Institute of Parallel and Distributed Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Master Thesis

Modeling Recommendations for Pattern-based Mashup Plans

Somesh Das

Course of Study:	Computer Science
Examiner:	Prof. Dr.-Ing. habil. Bernhard Mitschang
Supervisor:	Dipl.-Inf. Pascal Hirmer
Commenced:	October 17, 2017
Completed:	April 17, 2018
CR-Classification:	H.2.8

Abstract

Data mashups are modeled as pipelines. The pipelines are basically a chain of data processing steps in order to integrate data from different data sources into a single one. These processing steps include data operations, such as join, filter, extraction, integration or alteration. To create and execute data mashups, modelers need to have technical knowledge in order to understand these data operations. In order to solve this issue, an extended data mashup approach was created–FlexMash developed at the University of Stuttgart–which allows users to define data mashups without technical knowledge about any execution details. Consquently, modelers with no or limited technical knowledge can design their own domain-specific mashup based on their use case scenarios.

However, designing data mashups graphically is still difficult for non-IT users. When users design a model graphically, it is hard to understand which patterns or nodes should be modeled and connected in the data flow graph. In order to cope with this issue, this master thesis aims to provide users modeling recommendations during modeling time. At each modeling step, user can query for recommendations. The recommendations are generated by analyzing the existing models. To generate the recommendations from existing models, association rule mining algorithms are used in this thesis. If users accept a recommendation, the recommended node is automatically added to the partial model and connected with the node for which recommendations were given.

Contents

List of Figures

List of Tables

Listings

List of Algorithms

1 Introduction

Mashup plans that were introduced by Hirmer et al. [HRWM15], are graphical models that enable domain-specific modeling of data mashups. Based on Mashup plans an approach named FlexMesh is introduced by Hirmer et al. [HM16], that allows modeling and pattern based execution of data mashups. FlexMash provides a non-technical, domain-specific model where users can define data processing and integration scenarios based on their use case scenarios without the need of any implementation and execution details. This relieves non-technical users from the technical challenges that arise during implementing their own data processing and integration scenarios.

However, in FlexMash, designing a model graphically is difficult for domain-users who do not have enough technical knowledge. For example, non-technical users who want to design their own use case specific scenarios but do not know which patterns or nodes to use during the design of the graphical models. They might have a lack of knowledge which patterns or nodes they need in order to achieve their desired model. It might happen that, users do not know about the functionalities of the nodes offered by FlexMash. Typically, what people do, when they need a solution for a modeling problem, they ask more skilled people for help or search the internet. In this case, models that have been executed successfully in the past, could be the most useful information. Existing models could be analyzed to know which node to use in the model.

To solve this challenge, this master thesis aims to offer recommendations interactively to users during each step of their model design. These kinds of recommendations are generated by analyzing existing models. At each step, users can query for recommendations which node to use.

The goal of this thesis is to assist users during development of mashup models in a modeling environment as provided by FlexMash by recommending nodes similar to Amazon's recommendation feature that recommends products that other customers bought as well.

In this thesis, the FlexMash application is extended by implementing the modeling recommendation feature. An interactive recommendation feature is built in the original FlexMash application to assist users during mashup model development. The main goals of this thesis are:

- Increase usability of FlexMash for domain users.
- Increase the flexibility of the mashup model development.

- Assist users by providing node recommendations. And also automatically add the recommended node, selected by users into the modeling canvas on behalf of the users.

A key component that is most important in order to achieve these goals is, the algorithm for generating recommendations. In this thesis, association rule mining algorithms are used for this. Another important task is to transform all models into a generic modeling structure.

The thesis document has provides the following structure:

- **Chapter 2** provides the fundamentals needed to understand the concepts of this thesis including basic concepts of association rules mining and an overview of the FlexMash application.
- **Chapter 3** discusses the related work for this thesis.
- **Chapter 4** describes the conceptual overview of this thesis, the different steps taken in order to achieve the goals, such as creation of canonical models, selection of algorithms, approach for generating association rules, and integration into FlexMash.
- **Chapter 5** describes the implementation details of this work. It includes the description of the technologies used, code snippets, and screenshots of the developed user interfaces.
- **Chapter 6** discusses the overall summary of this thesis work and future works.

2 Fundamentals

In this chapter, the basic fundamentals that are needed for this thesis are described. This thesis work is done based on the concept of Association Rules. The chapter summarizes the theoretical concepts related to Association Rules and the algorithms for generating Association Rules. An overview of the FlexMash application is also given in this chapter.

