Towards Pattern-Based Service Identification

Veronica Gacitua-Decar and Claus Pahl

School of Computing, Dublin City University, Dublin 9, Ireland
vgacitua|cpahl@computing.dcu.ie

Abstract. Service-Oriented Architecture is a promising architectural approach to solve the integration problem originated by business process integration and automation requirements. The identification of the adequate services for the service architecture solution is a critical issue. Architecture abstractions, such as patterns, can capture design knowledge and allow the reuse of successful applied designs. The continual rise of abstraction in software engineering approaches have been a central driver of this work, placing the notion of patterns at business domain level. In this paper we propose a set pattern-based techniques for service identification. Graph-based pattern matching and pattern discovery are proposed to recommend the scope and granularity of services on process-centric description models. Matching of generalised patterns and hierarchical matching are discussed.

1 Introduction

Nowadays, evermore organizations are taking advantage of consolidating relations with service provider companies in order to improve competitiveness. This involves the merging of internal processes from provided and provider companies into inter-organisational processes shaped by a business chain value [1]. At technical level, business process integration creates an Enterprise Application Integration (EAI) problem.

Service-Oriented Architecture (SOA) is a promising architectural approach to solve the EAI problem. The definition of the services that will be the building blocks of the architecture solution is a critical issue. *Abstraction* is a principle that can address this challenge. Architecture abstractions like patterns and styles can capture design knowledge and allow the reuse of successfully applied designs and improve the quality of software [2]. Abstraction in software engineering approaches is a central driver; at the business level the reuse of successfully business designs is equally important.

Service identification is a central activity during the design of service architecture solutions. It involves the analysis of business models and their relation with the existing software support [3]. Existing software support might be implemented as services, or most frequently, as legacy applications. Thus, service identification might involve the *discovery* of existent services, adaptation of those services, or the *definition of new services*. Regarding service discovery -ideally- service requesters can find completely compatible services. However, in some practical scenarios, services that partially fulfill a request might also be of interest. A black box view of services is not sufficient, and considering structural and behavioural information beyond the signatures and effects of services can support a more flexible service discovery. Reuse of services within the limits of one organisation and its partners, providers and clients in close cooperation can be potentiated by planning in advance the services that will be available. In this manner, *reuse* of services is emphasised at design time - before implementation. This is specially relevant for large organisations where overlapping functionality offered by different services can rapidly grow, overshadowing the benefits of service reuse.

A number of contributions have addressed the problem of service identification. High level guidelines for the design of new services such as in [3] are useful, however they lacks of formality and techniques promoting automation that can be finally materialised as tool support. More specific approaches for service discovery, for example, based on matching of process-centric service descriptions such as in [4],[5],[6] goes in the line of automating the service discovery process. Architecture abstractions in the form of patterns has been exploited at technical level to improve the quality of software [2]. Less explored is the use of patterns at business level and their subsequent refinement to more technical levels. In this paper we present a set of pattern-based techniques and algorithms that focus on the identification of boundaries on process-centric models that recommend the scope and granularity of new services. Note that service discovery has not directly addressed here, however the proposed algorithms and related concepts could contribute to graph-based techniques for exact and partial service matching such as for example the work in [7].

– The definition of *new business-centric services* is addressed by means of *structural matching* between process patterns and process models. *Hierarchical* matching allows incremental levels of abstraction of process-centric matched patterns. Controlled vocabulary of business domains is considered by the matching of *generalised patterns*. *Partial* pattern matching provides flexibility to the proposed techniques.
– The other technique presented here, exploits the fundamental principle of *reuse* in software design. The intuitive idea is to find *frequent* process substructures -named *utility patterns*- within large process models. Process steps related with discovered utility patterns might be supported by existing software components, which can be rationalised, and subsequently encapsulated as reusable technical-centric services.

The remainder of this paper is organised as follows. Section 2 introduces a graph-based representation of process models and its relation with process patterns. Section 3 describes the different aspects of the process pattern matching problem and our proposed solutions. Section 4 describes our initial proposal for finding utility patterns in process models. Section 5 provides a preliminary evaluation of the proposed exact and partial pattern matching techniques. Finally, in sections 6 and 7 a review of the related work and conclusions are provided.

2 Graph-based representation of Business Process Models and Business Process Patterns

Graphs emerge as a natural representation of process-centric models [8],[9]. Graphs can capture both structure and behaviour, and allow abstractions such as patterns to be related to process-centric models .

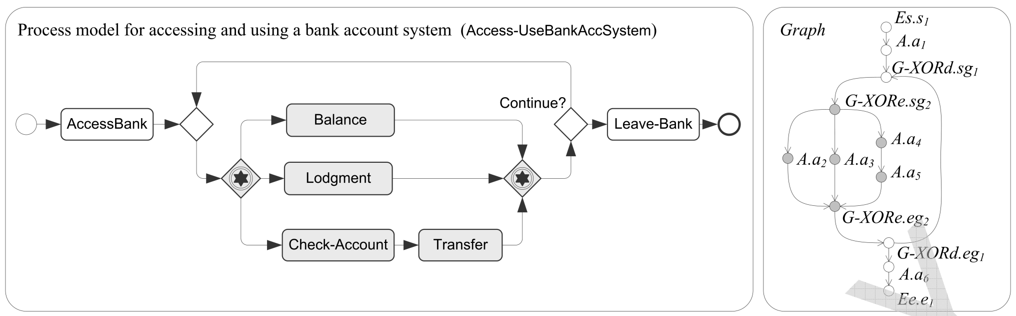

Fig. 1. Process model annotated with BPMN and a related graph-based representation.

