# Fast Pattern Matching in Conceptual Models – Evaluating and Extending a Generic Approach

Hanns-Alexander Dietrich, Matthias Steinhorst, Jörg Becker, Patrick Delfmann

European Research Center for Information Systems
University of Muenster
Leonardo-Campus 3
48149 Münster
{dietrich | steinhorst | becker | delfmann}@ercis.uni-muenster.de

**Abstract:** Identifying structural patterns in conceptual models serves a variety of purposes ranging from model comparison to model integration and exploration. Although there are a multitude of different approaches for particular modelling languages and application scenarios, the modelling community lacks an integrated approach suitable for conceptual models of arbitrary languages and domains. Therefore, a generic set-theory based pattern matching approach has recently been developed. To prove that this approach is beneficial in terms of performance, we conduct a statistically rigorous analysis of its runtime behaviour. We augment the original approach to include a caching mechanism that further increases performance. We are able to show that the original algorithm is able to identify arbitrary patterns within milliseconds. The caching extension further increases performance by up to fifty per cent given the model base and patterns we used.

## 1 Introduction

Structurally analysing conceptual models serves a wide variety of purposes. In the domain of Business Process Management (BPM) it helps detect process improvement potential [VTM08]. In case of mergers and acquisitions, identifying structural patterns in process models allows for comparing heterogeneous process landscapes to one another. Such a comparison can then be used to integrate various process models containing similar or equivalent structures [Di09]. Identifying patterns in conceptual models is also useful for business process compliance checking [Kn10] as well as determining the syntactical correctness of a particular model [Me07]. In the domain of database engineering, detecting structures in conceptual models addresses the problem of schema matching and integration [PS11; RS10]. In all of these application scenarios a manual analysis is extremely costly, since the number of models to be analysed may range in the thousands [YDG10]. Furthermore, each model may contain hundreds of elements. Structural pattern matching can therefore only be beneficial if it is conducted in an automated or at least semi-automated way. To that end, we have developed a generic set-theory based pattern matching approach that is able to find patterns in conceptual models of arbitrary modelling languages [De10].



79

The approach is based on the idea that any graph-based conceptual model can be regarded as the set of its objects and relationships. The approach was prototypically implemented as a plug-in for a meta-modelling tool that was available from a previous research project [DK07]. To prove that our approach is beneficial in terms of performance, we conduct a statistically rigorous analysis of its runtime behaviour. We evaluate the approach on two sets of Event-driven Process Chains (EPC) [Sc00] that were available from previous research projects as well. In total, we analyse process models having between 20 and 343 elements. The patterns we search for are based on the works of [Me07] who defines syntactical errors in EPCs and [Va03] who identify workflow patterns. We are able to demonstrate that the generic set-theory based approach allows for finding structural pattern matches within milliseconds. We augment the approach to include a caching algorithm that further improves matching performance by up to fifty per cent given the model base and patterns we used. The algorithm is based on the idea of storing frequently used sub-patterns in a data structure that allows access in constant time.

The remainder of the paper is structured as follows. In Section 2, we discuss related work. Our analysis indicates that there is no generic pattern matching algorithm that finds matches within milliseconds. In Section 3, we briefly elaborate on the approach to be evaluated. We continue by explaining the caching mechanism augmenting the original algorithm. In Section 4, we provide a detailed performance analysis of the approach both with and without the caching algorithm. We conclude by summarizing our main findings and providing an outlook to future research.

## 2 Related Work

Identifying structural patterns in conceptual models serves multiple purposes in the fields of database engineering and business process management. To identify relevant literature, the keywords ‘schema matching’, ‘business process’, ‘similarity’, ‘merge’, ‘check ’, ‘comparison ’, ‘compliance’, and ‘exploration’ were used in combination with Google Scholar. Primarily, recent literature from the last five years was analysed. We furthermore included some sources from unstructured search.

In the field of database engineering, various approaches have been proposed addressing the problem of schema matching and integration. Two or more schemas are taken as input and semantically as well as structurally equivalent elements are identified to create an integrated schema as output [RB01]. Some of these approaches assume the underlying schemas to exhibit a tree-like structure [RS10; TC06; Au05] others identify similar model parts [PS11; SDH08; DHY08].

In the field of business process management pattern matching is closely related to the topic of model analysis, which serves multiple purposes ranging from model comparison [Di10; YDG10; DDG09; AVW08; Kü08; VDM08; EKO07; SM07a; VAW06;] and model merge [La10; WDM10; Di09; SM07b; Gr05; UC04] to model exploration [We11; Kn10; Sm10; We10; Fa09; Sm09; WHM08; Aw07].



80

All of these approaches address very particular tasks related to the identification of patterns in conceptual models. Furthermore, some of them restrict themselves to specific modelling languages or do not provide exact matching algorithms but approximation-based heuristics. We therefore argue that the modelling community would benefit from a more generic pattern matching approach that is not limited to a particular modelling language or application scenario and that at the same time is able to identify exact pattern matches. Moreover, to be beneficial such an approach has to return results with good performance. By providing a statistically rigorous performance evaluation, this paper can therefore be seen as a continuation of the work of [De10].