2.1 Association Rules

The identification of association rules is a very important task in the field of data mining. It was first introduced by Agrawal et al. [AIS93]. The main objective behind association rules is to identify frequent patterns, associations or interesting correlations within the data stored in transactional databases. The idea of association rules is frequently used in many areas, such as market basket analysis, telecommunication networks, and inventory controls.

The most common example of association rules mining is market basket analysis. In market basket analysis, different buying habits of customers are discovered and analyzed to find out the associations between items that customers bought. Association rules mining helps the seller to figure out different types of marketing plans and inventory management strategies. Items that are frequently bought together can be placed in a bundle offer. For example, if the customer who buys bread also wants to buy cheese at the same time, the seller can offer a reduced prices for buying both bread and cheese together. These may help to increase the sale of both items. A simple association rule can be defined as follows:

$Bread \rightarrow Cheese[support = 0.2, confidence = 0.7]$.

This rule expresses a relationship between *Bread* and *Cheese*. The *support* measure defines that *Bread* and *Cheese* appeared together in 20% of all transactions. The possibilities of a transaction involving *Bread* also involving *Cheese* within the same transaction is defined by *confidence* measure. In this case, 70% of all recorded transactions involving *Bread* also involved *Cheese*. So It can be assumed that customers who buy *Bread* are also likely to buy *Cheese*.

Association rule mining is user-centric because its objective is to investigate interesting rules which can be used to discover knowledge. The meaning of interestingness of rules is that they are non-trivial and significant. These rules can be used for further interpretation by the user.

Argawal et al. in [AIS93] [AS94] described an algorithm for mining association rules. The fundamentals of association rules mining and itemset identification are well established and accepted.

2.1.1 Basics

The mining association rules problem can be stated as follows: Let, a set of items is $I = \{i_1, i_2, i_3, ..., i_m\}$ and a set of transactions is $T = \{t_1, t_2, t_3, ...t_n\}$. Each transaction contains items of the itemset I. So, each transaction t_i is a set of items such that $t_i \subseteq I$. An association rule is an implication of the form: $X \rightarrow Y$, where $X \subset I$, $Y \subset I$ and $X \cap Y = \emptyset$. X(or Y) is a set of items, called an *itemset* [Liu11]. For example, a simple association rule can be defined as $\{bread\} \rightarrow \{cheese\}$.

Assume an association rule of the form $X \rightarrow Y$, where X is called *antecedent* and Y is called *consequent*. It is obvious that the value of consequent is the implication of the value of antecedent. The antecedents are also called “left-hand side” of the rule, it can consist of either one item or a whole set of items. The consequents are called “right-hand side” and it also can be a single item or a whole set of items.

In the whole association rule mining process, the most complicated task is to generate frequent itemsets. Many different combinations of items have to be identified which requires computation-intensive tasks, especially in large databases. It requires an efficient algorithm that can extract itemsets in a minimal time. Often, between exploring all itemsets and computation time, a compromise has to be made. Generally, only those itemsets are taken into account that have certain support. Confidence and support are the two important measures for evaluation of the interestingness of a rule.

- **Support** : The support of the rule $X \rightarrow Y$ is defined by the percentage of transactions in T with $X \cup Y$. How frequent a rule is applicable to the transaction set T is determined by the support of this rule. The formula for representing support is, defined as follows :

$$support = \frac{(X \cup Y) \cdot count}{n}$$

  Support represents the frequency of the occurrence of the rule. The rules that cover only a few transactions might not be useful.

- **Confidence** : The confidence of a rule represents the percentage of transactions in T which contains X and also Y. It measures the conditional probability, $Pr(Y \mid X)$ [Liu11]. It is computed as follows:

$$confidence = \frac{(X \cup Y) \cdot count}{X \cdot count}$$

  This is an important measurement for calculating interestingness of a rule. It searches in all transactions which contain a certain item or itemset defined by the antecedent of the rule [Hel07]. Then, it calculates the percentage of the transactions which are also including all the items contained in the consequent.

2.1.1.1 The Process

The process of mining of association rules includes two main parts. First, identifying frequent itemsets in the data. Second, generation of rules from the identified frequent itemsets.