2.1 Structural Representation of Business Process Models as Graphs

In the context of this paper we use graphs to represent the structure of process models and process patterns. Graph vertices represent process elements such as activities, control flow elements, and so on. Graph edges represent the connectivity between process elements. Section 8 (annex) provides an introductory background on graphs and related notation that is used in this section and referred to the rest of the paper.

Graph-based business process model. Let the graph $PM = (V_{PM}, E_{PM}, \ell_{V_{PM}}, \ell_{E_{PM}})$ be a finite, connected, directed, labelled graph representing a business process model. V_{PM} is the set of vertices representing process elements and E_{PM} is the set of edges representing *connectivity* between process elements. The function $\ell_{V_{PM}} : V_{PM} \rightarrow L_{V_{PM}}$ is the function providing labels to vertices of PM, and $\ell_{E_{PM}} : E_{PM} \rightarrow L_{E_{PM}}$ is the function providing labels to edges of PM. $L_{V_{PM}}$ and $L_{E_{PM}}$ are the sets of labels for vertices and edges, respectively.

Note that in this paper *connectivity* between process elements is simplified by considering only the sequence flows between activities since we focus on *structural* matching of patterns on process. A more complete approach could capture on edges: inputs, outputs, pre and post conditions regarding execution of activities. The Fig. 1 provides an example of a intuitive graph-based representation of a business process model annotated with a well-known process modelling notation, i.e. Business Process Modelling Notation[1] (BPMN). An appropriate mapping function maps descriptions of process elements with graph labels. Note that similar graph-based models can represent executable processes described for instance in the standard WS-BPEL language[2].

2.2 Structural Representation of Business Process Patterns as Graphs

Business Process (BP) patterns are essentially common connectivity patterns in process models. BP patterns can be operator-oriented, e.g. a multi-choice pattern that allows the selection of a number of options instead of an exclusive selection based on the basic choice operator. These kind of process patterns are know in the literature as *workflow patterns* [10]. Other category of BP patterns consists of application context-oriented and

[1] Available from http://www.bpmn.org/Documents/BPMN 1-1 Specification.pdf
[2] Available from http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html

often more complex patterns derived from and specific to the business context. These kind of BP patterns can represent well-known process building blocks in reference models, abstracting a set of connected activities required to reach some business goal [11]. Application context-oriented business process patterns can be reused as previously implemented and successful designs and provide an integrated vision of processes among different participants. For instance, in Fig. 1 the Use-AccessBankAccSystem process has at its core, in gray colored vertices, a common set of account usage activities that can be represented in the form of a application-context oriented process pattern.

Beyond the previous types of BP patterns, a third category represent frequent process connectivity structures that are not specific to a business domain, but relates to some standard technology solution, for instance a typical authentication and authorisation processes to access a system. We named this patterns as *utility* patterns, borrowing the name from the definition of *utility* services in [3]. In the rest of the paper we will refer to *application context-oriented business process patterns* only as *patterns*. *Workflow patterns* are not addressed here. *Utility patterns* are the focus of section 4.

Graph-based business process pattern. Let the graph $PP = (V_{PP}, E_{PP}, \ell_{V_{PP}}, \ell_{E_{PP}})$ be the finite, connected, directed, labelled graph representing a business process pattern model. Elements of V_{PP} represent process pattern roles and elements in E_{PP} represent connectivity between pattern roles. Note that the graph-based representation for business patterns, utility patterns and business processes is structurally the same.

2.3 Instantiation of Process Patterns in Process Models

Process patterns have been described in the same way as process models. Now, we discuss the relation between process patterns and process models. In particular, we are interested in the abstraction that patterns represent for process models and concretely, in the notion of *instantiation* of process patterns in process models. *Pattern Instantiation* in a concrete model indicates that the structural relations described in the pattern hold in the model. The structural preserving relations that graph homomorphisms represent help us to capture the notion of pattern instantiation. In particular, instantiation of a BP pattern in a process model can be captured by the definition of a *locally surjective graph homomorphism* [12] between a subgraph PM_S of the graph process model PM and the pattern graph PP, i.e. $PM_S \xrightarrow{S} PP$. *Surjection* allows that several process elements (vertices of PM) play the role of one pattern element (vertex of PP). Moreover, model elements can belong to more than one pattern when considering this approach. Note that we have used the notation from the previous section and the Annex. We will continue using this notation along the paper.

3 Process Pattern Matching

We have discussed in Section 1 the potential that discovering instances of patterns in concrete models can provide to the definition of new services. Matching a pattern in a concrete model involves the identification of instances of that pattern in the concrete model. In this manner, the *pattern matching problem* can be referred as the detection of a graph homomorphism between the graph representing a concrete model and the graph representing the pattern.

3.1 Exact, Inexact and Partial Pattern Matching

In realistic scenarios where an *exact* match of a pattern is unlikely, *partial* and *inexact* matching become relevant. *Inexact pattern matching* provides *good*, but not exact solutions to the matching problem. In this case, pattern instances can incorporate additional elements not described in the pattern, nevertheless they must not affect the structural properties of the pattern. *Partial pattern matches* identify exact but *incomplete* matches of patterns. Partial instances of patterns might exist due to a modification or evolution of a previously instantiated pattern. However, when patterns have not previously considered as part of the design, partial matches indicate an opportunity to improve the design through incorporating the whole pattern. *Partial* and *inexact* matches are also important due to the fact that process models and their implementations as services might be highly similar but not exactly the same from organisation to organisation and to identify commonalities can save costs and encourage reuse.

In order to formalise and later on implement our proposed techniques as concrete tool support we will define exact, partial and inexact pattern matching in terms of the graphs representing processes and patterns and their structural relations. Formalisation can provide guaranties of correctness and improve the confidence on tools.