## 3 Set-Theory Based Pattern Matching

### 3.1 Research Design

Our research follows the design science approach [He04] and more specifically the design science research methodology put forth by [Pe07]. Design science is concerned with the creation of innovative artefacts solving previously unsolved problems. In our case the artefact constitutes a generic pattern matching algorithm that is able to find structures in conceptual models created in arbitrary modelling languages. The methodology of [Pe07] stipulates six steps for the process of conducting design science, namely problem identification and motivation, definition of the objectives for a solution, design and development, demonstration, evaluation, and communication. In this paper, we conduct a statistically rigorous performance evaluation of the artefact presented by [De10]. We assess its capability of efficiently contributing to the analysis of conceptual models. In an additional development iteration we augment the artefact to include a caching algorithm that further improves performance.

### 3.2 Pattern Matching Approach

The idea of this approach is to apply set operations to a set of model elements, representing the model to be analysed. Based on graph theory, the approach recognizes any conceptual model as a graph $G$, consisting of vertices $V$ and edges $E$, where $G=(V,E)$ with $E \subseteq V \times V$. Therefore, the approach distinguishes model objects, representing nodes, and model relationships, representing edges, interrelating model objects. Starting from a particular basic set, the approach searches for pattern matches by performing set operations on this basic set. We define the set O of all objects, the set R of all relationships, and the set $E = O \cup R$ to be the basic sets required for this approach. By combining different set operations, patterns are built up successively. Given a pattern definition, the matching process returns a set of model subsets representing the pattern matches found. Every match found is put into a separate subset. In the following, we introduce the available operations of the approach briefly. A detailed formal specification can be found in [De10].

Each operation has a defined number of input sets and returns a resulting set. In the explanation of the operations, we use additional sets (*X*: arbitrary set of elements; *Y*: arbitrary set of objects; *Z*: arbitrary set of relationships) specifying which kinds of inputs an operation expects.

- *ElementsOfType(X,a)* returns a set of all elements of *X*, belonging to the given element type *a*.
- *ElementsWithTypeAttributeOfValue(X,a,b)* returns a set of all elements of X whose type is assigned an attribute a. In addition, the instance of the type attribute has to be of value b. A type attribute might be the label of an element. The function then returns all elements having a particular label b.

In order to assemble complex pattern structures successively, the following operations combine elements and their relationships and elements being related, respectively:

- *ElementsWith{In|Out}Relations(X,Z)* returns a set of sets containing all elements of *X* and their {ingoing | outgoing} relationships of *Z.*
- *ElementsDirectlyRelated($X_1$,$X_2$)* returns a set of sets containing all elements of $X_1$ and $X_2$ that are connected directly via undirected relationships of *R*, including these relationships. Each inner set contains one occurrence.
- *DirectSuccessors($X_1$,$X_2$)* is the directed analogue to *ElementsDirectlyRelated*.

A further category of operation is needed to build patterns representing recursive structures (e.g. a path of arbitrary length):

- *{Directed}Paths($X_1$,$X_n$)* returns a set of sets containing all sequences with undirected {directed} relationships, leading from any element of $X_1$ to any element of $X_n$. The elements that are part of the paths do not necessarily have to be elements of $X_1$ or $X_n$, but can also be of $E\backslash X_1 \backslash X_n$. Each path found is represented by an inner set.
- *{Directed}Loops(X)* is defined analogously to *{Directed}Paths*. It returns a set of sets containing all undirected {directed} sequences, which lead from any element of *X* to itself.

To avoid infinite sets, only finite paths and loops are returned. To provide a convenient specification environment for structural model patterns, we define some additional functions that are derived from those already introduced:

- *ElementsWith{In|Out}RelationsOfType(X,Z,c)* returns a set of sets containing all elements of *X* and their {undirected} directed, {incoming | outgoing} relationships of *Z* of the type *c*. Each occurrence is represented by an inner set.
- *ElementsWithNumberOf{In|Out}Relations(X,n)* returns a set of sets containing all elements of *X*, which are connected to the given number *n* of {undirected} directed {incoming | outgoing} relationships of *R*, including these relationships. Each occurrence is represented by an inner set.
- *ElementsWithNumberOf{In|Out}RelationsOfType(X,c,n*) returns a set of sets containing all elements of *X*, which are connected to the given number *n* of {undirected} directed {incoming | outgoing} relationships of *R* of the type *c*, including these relationships. Each occurrence is represented by an inner set.

- *{Directed}PathsContainingElements($X_1$,$X_n$,$X_c$)* returns a set of sets containing elements that represent all undirected {directed} paths from elements of $X_1$ to elements of $X_n$, which each contain at least one element of $X_c$. The elements that are part of the paths do not necessarily have to be elements of $X_1$ or $X_n$, but can also be of $E\backslash X_1\backslash X_n$. Each such path found is represented by an inner set.
- *{Directed}PathsNotContainingElements($X_1$,$X_n$,$X_c$)* is defined analogously to *{Directed}PathsContainingElements*. However, it returns only paths that do not contain any element of $X_c$.
- *{Directed}Loops{Not}ContainingElements(X,$X_c$)* is defined analogously to *{Directed}Paths{Not}ContainingElements.*
- *{Longest|Shortest}{Directed}Path{Not}ContainingElements($X_1$,$X_n$,$X_c$)* is defined analogously to *{Directed}Paths{Not}ContainingElements*. However, the function only returns the shortest or longest paths from elements of $X_1$ to elements of $X_n$ containing or not containing at least one element of $X_c$.

