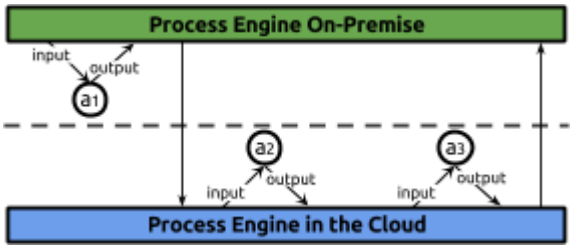
Cloud-based BPM, and particularly the distribution of business process activities and data over the user premises and a cloud-based BPM environment, has been initially investigated by Han *et al.* in [6], who defined the so called Processes Enactment Engine, Activities, and Data Storage (PAD) model. This model considers the distribution of activities and data of a business process, but ignores the distribution of the process engine itself. This model was extended by Duipmans *et al.* in [7] to include the distribution of the process engine, so that in some circumstances the communication between the sub-processes could be reduced.

Fig. 1 shows a business process executed partially on premise and partially in a cloud environment. Fig. 1(a) shows the message exchange when there is a single process engine, in this case on premise, while Fig. 1(b) shows the message exchange when process engines are available on both sides. Since all data exchange happens through a single process engine in case only one process is available, the performance of the process execution tends to decrease. Further, costs tend to increase in case only one process engine is available due to the amount of data that needs to be transferred to and from the cloud.

To cope with these problems, our approach to distribute activities, data, and the process engine on premise and in the cloud considers performance, privacy, and cost requirements. Fig. 2 shows that this approach consists of four steps: lifting, selection, decomposition, and grounding.



(a)



(b)

Fig. 1. Process engine distributed (a) only on premise and (b) on both sides.

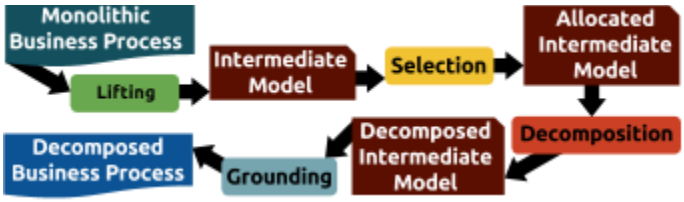


Fig. 2. Steps of the proposed approach.

In the lifting step, a monolithic business process specification is transformed to an intermediate model, taking into account the variables and data flows amongst the activities. In the selection step, the optimal distribution of the business process activities into intermediate model is calculated, and the location, on premise or in the cloud, of each activity and associated data is determined. In the decomposition step, the business process represented into intermediate model is decomposed, based on the results obtained in the precedent step, and by applying a set of decomposition rules. In the grounding step, the resulting sub-processes represented into intermediate model are transformed into business sub-process specifications.

III. Intermediate Model

The intermediate model has been designed to support most business process languages available nowadays, making the decomposition procedure language-independent. To this end, we designed our model to be able to represent eighteen Workflow Control-Flow Patterns (WCP) [8], nine Workflow Data Flow Patterns (WDP) [9], six Workflow Exception Handling Patterns (WEP) [10] and two Communication Patterns (CP) [11]. The intermediate model has been designed based on directed graphs, and hence we call it Graph-based Workflow Model (GWM). The GWM constructs are able to represent business processes specified in WS-BPEL 2.0, BPMN 2.0, Yet Another Workflow Language (YAWL), Web Services Choreography Description Language (WS-CDL), and Architectural Modelling Box for Enterprise Redesign (AMBER).

In GWM, activities are represented as nodes, and the relationships between activities are represented as edges. GWM defines nodes and edges of different types. Particularly control edges, data edges, communication edges, and exception edges enable the representation of different types of structures.

GWM also enables the validation of the data constraints in distributed business processes, and the calculation of the activity costs, by reasoning on the input and output data of activities. Eventually, unexpected behavior within an activity interrupts the normal flow, and triggers an exception of some type, which is captured by its outgoing exception edge.

In the sequel we present the GWM constructs in terms of four major concerns that are addressed in business process models, namely Control Flow, Data Flow, Communication and Exception Handling. We discuss each major concern in terms of simpler concerns that correspond to one or more WCP, WDP, WEP or CP patterns. The corresponding patterns for each concern are indicated between parentheses. The formal definition of the GWM is presented in [12].

A. Control Flow

Control flow constructs define the activities coordination, i.e. order and conditions. GWM supports the following control flow concerns: Sequence, Conditional Branch, Parallel Branches, Partial Join, Loop, External Exclusive Choice, Implicit Finalization, Explicit Finalization, and Multiple Instances.

In GWM, activities are represented as nodes, and the relationships between activities are represented as edges. The Sequence concern (WCP#1) represents the sequential execution of activities. The execution order is determined by the directed control edges that link the nodes.

The Conditional Branch concern (WCP#2, WCP#3 and WDP#40), which is modeled by *if* and *eif* nodes, represents the choice between two alternative branches. The *if* node splits a branch into two branches, has one of its outgoing edges labeled true and the other false, and a condition attached to it. After the evaluation of this condition, only one of these edges is taken. The *eif* node is used to join the conditional branches into a single outgoing branch. Fig. 3 illustrates this concern.

