An explicitly modeled algorithm for mining frequent item sets in MDE settings

T. Leys
tim.leys@student.uantwerpen.be

University of Antwerp, Belgium

Abstract

In this paper, we will take a closer look into how we can use data mining techniques in combination with DSL's. The frequent dataset mining algorithm will be used.
We will discuss how the algorithm can be explicitly modeled. By explicitly modeling the algorithm, we elevate the level of reasoning to the same level as the DSL. This way domain experts can work with and custom tailor the algorithm in an environment they are familiar with. The approach is presented through a scenario exemplifying a typical Industry4.0 context, in which data mining algorithms are frequently used.

Keywords: MDE, Frequent Itemset Mining, Explicit Modeling, Supply Chain Management

1. Introduction

Recommender systems has become a ubiquitous concept in the world of computer science. By analyzing past behavior, a recommender system tries to recommend certain items to a user. A typical applications of recommender systems can be found in e-commerce sites like amazon.com. A user will be given a set of items that he is likely to be interested in. Dyck et al. [1] have mentioned the fact that modeling environments might benefit from recommender systems, for example for syntax completion. However current research in this field is limited.

A widely used technique in recommender systems is *frequent dataset mining* [3]. Given a database of transaction, a frequent dataset mining algorithm

attempts to identify which items appear frequently together in a transaction. A transaction set is a set of items, in the context of e-commerce a transaction can be interpreted as a set of items a user buys together. In a modeling environment, we can interpret such a transaction set as a set of all elements used in a model. Many algorithms have been developed to deal with this problem, but they all operate on textual transactions. In model driven engineering however, formalisms often have a visual concrete syntax. To use the existing algorithms we would have to transform models into a textual representation. It is hard enough to reason about the importance of visual elements for the algorithm, that transforming those features into a textual representation (and back) adds unnecessary complexion. In this research we will come up with a meaningful visual representation of transactions that can be used as input to an algorithm.

Kühne et al. [4] have stated the importance of explicitly modeling models as well as model transformations. In this paper we will explicitly model the frequent dataset mining algorithm. Explicitly modeling the algorithm has some advantages. The specification is not hidden away in code, the specification can be altered on the fly and its easier to reason about it. We will show that our algorithm will be able to deal with the visual representation of transactions.

We also apply the algorithm to a motivating example. The example will show the possible benefits of applying data mining techniques to modeling formalisms and DSL's and why an explicit model of the algorithm is useful.

The reminder of the paper is structured as follows. In section 2, we will elaborate the motivational example. In section 3, we discuss the explicitly modeled implementation of the fp-growth algorithm. In section 4, we will apply the algorithm to the example.

2. Motivational Example

To demonstrate the usefulness of using the frequent dataset mining algorithm, we present a motivational example of a tile factory. The example will show what the frequent dataset mining algorithm aims to solve.

2.1. The Tile Factory

The example tile manufacturer consists of three parts, the factory, the storage facility and the retail. This example is simplified version of a supply