By nesting the functions introduced above, it is possible to build structural model patterns successively. The results of each function can be reused adopting them as an input for other functions. In order to combine different results, the basic set operators *union* ($\cup$), *intersection* ($\cap$), and *complement* (\) can generally be used. Since it should be possible to not only combine sets of pattern matches (i.e., sets of sets), but also the pattern matches themselves (this refers to the inner sets), the approach incorporates additional set operators. These operate on the inner sets of two sets of sets respectively. The ***Join*** operator performs a *union* operation on each inner set of the first set with each inner set of the second set having at least one element in common. The ***InnerIntersection*** operator *intersects* each inner set of the first set with each inner set of the second set. The ***InnerComplement*** operator applies a *complement* operation to each inner set of the first outer set combined with each inner set of the second outer set. Only inner sets that have at least one element in common are considered. As most of the set operations introduced expect simple sets of elements as inputs, further operators are introduced that turn sets of sets into simple sets. The ***SelfUnion*** operator merges all inner sets of one set of sets into a single set performing a ***union*** operation on all inner sets. The ***SelfIntersection*** operator performs an ***intersection*** operation on all inner sets of a set of sets successively. The result is a set containing elements that each occur in all inner sets of the original outer set.

```
DirectSuccessors(ElementsOfType(O, Function),ElementsOfType(O, Event))
```

The example of an EPC pattern given above illustrates the application of the approach. This pattern represents all functions that are directly succeeded by an event. The *ElementsOfType* calls return the set of all functions and events respectively. They take the basic set *O* as input which represents the set of all objects contained in the model to be searched. The second parameter specifies the type of the objects contained in the intermediate result sets. These are then passed on to the *DirectSuccessors* call returning a set of sets. Each inner set contains a function, the succeeding event and the edge between these two objects.



83

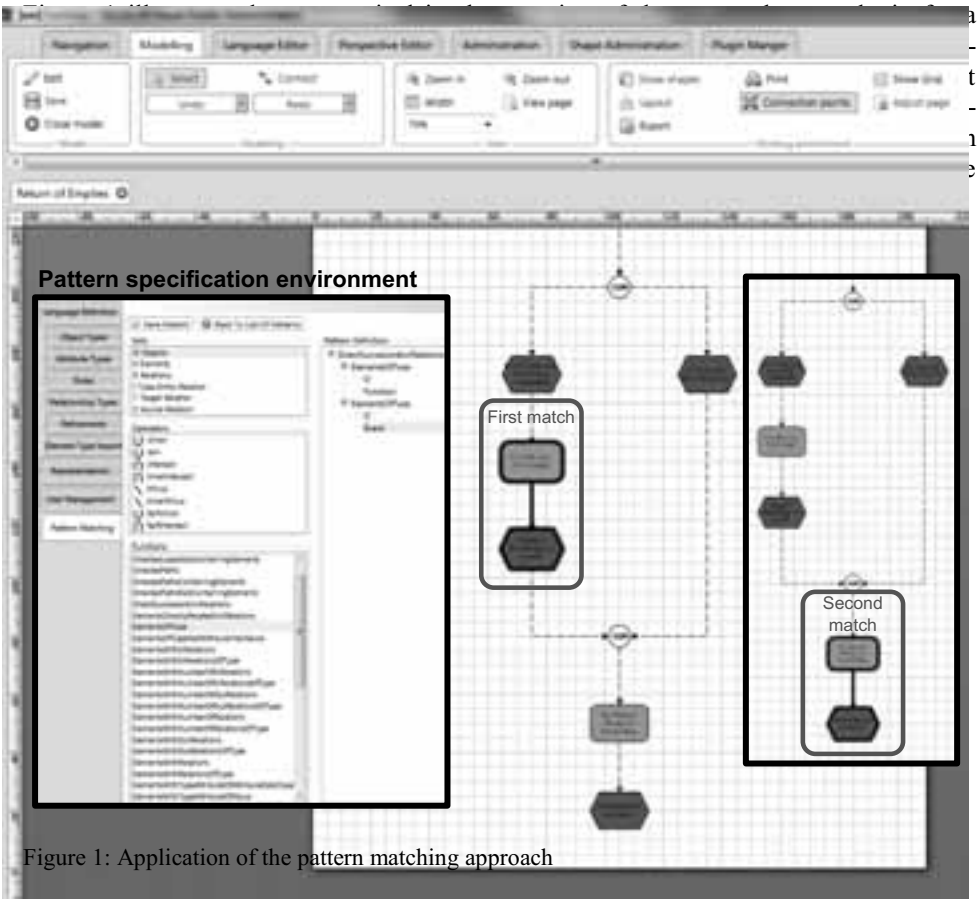

Figure 1: Application of the pattern matching approach

### 3.3 Extensions of the approach

Our approach is implemented using the visitor design pattern known from software engineering [Ga95]. A nested pattern definition is represented as a search tree. The algorithm performs a depth-first search on that tree calculating the result of the leaf nodes first and then returning this intermediate result to the next higher level of the tree. This structure allows for efficiently caching intermediate results, since a given sub-tree may appear multiple times within one pattern definition. Before calculating the result on any given level in the search tree, the algorithm checks a hash table to see if the given subtree has already been calculated. This hash table can be read in constant time, thus allowing for extremely fast access.



