Pattern Discovery in Conceptual Models Using Frequent Itemset Mining

Mattia Fumagalli[1(✉)], Tiago Prince Sales[1], and Giancarlo Guizzardi[1,2]

[1] Conceptual and Cognitive Modeling Research Group (CORE), Free University of Bozen-Bolzano, Bolzano, Italy
{mattia.fumagalli,tiago.princesales,giancarlo.guizzardi}@unibz.it
[2] Services and Cybersecurity Group, University of Twente, Enschede, The Netherlands

Abstract. Patterns are recurrent structures that provide key insights for Conceptual Modeling. Typically, patterns emerge from the repeated modeling practice in a given field. However, their discovery, if performed manually, is a slow and highly laborious task and, hence, it usually takes years for pattern catalogs to emerge in new domains. For this reason, the field would greatly benefit from the creation of automated data-driven techniques for the empirical discovery of patterns. In this paper, we propose a highly automated interactive approach for the discovery of patterns from conceptual model catalogs. The approach combines graph manipulation and Frequent Itemset Mining techniques. We also advance a computational tool implementing our proposal, which is then validated in an experiment with a dataset of 105 UML models.

Keywords: Modeling patterns · Pattern discovery · Itemset mining

1 Introduction

For a while now, patterns have been widely used by the modeling community for a range of different purposes, including to understand how languages are used in practice [3,8]. Their popularity is evinced, among other factors, by the growing number of pattern catalogs[1] for different modeling languages. Pattern discovery, however, if performed manually, is highly laborious and, hence, it usually takes years for pattern catalogs to emerge. First, because of the sheer size of data to be analyzed. Second, searching for patterns consists of cognitively demanding steps, such as partitioning models into smaller fragments, calculating the frequency of candidate patterns, and filtering out constructs of interest (e.g. when one is looking for taxonomic structures in domain models).

In this paper we propose a highly-automated interactive approach for the empirical discovery of patterns from conceptual model catalogs. In particular,

[1] E.g. https://github.com/wilmerkrisp/patterns, http://www.bpmpatterns.org.

This work was supported by Accenture Israel Cybersecurity Labs.

J. Ralyté et al. (Eds.): ER 2022, LNCS 13607, pp. 52–62, 2022.
https://doi.org/10.1007/978-3-031-17995-2_4

we focus on the discovery of *recurrent modeling structures* that can be defined by a fixed combination of the constructs of a language. Our approach, developed using the *Design Science Methodology* [7], aims at supporting language designers throughout all activities of the pattern discovery process, namely: *i) input data preparation*, when the input conceptual models data are manipulated to feed the mining process; *ii) mining process customization*, when the parameters for the mining process are provided; *iii) pattern mining*, the actual mining process; and *iv) output assessment*, when the user assess the discovered patterns. For this purpose, we combine *graph manipulation* techniques with the *Frequent Itemset Mining* algorithm [1].

We implemented our approach in a proof-of-concept application, which was validated according to a set of requirements gathered from expert language designers about *what* our approach should do, as well as *how* it should do it. We tested these criteria using a catalog of 105 domain models [2] specified in OntoUML [5], a pattern-based well-founded extension of *UML Class Diagrams*.

The remainder of this paper is structured as follows. Section 2 lists the requirements that drove the design of our approach. Section 3 describes the pattern discovery method embedded in our approach. In Sect. 4, we report on the experiments we conducted to validate our solution. Then, in Sect. 5, we position our contribution with respect to the state of the art. Finally, in Sect. 6, we reflect on our results and discuss some future work.