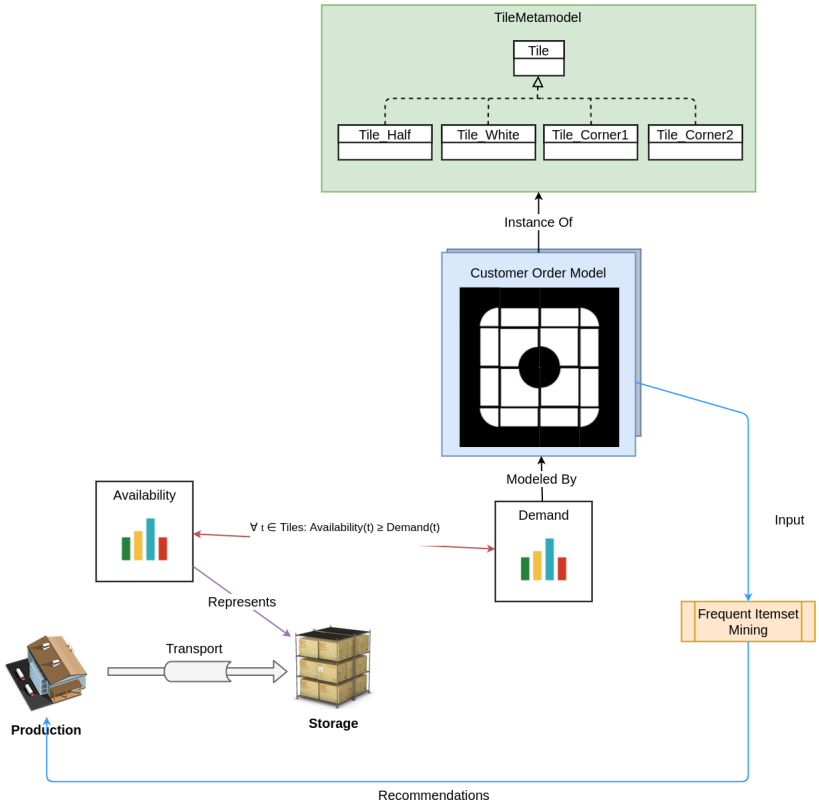

Figure 1: Concrete syntax of the tile formalism.

chain. In figure 1, the full diagram of the tile factory is shown.

To improve customer experience, the company decided to use a visual domain specific language in which users can model their floors and how they want to arrange the tiles. This allows customers to have an idea of the final result and to make better estimations for the amount of tiles they need. In the diagram of figure 1, the metamodel of this DSL can be seen in the *TileMetamodel* box, an example of an instance model can be seen in the *Customer Order Model*.

For the storage facility, a domain specific language is used to model the current availability of each tile. This model imposes constraints on the tile order models. After a certain amount of time a new order will be placed with the factory.

A very well known problem in such a supply chain is the high cost of storing

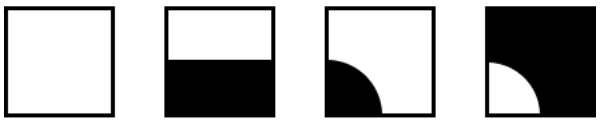

Figure 2: Concrete syntax of the tile formalism.

the parts. Companies thus need a smart way of making the orders. A recommender system can inspect the database of order models. By using the frequent dataset mining algorithm, we can compute the confidence of associations rules like: If tile A and B are frequently bought together, it likely that tile C will also be bought. This information can be sent to the production process and this will allow the production process to make a better estimation on which tiles are needed.

In the next chapter we will discuss how the algorithm will be implemented, using an explicit model.

3. The Explicitly Modeled Algorithm

In this section we will take a closer at an algorithm for frequent dataset mining, the FP-Growth algorithm, and how we made an explicit model of this algorithm.

3.1. The FP-Growth Algorithm

The algorithm used for this example was the FP-Growth algorithm [2]. This algorithm tries to deal with both time efficiency as well as memory efficiency. In this chapter we will discuss how the algorithm operates and how it was explicitly modeled.

3.1.1. The Existing Algorithm

The FP-Growth algorithm uses frequent pattern trees (or FP-tree) to represent the frequent datasets. Each node in the fp-tree represents an item that can occur in a transaction set and a path represents a possible way of combining items in a transaction set. Each node also has an integer that

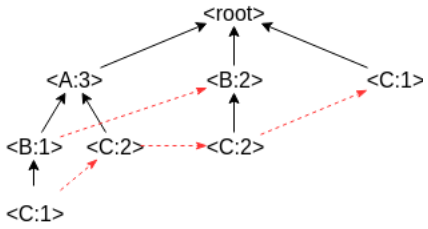

Figure 3: An example of an FP-tree.

represents the amount of occurrences. Note that the term transaction set, in this context, is used to denote a set of items that are bought together in a transaction.