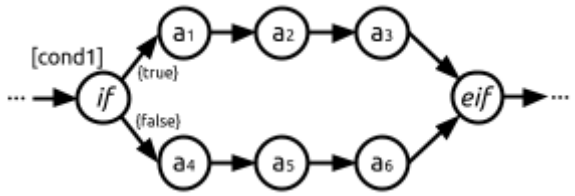


Fig. 3. Conditional Branch concern.

The Parallel Branches concern (WCP#4 and WCP#5), which is modeled by *par* and *epar* nodes, represents the execution of multiple branches in parallel. The *par* node splits a branch into multiple branches, and has at least two outgoing control edges. The *epar* node is used to join the triggered

parallel branches into a single outgoing branch. Fig. 4 illustrates this concern with two parallel branches.

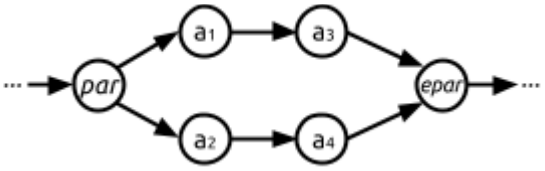


Fig. 4. Parallel Branches concern.

The Partial Join concern (WCP#9, WCP#19 and WCP#37), which is modeled by *and* or *or* nodes, enables the synchronization of a subset of parallel branches within a Parallel Branches concern. The outgoing branch of an *and* node is executed only after all its incoming branches have been executed. The outgoing branch of an *or* node is executed only after one of its incoming branches has been executed, which leads to abortion of the executions of others incoming branches. Fig. 5 illustrates this concern with *and* and *or* nodes.

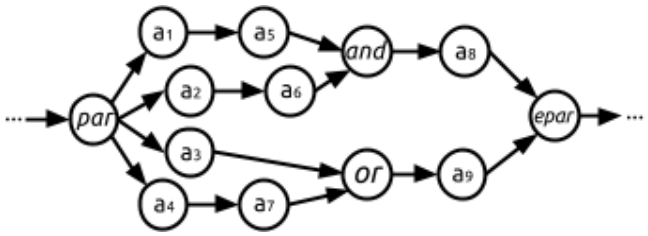


Fig. 5. Partial Join concern.

The Loop concern (WCP#21 and WDP#40), which is modeled by a *loop* node with a condition attached to it, represents iterative branches. After evaluating the condition, the iterative branch is taken or abandoned. A *loop* node can be placed before or after the iterative branch: in the first case this branch will be executed zero or more times, while in the second case will be executed at least once. Fig. 6 illustrates this concern with the two possibilities for placing the *loop* node.

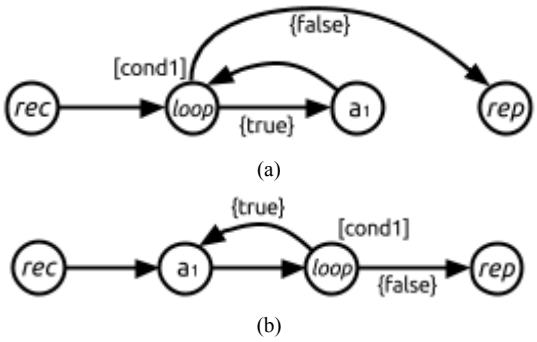


Fig. 6. Loop concerns with *loop* node before (a) and after (b) the iterative branch.

The External Exclusive Choice concern (WCP#16, WCP#24, CP#2, and WDP#38), which is modeled by *xor* and *exor* nodes, represents the choice of a branch via an interaction with an external partner. The *xor* node splits a branch into multiple branches, which are labeled with operations that can be invoked by external partners. The *exor* node is used to join the multiple branches into a single outgoing branch. Fig. 7 illustrates this concern with three possible branches.

The Implicit Finalization concern (WCP#11), which has no specific node for modeling its behavior, implicitly finalizes a workflow instance when no activities are being executed. The Explicit Finalization concern (WCP#19), which is modeled by an *exit* node, explicitly finalizes a workflow instance.

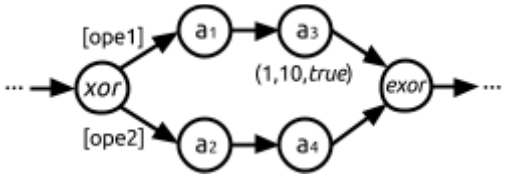


Fig. 7. External Exclusive Choice concern.

The Multiple Instances concern (WCP#12, WCP#13 and WCP#14), which can be asynchronous or synchronous, defines a replication rule, via the 3-tuple *(s, e, w),* for generating multiple parallel instances of an activity set. The number of parallel instances is defined by $(e - s + 1)$, if $s > e$ there are no parallel instances, and w is a Boolean variable which defines whether the parallel instance executions are synchronized or not. Fig. 7 illustrates this concern nested at an External Exclusive Choice concern, where ten instances of a_3 are created.

B. Data Flow

Data flow constructs deal with the data aspects of business processes. GWM supports three data flow concerns: Local Variable, Global Variable, and Data Exchange.

The Local Variable concern (WDP#3), which has no specific node for modeling its behavior, defines variables that are visible for a set of activities. The Global Variable concern (WDP#5), which also has no specific node for modeling its behavior, defines variables that are visible for the entire business process.

The Data Exchange concern (WDP#9, WDP#15, WDP#16, WDP#27 and WDP#28), which is modeled by a labeled data edge, represents the data exchange between internal activities, or between the business process and its external partners. Fig. 8 illustrates this concern with the data edges labeled v_1 and v_2.