2 Requirements

Following the Design Science methodology [7], we grounded the design of our approach on a preliminary *problem identification* activity. In this phase, we interviewed five senior researchers who developed conceptual modeling languages. We asked them, in open-ended interviews, about: *i) the relevance of an approach for facilitating the empirical discovery of structural patterns in conceptual models* and *ii) what is required to facilitate this discovery process*. From their feedback, we defined the following requirements:

- **Interest (R1)**: the approach should be able to discover *subjectively interesting* patterns. Here, the notion of "subjectively interesting" is inspired by the work from Silberschatz and Tuzhilin [14], where a pattern is ranked as interesting by a user if: *a)* it is considered *exploitable* for modeling activities, *b)* it contradicts some user's expectations.
- **Customization (R2)**: the approach should support the manipulation of input models so that one can look for a particular type of pattern. For instance, from class diagrams, one should be able to filter out everything but classes and generalizations to look for taxonomic patterns.
- **Comprehension (R3)**: the approach should support the assessment and analysis of the output patterns by generating human-readable visualizations and providing their *absolute frequency* (i.e. how many times the pattern occurs in all models of the catalog) and their *model frequency* (i.e. how many models in the catalog have at least one occurrence of the pattern).

- **Reliability (R4)**: the approach should accurately calculate the absolute and model frequencies for all the patterns it finds. More precisely, the ratio between the number of occurrences retrieved by the approach and the number of actual occurrences should be at least 0.5.
- **Performance (R5)**: the processing and mining steps should *happen* in a reasonable amount of time, even with a *large* set of models. By "a reasonable amount of time" we mean an amount that would not discourage language designers to interact with such a tool, and naturally, that is lower than the time it would take for them to produce the same outputs manually. For now, we are assuming a threshold of 5 min to mine patterns from 100 models. By "large set of models", we mean between 100 and 10000 models, as we do not expect model catalogs to be much bigger than that.
- **Compatibility (R6)**: the approach should be generic enough such that it works with any conceptual modeling language.

3 Discovering Frequent Patterns

We represent our approach as a workflow composed of 7 main tasks, whose inputs, outputs, and dependencies are combined as from Fig. 1 below.

The first task is *Filtering (0)*, where the user can select what language constructs to filter out from the models. For instance, in the case of OntoUML, one may want to look for patterns only involving classes decorated with certain stereotypes, or involving only classes, generalizations, and generalization sets.

The *Abstraction (1)* task allows the user to input a set of transformation functions to be applied to the models, by which certain constructs can be abstracted into more general constructs. For instance, in ArchiMate, business processes, business functions, and business collaborations may be abstracted into business internal behavior elements. This step allows users to look for more general patterns that apply to several types of constructs.

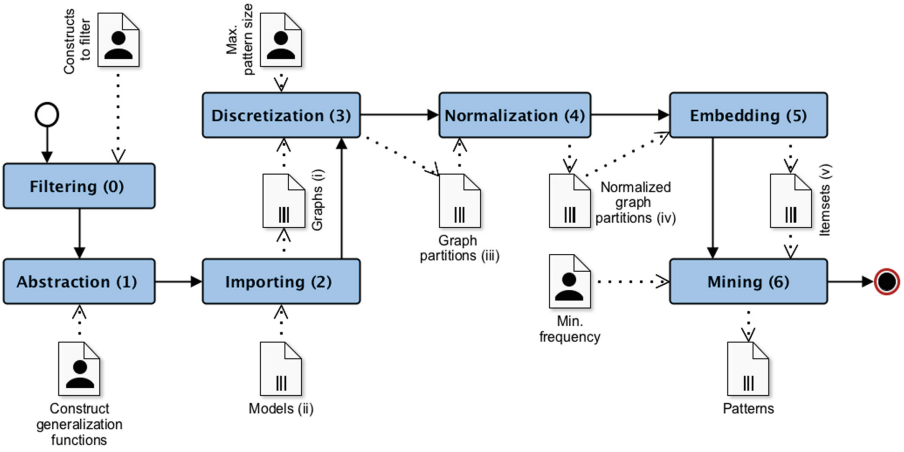

Fig. 1. The frequent patterns discovery workflow.

The *Importing (2)* task consists of taking a set of conceptual models M encoded in a given language (e.g., UML or BPMN) and transforming each model $m_i \in M$ into a graph g_j. This may seem trivial at a first glance, after all, conceptual models are basically graphs. That, however, is not always the case. Consider, for instance, the transformation of UML class diagrams into graphs. The simple solution is transforming classes into nodes and generalizations and associations into edges. Still, if we want to convert, generalization sets, association classes, generalizations between associations, cardinalities, and several other constructs, that no longer works. This task is language-dependent and requires an *ad hoc* transformation for each source conceptual modeling language.