84

# 4 Performance Evaluation

## 4.1 Model Base

As model base, we chose 19 different EPC models that were available from two previous research projects in the fields of public administration (nine models) [Be05] and the retail industry (ten models) [BS04]. In terms of the former, the largest process model contains 343 elements and describes the process of refunding student travel expenses. The smallest model consists of 26 elements and depicts the application for a dog licence fee. As far as the process models of the retail industry are concerned, the largest model contains 150 elements and depicts the process of goods receipt. The smallest model consists of 20 elements representing the return of empties. In terms of the structure of the models, in both model pools the large models contain a disproportionately high number of split-connectors.

We chose these model pools for two reasons. First, these processes were surveyed in real life scenarios. Consequently, they allow us to test the pattern matching approach in a realistic context. Second, these pools explicitly include very small process models having a minimum of 20 elements as well as extremely large models of up to 343 elements. This range of different model sizes allows us to evaluate the scaling characteristics of the pattern matching approach. We argue that, other than the complexity of the pattern and the abovementioned structure of the model, the size of the model is the factor that most influences runtime performance. Additionally, we argue that most conceptual models exhibit model sizes that fall into the range we cover here. We therefore conclude that the results we obtained for EPC models are also representative of other process modelling languages as well as application domains.

## 4.2 Patterns

In total we searched for seven patterns in each of the 19 process models. All but one pattern check the syntactical correctness of EPCs. Although the matching approach can be used in a variety of scenarios, we restrict ourselves to syntactical soundness checking as this task requires the most complex patterns. Furthermore, the patterns we searched for are based on the workflow patterns presented by [Va03] and common syntactical errors in EPCs described by [Me07].

The first pattern we searched for is the “AND might not get control from XOR/OR” (AND)-pattern reported in [Me07]. It returns paths of arbitrary length that begin in an XOR/OR-split and end in an AND-join which is the successor of an event having no incoming edges. The exact definition of that pattern is given below. Another very common syntactical error in EPCs constitutes a decision split after an event (DSAE) [Me07]. Other than that, we also searched for the syntactically correct version of that pattern, the decision split after a function (DSAF). The exact pattern definitions can be found in [De10]. Furthermore, we searched for connector loops (CL) which are paths of arbitrary length that start and end in the same element and contain only connectors.

```
DirectedPaths(
   COMPLEMENT(
      UNION(ElementsOfType(O,'OR'),ElementsOfType(O,'XOR')),
      UNION(
        SELFUNION(INNERINTERSECT(UNION(ElementsOfType(O,'OR'),
               ElementsOfType(O,'XOR')),ElementsWithNumberOfOutRelations(
               UNION(ElementsOfType(O,'OR')ElementsOfType(O,'XOR')),1)))
        SELFUNION(INNERINTERSECT(UNION(ElementsOfType(O,'OR'),
               ElementsOfType(O,'XOR')),ElementsWithNumberOfOutRelations(
               UNION(ElementsOfType(O,'OR'),ElementsOfType(O,'XOR')),0)))
   )),
   INNERINTERSECT(ElementsOfType(O,AND),DirectSuccessors(
      ElementsWithNumberOfInRelations(ElementsOfType(O,'Event'),0),
      ElementsOfType(O,'AND'))))
```

We determined the longest sequence of activities (LSA) in each model which we define as the longest paths not containing connectors. Additionally, we checked if every XOR-split is joined by the appropriate counterpart XOR (XOR) [Me07]. This pattern returns all paths from XOR-split connectors to XOR-join connectors not containing AND- or OR-join connectors. We also searched for all paths not containing elements that have a particular label (PRC). We define this pattern to calculate all paths from the start to the end elements of the model. From this set of paths we subtract all paths that do contain elements having that particular label. This second set of paths is calculated by joining the set of paths from all start to all end elements with the set of elements exhibiting the label. By defining the pattern in this manner, we calculate the paths from all start to all end elements twice. In doing so, we are able to demonstrate the effect of the caching mechanism on a computationally expensive operation like a path search. For reasons of brevity we omit the exact definitions of the CL-, LSA-, XOR-, and PRC-patterns here.

### 4.3 Evaluation Process

To obtain the performance data in a statistically rigorous manner, we applied the steady-state performance methodology presented by [GBE07]. The methodology was developed to control measurement errors in Java programs caused by the non-deterministic execution in the virtual machine due to garbage collection, just-in-time (JIT) compilation and adaptive optimisation. As described above, the pattern matching approach was implemented as a plugin for a meta-modelling tool written in C#. This programming language suffers from the same measurements errors. Therefore we adapted the approach of [GBE07] to C#. To compensate for the non-deterministic behaviour of the virtual machine, we chose the so-called steady-state performance measurement determining program performance after JIT compilation and optimisation.