Exact Pattern Matching. A exact pattern match of a specific pattern PP in an arbitrary process model PM refers to the detection of $PM_S \xrightarrow{S} PP$ with $PM_S \subseteq PM$. The mapping function φ defines an individual instantiation of the pattern PP in the process model PM with $\varphi : V_{PM_S} \rightarrow V_{PP}$ satisfying that for all $u \in V_{PM_S} : \varphi(N_{PM_S}(u)) = N_{PP}(\varphi(u))$ and with mapping $\lambda_S : L_{V_{PM_S}} \rightarrow L_{V_{PP}}$ a bijective function indicating a semantic correspondence between the labels of two mapped vertices.

Partial pattern matching. Partial matches restrict the matching problem allowing incomplete matches. Incomplete pattern matches maps elements from PM to a reduced number of elements considered in the original codomain (V_{PP}). In this manner, the original function φ defined by the exact matching case is now restricted to the function $\varphi_{PARTIAL} : V_{PM_{S^*}} \rightarrow V_{PP_{PARTIAL}}$ satisfying that for all $u \in V_{PM_{S^*}}$:
$\varphi_{PARTIAL}(N_{PM_{S^*}}(u)) = N_{PP_{PARTIAL}}(\varphi_{PARTIAL}(u))$ with $PP_{PARTIAL} \subseteq PP$ and $PM_{S^*} \subseteq PM_S$.

Inexact pattern matching. Inexact pattern matching relaxes the definition of *neighborhood* in the Annex (Section 8) by a set $N^*_{PM_S}(u)$ allowing other vertices not only in the neighborhood of a vertex u ($N_{PM_S}(u)$) but also in the *path* between u and v with $\varphi(u)$ adjacent with $\varphi(v)$ and $\varphi : V_{PM_S} \rightarrow V_{PP}$.

Algorithm for Exact and Partial Matching. We propose an algorithm for exact and partial process pattern matching. The pseudo-code of the proposed algorithm is described in **ALGORITHM 1** (uEP-PMA). The algorithm starts matching each vertex in V_{PP} with vertices in V_{PM} such that the labels in $L_{V_{PP}}$ are semantically correspondent with labels in

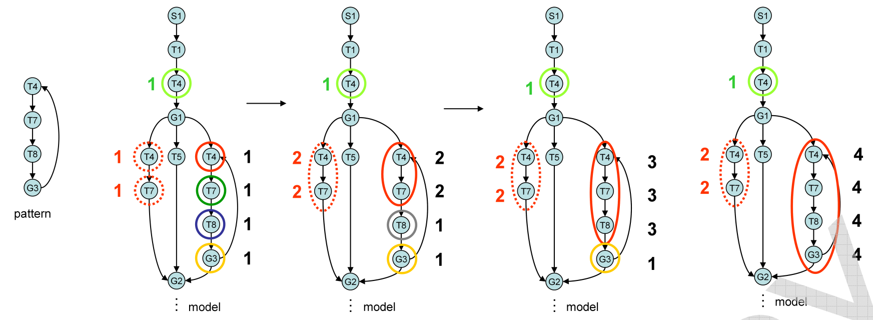

Fig. 2. Matching expansion steps. One exact match and two partial matches are found.

$L_{V_{PM}}$. Semantic correspondence in *u*EP-PMA refers to a one to one (bijective) mapping λ between a subset in $L_{V_{PM}}$ of giving labels to matched vertices in V_{PM} and labels in $L_{V_{PP}}$. Each initial match is considered a temporal pattern matching defining a temporal subgraph in *PM* that we denote as *tPM*. Subsequently, *tPM* is expanded until all its neighbors that hold a structural relation defined by φ or at least $\varphi_{PARTIAL}$ are added.

Fig. 2 illustrates the expansion steps. The algorithm terminates when no more expansion steps can be done. The result is a *score* vector. Each vertex in *PM* has a score that indicates the number of vertices of the matched pattern to which it belongs.

Algorithm 1: *u*EP-PMA - *u*ndirected Exact and Partial - Pattern Matching Algorithm.

Input: Target Graph (*PM*), Pattern Graph (*PP*)
Output: Score Vector (*score*).

```
1 :  For each vertex $m$ in $V_{PM}$ do
2 :    For each vertex $p$ in $V_{PP}$ do
3 :      If $\lambda \circ \ell_{V_{PM}}(m) = \ell_{V_{PP}}(p)$ == true then
4 :        $tPM(m) \leftarrow$ initial temporal match centred in vertex $m \in PM$
5 :        $score \leftarrow 1$ (score for vertices in $tPM(m)$)
6 :      end if
7 :    end for
8 :  end for
9 :  Do while $ExpansionCondition$ == true
10 :   For each vertex $i \in tPM(m)$ do
11 :     If $\ell^{-1}_{V_{PP}} \circ \lambda \circ \ell_{V_{PM}}(N_{tPM(m)}(i)) = N_{PP}(\ell^{-1}_{V_{PP}} \circ \lambda \circ \ell_{V_{PM}}(i))$ && $N_{tPM(m)}(i) \notin tPM(m)$ then
12 :       Expand $tPM(m)$ with $N_{tPM(m)}(i)$
13 :       $score \leftarrow score + 1$
14 :       $ExpansionCondition \leftarrow$ true
15 :       Else if $ExpansionCondition \leftarrow$ false end else if
16 :     end if
17 :   end for
18 :  end do while
```

Note that several exact or partial instances of *PP* in *PM* might exist. If different pattern instances share edges in *PM*, we say that there are *overlaps* of the pattern *PP* in *PM*. The *u*EP-PMA algorithm identifies the connected subgraphs in *PM* containing overlaps as a one single subgraph PM_O. The score of the vertices in the overlap is the number of vertices in PM_O. Additionally, in order to consider the directionality of the graphs representing concrete models and patterns the *u*EP-PMA algorithm can also be performed on the *undirected* version of *PM* and *PP*. In this manner, matches not only considers vertices, but also arcs.