Discretization (3) takes each graph g_j and splits it into graph partitions gp_k that represent subsets of the input conceptual model m_i. We do this by repeatedly executing the *Kernighan Lin Bisection Algorithm* [9], which splits a graph into two balanced bisections[2], until we obtain graph partitions with at most N nodes–a threshold value provided by the user. Note, however, that we lose some edges in the bisection process. To counter this effect, after generating our graph partitions, we restore some removed edges back to them. A removed edge e_l is restored to a partition gp_k if gp_k contains at least one of the two nodes connected by e_l. If one such node is not part of gp_k, it is also restored to it. This discretization task is completely language-independent.

Through the *Normalization (4)* task, the graph partitions are relabeled and indexed to enable the detection of patterns across them. The relabeling of the graph partition occurs by firstly associating both a label and an index to each node, where, originally, the node corresponds to an *id* and the label corresponds to the language construct associated with that node. Once the relabeling is applied, the index of the nodes with the same label is normalized (from 0 to n). Normalization is also a language-independent task.

Embedding (5), still a language-independent task, converts normalized graph partitions into item sets. Here each graph edge is transformed into an item, thus enabling to mine all the information encoded by the node and the edge labels. Notice that this allows accounting for the rich amount of information encoded by conceptual models. For instance, an item can easily represent nodes and edges labels, association source and target cardinalities, and edges directionality.

Figure 2 provides an example of how a conceptual model '*.i*' is converted through the steps described so far. The importing step produces the graph shown by '*.ii*'. The discretization task produces the partitions shown by '*.iii*'. The normalization task produces the normalized partitions '*.iv*' and the embedding returns the set of item sets represented by '*.v*'.

Notice that each column of the item set table '*.v*' represents an item i_j, namely a graph edge with the standardized nodes. Each record represents an input partition graph. I_1, for instance, encodes the left-side graph in of Fig. 2.'*.iii*' and I_2 encodes the graph on the right side. The index of the labels was standardized from "0" to "3" for the larger partition and from "0" to "2" for the

[2] The bisections are balanced in terms of the number of nodes and edges.

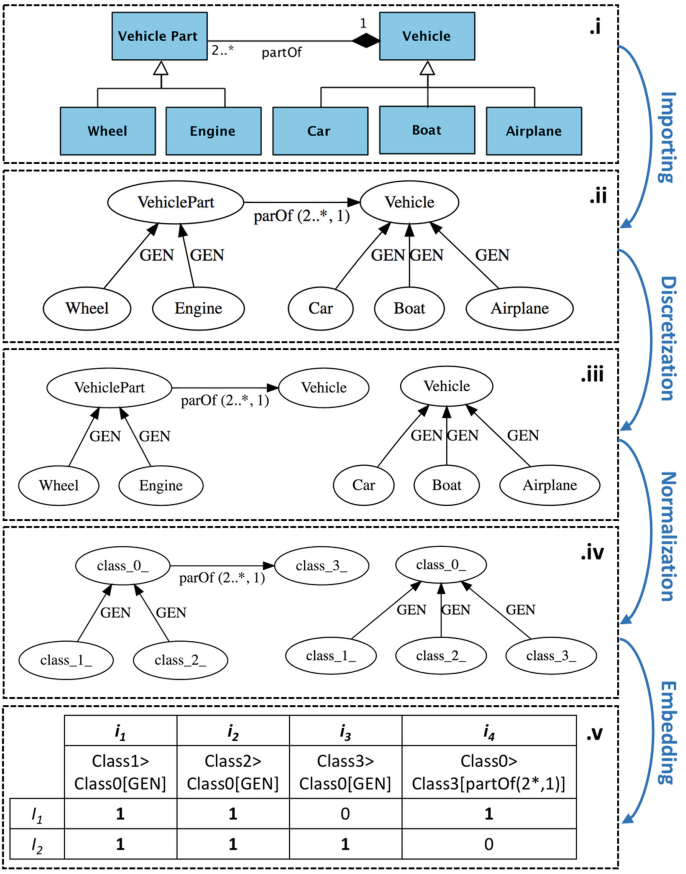

Fig. 2. From *importing* to *embedding*: *.i* input conceptual model, *.ii* transformed graph, *.iii* graph partitions, *.iv* normalized graph partitions, *.v* set of item sets.

smaller one, thus enabling the detection of recurrent items across the item sets. For instance, in '*.v*' both i_1 and i_2 occur in both I_1 and I_2.