- **Mining Frequent Patterns** In this step, all the itemsets that appear as frequent as the minimum support specified by the user needs to be discovered. Computation time is an important issue because in case of large databases, lots of possible itemsets need to be evaluated. There are different algorithms for finding frequent patterns efficiently. Some of those are discussed in Section 2.1.4.
- **Discovering Association Rules** After generation of all patterns according to minimum support requirements, rules can be generated. A minimum confidence is required to do so. All possible rules have to be generated out of the frequent itemsets and their confidence has to be compared with the minimum confidence defined by the user. The rules which meet this requirement are considered as interesting. At the end, all the discovered rules can be presented to the user with their support and confidence values.

2.1.2 Binary Association Rules

The term binary association rules indicates the classical association rules in market basket analysis. Here, it is possible to define with a boolean value whether a product is in a transaction or not (true or false, represented by 1 and 0). Therefore, every transaction can be represented as a binary attribute with domain $\{0, 1\}$. The formal model is represented in [AIS93] as follows: *"Let $I = i_1, i_2, ..., i_m$ be a set of binary attributes, called items. Let T be a database of transactions. Each transaction t is represented as a binary vector, with $t[k] = 1$if t bought the item i_k , and $t[k] = 0$ otherwise. There is one tuple in the database for each transaction. Let X be a set of some items in I . We say that a transaction t satisfies X if for all items $i_k \in X$, $t[k] = 1$."*

As already discussed, an association rule is an implication of the form $X \rightarrow Y$ where X and Y are itemsets that are contained in itemsets I and X does not include *Y*. A rule in transactions T with the confidence factor $0 \leq c \leq 1$ is called *satisfied* if the percentage of transactions, contained in T that support X also support Y, is equal to the factor c [Hel07]. The notation $X \rightarrow Y \mid c$ can be used to represent the rule with the confidence factor of c.

In [AIS93], Argawal et al. divided the problem of rule mining into two subproblems:

- All combination of items containing support above user-defined minimum support have to be identified. The sets of itemsets which show sufficient support are called large or frequent itemsets and the itemsets which do not shows sufficient support are called small itemsets. It is also possible to consider syntactic constraints, for example, only those rules are taken into consideration that contain a certain item in the antecedent or the consequent [Hel07].

ID	Age	Salary
101	18	9000
102	35	15000
103	26	21000
104	39	25000
105	31	11000

Table 2.1: Sample Database

After discovering the itemsets that meet the minsupport requirements, it is also necessary to check whether it meets the requirement of the confidence factor c. At this stage, only the large itemsets that were defined previously have to be considered. The confidence is the ratio of the support of the whole itemset and the support of the antecedent.

- After solving the first problem, the solution of the second subproblem is quite straightforward. The Apriori algorithm was developed as first and best-known algorithm nowadays for mining association rules. The Apriori and FP-Growth algorithms are discussed in detail in Section 2.1.4

2.1.3 Quantitative Association Rules

An overview of binary association rules is given in the previous section, where boolean values are used to represent the items. However, in reality, databases not only contain boolean attributes but also quantitative or categorical ones that cannot be mined using the classical technique. Identifying rules in such kind of data can be represented as the quantitative association rules problem [SA96]. To deal with quantitative attributes a possible approach is replacing them with several boolean attributes. It is straightforward to map quantitative attributes into binary values if they are categorical or if there are only a few values.

For example, the value of boolean field corresponding to $(attribute1, value1)$ would be "1" if $attribute1$ had $value1$ in the original data, and "0" otherwise [SA96]. This only works if the original data has a very small number of values. It is required to split the values into intervals and map each attribute to the corresponding new boolean attribute when the number of different values increases. Now, to identify association rules, a binary algorithm can be used.

In Table 2.1, a sample database is shown. Here we can see that all the attributes are numeric so it is important to create intervals for all attributes. For each record of the table, an appropriate interval needs to be chosen. Then, each record is mapped to the corresponding new binary attribute. The number of columns in the new table is equal to the number of intervals chosen for each attribute. A mapping table with sample intervals chosen for each attribute is shown in Table 2.2.

ID	Age: <20	Age: 20-32	Age: >32	Sal: <10000	Sal:10000-19999	Sal: >19999
101	1	0	0	1	0	0
102	0	0	1	0	1	0
103	0	1	0	0	0	1
104	0	0	1	0	0	1
105	0	1	0	0	1	0

Table 2.2: Mapping Table

With this mapping method, two problems arise [SA96]:

- **"MinSupport"** : If the interval count for a single quantitative attribute is high, there could be a single interval which has low support. It is necessary to take larger intervals, otherwise, some existing rules containing this attribute may not be identified because they lack minimum support.
- **"MinConfidence"**: If a larger interval is taken in order to solve the first problem, another problem arises. The rate of information loss increases if the interval sizes become larger. Then, the appearance of determined rules might be different to the original data.