According to [13], for a connected simple graph H, the problem of detecting a locally surjective homomorphism between an arbitrary graph and H is solvable in polynomial time if and only if H has at most two vertices. In all other cases the problem is NP-complete. The complexity of the latter problem, which is directly related with the pattern matching problem, made us aware of performance issues. In Section 5 we show a preliminary evaluation where instances of specific graph patterns are identified on arbitrary random graphs. The results show that the time required to solve the problem is quadratic in relation to the size of the random graphs and it has a small constant that conveniently modulates the response time for small and medium size graphs. Scalability, in terms of processing several patterns over one or more target graphs, could be addressed by implementing a refined version of the algorithms to allow parallel processing of each pattern to be matched on a target model.

3.2 Matching of Generalised Patterns

Consideration of restricted vocabulary for different vertical business domains can add additional benefits for the practical use of BP pattern matching solutions. There are cases where descriptions of process elements (or pattern elements) have the same syntax, but different semantic and vice versa. Moreover, processes and patterns might be described with different structures, while they behave in the same way. Regarding the vocabulary used to describe process and pattern elements, we have extended the *u*EP-PMA algorithm with the *u*G-PMA algorithm allowing semantic correspondence beyond the one to one mapping (λ) previously considered. The structure of the algorithm remains relatively invariant, but the functions $\ell_{V_{PM}}$, $\ell_{V_{PP}}$ and λ are modified. In this case, the two $\ell_{(\cdot)}$ functions are mapping vertices from *PM* to labels that are organised in a tree-like structured taxonomy. The labels in the taxonomy refer to concepts from a particular business domain. In this manner, *generalised patterns* are considered as families of patterns where the parent pattern contains the roots of tree-structured taxonomies for business concepts. Child patterns contains one or more child concepts connected to root concepts in the hierarchy defined by the taxonomy. Note that using the *u*G-PMA algorithm requires the existence of an implemented taxonomy from where the algorithm can search for semantically correspondent terms.

3.3 Hierarchical Pattern Matching

In the previous sections we have addressed the exact and partial matching problem on flat process models (and patterns). However, processes and patterns are commonly composed by more fine-grained process-centric structures. In this section we outline a solution to the problem of pattern matching considering different levels of abstraction.

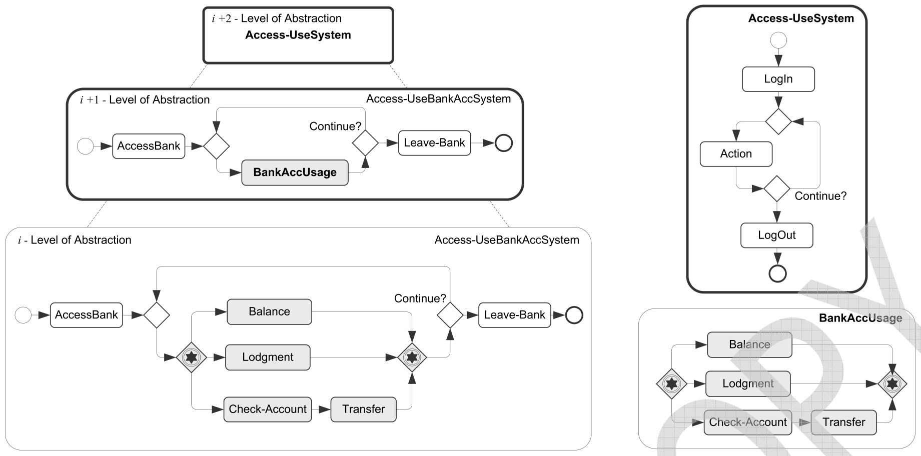

Fig. 3. Hierarchical pattern matching.

Algorithm for hierarchical pattern matching. The pseudo code of the proposed algorithm named *u*H-PMA is described in **Algorithm 2**. The algorithm starts matching at a certain level of granularity on a target model *PM* different patterns PP_j from a set of patterns *setPP*. The index *j* identifies an specific pattern in *setPP*. Subsequently, *PM* is transformed to an abstracted representation PM^i where *i* represent a particular level of abstraction. Subsequently, subgraphs of PM^i that have been matched with any PP_j are replaced by vertices $\mathtt{p}_j$ of type *pattern*[3] in the graph one level of abstraction up (PM^{i+1}). Thus, the complexity of a matched subgraph is hidden in a *pattern vertex* $\mathtt{p}_j$. Note that representative labels are assigned to pattern vertices. Once the target model is abstracted with pattern vertices from matched patterns at a specific level of abstraction, other patterns at a higher level might appear. In this way, the abstraction process can be performed iteratively, abstracting a process graph PM^i into a process graph PM^{i+1} which is one level of abstraction up, and so on. The algorithm terminates when no more matches are found or when the process graph has only one vertex.

The Fig. 3 illustrates the idea of hierarchical pattern matching making use of the process model from Fig. 1. The Fig. 3 shows two patterns: BankAccUsage and Access-UseSystem that are consecutively matched in two different levels of abstraction. The pattern BankAccUsage describes a set of common bank account usage activities and the pattern Access-UseSystem represents a typical -simplified- set of steps to access a generic system. Note that BankAccUsage is focused on the banking industry, however the Access-UseSystem pattern can be valid across different industries since it has a technology-oriented and business-agnostic nature [11]. The first match involves a mapping from elements in the process model Access-UseBankAccSystem to elements in the BankAccUsage pattern. The resultant abstracted process model is subsequently matched with the Access-UseSystem pattern. In this case, checking semantic correspondence is enhanced by using

[3] Typed graphs are graphs that holds a complete mapping to a set of types. Mappings for typed graphs can consider vertices and edges. The mapping function considered for pattern vertices is a global surjective function from the set of graph vertices to the set of types.

a taxonomy for business concepts. The result of the hierarchical pattern matching process is a single vertex referring to the access and use of a system.