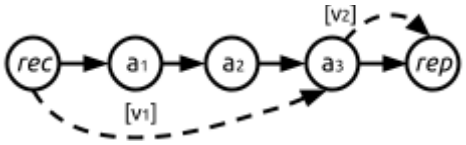


Fig. 8. Data Exchange concern.

C. Communication

The Receive Message concern (CP#2 and WDP#38), which is modeled by a *rec* node labeled with an operation, allows messages from external partners to be received. Fig. 6 illustrates two instances of this concern, where the *rec* node is placed before a Loop concern. The Reply Message concern (CP#1), which is modeled by a *rep* node, allows the process to reply to requests from external partners. Fig. 6 illustrates two instances of this concern, where the *rep* node is placed after a Loop concern.

The Request Service concern (CP#1, CP#2 and CP#4) is modeled by a *req* node when a message is sent to an external partner without an expected response, or by a combination of a

req node and a *get* node when a message is sent to an external partner and a response is expected. Fig. 9 illustrates two instances of this concern, where an external partner is accessed via operation *ope2*.

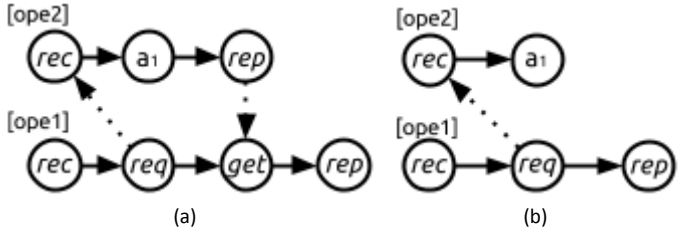


Fig. 9. Request Service concern (a) with and (b) without response.

D. Exception Handling

The Fault Handling concern (SFF-CWC-COM, SFF-CWC-NIL, SFF-RCC-COM. and SFF-RCC-NIL), which is modeled by *exp* and *eexp* nodes, allows the choice between multiple branches based on an exception of some type. An exception is received by an *exp* node, via its incoming exception edge, which sets the outgoing exception branch to be taken based on a matching exception. When none of the outgoing exception branches matches the received exception, an optional outgoing exception branch can be taken by default, which has the incoming edge labelled as otherwise. The *eexp* node is used to join the exception branches into a single outgoing branch. Fig. 10 illustrates this concern, where the *exp* node has an outgoing exception branch for the *exp1* exception, and a default outgoing exception branch.

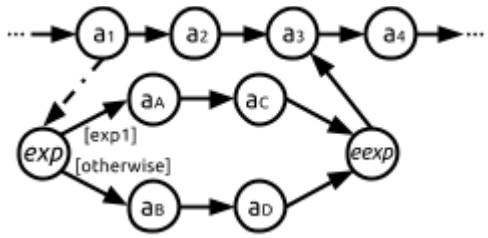


Fig. 10. Exception Handling concern.

The Deadline Handling concern (SCE-CWC-COM, SCE-CWC-NIL, SFF-CWC-COM, SFF-CWC-NIL, SFF-RCC-COM and SFF-RCC-NIL), which is modelled by *ddl* and *eddl* nodes, allows an alternative branch to be taken when a timeout for a *xor* or *rec* node is reached. A *ddl* node captures a timeout exception by means of an exception edge, and triggers an alternative branch. The *eddl* node is used to join the alternative branch. Fig. 11 illustrates this concern with an External Exclusive Choice concern, which has a deadline t_d.

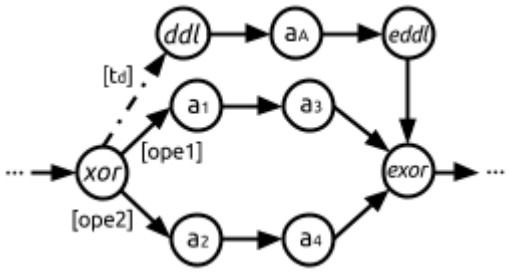


Fig. 11. Deadline Handling concern.

The Finalization Handling concern (SFF-CWC-COM, SFF-CWC-NIL, SFF-RCC-COM and SFF-RCC-NIL), which is modelled by *fin* and *efin* nodes, allows a branch to be taken after the execution of a set of activities, possibility triggering an exception. Fig. 12 illustrates this concern with the a_3 activity in combination with the Conditional Branch concern.

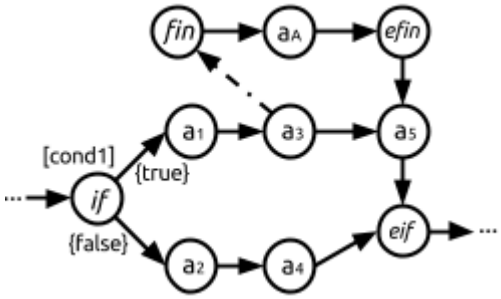


Fig. 12. Finalization Handling concern.