To limit the size of the tree, two approaches are used. First, the fp-tree orders its transaction sets. Since the transaction set $< A, B >$ is the same as $< B, A >$, they are both represented by the same path in the tree. The algorithm thus expects an ordered set. The second approach is that the tree is an extended prefix tree structure. If two transaction sets have a common prefix, they share the nodes that correspond to the items in the common prefix.

The FP-tree is then constructed as follows. The algorithm handles every transaction set in a single scan of the database. For every transaction set there exists a unique corresponding path, starting from the root, in the FP-tree. The amount of occurrences of each node in that path is then updated.

3.1.2. The FP-Tree Formalism

As mentioned before, the idea is to explicitly model a frequent dataset mining algorithm. The FP-Growth algorithm looked very promising, since the FP-Tree can be considered a model of the dataset.

The metamodel of the FP-tree formalism can be seen in figure 4. On this modeling language, transformation rules can be defined to compute the confidence of certain associations rules.

To apply this algorithm on an instance model of some formalism, we need a way to express how the elements in the instance model of the source language correspond to the nodes in the FP-Tree model. For this correspondence model we created the Transaction formalism, we will later discuss how the

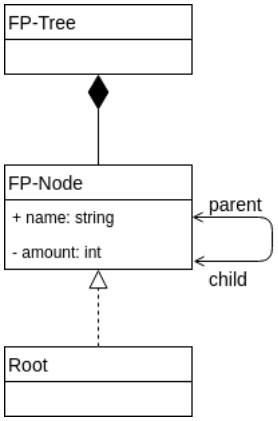

Figure 4: The metamodel of the FP-tree formalism.

instance model can be transformed into Transaction models. Once we have a transaction set model, we will match it to a path from the FP-tree and update the amount of occurrences.

3.2. The Transaction Set Formalism

The transaction set formalism is used to model a transaction sets and is the glue between the source model and the FP-tree model. In this section we will discuss how the formalism works and how the source model can be transformed into a transaction set model.

Usually this algorithm is used in a context with a lot of classes and only a small amount of instances. If the set $I = \{a_1, a_2, ..., a_n\}$ denotes the different classes that can be ordered in a transaction, a transaction set is a subset of I, ignoring the number of occurrences of each item in a single transaction. In the context of DSL's and modeling languages, the number of classes is often much smaller and the amount of instances of each class much larger, they also have a very specific semantics. Therefore we need to take into account that the generated transaction sets represent the original model in a meaningful way. This is where the explicit modeling approach really shows its benefit. The implementation is visual and can be altered directly, without having knowledge of any programming language. People who have an extensive knowledge about the semantic domain, but not about computer science, can easily tailor the algorithm to their needs and reason about how the transactions should be represented.

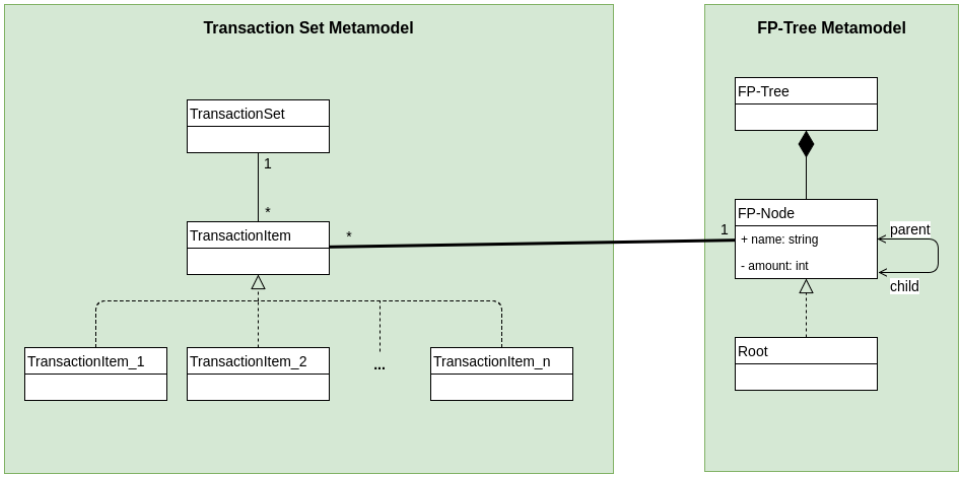

Figure 5: The Transaction Set Metamodel, the FP-Tree metamodel and their correspondence link.