Note that we have not addressed the problem of *overlaps* yet. How to abstract two matches that share vertices and edges in the target model? Our basic representation of processes and patterns as graphs restricts the possibility of representing two overlapped matched patterns as two different pattern vertices. One idea that we will explore is the representation of matched patterns as hyperedges of a hypergraph. The vertices of the hypergraph are the same vertices of the graph representing the process model.

ALGORITHM 2: *u*H-PMA - *u*ndirected Hierarchical - Pattern Matching Algorithm.

Input: Target Graph (*PM*), Set *setPP* of *n* pattern graphs ($setPP = \{PP_1, ..., PP_n\}$)
Output: *scoreMatrix*[4].

```
1 : Do while $IterationCondition \&\& change ==$ true
2 :   For each pattern $PP_j \in setPP$ do
3 :     uEP-PMA($PM^i, PP_j$) (or uG-PMA if generalised pattern matching is desired)
4 :     If $score(u) = |V_{PP_j}|$ with $u \in PM^i_{S_j}$ && $exact\ match ==$ true then
5 :       $PM^i_{S_j} \leftarrow \mathtt{p}_j$
5 :       $change \leftarrow$ true
5 :       If $|V_{PM^i}| <= 1$ then
5 :         $IterationCondition \leftarrow$ false
5 :       end if
6 :     end if
6 :     Else if
6 :       $change \leftarrow$ false
6 :       $i \leftarrow i + 1$
7 :   end for
8 : end do while
```

4 Discovering Frequent Utility Patterns in Process Models

Previous sections described techniques for identifying services based on the matching of known application context-oriented process patterns in process models. In this section we are interested in discovering frequently occurring substructures on large scale business process models. Process steps might be supported by existing software components and identifying reoccurring connected process steps provide a medium to define potential reusable software components as encapsulated services. The idea is to exploit the basic principle of reuse in SOA. Finding frequent -not necessarily known- *utility patterns* in large process models can help to the definition of reusable technical-centric services.

There are two distinct problem formulations for frequent pattern discovery in graphs: graph-transaction setting and single-graph setting [14]. The latter refers to the discovery of subgraphs that occur multiple times in a single input graph. The other refers to the

[4] *scoreMatrix* is a matrix where each element is a *score* vector derived from algorithm *u*EP-PMA. Rows in *scoreMatrix* refer to the level of granularity *i* of the model *PM* and the columns refer to the different matched patterns $PP_j \in setPP$.

discovery of subgraphs that occur frequently across a set of small graphs. We present an algorithm focused on single-graph setting scenario for pattern discovery in graphs.

Algorithm for Pattern Discovery. The algorithm attempts to find frequent -exact and partial- occurrences of subgraphs in a single input graph PM. A discovered frequent subgraph -utility pattern- is an induced subgraph PP_U homomorphic with all occurrences of a frequent subgraph of PM. Homomorphism detection in the proposed algorithm (named *u*EP-FPDA) relies on the pattern matching algorithm *u*EP-PMA described in Section 3.1. The pseudo code of *u*EP-FPDA is described in ALGORITHM 3.

ALGORITHM 3: *u*EP-FPDA - *u*ndirected Exact and Partial - Frequent Pattern Discovery Algorithm.

Input: Target Graph - undirected version (*uPM*), Threshold (Th), number of expansion steps (k)
Output: *score*, *FreqM*

```
1 :  For each vertex $u$ in $uPM$ do
2 :    $PP_{pivot(u,1)} \leftarrow u$
3 :    $seeds_{(u,1)} \leftarrow$ uEP-PMA($PM$, $PP_{pivot(u,1)}$)
4 :    $score_{(u,1)} \leftarrow seeds_{(u,1)}$
5 :    For each $i$ in $seeds_{(u,1)}$ do
6 :      If $score_{(u,1)}(i)/|PP_{pivot(u,1)}| >= Th$ then
7 :        $cnt_{(u,1)} \leftarrow cnt_{(u,1)} + 1$
8 :      end if
9 :    end for
10 :   $FreqM(u,1) \leftarrow cnt_{(u,1)}/|PP_{pivot(u,1)}|$
11 :   If $k >= 1$ do
12 :     For $j : 2 \rightarrow k$
13 :       $PP_{pivot(u,j)} \leftarrow expand(PP_{pivot(u,j-1)})$
14 :       $seeds_{(u,j)} \leftarrow$ uEP-PMA($PM$, $PP_{pivot(u,j)}$)
15 :       $score_{(u,j)} \leftarrow seeds_{(u,j)}$
16 :       For each $i$ in $seeds_{(u,j)}$ do
17 :         If $score_{(1)}(u,j)/|PP_{pivot(u,j)}| >= Th$ then
18 :           $cnt_{(u,j)} \leftarrow cnt_{(u,j)} + 1$
19 :         end if
20 :       end for
21 :       $FreqM(u,j) \leftarrow cnt_{(u,j)}/|PP_{pivot(u,j)}|$
22 :     end for
23 :   end if
24 : end for
```