The performance evaluation was conducted on an Intel® Core™ 2 Duo CPU E8400 3.0 GHZ with 3.25 GB RAM and Windows 7 (32-Bit edition). We disabled the energy saving settings in Windows and executed the process as a real-time process to avoid any unnecessary hardware slow down or process switching. Prior to each search run a complete garbage collection was forced. In doing so, we further eliminate systematic errors. For the time measurement we used the high resolution QueryPerformanceCounter-API in Windows in concert with the adapted methodology.



86

During the initial warm-up phase, the JIT compiler optimized the pattern search algorithm. This implies running the algorithm until the variation coefficient drops under two per cent [GBE07]. This coefficient is calculated by dividing the standard deviation of all runs by its mean execution time. This led to searching each pattern in each model at least five times. Once this warm-up phase was completed and a steady state was reached, the measurement phase started. During this phase, each pattern was searched in each model ten times. After ten search runs, the meta-modelling tool was restarted triggering another ten search runs. This process was repeated 35 times to compensate for the non-deterministic behaviour of the virtual machine [GBE07]. For each process, the mean execution times were determined for each combination of pattern and model with caching enabled and disabled respectively. This leads to a total of 70 measurements per combination of pattern and model, 35 for caching enabled and 35 for caching disabled.

### 4.4 Performance Data

We first consider the original algorithm without the caching extension. Runtime measurements are depicted in Figure 2. Each model is represented on the abscissa by its number of elements. In each model all seven patterns were searched for resulting in one bar for each pattern. The ordinate represents the time unit in milliseconds on a logarithmic scale. Each measurement depicts the mean execution time of all 35 search processes. In addition, each measurement exhibits a confidence interval smaller than one per cent given a confidence level of 99 per cent.

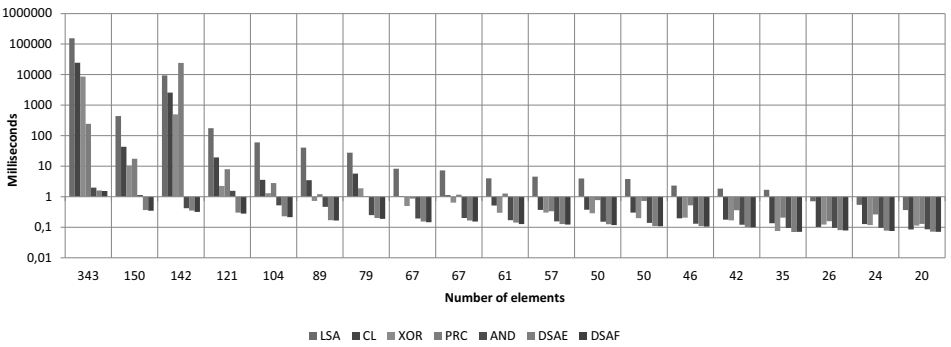


Figure 2: Performance evaluation of the original algorithm

Figure 2 demonstrates that the majority of patterns are found within less than one millisecond. In all but six cases results were obtained in considerably less than one second. The data suggest that our pattern matching algorithm returns results with an overall acceptable performance. Furthermore, the figure suggests that the performance of the matching algorithm depends on three factors, namely the size and the structure of the model (cf. Section 4.1) as well as the complexity of the pattern. The latter refers to what and how many functions and operators are used in the pattern definition and to what extent they are nested. Since the measurements are right-skewed, Figure 2 shows that the larger the model the longer takes the matching process.



87

The structure of the model refers to its number of split-connectors. A linear sequence of elements can be more efficiently searched than a highly branched model. This explains the execution times observed in the third largest model representing a procurement process in the retail industry. This process model contains a large number of splitting XOR connectors, thus prolonging the matching process. In terms of the pattern complexity, in all but one model the LSA pattern took the longest to compute. Furthermore, identifying events or functions that are followed by an XOR-split connector can be computed in less than one millisecond in all models except the largest one. The interaction of these three factors influences runtime performance. Although overall performance is acceptable, Figure 2 indicates that in case of large, highly branched models in combination with complex patterns the execution time increases exponentially. This is especially the case if path functions are used in the pattern definition. As explained in Section 3.2 these functions determine all paths from all start to all target elements. Depending on the number of objects in a given model in combination with the number of their relationships, an exponential number of paths have to be calculated. This theoretical complexity explains the execution times of the LSA and PRC patterns in the large, highly branched models.

The exponential increase in runtime performance motivates augmenting the original matching approach to include a caching mechanism that stores previously calculated sub patterns. The execution times of the augmented algorithm are depicted in Figure 3 and Figure 4. Each figure depicts runtime performance of only one pattern, namely the PRC pattern in Figure 3 and the XOR pattern in Figure 4. The models are again represented on the abscissa by their number of elements. The figures depict two measurements for each model, one with caching enabled and one with caching disabled. The execution times of the original algorithm without the caching mechanism are taken as a reference value for the measurements of the extended version. Again, the measurements represent the mean value of all 35 search processes.

Figure 3 suggests that the caching mechanism improves the overall search performance in every model. It represents measurements for the PRC pattern that allows for caching a computationally expensive path search (cf. Section 4.2). The data suggest that the caching mechanism works particularly well on large and highly branched models in combination with complex patterns allowing to cache expensive operations. In the five largest models the execution time decreased by approximately 50 per cent. Given this pattern, the performance gain tends to decrease with the size of the model. For the smallest model only a 20 per cent gain was measured.