In the next section, we will discuss how we will apply this explicitly modeled algorithm to the motivational example.

4. Applying the algorithm to the example

We will now discuss how we can apply the algorithm to the example discussed in section 2. We will also discuss the necessary steps to auto generate the transaction formalism.

4.1. From Order Model To Transaction Model

In section 3.2, we discussed that transforming the source model into a transaction is not so trivial. The problem is that DSL's often have a small amount of classes, but a big amount of instances, the tile formalism is perfect example for this. If we would apply the normal approach of making a single transaction for each model, containing only one item for each class of tile, if it is present in the model, the output of the algorithm won't have a significant output.

If we look at an example model in figure 6, we want to express that the items half tile and the the black corner tile appear more frequently together than the half tile and the black white tile. The approach we used, was to

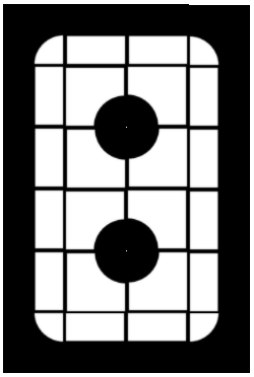

Figure 6: An example of an order model.

generate as much of the largest possible transaction set as possible, without reusing tile instances. Then repeat the process for smaller sets, until every tile instance is represented in a transaction.

If this approach would not fit the needs of the factory, the algorithm can be altered immediately.

When the model of the transaction set is created, we can again use explicitly modeled transformations. These transformations are very general, due to the fact that there is a superclass for transaction items and that every transaction set has a unique corresponding path in the FP-tree.

4.2. Dealing With Composite Patterns

A problem specific for visual formalisms is that a user might be interested in a transaction item that represents a composition of elements from the source language. An example in the tile formalism is a composition of corner tiles that together form a circle, this pattern can be seen twice in figure 6.

Since a composite pattern is composed of atomic elements, the question raises whether we are still interested every atomic element, or only the elements that are not part of a composite pattern.
In the example, we considered every element, whether is part of a pattern or not.

Creating these patterns and transforming them into the transaction set items requires extensive domain knowledge. This is another example of the usefulness of giving the domain experts the tools to custom tailor the algorithm.

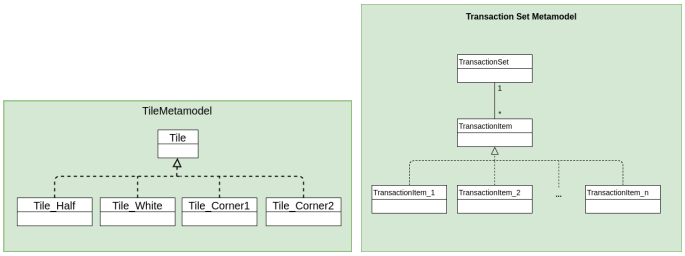

Figure 7: The tile metamodel (left) and the transaction metamodel (right).

In our example the transaction set formalism and the transformations on the formalism are implemented by hand. However, it should be feasible to auto generate this language. For every non abstract class in the source language, there should be a corresponding transaction item class, that is a subclass of TransactionItem. For the composite patterns, an extra formalism is needed in which the domain knowledge can be captured in patterns. The technique for every non abstract class can be used for every instance model in the composite pattern formalism. Afterwards, a transformation should be generated, that transforms an instance of a class or pattern in the source model into the corresponding transaction item.
Due to some limitations in the model transformations of AToMPM, we weren't able to auto generate the language, but the approach should work with some changes to the transformation mechanics or by using code generation to create the json files of the models.