If the intervals are too large, it might not be possible to reach the minimum confidence, if they are too small, it might not be possible to achieve minimum support. The "MinSupport" problem can be solved by considering all potential continuous ranges over the ranges of the quantitative attribute. Increasing the number of intervals, without encountering the "MinSupport" problem, could be a solution for the "MinConfidenece" problem. By increasing the number of intervals and combining adjacent intervals simultaneously, creates two new problems:

- **"ExecTime"**: By using the above method, the number of items per record increases, so the execution time will increase as well.
- **"ManyRules"**: If a value has minimum support, any range that contains this value will also have minimum support. Hence, the number of rules will increase and many of them will not be interesting.

There is a trade-off between the above mentioned problems. If more intervals are built to cope with the "MinConfidence" problem, then execution time will increase and additionally, many rules might be generated which are not interesting.

2.1.4 Algorithms for Finding Association Rules

Since the introduction of the Apriori algorithm [AIS93], several algorithms have been developed. Those algorithms focus on the efficiency of finding a frequent pattern or

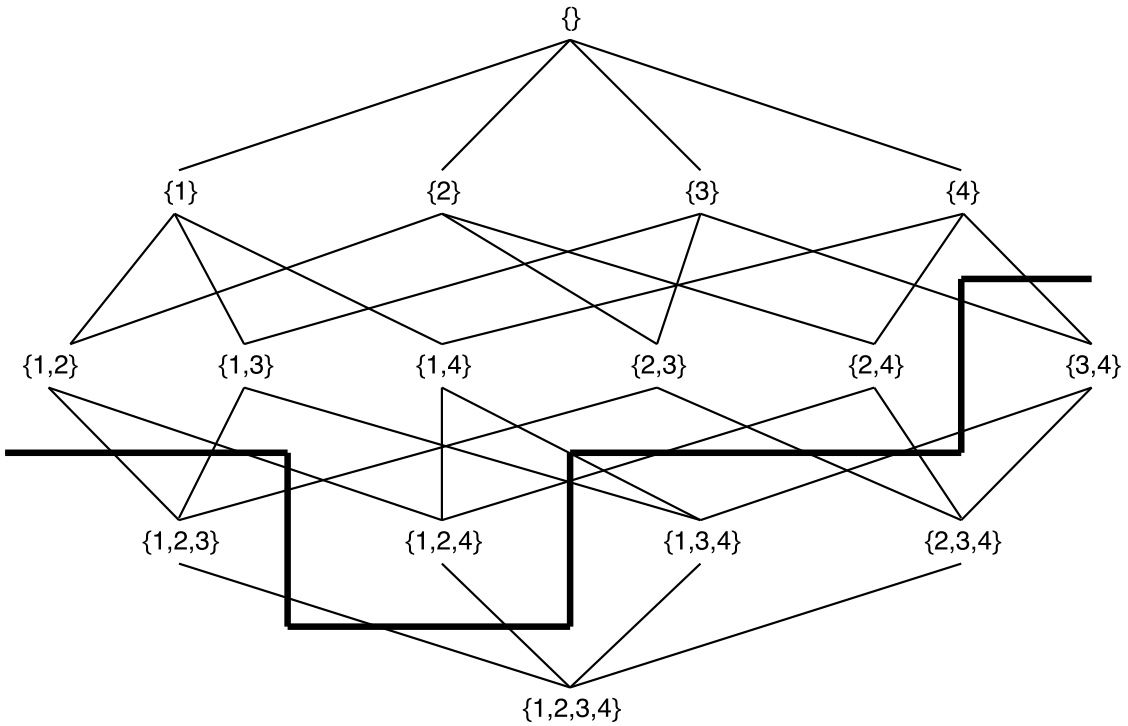

Figure 2.1: Representation of the itemsets [HGN00]

association rule identification. Apriori provides solutions for both problems. In this section, a brief overview of some important mining algorithms is given. Most of the algorithms work with binary association rules, but they also work with the quantitative association rules as well.