IV. Location Selection

In order to semi-automatically determine the location of each activity and the associated data, we built a location selection framework based on privacy policies, monetary costs, and performance metrics (e.g., response time, throughput). This framework simplifies the GWM generated in the lifting step by reducing the number of nodes. Using this simplified GWM, and based on the integer optimization model defined by Han *et al.* [6], the selection framework calculates the cost of the business process for each possible alternative combination of locations of the nodes (on premise and cloud) as follow:

$$cost_d = w_e \times cost_e + w_m \times cost_m + w_p \times cost_p \quad (1)$$

where $cost_e$, $cost_m$, and $cost_p$ stand for the execution cost, monetary cost, and privacy cost, and w_e, w_m, and w_p are their weight factors defined to represent the relative importance of each of these factors in the total costs calculation. For calculating these costs, we assume that:

- let $\boldsymbol{A} = \{a_1, a_2, \ldots, a_n\}$ be the set of activities, and $\boldsymbol{D} = \{d_1, d_2, \ldots, d_m\}$ be the set of data items in the simplified GWM;
- let $\boldsymbol{s} = (s_1\ s_2\ \ldots\ s_n)$ be the location vector of activities, where $s_i = 1$ means that activity a_i is located in the cloud, and $s_i = 0$ on premise;
- let $\boldsymbol{R}$ be the relation sparse matrix of activities and data items, where $R(i,j) = 1$ means that activity a_i has a direct relation with data item d_j, and $R(i,j) = 0$ means that the activity and the data item are unrelated;
- let $\boldsymbol{V}$ be the 3-dimensional sparse matrix for data exchange, where $V(i,j,k) = 1$ means that activity a_i sends the data item d_j to activity a_k, and $V(i,j,k) = 0$ means that this data item is not sent from activity a_i to activity a_k;
- let $\boldsymbol{Q}$ be the sparse matrix for data location with $Q(o,p) = \sum_{i=1}^{m}[V(i,o,p) \times s_i]$, where data item d_j is located on premise before being sent to activity a_i whether $Q(i,j) = 0$, and in the cloud if $Q(i,j) = 1$;

- let $C_{premise} = (ram_p, hdd_p, cpu_p, f_p, b)$ be the 5-tuple that represents the server configuration on premise, where ram_p is the amount of RAM in MB, hdd_p is the amount of disk in GB, cpu_p is the number of vCPUs, f_p is the frequency of each vCPU in GHz, and b is the related bandwidth in Bps between premise and cloud;
- let $C_{cloud} = (ram_c, hdd_c, cpu_c, f_c, cost_t, cost_h, cost_s)$ be the 7-tuple that represents the cloud server configuration, where ram_c is the amount of RAM in MB, hdd_c is the amount of disk in GB, cpu_c is the number of vCPUs, f_c is the frequency of each vCPU in GHz, $cost_t$ is the cost in US$ per byte transferred to the cloud, $cost_h$ is the cost in US$ per hour of the cloud server, and $cost_s$ is the cost in US$ per byte stored in the cloud; and
- let $exec_c(a_i)$ be the execution time of activity a_i in the cloud defined as $exec_c(a_i) = 1/2 \times (cpu_p \times f_p / cpu_c \times f_c + ram_p/ram_c) \times exec_p(a_i)$, where $exec_p(a_i)$ is the execution time of the activity a_i on premise that must be provided by an external entity (e.g., user or BPMS component).

Function $exec_c(a_i)$ defines that increasing RAM or increasing the processing power cause a reduction in the execution time of activities in the cloud. This assumption is acceptable because there is no prior knowledge whether an activity is memory intensive, CPU intensive or disk intensive.

Consequently, $exec_c(a_i) < exec_p(a_i)$, in case the cloud server configuration has more processing power than the on premise configuration (often the case). However, reducing the execution time of the activities of a business process may not be a real benefit, if the data transfer time to and from the cloud for this business process outweighs the achieved reduction. This time is defined as $trans_t = \sum_{i=1}^{n}\sum_{j=1}^{m} size(d_j) \times b^{-1} \times R(i,j) \times |s_i - Q(i,j)|$, where *size* is the data size in bytes, provided by an external entity, and the modulo $|s_i - Q(i,j)|$ yields *1*, if the data item d_j needs be transferred to be used by activity a_i, and *0* otherwise. So the execution cost of a business process is defined as

$$cost_e = \sum_{i=1}^{n} [exec_c(a_i) \times s_i + exec_p(a_i) \times (1 - s_i) + trans_t] \quad (2)$$

Usually, the time that a server remains active in the cloud, the data transfer to and from the cloud, and the data stored into the cloud are charged by the cloud provider. The monetary cost for executing business process activities in the cloud is defined as $monet_c = \sum_{i=1}^{n}[cost_h \times exec_c(a_i) \times s_i]$. Let $\boldsymbol{P}$ be the sparse matrix, provided by an external entity, where $P(i,j) = 1$ if data item d_j is persistent in activity a_i and *0* otherwise. The monetary cost for storing data in the cloud of a business process is defined as $monet_s = \sum_{i=1}^{n}\sum_{j=1}^{m}[cost_s \times size(d_j) \times s_i \times P(i,j)]$. The monetary cost for transferring data into or out of the cloud is defined as $monet_t = \sum_{i=1}^{n}\sum_{j=1}^{m}[cost_t \times size(d_j) \times |s_i - Q(i,j)|]$. So, the monetary cost of a business process is defined as

$$cost_m = monet_c + monet_s + monet_t \quad (3)$$

Data privacy is another issue for executing business processes in the cloud. Let $\boldsymbol{c} = (c_1\ c_2\ \ldots\ c_m)$ be a constraint vector for data items, provided by an external entity, where $c_j = 1$ if the data item d_j is sensitive and *0* otherwise. So the privacy cost of a business process is defined as

$$cost_p = \sum_{i=1}^{n}\sum_{j=1}^{m}[c_j \cdot R(i,j) \cdot s_i] \quad (4)$$

After calculating the cost for each alternative combination of node locations (on premise and in the cloud), the lowest cost is selected. The vector $\boldsymbol{s}$ associated to this cost is used for marking the location of each node of the GWM generated in the lifting step. This marked GWM is used as input to the decomposition step.