The size of the induced subgraphs and a parameter that relaxes the way of counting the frequency of occurrences of induced subgraphs are parameterised in k and Th, respectively. The constant k refers to the amount of times that an initial arbitrary subgraph in PM will be *expanded* and compared with other subgraphs in PM to check for homomorphisms. Th refers to a threshold for the ratio between the number of vertices of two non exact occurrences of PP_U. If Th is equal to one, the frequent occurrences of subgraphs in PM must to be isomorphic between them. The output of *u*EP-FPDA are two matrices *score* and $FreqM$. In the matrix *score* rows represent each vertex u in PM and columns the results for different size of pattern. If u belongs to a highly frequent

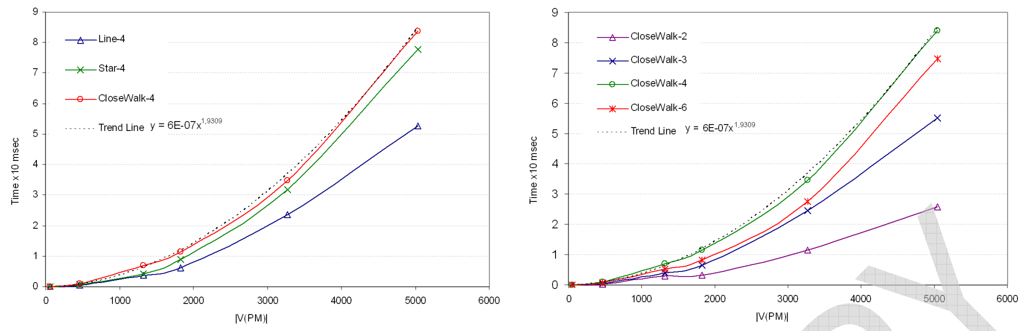

Fig. 4. Average response time of uEP-PMA on arbitrary random graphs for different pattern structures (left side) and different pattern sizes (right side).

subgraph in PM of size j then $score(u, j)$ will be also high. $FreqM$ is a matrix with $|V_{PM}|$ rows and k columns, where each cell indicates the frequency of a discovered pattern centred the vertex indicated by the row and with size indicated by k.

The uEP-FPDA algorithm starts defining an arbitrary vertex u from the target graph PM as the first temporal pattern (pivot pattern or $PP_{pivot(u,1)}$) and matching $PP_{pivot(u,1)}$ with the rest of the target graph. The matrices $score$ and $FreqM$ are initialised with the results of the matching for the initial pattern of size 1. The next steps are repeated for each vertex in PM. The subgraph $PP_{pivot(u,1)}$ is expanded with its neighbors, together with expanding each of the vertices in PM whose were matched with the initial $PP_{pivot(u,1)}$. These first matched vertices are called $seeds_{(u,1)}$. The algorithm continues the expansion of PP_{pivot} while checking if there exist an homomorphism between the expanded PP_{pivot} and subgraphs in PM. The counting for measuring the frequency of the matched subgraphs -expanded $seeds$- depends on the satisfaction of the threshold Th parameter. The expansion process continues until k times or no more homomorphisms are detected. The results contained in $score$ and $FreqM$ indicates the set of induced subgraphs PP_U -discovered utility patterns- centred in the initial $seeds$ and the PP_{pivot}.

Based on the results obtained in the preliminary evaluation (Section 5) indicating the quadratic complexity order of uEP-PMA, it is expected that for uEP-FPDA the complexity order grown up to $O(kV^3)$, where V the size of the problem in term of the number of vertices and k the number of times the temporal patterns in uEP-FPDA is expanded.

5 Evaluation

We have performed a preliminary evaluation for the exact and partial matching algorithm (uEP-PMA). The experiments consider seven specific patterns over arbitrary random graphs with approximate sizes of 60, 450, 1300, 1800, 3200 and 5000 vertices. The experiments were run in a Intel machine 2 GHz and 2GB RAM on WinXP-SP3. Labels in patterns and random graphs can be of three different types: A, B or C. The used patterns are a four close-walk of 2, 3, 4 and 6 vertices; two line-like patterns of 3 and 4 vertices and a star-like pattern with 4 vertices.

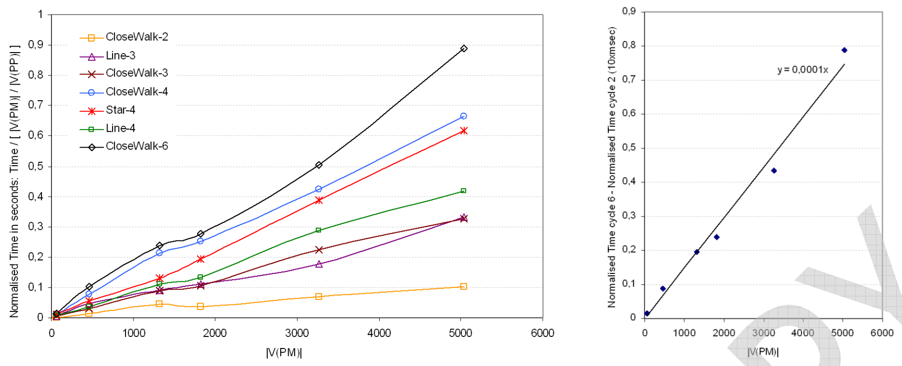

Fig. 5. Average response time of *u*EP-PMA algorithm on arbitrary random graphs for matching a star-like pattern, a line-like pattern and a pattern with a close-walk structure.