The fact that the caching mechanism improves performance particularly well for computationally expensive operations is also supported by Figure 4. Here the execution times of the XOR pattern are compared. This pattern only allows for caching relatively cheap calls to the *ElementsOfType* function. The expensive path search is performed only once. In this case, the caching mechanism yields hardly any performance increase in large models. For smaller models, however, execution times can be decreased by up to 25 per cent. Since in models of up to 100 elements this pattern can be searched for in less than one millisecond anyway, the actual performance gain is negligible. Still, the data presented in Figure 4 indicates that execution times decrease in all but two models.



88

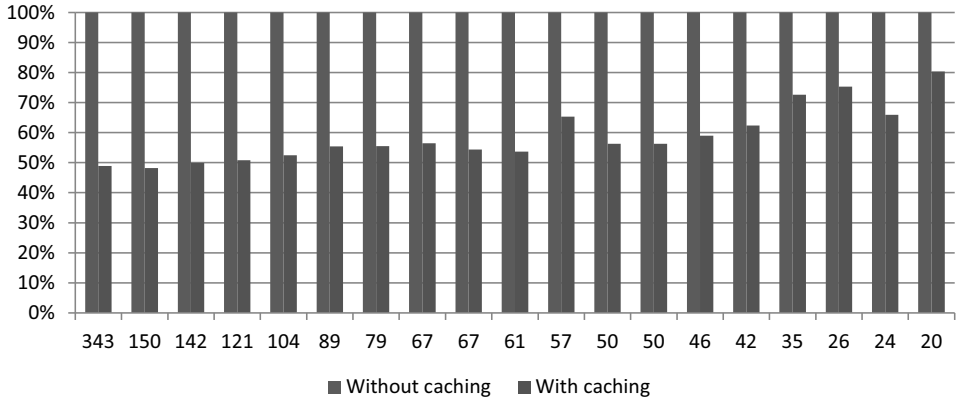


Figure 3: Comparison of the caching algorithm to the original algorithm executing the PRC pattern

These performance statistics prove that although our pattern matching algorithm is generic and not optimized to suit a particular application scenario it returns results with acceptable performance. The caching extension further decreases execution time. Its performance gain depends on the size and structure of the models to be searched as well as the complexity of the patterns. It is particularly beneficial on large, highly branched models in combination with patterns that contain computationally expensive operations like the path search. In those cases, the caching mechanism increases runtime performance by up to fifty per cent. In the worst case, caching previously calculated sub-patterns does not yield any performance gain. As expected, however, it does not decrease execution time either.

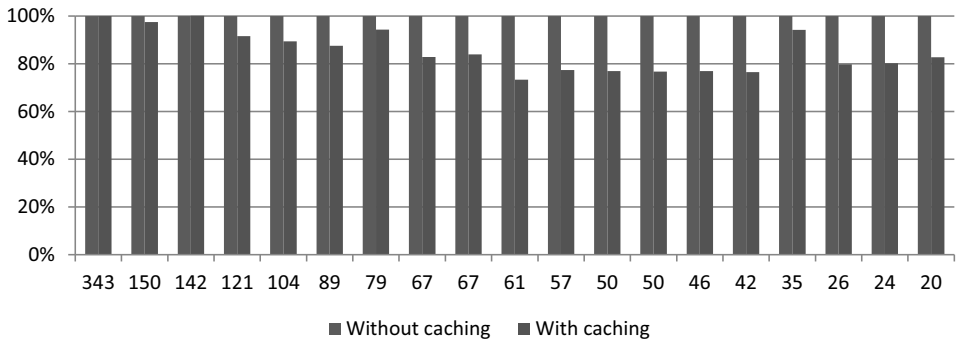


Figure 4: Comparison of the caching algorithm to the original algorithm executing the XOR pattern



89

## 5 Summary and Outlook

In this paper, we conduct a statistically rigorous performance evaluation of a generic set-theory based pattern matching approach. On the basis of a literature review we conclude that there is no generic pattern matching approach suited for any application scenario that at the same time is able to identify structures in conceptual models of arbitrary modelling languages. To prove that such a generic approach is beneficial in terms of performance, a detailed analysis of its runtime behaviour is required. After briefly introducing the matching approach, we augmented the algorithm by a caching mechanism that stores previously calculated sub-patterns. We introduced the model base as well as the patterns we used to conduct the evaluation. Since the size of the models we analysed ranges from 20 to 343 elements, we argue that the results we obtained are also representative of other process modelling languages as well as application scenarios. The evaluation process was conducted according the methodology presented by [GBE07]. Our results were therefore obtained in a statistically rigorous manner. The performance data suggest that despite its generic nature our pattern matching approach returns results with acceptable performance. Most of the patterns could be identified within a few milliseconds. The caching mechanism we introduce continues to decrease execution time by up to fifty per cent.

Having conducted a statistical analysis of its runtime behaviour, future research activities will focus on a theoretical performance evaluation of the matching approach to determine the complexity class of the algorithm. To confront the exponential increase in execution time given large, highly branched models in combination with complex patterns, we furthermore intent to introduce an additional function called *ElementsOfTypeFast*. This function will work on a similar basis than the caching mechanism presented in this paper. It will access a data structure that holds all elements of the respective element types of a given modelling language. This data structure will furthermore allow for access in constant time. The idea is to fill this data structure only once before the actual calculation of the pattern matches. *ElementsOfTypeFast* will then call this structure and thus return all elements of a particular type in constant time which will lead to a further increase in execution time.