V. Lifting and Grouding

In order to enable the lifting and grounding transformations between GWM and business processes specified in BPMN or WS-BPEL, we defined a mapping between the BPMN and WS-BPEL patterns and the GWM concerns.

Based on this mapping, we defined transformation algorithms to perform the lifting and grounding, using the approach proposed by Povoa *et al.* [5], which converts tree structures into graph structures and vice-versa. In particular, the transformation algorithms for BPMN business processes assume that the input process is well structured, i.e., for every node with multiple outgoing edges (a split) there is a corresponding node with multiple incoming edges (a join), and vice-versa [13].

In order to generalize these transformations, we defined recursive algorithms for the lifting and grounding, which both have a general part and specific parts.

The general part of the lifting algorithm takes a tree structure as input and identifies the root node type, after which it calls an appropriated specific part for that node type, passing a sub-tree from the root node as input. All specific parts know the structure from the root node and proceed towards the child nodes, generating an equivalent construction in a graph structure. When a specific part comes across a node type for which no transformation specification is available in this scope, it calls recursively the general part, passing to the general part a sub-tree from the current node.

The general part of the grounding algorithm takes as input a graph structure (GWM) and identifies its top-level concern in order to call the specific part that can handle this concern. The specific parts proceed towards the nested nodes and generate a business process structure that is equivalent to tree structure being processed. Recursively, the general part is called when the specific part finds a node type for which no transformation specification is known in this scope.

Fig. 13 shows an instance of execution of the lifting algorithm on the *Exclusive Gateway* pattern in BPMN and on the *if* pattern in WS-BPEL and of the grounding algorithm on the Conditional Branch concern. In BPMN, the control flow is defined by child nodes of type *incoming* and *outgoing*, which are shown as dashed lines in Fig. 13, illustrating the control flow defined by them. In WS-BPEL, the control flow is simply

defined by the order of nodes (the first node is executed first, and so on).

Therefore, the lifting algorithm for BPMN looks at outgoing child nodes to decide which is the next node to be analyzed and at incoming child nodes to check whether the flow is correct, defining thus the control flow in GWM. The lifting algorithm for WS-BPEL only follows the order of the nodes to define the control flow in GWM.

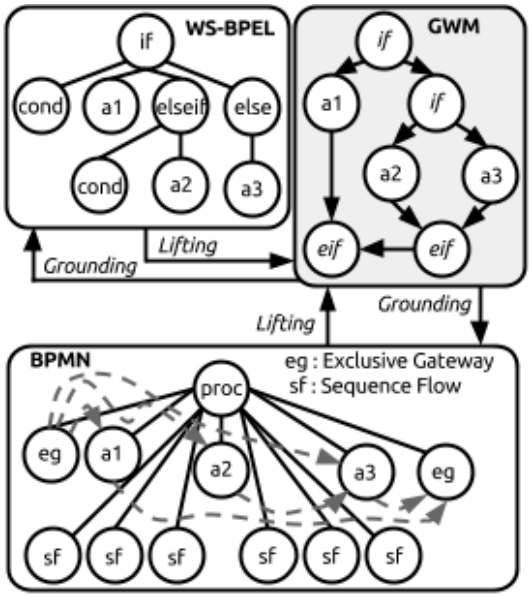


Fig. 13. Lifting and grounding transformations for the Conditional Branch.

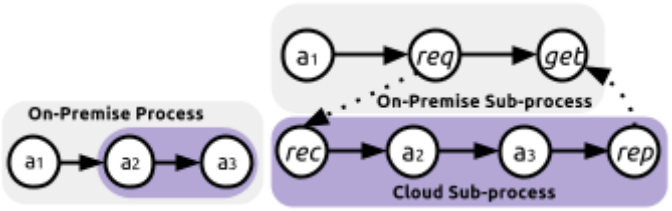
Moreover, BPMN *sequenceFlow* nodes get properties derived from the directed edges, which for outgoing edges of an Exclusive Gateway represents an execution condition. In WS-BPEL, a condition of an *if* or *elseif* pattern is defined in a condition node.

VI. Decomposition

In order to perform business process decomposition, we defined six decomposition rules taking into account that this process is hosted on premise and have activities to be allocated in the cloud, or vice-versa.

The first rule allocates the Sequence, Conditional Branch, Parallel Branches, or Loop concerns as a whole in a new sub-process. The selected concern in the monolithic process is replaced by *req* and *get* nodes connected via a control edge in the on premise sub-process. The cloud sub-process starts with a *rec* node and ends with a *rep* node. The *rec* and *req* nodes, and the *rep* and *get* nodes are connected by communication edges. Fig. 14 illustrates this rule for the Sequence concern.