The Fig. 4-left side shows the average response time of *u*EP-PMA for the matching of three patterns with different structures and the same number of vertices on arbitrary random graphs. The line-like pattern requires less time in comparison with the star-like and close-walk patterns, providing an indication that the structure of the matched patterns influence the response time. The right side of the Fig. 4 shows the average response time of *u*EP-PMA on arbitrary random graphs for a same pattern and different number of vertices. The number of vertices of the pattern also influence the time response. In order to visualise the trend of the time response more clearly, we divided the time that the algorithm requires to compute a solution by the ratio between the number of vertices in the random graph (target graph) and the number of vertices in the pattern. The Fig. 5 - left side shows the trend of the normalised response time for all the different patterns considered in the experiment. The right side of the Fig. 5 illustrates the trend of the normalised time response for two patterns with different number of vertices. The trend lines in the two graphics of the Fig. 4 indicate that the time to solve the problem increase quadratically with the number of vertices on the target graph. The constant 6^{-7} suggest advantageous performance characteristics regarding the response time of the algorithm for small and medium size graphs.

6 Related work

Matching of process-centric descriptions is an activity in the context of service discovery and modelling of new services are activities. Patterns provide a notion of abstraction in models, and they can play a role in the reuse of previously implemented designs. A number of papers have proposed graph-based approaches for matching of processes and patterns. In [5] a technique for partial matches on behavioral models is presented. The proposal provides measures of semantic distance between resultant matches and user requirements. Several issues regarding complexity of the proposed algorithm are reported to be improved. However, the experimental results indicate a response time of approximately thirty seconds for a target graph of fifty vertices, which can be prohibitive for large processes. In [15] a method to measure distance between process definitions of web services is presented. The method relies on a distance measure of normalised matrices representing graph-based process models. The proposed normalised matrices lack of flexibility in relation with chosen data structure. Optimisations on the computation of the normalised matrices, e.g. considering a more efficient data structure such as sparse matrices, is not mentioned. In [7] various types of structural matches for BPEL processes supporting dynamic binding of services are defined. BPEL processes are modelled as process trees where each tree node is an interaction. Activities which are not interactions are abstracted into internal steps and can not be matched. Duplicate interaction activities are not allowed in the tree. *Plugin* matching is presented as an approach based on a process simulation notion, however such as the authors indicate, the proposal requires further semantic analysis to decide if a process can replace other after a matching. In [6] the authors propose a measurement to compare two process models based on their observed behavior. Observed behavior is restricted to logs of process executions. Mining techniques are only applied over sequences of process steps rather than graphs representing process models. In [16] an best-effort method to exact and partial pattern matching is presented. The results of a matching process are presented to users ordered according to a proposed *goodness* measurement. The proposed method finds partial subgraphs in time linear on the size of the target graph. However, the pattern queries are limited in size and structure, and attributes or labels on edges are not considered. Neither overlaps nor hierarchical matching are considered.

7 Conclusion

In this paper we have discussed the benefits, the considerations and some possible solutions for a pattern-based approach for service identification. In its core, the approach uses a set of graph-based pattern matching algorithms. We discussed some considerations for exact, inexact, partial, generalised and hierarchical pattern matching. We provided a solution for exact and partial matching (*u*EP-PMA algorithm). We extend *u*EP-PMA by adding hierarchical pattern matching with the *u*H-PMA algorithm, and outlined a proposal for matching of generalised patterns (*u*G-PMA.). A solution for discovering frequent pattern in graphs (*u*EP-FPDA) was proposed. The solution attempts to discover utility patterns, which could provide a recommendation for designing new reusable technical-centered services.

Our initial motivation for this work was based on the potential benefits that pattern matching and pattern discovery techniques could provide to business analysts and architects during the definition of new services based on the analysis of process-centric models. Process models could be annotated with matches of process patterns and presented to the designers on standard modelling tools. This investigation assume the availability of process models or process-centric service descriptions and related patterns. The availability of process documentation - and with a unique type of process description- might be thought as quite difficult to find in real cases. However, we believe that business and architectural documentation in the form of process-centric models is becoming more and more relevant in the context of service architecture implementations. Models documenting real case scenarios are complex, numerous and often large. Thus, our proposal attempts to support designers by automating some of the steps during the analysis and design activities of business process models and process-centric service architectures descriptions.

We believe that architecture abstractions, such as patterns, are a powerful concept that can be exploited to improve the design of new services and pattern matching techniques can help with the discovery of already implemented services. Further work regarding performance and scalability of our proposed algorithms is in development. We plan investigate their applicability to dynamic service composition.