The *Mining (6)* task represents the final part of the workflow and is aimed at *a)* generating the candidate patterns and *b)* making the output accessible to the user for the final assessment. This task allows for another interaction with the user, who can select the frequency threshold for the output patterns (e.g., filter out patterns that occur less than 30 times) or some *ad hoc* parameters of the mining algorithm (e.g., avoid sub-patterns with the same frequency). The output of the mining task will then consist of a *1)* list of the discovered patterns in a format that eases the final assessment[3]; *2)* a set of frequency measures, for each output pattern, namely: *2.1)* the *absolute frequency*, calculated as the number of pattern occurrences over the total number of item sets generated through

[3] Example at https://purl.org/mining-cm-patterns/pattern-example.

embedding; *2.2)* the *model frequency*, calculated as the number of pattern occurrences over the number of conceptual models used as inputs of the whole process. For instance, given 5 models we can have a pattern occurring with an absolute frequency of '10', but the model frequency cannot be more than '5'.

4 Evaluation

To evaluate our approach, we implemented it as a Python command-line application[4] in which the user can interactively set up the process, manipulate the data, and assess the output of the mining algorithm. The implementation is built on top of two main packages, namely *NetworkX* and *PrefixSpan*. NetworkX is a comprehensive, open-source, graph analytics and processing toolkit, independently developed and maintained by a large and lively community of developers. PrefixSpan is a very simple yet flexible implementation of the homonymous algorithm. With this application, we evaluated our requirements by running the following two experiments.

4.1 Experiment 1

This experiment assesses our solution w.r.t. **R1**, **R2**, **R3**, and **R4**.

(i) Data. As input data, we used **105** models from a catalog of *OntoUML* models [2], a pattern-based language that extends *UML Class Diagrams* [5].

(ii) Setup. For the validation, we used, as "litmus test", 6 common OntoUML patterns, which were previously manually identified by the designers of the language within multiple example models, retained to be useful for building OntoUML models [6,13]. The selected patterns are represented in Fig. 3.

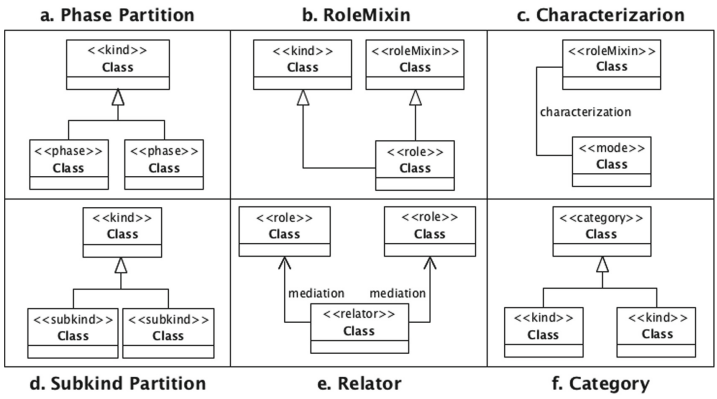

Fig. 3. OntoUML modeling patterns examples [13].

[4] Source code is available at https://purl.org/krdb-core/mining-cm.