For developing an association rule mining algorithm there exist two main approaches, one is called breadth-first search (BFS) and depth-first search (DFS)[HGN00]. A lattice including all possible combinations of an itemset is shown in Figure 2.1.

The border between frequent and infrequent itemsets is represented by the bold line. All the items that lie above the border satisfy the minimum support requirements. The algorithms locate this border. In BFS, first, the support is computed for all itemsets in a specific level of depth, whereas DFS recursively subsides the structure by several depth levels. The algorithms for association rule mining can be systematized as depicted in Figure 2.2.

2.1.4.1 Apriori

The Apriori algorithm was the first developed algorithm for mining association rules out of a large dataset. It has been first introduced by Agrawal et al. [AS94]. The algorithm can find frequent patterns and generates association rules out of the frequent patterns.

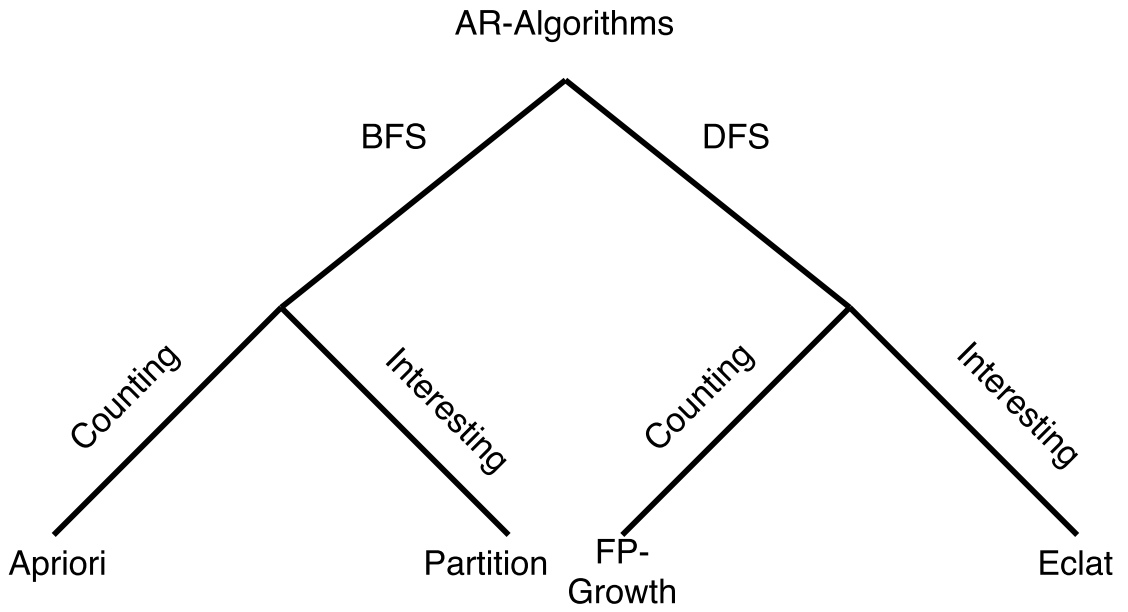

Figure 2.2: Systematization of Algorithms [HGN00]

k -itemset	An Itemset having k items
L_k	"*Set of large k -itemsets (those with minimum support). Each member of this set has two fields: i) itemset and ii) support count.*"
C_k	"*Set of candidate k -itemsets (potentially large itemsets). Each member of this set has two fields: i) itemset and ii) support count.* "

Table 2.3: Notation [AS94]

2.1.4.1.1 Discovering Frequent Itemsets A fact that is used to generate frequent itemsets is that any subset of large itemset must also be large as well. The size of an itemset is determined by the number of items contained in the itemset, an itemset is called $k-itemset$ where k is the size. The items in the itemset are stored in lexicographic order. The notation shown in Table 2.3 is used to represent the algorithm.

Each itemset associates a count field where the value of support is stored. Algorithm 2.1 shows the pseudocode of the Apriori algorithm. At first, the database is supplied for counting the occurrences of single elements. If the support value of a single element is below the minimum support that is defined by the user, it is not taken into consideration anymore. A subsequent pass k is composed of two phases:

- To generate the candidate itemsets, C_k for the current pass, the discovered large itemsets of pass k-1 , are used.