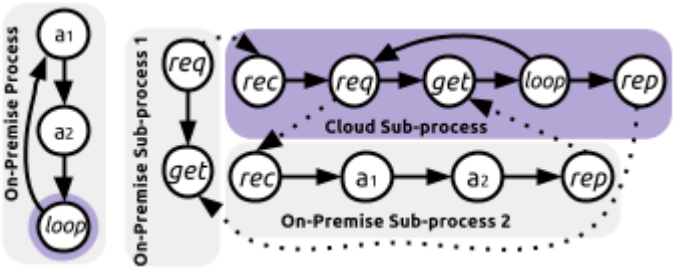
The second rule generates three sub-processes, where two are hosted on premise and the other one in the cloud, for allocating the *if* and *eif* nodes of a Conditional Branch concern or the *par*, *epar*, *and* and *or* nodes of a Parallel Branches concern to the sub-process in the cloud. In the first step, the concern is allocated as a whole to the sub-process in the cloud and replaced by *req* and *get* nodes in the sub-process on premise. Then the branches between *if* and *eif* nodes or between the *par*, *epar*, *or* and *and* nodes are replaced by *req* and *get* nodes, in the sub-process in the cloud, and allocated nested at the Exclusive External Choice concern in the new sub-process on premise.



(a) Monolithic (b) Decomposed

Fig. 14. Decomposed rule applied to the Sequence concern.

The third rule also generates three sub-processes for allocating the *loop* node of the Loop concern in the sub-process in the cloud. Fig. 15 illustrates this rule, where the first step is exactly the same as in the third rule. Then the iterative branch of the Loop concern is replaced by *req* and *get* nodes in the sub-process in the cloud, and allocated between the *rec* and *rep* nodes in the new sub-process on premise.



(a) Monolithic (b) Decomposed

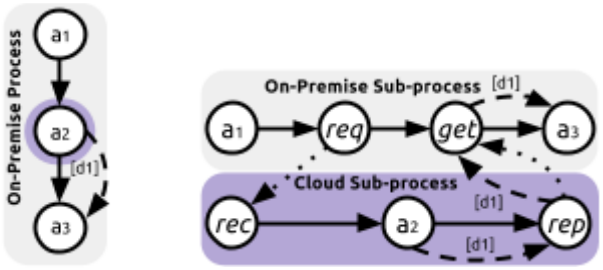
Fig. 15. Decomposition by moving the *loop* node.

The fourth rule combines the first and second rules and is applied on the Conditional Branch and Parallel Branches concerns. This rule also generates three sub-processes, where two are hosted on premise and the other one in the cloud, for allocating the *if* and *eif* nodes and one nested branch of a Conditional Branch concern or the *par*, *epar*, *and* and *or* nodes and a non-empty subset, not equals to the set, of nested branches of a Parallel Branches concern to the sub-process in the cloud.

The fifth rule is applied to communication nodes, and allocates the *req* and *get* nodes of the Request Service concern to the sub-process in the cloud. This rule reduces communication costs when invoking an external partner that requires some data item to be moved to the cloud.

The sixth rule moves data items associated to activities that originally were on one side (i.e., cloud or premise) and have been allocated to the other side. Fig. 16 illustrates this rule applied to the data item d_1 exchanged between a_2 and a_3.

Fig. 16(a) shows a monolithic process on premise where the activity a_2, which was allocated to the cloud, sends the data item d_1 to the activity a_3. Fig. 16(b) shows the decomposed process where the activity a_2, now in the sub-process in the cloud, sends the same data item d_1 to the activity a_3 in the sub-process on premise. In this example, the a_2 outgoing data edge is connected to the *rep* node, and two new data edges are created for exchanging d_1: one between the *rep* and *get* nodes, and another one between the *get* node and activity a_3.



(a) Monolithic (b) Decomposed

Fig. 16. Decomposition by moving the Sequence concern exchanging a data.

VII. Case Study

Our case study is based on Picture Archiving and Communication Systems (PACS) [14], which supports a business process in the healthcare domain. Its goal is to persist and analyze breast tomographies, accepting as input a non-empty tomography set with their diagnostics and identifiers. Each tomography is persisted along with its diagnostic, analyzed for searching nodules, and the tomography set with possible nodes is sent back to the requester. We performed the decomposition of the PACS process via a prototype of our approach implemented in Java 7.

Although the PACS process was specified in BPMN and WS-BPEL [12], this paper deals only with its BPMN specification. Fig. 17 illustrates the monolithic PACS in BPMN, and Fig. 18 in GWM, where the last one was obtained from the first by performing the lifting algorithm.

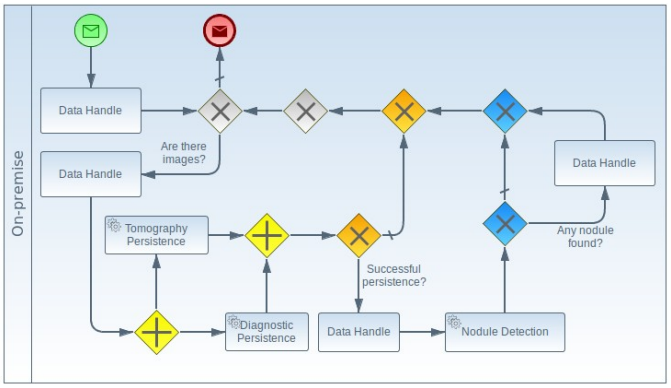


Fig. 17. Monolithic PACS Business Process in BPMN.