We run the application **6** times and we checked whether the proposed solution is able to discover the pre-identified interesting patterns (**R1**). Moreover, we checked the role of the customization steps in supporting the discovery process (**R2**) and the level of comprehensibility of the outcome (**R3**). For each trial, we selected different parameters. In the first three trials, we just customized the number of nodes allowed in the graph partition, selecting *12*, *8*, and *4* as reference sizes. The partition size parameters were selected based on the average number of classes in OntoUML patterns. In the last three trials, we reused the same partition parameters by filtering out stereotypes and edge types that are not used in the pattern. Finally, we queried the input models to count the real-existing number of occurrences for each pattern and we compared the results with the occurrences found by our application. The level of reliability (**R4**) was then simply calculated through an application of the *Jaccard index* $J(A, B) = \frac{|(A \cap B)|}{|(A \cup B)|}$ [4,16], where A is the set of manually found pattern occurrences and B is the set of occurrences for the same pattern found by the process.

(iii) Results. Figure 4 resumes the output data of experiment 1. The first observation is that the approach is able to find the patterns we selected beforehand (**R1**), which are clearly mapped into the output pattern graphs (**R3**). All the discovered patterns (with examples of more complex patterns as well) can be found at our git repository[5]. Secondly, it can be noticed that the partition parameters have an impact on the reliability of the process. This seems to be dependent on the structure of the pattern. For instance, in the case of hierarchical patterns, such as phase, subkind, and category patterns (see Fig. 3), the discovery better performs with larger graphs; differently, in the case of patterns that are not characterized by a taxonomical structure, the behavior seems opposite (see roleMixin, relator and characterization patterns). Moreover, in the trials where the user interacts with the application to filter out information, the reliability significantly improves in most of the cases, thus demonstrating the key role of the customization features (**R2**) implemented in the approach. Finally, for what concerns the reliability overall (**R4**), the current implementation is evidently better in calculating the model frequency (for roleMixin and subkind patterns we have 100% reliability). Still, considering taxonomic patterns, the approach returns high Jaccard index scores w.r.t. the absolute frequency (the scores for phase, subkind, and category patterns were *0.91*, *0.9* and *0.7*, respectively).

(iv) Threats to Validity. We see one main threat to our claim that our approach is able to discover interesting patterns (**R1**). The risk stems from the selection of patterns that in our experiment are relatively small and in other cases may have a more complex structure, with more nodes and edges. However, the examples we used are recognized as the most common in OntoUML and are very similar in terms of size and structure to most of the OntoUML modeling patterns [13]. Moreover, by analysing the whole output of the trials we ran, we observed bigger

[5] https://purl.org/mining-cm-patterns/experiment.