References

1. Abramovsky, L., Griffith, R.: Outsourcing and offshoring of business services: How important is ICT? J. of the European Economic Association **4**(2-3) (2006) 594–601
2. Buschmann, F., Henney, K., Schmidt, D.C.: Pattern-Oriented Software Architecture: On Patterns and Pattern Languages. Wiley and Sons (2007)
3. Erl, T.: Service-oriented architecture: Concepts, Technology, and Design. Prentice Hall (2004)
4. Wombacher, A., Rozie, M.: Evaluation of workflow similarity measures in service discovery. In Schoop, M., Huemer, C., Rebstock, M., Bichler, M., eds.: Service Oriented Electronic Commerce. Volume 80. GI (2006) 51–71
5. Corrales, J., Grigori, D., Bouzeghoub, M.: Bpel processes matchmaking for service discovery. In: On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE. (2006) 237–254
6. Aalst, W.M.P.v.d., Medeiros, A.d., Weijters, A.: Process equivalence: Comparing two process models based on observed behavior. In: Business Process Management. (2006) 129–144
7. Eshuis, R., Grefen, P.: Structural matching of bpel processes. In: Fifth European Conference on Web Services. ECOWS07. (2007) 171–180
8. Ehrig, H., Engels, G., Kreowski, H.J., Rozenberg, G., eds.: Handbook of Graph Grammars and Computing by Graph Transformation, volume 2: Applications, Languages and Tools. World Scientific (1999)
9. Sadiq, W., Orlowska, M.E.: Analyzing process models using graph reduction techniques. Information Systems **25**(2) (2000) 117–134
10. Aalst, W.M.P.v.d., Hofstede, A.H.M.t., Kiepuszewski, B., Barros, A.P.: Workflow patterns. Distributed and Parallel Databases **14**(1) (2003) 5–51
11. Fettke, P., Loos, P.: Reference Modeling for Business Systems Analysis. IGI Publishing (2006)
12. Fiala, J.: Structure And Complexity of Locally Constrained Graph Homomorphisms. PhD thesis, Charles University, Faculty of Mathematics And Physics (2007)
13. Fiala, J., Kratochvíl, J.: Locally constrained graph homomorphisms–structure, complexity, and applications. Computer Science Review **2**(2) (2008) 97–111
14. Michihiro, K., George, K.: Finding frequent patterns in a large sparse graph*. Data Min. Knowl. Discov. **11**(3) (2005) 243–271
15. Bae, J., Liu, L., Caverlee, J., Rouse, W.B.: Process mining, discovery, and integration using distance measures. In: Proc. IEEE International Conference on Web Services (ICWS'06), IEEE Computer Society (2006) 479–488
16. Tong, H., Faloutsos, C., Gallagher, B., Eliassi-Rad, T.: Fast best-effort pattern matching in large attributed graphs. In: Proc. 13th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining. KDD07, San Jose, California, USA, ACM (2007) 737–746
17. Hell, P., Nesetril, J.: Graphs and homomorphisms. Oxford Lecture Series in Mathematics and Its Applications **28** (2004)

8 Annex: Graphs

This annex is based on Nešetřil, Fiala and Hell's work [17],[13], [12].
A ***graph*** G is a set V_G of vertices together with a set E_G of edges, where each edge is a two-element set of vertices. If V_G is finite, the graph G is called a ***finite*** graph. If the graph has orientation, it is called ***directed*** graph, and each edge is called an *arc*. An arc can have one of the two orientations (u,v) or (v,u) with $u,v \in V_G$. If loops on vertices are allowed, then edges consist of only one vertex, written (u,u) with $u \in V_G$. A sequence of vertices of a graph G, such that the consecutive pairs are adjacent, is called a *walk* in G. If all vertices in a walk are distinct, then it is called a *path*. A graph G is called a ***connected*** graph if for every pair of vertices $u,v \in V_G$ there exists a finite path starting in u and ending in v. For a vertex u in a graph G, the set of all vertices adjacent to u are called the ***neighborhood*** of u and is denoted by $N_G(u)$, with $N_G(u) = \{v|(u,v) \in E_G\}$. Consequently, a vertex v is a neighbor of u if u and v are adjacent. A graph G is a ***subgraph*** of H if $V_G \subseteq V_H$ and $E_G \subseteq E_H$.

Homomorphisms. Graph homomorphisms are edge preserving vertex mapping between two graphs. A *graph homomorphism* from G to H denoted by $G \to H$ is a vertex mapping $f : V_G \to V_H$ satisfying $(f(u), f(v)) \in E_H$ for any edge $(u,v) \in E_G$. According to [13], whenever a homomorphism $G \to H$ is hold, then the image of the neighborhood of a vertex from the source graph V_G is contained in the neighborhood of the image of that vertex in the target graph V_H, i.e. $f(N_G(u)) \subseteq N_H(f(u))$ for all $u \in V_G$. Composition of two homomorphisms $f : F \to G$ and $g : G \to H$ is another homomorphism $g \circ f : F \to H$. If a homomorphism $f : G \to H$ is an one-to-one mapping and f^{-1} is also a homomorphism, then f is called an ***isomorphism***. In such a case is said that G and H are isomorphic and it is denoted by $G \simeq H$. An isomorphism $f : G \to G$ is called an automorphism of G, and the set of all automorphisms of G is denoted by $AUT(G)$. Using the latter notation, for graphs G and H three kind of homomorphic mapping are defined as:
- $G \xrightarrow{B} H$ if there exist a *locally bijective homomorphism* $f : V_G \to V_H$ that satisfies for all $u \in V_G : u \in V_G : f(N_G(u)) = N_H(f(u))$ and $|f(N_G(u))| = |N_G(u)|$.
- $G \xrightarrow{I} H$ if there exist a *locally injective homomorphism* $f : V_G \to V_H$ that satisfies for all $u \in V_G : |f(N_G(u))| = |N_G(u)|$.
- $G \xrightarrow{S} H$ if there exist a *locally surjective homomorphism* $f : V_G \to V_H$ that satisfies for all $u \in V_G : f(N_G(u)) = N_H(f(u))$.

Note that for the mappings above, locally bijective homomorphism is both locally injective and surjective. The mappings are also known in the literature as (full) covering projections (bijective), or as partial covering projections (injective), or as role assignments (surjective). Additionally, any locally surjective homomorphism f from a graph G to a connected graph H is globally surjective, and any locally injective homomorphism f from a connected graph G to a forest H is globally injective [12].

Labelled Graphs. The graph $G = (V_G, E_G, \ell_{V_G}, \ell_{E_G})$ is a graph where the vertices in V_G and edges in E_G have labels. The functions assigning labels to vertices and edges are surjective homomorphisms $\ell_{V_G} : V_G \to L_{V_G}$ and $\ell_{E_G} : E_G \to L_{E_G}$ for all the vertices in V_G and the edges in E_G, respectively. L_{V_G} and L_{E_G} are the sets of vertex labels and edge labels, respectively. Note that surjection allow the existence of a same label in $L_{V_G}(L_{E_G})$ for several vertices(edges).