5. Related Work

Models and modeling formalisms have been used widely in the field of supply chains. A multiagent approach for modeling and simulating a supply chain was proposed by Swaminathan et al. [6]. The multi-agent system allows the different components to make autonomous decisions. Their formalism is used to model and simulate supply chains, our approach is more focused gathering information and making recommendations in real time.

Waller et al. [7] discussed the need for experts in data science in a supply chain management context. They mention a big lack of research in this field. Our approach tries to deal with this problem by making the data mining algorithms more accessible to the domain experts.

The benefits of explicit modeling has been shown by Khüne et al.[4]. They proposed a way of explicitly modeling model transformations.

Augmenting DSLs with lower-level constructs is not a novel idea. Meyers et al. [5] investigate the possibility of incorporating property-based reasoning on the DSL level. In their case, the reasoning typically captured LTL and CTL formulas is elevated to the level of the DSL, resulting in much easier design workflows for the stakeholders working with the DSL. Our approach is similar in a sense that we also incorporate a reasoning typically captured below the level of the DSL.

6. Conclusion And Future Work

In this paper we proposed an approach to deal with information gathering in modeling languages and DSL's by explicitly modeling the data mining algorithm. For the scope of this project, we only considered the frequent dataset mining algorithm.

The idea was to create an explicit and visual model that models the algorithm under study. The explicitly modeled algorithm has the advantage that both model and algorithm are on the same level of reasoning. The implementation can also be altered on the spot, allowing fast adaptations to changes in the industry.

We also applied the algorithm to an example of a simplified production line. This example showed that using data mining the order models can reduce the storage cost by making smarter orders to the production factory. We showed that a problem in this setup is that the mining algorithm often needs custom tailoring, but the domain experts often don't have a strong background in computer science. The explicitly modeled algorithm offers the domain experts a means of tailoring the algorithm to their needs, without the need of extensive knowledge on programming.

In future work, we will be generalizing this approach. We will aim to propose a framework in which the DSL and the mining algorithm can be changed independently. We will also take a closer look into the auto generation of the correspondence models (in the example, this was the transaction set formalism).

By explicitly modeling the algorithm, we create a certain overhead, which

has an impact on performance. We will try to analyze this cost in performance and how we can optimize it.

Bibliography

[1] Andrej Dyck, Andreas Ganser, and Horst Lichter. Model recommenders for command-enabled editors. *MDEBE2013*, 2013.

[2] Jiawei Han, Jian Pei, and Yiwen Yin. Mining frequent patterns without candidate generation. In *ACM sigmod record*, volume 29, pages 1–12. ACM, 2000.

[3] Dietmar Jannach, Markus Zanker, Alexander Felfernig, and Gerhard Friedrich. *Recommender systems: an introduction*. Cambridge University Press, 2010.

[4] Thomas Kühne, Gergely Mezei, Eugene Syriani, Hans Vangheluwe, and Manuel Wimmer. Explicit transformation modeling. In *International Conference on Model Driven Engineering Languages and Systems*, pages 240–255. Springer, 2009.

[5] Bart Meyers, Romuald Deshayes, Levi Lucio, Eugene Syriani, Hans Vangheluwe, and Manuel Wimmer. Promobox: A framework for generating domain-specific property languages. In *International Conference on Software Language Engineering*, pages 1–20. Springer, 2014.

[6] Jayashankar M Swaminathan, Stephen F Smith, and Norman M Sadeh. Modeling supply chain dynamics: A multiagent approach. *Decision sciences*, 29(3):607–632, 1998.

[7] Matthew A Waller and Stanley E Fawcett. Data science, predictive analytics, and big data: a revolution that will transform supply chain design and management. *Journal of Business Logistics*, 34(2):77–84, 2013.