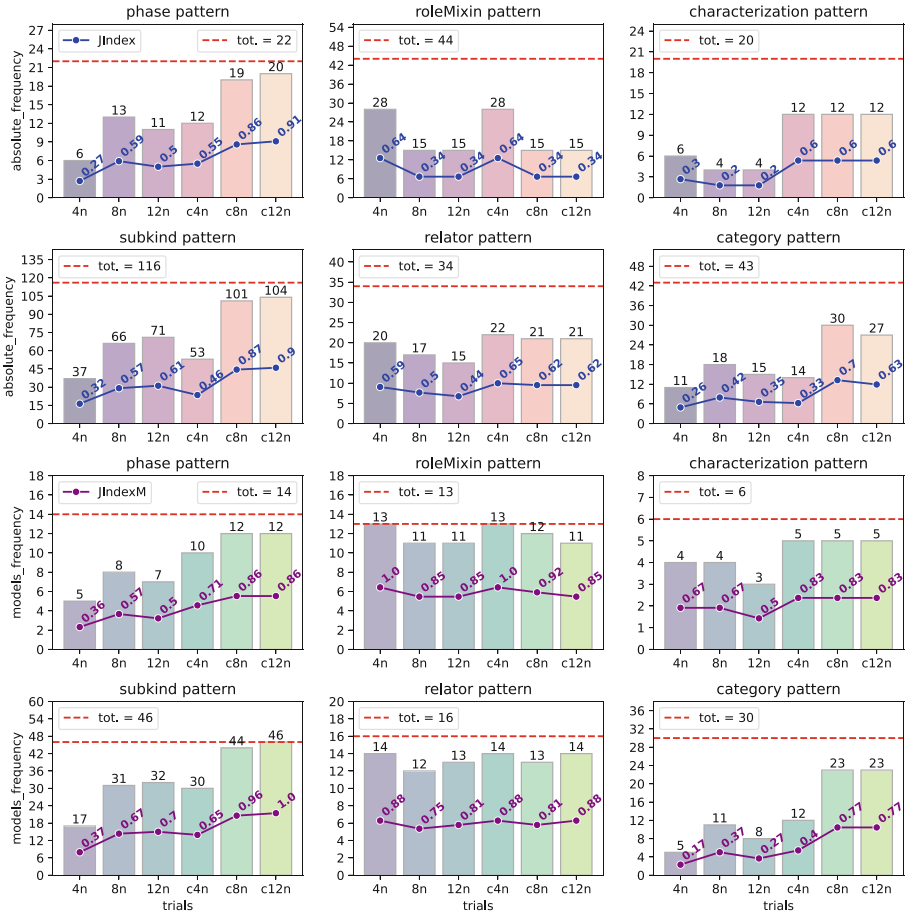

Fig. 4. Trial 1 results. Each chart shows the data for a pattern. Bars represent found occurrences no.; the first six charts refer to *absolute frequency* data, while the last six refer to *model frequency* data. The red line in each chart is the total number of occurrences found with the queries. The blue and purple lines represent the *jaccard index* for the *absolute frequency* and the *model frequency*, respectively. (Color figure online)

but less frequent patterns than those presented in Fig. 3, thus suggesting that the approach can discover more complex structures.

4.2 Experiment 2

The second experiment asseses our approach w.r.t. **R5**. Information about *(i) Data*, *(ii) Setup*, *(iii) Results* and *(iv) Threats to validity* can be found in our git repository[6]. In a nutshell, processing and mining the 105 models in our catalog

[6] https://purl.org/mining-cm-patterns/performance.

takes approximately 2 min in a MacBook Pro (Retina, 13-in. Early 2015) with CPU 2,7 GHz Intel Core i5, 8 GB RAM.

5 Related Work

There is extensive literature on pattern discovery and its applications in a variety of domains, including software code and databases. The application of pattern discovery techniques in conceptual models, however, is much more restricted. In this focused area of research, the closest work to what we propose is that of Skouradaki et al. [15], who designed a pattern mining algorithm for *BPMN*. Still, the goal of our contribution is not to provide a new mining algorithm. Our focus is indeed on the combination of well-established itemset mining (PrefixSpan) and graph manipulation techniques. Furthermore, a considerable amount of effort from our side concerns the definition of an interactive process where users can participate in the discovery activities, thus affecting the reliability of the final output. Last but not least, we designed the approach with the scope of covering different conceptual modeling languages, by keeping all the functions of the approach as language-independent.

Ławrynowicz et al. [10] seek to discover domain patterns, related to specific areas of information and independent of the modeling language constructs, that recur across *OWL ontologies* by applying a *tree-mining* technique. The contribution is divided into two main steps, which partially resemble aspects of our strategy, namely: a transformation step - where ontology axioms are transformed into tree structures; and an association analysis step - where co-occurring axioms are extracted to discover ontology patterns. This research is applied over a set of ontologies from the *BioPortal* repository and is very similar to ours in spirit. However, our solution presents key differences. Firstly, for the mining step, we adopted the *frequent itemset mining* algorithm, thus involving a completely different input preparation step. Secondly, we devised our approach with the main goal of discovering *structural modeling patterns*, namely patterns defined simply by the combination of constructs of a modeling language. In [10] the discovered patterns concern primarily domain-specific information that may recur within or across ontologies (e.g., what are the recurrent properties of the class "person"). Again, the interaction capabilities we proposed are out of their scope.

In the same direction, Lee et al. [11] seek to discover domain patterns across and within ontologies. However, to address this challenge, two different steps are adopted: a step where sub-graphs are extracted through candidate generation and chunking processes; a step where *frequent sub-graphs mining* [12] is adopted. This work also focuses on domain-specific patterns and one of its priorities is to allow the processing of large-scale knowledge graphs. Moreover, no account of how to handle an interactive discovery process is provided.