## References


[Au05] Aumueller, D. et. al.: Schema and Ontology Matching with COMA++. In: Proc. of the 2005 ACM SIGMOD Int. conf. on Management of data, Baltimore, 2005; pp. 906-908.

[AVW08] Alves de Medeiros, A.K.; van der Aalst, W.M.P.; Weijters, A.J.M.M.: Quantifying Process Equivalence Based on Observed Behavior. Data & Knowledge Engineering 64, 2008; pp. 55-74.

[Aw07] Awad, A.: BPMN-Q: A Language to Query Business Processes. In (Reichert, M., Strecker, S., Turowski, K. Eds.): Proc. of the 2nd Int. Workshop on Enterprise Modelling and Information Systems Architectures, St. Goar, 2007; pp. 115-128.

[Be05] Becker, J. et. al.: Referenzmodellierung in öffentlichen Verwaltungen am Beispiel des prozessorientierten Reorganisationsprojekts Regio@KomM. In (Ferstl, O. K.; Sinz, E. J.; Eckert, S.; Isselhorst, T. Eds.): Wirtschaftsinformatik 2005, Physica-Verlag HD, Heidelberg, 2005; pp. 729-745.

[BS04] Becker, J.; Schütte, R.: Handelsinformationssysteme. 2nd edition, Redline, Frankfurt, 2004.

[DDG09] Dijkman, R. M., Dumas, M., García-Bañuelos, L.: Process Model Similarity Search. In (Dayal, U., Eder, J., Koehler, J., Reijers, H. A. Eds.): Proc. of the 7th Int. Conf. on Business Process Management, Ulm, 2009; pp. 48-63.

[De10] Delfmann, P. et. al.: Pattern Specification and Matching in Conceptual Models. A Generic Approach Based on Set Operations. In: Enterprise Modelling and Information Systems Architectures 5, 2010; pp. 24-43.

[DHY08] Dong, X. L.; Halevy, A.; Yu, C.: Data Integration with Uncertainty. In: Proc. of the 33rd Int. Conf. on Very large data bases, Vienna, 2008; pp. 687-698.

[Di09] Dijkman, R. et. al.: Aligning Business Process Models. In: Proc. of the 13th IEEE Int. Conf. on Enterprise Distributed Object Computing, Auckland, 2009; pp. 40-48.

[Di10] Dijkman, R. et. al.: Similarity of Business Process Models. Metrics and Evaluation. In: Information Systems 36, 2010; pp. 498—516.

[DK07] Delfmann, P.; Knackstedt, R.: Towards Tool Support for Information Model Variant Management – A Design Science Approach. In: Proc. of the 15th European Conf. on Information Systems, St. Gallen, 2007; pp. 2098-2109.

[EKO07] Ehrig, M., Koschmider, A. Oberweis, A.: Measuring similarity between semantic business process models. In (Roddick, J.F., Hinze, A. Eds.): Proc. of the 4th Asia-Pacific Conf. on Conceptual Modelling, Ballarat, 2007; pp. 71–80.

[Fa09] Fahland, D. et. al.: Instantaneous Soundness Checking of Industrial Business Process Models. In (Dayal, U., Eder, J., Koehler, J., Reijers, H. A. Eds.): Proc. of the 7th Int. Conf. of Business Process Management, Ulm, 2009; pp. 278-293.

[Ga95] Gamma, E. et. al.: Design Patterns. Elements of Reusable Object-Oriented Software. Addison-Wesley Longman, Amsterdam, 1995.

[GBE07] Georges, A., Buytaert, D., Eeckhout, L.: Statistically rigorous java performance evaluation. ACM SIGPLAN Notices, vol. 42, 2007; pp. 57-76.

[Gr05] Grossmann, G. et. al.: Behavior based integration of composite business processes. In (van der Aalst, W.M.P., Benatallah, B., Casati, F. Curbera, F. Eds.): Proc. of the 3rd Int. Conf. on Business Process Management, Nancy, 2005; pp. 186-204.

[He04] Hevner, A. R. et. al.: Design Science in Information Systems Research. MIS Quarterly 28, 2004; pp. 75-105.

[Kü08] Küster, J. et. al.: Detecting and Resolving Process Model Differences in the Absence of a Change Log. In (Dumas, M., Reichert, M., Ming-Chien, S. Eds.): Proc. of the 6th Int. Conf. on Business Process Management, Milan, 2008; pp. 244-260.

[Kn10] Knuplesch, D. et. al.: On Enabling Data-Aware Compliance Checking of Business Process Models. In (Parsons, J., Saeki, M., Shoval, P., Woo, C. C., Wand, Y. Eds.): Proc. of the 29th Conf. on Conceptual Modeling, Vancouver, 2010; pp. 332-346.

[La10] La Rosa, M.et. al.: Merging Business Process Models. In (Meersmann, R., Dillon, T., Herrero, H. Eds.): Proc. of the 2010 Int. Conf. on the move to meaningful internet systems, Hersonissos, 2010; pp. 96-113.