Algorithm 2.1 Apriori algorithm Candidate Generation [AS94]

1: F_l=(Frequent itemsets of cardinality 1);
2: **for** ($k = 1; F_k \neq \phi; k++$) **do**
3: C_{k+1} = apriori-gen(F_k); // New Candidates
4: **for** all transactions $t \in Database$ **do**
5: $C'_t = subset(C_{k+1}, t)$; // Candidates contained in t
6: **for** all candidates c $\in C'_t$ **do**
7: c.count++;
8: **end for**
9: F_{k+1} = $\{C \in C_{k+1} | c.count \geq minimumsupport\}$;
10: **end for**
11: **end for**
12: Answer $\cup_k F_k$

- The database is searched again determining the support for the candidate itemsets C_k. The candidates which have a support above the minimum support will be included to the large itemsets. To prevent a long counting duration, identifying the right candidates is important.

The function *apriori-gen* shown in Algorithm 2.1 takes the itemsets of the previous iteration as an input. These itemsets are joined together, composing itemsets with one more item than in the previous step. Then in the prune step, those itemsets will be removed whose sub-combinations have not been part of the discovered sets in previous iterations. A hash-tree is used to store the candidate sets. This tree can either have a list of itemsets, called a leaf node, or a hash table, which is called an interior node.

The function *subset* starts traversing the hash-tree from the root node and traverses until the leaf nodes for finding all candidates that are in a transaction t. The function will ignore the itemsets which start with an item that is not in t.

2.1.4.1.2 Discovering Association Rules Association rules can have multiple elements in the antecedent and also in the consequent. To generate the association rules, only large itemsets are used. The first step of the procedure is to find all possible subsets of the large itemset l. A rule is defined in the form $a \rightarrow (l - a)$ for each of those identified subsets. If the confidence of the rule is greater than the user-defined minimum confidence, then the rule is considered as interesting. All subsets of l are discovered so that any possible dependencies are not missed. The algorithm for generating association rules is shown in Algorithm 2.2.

2.1.4.2 Frequent Pattern Growth (FP-Growth)

The FP-Growth algorithm generates frequent itemsets. It tries to avoid generating a large candidate set like the Apriori algorithm. The basis of this algorithm is a solid representation

Algorithm 2.2 Apriori algorithm Association Rule Generation [YKTZ11]

```
1: for each frequent itemsets i_k (k ≥) do
2:     H_1 = {h ∈ l_k}|cf(l_k − {h} ⇒ {h}) ≥ min_cf
3:     Call Ap-GENRULE(l_k, H_1);
4: end for
5: procedure AP-GENRULE(l_k, H_m )
6:     if k > m + 1 then
7:         H_{m+1} = apriori_gen(H_m);
8:         for all h_{m+1} ∈ H_{m+1} do
9:             cf = sp(l_k)/sp(l_k − h_{m+1})
10:            if cf ≥ min_cf then
11:            else H_{m+1} := H_{m+1} − {h_{m+1}};
12:            end if
13:            Ap-GENRULE(l_k, H_{m+1});
14:        end for
15:    end if
16: end procedure
```

of the original data set without losing any information. This is done by constructing a tree, using the data. This tree is called the Frequent Pattern Tree, FP-Tree in short. The FP-Growth algorithm has been introduced in [HPY99]. Here, the FP-Tree is constructed out of the original data set first, and then the frequent patterns are generated from the tree. Before applying the algorithm, the data should be preprocessed in order to minimize the execution time.

2.1.4.2.1 Preprocessing the Data The FP-Growth algorithm applies following preprocessing steps for efficiency [Bor05]:

- First, the initial dataset is scanned and for each item, the support is calculated. Then, all items that have a support below the user defined minimum support are discarded from the transactions.
- The remaining items are stored in a decreasing order according to their support.

Although, the algorithm works fine without sorting, it works much faster after sorting [Bor05]. If increasing order is used instead of decreasing order, it performs worse than using a random order [Bor05]. Table 2.4 provides an example of preprocessing steps for a transaction of the FP-Growth algorithm.