Unlike the above-presented approaches, our goal is mainly to offer an interactive tool for pattern discovery. Our approach finds recurrent modeling structures, which do not represent necessarily examples of good or bad modeling practices. Finally, one key aspect of our solution is to apply *frequent itemset mining*. This

technique enables us to mine information (e.g., cardinalities, edge labels, class labels vs. stereotypes) that, with more orthodox approaches (e.g., *frequent subgraph mining* [12]), which are mostly aimed at mining unlabeled undirected graphs, could not be fully exploited.

6 Final Considerations

This paper presents an interactive approach for automating the empirical discovery of modeling patterns in conceptual models by combining graph manipulation techniques and frequent itemset mining. By doing so, we move towards automating the construction of pattern catalogs for modeling languages and we create a mechanism for helping language designers to create higher-granularity primitives in their languages, i.e., modeling patterns that can become part of the grammar and tools of that language [6].

Based on the encouraging results from our evaluation with 105 OntoUML models, we envision a series of next steps. First, we will test, in collaboration with language designers, if our approach can find unexpected patterns in OntoUML. Second, we are going to extend the set of constructs to be encoded in the input graphs (e.g., *generalization sets*). Third, we will test our approach with models encoded in different modeling languages, such as BPMN and ArchiMate (**R6**).

References

1. Agrawal, et al.: Mining association rules between sets of items in large databases. In: ACM SIGMOD International Conference on Management of Data, pp. 207–216 (1993)
2. Barcelos, P.P.F., et al.: A FAIR model catalog for ontology-driven conceptual modeling research. In: Conceptual Modeling, ER 2022 (2022)
3. Gangemi, A., Presutti, V.: Ontology design patterns. In: Staab, S., Studer, R. (eds.) Handbook on Ontologies. IHIS, pp. 221–243. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-540-92673-3_10
4. García-Vico, et al.: A big data approach for the extraction of fuzzy emerging patterns. Cogn. Comput. **11**(3), 400–417 (2019)
5. Guizzardi, et al.: Types and taxonomic structures in conceptual modeling: a novel ontological theory and engineering support. Data Knowl. Eng. **134**, 101891 (2021)
6. Guizzardi, G.: Ontological patterns, anti-patterns and pattern languages for next-generation conceptual modeling. In: ER 2014, vol. 8824, pp. 13–27 (2014)
7. Hevner, A.R., March, S.T., Park, J., Ram, S.: Design science in information systems research. Manag. Inf. Syst. Quart. 28(1), 75–105 (2004)
8. Hitzler, P., Gangemi, A., Janowicz, K.: Ontology engineering with ontology design patterns: foundations and applications, vol. 25. IOS Press (2016)
9. Kernighan, B.W., Lin, S.: An efficient heuristic procedure for partitioning graphs. Bell Syst. Tech. J. **49**(2), 291–307 (1970)
10. Ławrynowicz, A., Potoniec, J., Robaczyk, M., Tudorache, T.: Discovery of emerging design patterns in ontologies using tree mining. Semant. web **9**(4), 517–544 (2018)
11. Lee, K., Jung, H., Hong, J.S., Kim, W.: Learning knowledge using frequent subgraph mining from ontology graph data. Appl. Sci. **11**(3), 932 (2021)

12. Ramraj, T., Prabhakar, R.: Frequent subgraph mining algorithms-a survey. Procedia Comput. Sci. **47**, 197–204 (2015)
13. Ruy, F.B., et al.: From reference ontologies to ontology patterns and back. Data Knowl. Eng. **109**, 41–69 (2017)
14. Silberschatz, A., Tuzhilin, A.: On subjective measures of interestingness in knowledge discovery. In: KDD, vol. 95, pp. 275–281 (1995)
15. Skouradaki, M., Andrikopoulos, V., Kopp, O., Leymann, F.: RoSE: reoccurring structures detection in BPMN 2.0 process model collections. In: Debruyne, C., et al. (eds.) OTM 2016. LNCS, vol. 10033, pp. 263–281. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-48472-3_15
16. Tan, P., et al.: Selecting the right interestingness measure for association patterns. In: International Conference on Knowledge Discovery and Data Mining, pp. 32–41 (2002)