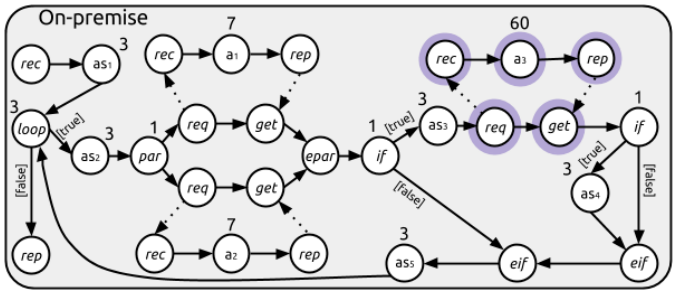


Fig. 18. Monolithic PACS Business Process in GWM.

The value marked in each node of Fig. 18(b) is the on premise execution time of the corresponding activity, which was used in the selection step. Furthermore, we considered for calculating the decomposition cost an on premise server with 2 GB of RAM, 50 GB of disk, 1 virtual CPU (vCPU) with 0.8 GHz of frequency, and a bandwidth of 125 Mbps, and three distinct cloud server configurations, which are shown in Table I. We also considered for this calculation a workload composed by two breast tomographies of 11.7 MB each, two integers of 32 bits, and two textual diagnostics of 64 KB, where both activities *a1* and *a2* persist their associated data. Our location selection framework produced the same result for the three cloud server configurations, which is illustrated in Fig. 18, where the highlighted activities have been selected to be deployed in the cloud.

TABLE I. Amazon EC2's configurations of cloud servers.

Instance name	vCPUs (Unity)	Frequency (GHz)	RAM (GB)	HD (GB)	Price/hour (US$)
c1.xlarge	8	2.75	7.00	4 x 420	0.58
m2.4xlarge	8	3.58	68.40	2 x 840	1.64
hs1.8xlarge	16	2.41	117.00	45 x 2048	4.60

After performing the decomposition step on the monolithic GWM, we obtained the decomposed GWM illustrated in Fig. 19. Finally, the grounding step was applied to this decomposed GWM to obtain the decomposed PACS in BPMN illustrated in Fig. 20.

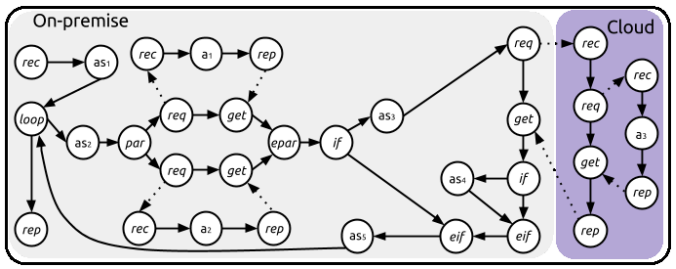


Fig. 19. Decomposed PACS Business Process in GWM.

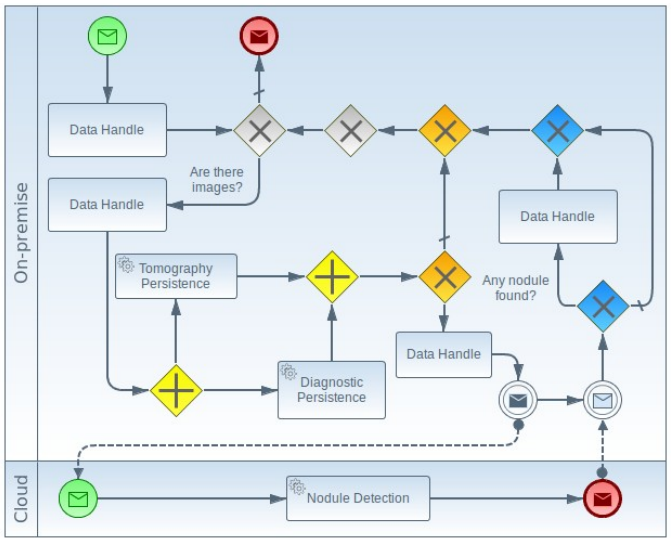


Fig. 20. Decomposed PACS Business Process in BPMN.

VIII. Related Work

In Nanda *et al.* [15] new orchestrations are created for each service employed by a business process, resulting in direct communication between them instead of having a single

coordinator. The WS-BPEL process is converted to a control flow graph, which generates a Program Dependency Graph (PDG) from which the transformations are performed, and the new generated graphs are reconverted to WS-BPEL sub-processes. Since each service in the original business process corresponds to a fixed node to which a sub-process is generated, this work is not suitable to support our decomposition requirements because it creates sub-processes with multiple services.

The approach described by Koop *et al.* in [16] focuses on decentralizing the orchestration of processes in WS-BPEL, and employs the Dead Path Elimination (DPE) model to ensure the end of the decentralized processes executions. However, since DPE is an issue that rises only for WS-BPEL, the use of this model makes this approach dependent of this language. In contrast, our approach is founded on the GWM, which is independent of any business process language, and employs a reusable set of decomposition rules, requiring only the development of the lifting and grounding transformations for the chosen business process language.