[Me07] Mendling, J.: Detection and Prediction of Errors in EPC Business Process Models. Doctoral Thesis, Vienna University of Economics and Business Administration. Vienna, 2007.

[Pe07] Peffers, K. et. al.: A Design Science Research Methodology for Information Systems Research. Journal of Management Information Systems 24, 2007; pp. 45-77.

[PS11] Po, L.; Sorrentino, S.: Automatic generation of probabilistic relationships for improving schema matching. Information Systems 36, 2011; pp. 192-208.

[RB01] Rahm, E.; Bernstein, P. A.: A survey of approaches to automatic schema matching. The Very Large Data Base Journal 10, 2001; pp. 334-350.

[RS10] Rajesh, A.; Srivatsa, S. K.: XML Schema Matching – Using Structural Information. International Journal of Computer Applications 8, 2010; pp. 34-41.

[Sc00] Scheer, A.-W.: ARIS – Business Process Modelling. 3rd edition, Springer, Berlin, 2000.

[SDH08] Sarma, A.; Dong, X.; Halevy, A.: Bootstrapping Pay-As-You-Go Data Integration Systems. In: Proc. of the 2008 ACM SIGMOD Int. Conf. on Management of data, Vancouver, 2008; pp. 861-874.

[SM07a] Simon, C., Mendling, J.: Integration of Conceptual Process Models by the Example of Event-driven Process Chains. In Proc. of the 8th Int. Tagung Wirtschaftsinformatik, Karlsruhe, 2007; pp. 677-694.

[SM07b] Simon, C., Mendling, J.: Integration of Conceptual Process Models by the Example of Event-driven Process Chains. In (Oberweis, A., Weinhardt, C., Gimpel, H., Koschmider, A., Pankratius, V., Schnizler, B. Eds.): Proc. of the 8th Int. Tagung Wirtschaftsinformatik, Karlsruhe, 2007; pp. 677-694.

[Sm09] Smirnow, S. et. al.: Action Patterns in Business Process Models. In (Baresi, L., Chi-Hung, C., Suzuki, J. Eds.): Proc. of the 7th Int. Conf. on Service-Oriented Computing, Stockholm, 2009; pp. 115-129.

[Sm10] Smirnov, S. et. al.: Meronymy-Based Aggregation of Activities in Business Process Models. In (Parsons, J., Saeki, M., Shoval, P., Woo, C. C., Wand, Y. Eds.): Proc. of the 29th Int. Conf. on Conceptual modelling, Vancouver, 2010; pp. 1-14.

[TC06] Tansalarak, N.; Claypool, K.T.: QMatch – Using paths to match XML schemas. Data & Knowledge Engineering 60, 2006; pp. 260-282.

[UC04] Uchitel, S., Chechik, M.: Merging Partial Behavioural Models. SIGSOFT Software Engineering Notes 29, 2004; pp. 43-52.

[Va03] Van der Aalst, W. et. al.: Workflow Patterns. Distributed and parallel databases 14, 2003; pp. 5-51.

[VAW06] van der Aalst, W. M. P., Alves de Medeiros, A. K., Weijters, A. J. M. M.: Process Equivalence: Comparing Two Process Models Based on Observed Behavior. In (Dustdar, S., Fiadeiro, J.L., Sheth, A. Eds.): Proc. of the 4th Int. Conf. on Business Process Management, Vienna, 2006; pp. 129-144.

[VDM08] van Dongen, B., Dijkman, R., Mendling, J.: Measuring Similarity between Business Process Models. In (Bellahsène, Z., Léonard, M. Eds.): Advanced Information Systems Engineering, LNCS 5074, Heidelberg, 2008; pp. 450-464.

[VTM08] Vergidis, K., Tiwari, A., Majeed, B.: Business process analysis and optimization: beyond reengineering. IEEE Transactions on Systems, Man, and Cybernetics 3, 2008; pp. 69–82.

[WDM10] Weidlich, M., Dijkman, R., Mendling, J.: The ICoP Framework. Identification of Correspondences between Process Models. In: (Pernici, B. Eds.): Proc. of the 22nd Conf. on Advanced Information Systems Engineering, Hammamet, 2010; pp. 483-498.

[We10] Weidlich, M. et. al.: Process Compliance Measurement based on Behavioral Profiles. In (Pernici, B. Ed.): Proc. of the 22nd Conference on Advanced Information Systems Engineering, Hammamet, 2010; pp. 499-514.

[We11] Weber, B. et. al.: Refactoring large process model repositories. Computers in Industry 62, 2011; pp. 467-486.

[WHM08] Weber, I., Hoffmann, J., Mendling, J.: Semantic Business Process Validation. In (Hepp, M., Stojanovic, N., Hinkelmann, K., Karagiannis, D., Klein, R. Eds.): Proc. of the 3rd Int. Workshop on Semantic Business Process Management, Tenerife, 2008; pp. 22-36.

[YDG10] Yan, Z.; Dijkman, R.; Grefen, P.: Fast business process similarity search with feature-based similarity estimation. In (Meersmann, R., Dillon, T., Herrero, H. Eds.): Proc. of the 2010 Int. Conf. on the move to meaningful internet systems, Hersonissos, 2010; pp. 60-77.