Duipmans *et al.* [17] presents an approach for the decomposition of business processes described in AMBER, a proprietary specification language. This decomposition is founded on a graph-based model, and is guided by a distribution list of activities that must be provided by the user. However, this model employs only six workflow control-flow patterns, uses data flow and communication features in an informal way, and does not consider exception handling patterns. Our approach considers two standardized business process specification languages, the decomposition is founded on GWM, and the distribution of activities is determined by a location framework. Furthermore, GWM supports fourteen workflow control-flow patterns, nine data flow patterns, three communication patterns, and also six exception handling patterns.

Finally, Fdhila *et al.* [18] report that most work on the decentralization of orchestrations focus too much on specific business process languages. In our work we do not focus so much on business process languages, and do not only concentrate on performance issues, but also consider safety requirements.

IX. Final Remarks

This paper presented an approach to decompose business processes for cloud deployment. This paper contains three main contributions: (a) the definition of a language-agnostic intermediate model, which allows our approach to be independent of business process languages, and that supports several control flow, data flow, communication and exception handling concerns, which allows our approach to cover the majority of the available business process languages; (b) the location selection framework that semi-automatically determines an optimal location for activities and their associated data, considering allocation data restrictions, performance, and monetary costs; (c) the definition of a set of rules for the business process decomposition for generating a decomposed process that displays a behavior that is equivalent to the behavior of the original monolithic process. The details of the proposed approach, which were omitted in this paper due to the pages limitation, are described in [12].

As future work, we intend to develop new decomposition rules and selection frameworks, in order to cope with multiple cloud servers in different cloud providers, as well as investigate effects of executing the decomposed parts in parallel. We are also interested in integrating our approach to a BPMS for providing all needed information to perform the decomposition automatically, which can be an initial step for performing business process decomposition and deployment on demand.

References

[1] Deelman, E.: Grids and Clouds: making workflow applications work in heterogeneous distributed environments. Int. J. High Perform. Comput. Appl. 24, 284–298 (2010)

[2] Van der Aalst, W.M.P., Hofstede, A.H.M.T., Weske, M.: Business process management: a survey. In: Proceedings of the BPM'03, pp. 1–12. Springer-Verlag, Eindhoven, The Netherlands (2003)

[3] Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., Zaharia, M.: A view of cloud computing. Commun. ACM. 53, 50–58 (2010)

[4] Buyya, R., Yeo, C.S., Venugopal, S., Broberg, J., Brandic, I.: Cloud computing and emerging IT platforms: vision, hype, and reality for delivering computing as the 5th utility. Future Gener. Comp. Sy. 25, 599–616 (2009)

[5] Venezian Povoa, L., de Souza, W.L., Ferreira Pires, L., Duipmans, E.F., do Prado, A.F.: An approach to business processes decomposition for cloud deployment. In: 27th SBES, pp. 124–133. Brasília, Brazil (2013)

[6] Han, Y.B., Sun, J.Y., Wang, G.L., Li, H.F.: A cloud-based BPM architecture with user-end distribution of non-compute-intensive activities and sensitive data. J Comp. Sci. Techn. 25, 1157–1167 (2010)

[7] Duipmans, E.F., Ferreira Pires, L., da Silva Santos, L.O.B.: Towards a BPM cloud architecture with data and activity distribution. In.: 16th IEEE EDOC Workshops, pp. 165–171. Helshinki, Finland (2012)

[8] Russell, N., ter Hofstede, A., van der Aalst, W., Mulyear, N.: Workflow control-flow patterns: a revised view. BPMcenter.org (2006)

[9] Russell, N., ter Hofstede, A.H.M., Edmond, D., van der Aalst, W.M.P.: Workflow data patterns: identification, representation and tool support. In: 24th ER, pp. 353–368. Springer, Netherlands (2005)

[10] Russell, N., Aalst, W., Hofstede, A.: Workflow exception patterns. In.: Advanced Information Systems Engineering. LNCS, vol. 4001, pp. 288–302. Springer, Heidelberg (2006)

[11] W. Ruh, F.M., Brown, W.: Basic Building Blocks. In: Enterprise application integration: a Wiley tech brief, pp. 39–60. John Wiley & Sons (2002)

[12] Venezian Povoa, L.: An approach to business processes decomposition for execution in cloud environments. (Uma abordagem para a decomposição de processos de negócio para execução em nuvens computacionais). Master Thesis. Federal University of São Carlos – Department of Computer Science. (forthcoming).

[13] Polyvyanyy, A., García-Bañuelos, L., Fahland, D., Weske, M.: Maximal Structuring of Acyclic Process Models. The Computer Journal. 57(1), pp. 12-35. (2014).

[14] Oosterwijk, H.: PACS fundamentals. O Tech Incorporated (2004)

[15] Nanda, M. G., Chandra, S., Sarkar, V.: Decentralizing execution of composite web services. SIGNPLAN Notices. 39, 170-187 (2004)

[16] Kopp, O., Khalaf, R., Leymann, F.: Deriving Explicit Data Links in WS-BPEL Processes. In.: 2008 IEEE SCC, pp. 367-376. IEEE Computer Society, Honolulu, Hawaii (2008)

[17] Duipmans, E.F., Ferreira Pires, L., da Silva Santos, L.O.B.: A transformation-based approach to business process management in the cloud. Journal of Grid Computing. 1–296 (2013)

[18] Fdhila, W., Yildiz, U., Godart., C.: A Flexible Approach for Automatic Process Decentralization Using Dependency Tables. In.: 7th IEEE ICWS, pp. 847-855. Miami, Florida (2009)