2.1.4.2.2 Constructing the FP-Tree An FP-Tree can be constructed out of the preprocessed data. A scan over the database is done for adding each itemset to the tree. The first branch of the tree will be the first itemset. In case of the transaction shown in Table 2.4, the items *b*,*d* and a would be the first branch. The second transaction has the prefix *bd* which already exists in a set of the tree. In this case, the count of each node along the path

OriginalDB		**PreprocessedDB**
abd	support(b)=6	bda
bcde	support(d)=5	bde
bd	support(e)=5	bd
ade	support(a)=4	dea
ab	support(c)=2	ba
abe		bea
cde	minsupport=3	dec
be		be

Table 2.4: FP-Growth Preprocessing

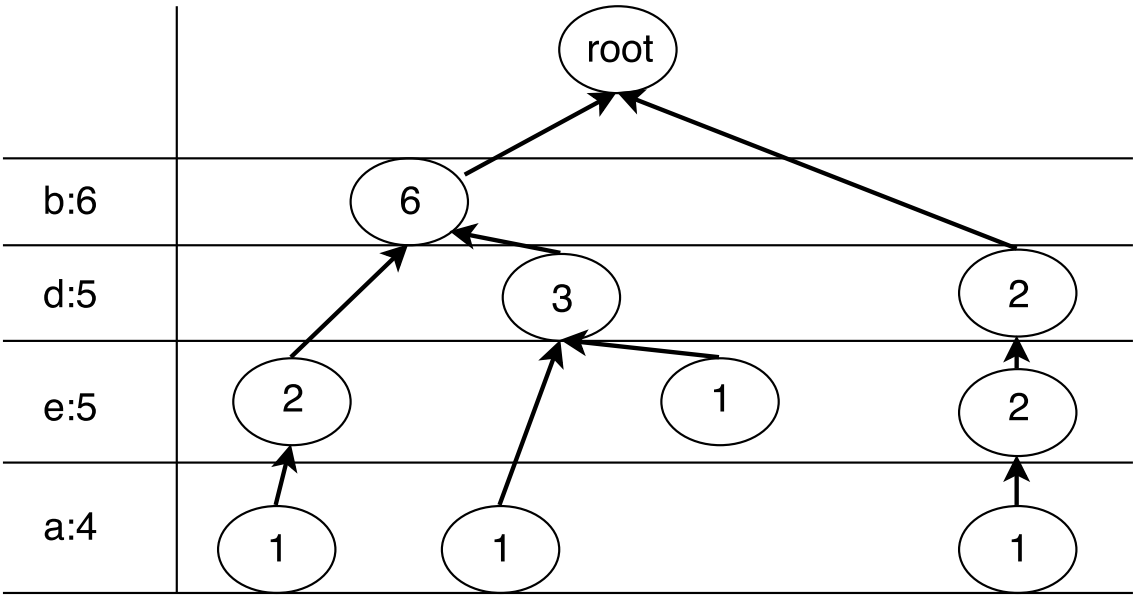

Figure 2.3: FP-Tree [HGN00]

of the common prefix is increased by one, and for the remaining items, new nodes will be created and linked as a child. In case of the example shown in Table 2.4, only a new node for *e* will be created and linked as a child of *d*. For the transaction database of Table 2.4, the generated tree is shown in Figure 2.3. It represents the database without losing any information.

Each node of the FP-Tree has three fields [HPY99]:

- item-name: The name of the item representing the node is stored in this field.
- count: The accumulated support of the node within the current path is represented by the count field.

item	conditional pattern base
a	$\{\langle b, e\colon 1\rangle, \langle b, d\colon 1\rangle\langle d, e\colon 1\rangle\}$
e	$\{\langle b\colon 2\rangle, \langle b, d\colon 1\rangle\langle d\colon 1\rangle\}$
d	$\{\langle b\colon 3\rangle\}$
b	$\{\emptyset\}$

Table 2.5: Conditional Pattern Bases

- node-link: It represents the link between the nodes. It stores the ancestor of the current node, and null in case there is none.

After building the FP-Tree, the database is not required anymore for mining. Now the FP-Tree can be used. The support of an itemset can easily be calculated by traversing the path and using the minimum value of count from the nodes. For example, the support of itemset $\{b, e\}$ will be 2 and the support of itemset $\{b, e, a\}$ will be 1.

2.1.4.2.3 Mining the FP-Tree using FP-Growth The FP-Tree provides an efficient structure for mining, however one may still encounter the combinatorial problem of candidate generation which needs to be solved. For identifying all frequent itemsets, the FP-Growth algorithm takes a look at each level of depth of the tree [PEl16]. It starts from the bottom and generates all possible itemsets that include nodes in that specific level. After generating frequent patterns for each level, they are stored in the complete set of frequent patterns. The procedure of the algorithm is shown in Algorithm 2.3.

The algorithm is executed at each of these levels. The tree is first checked for the number of paths it contains for finding all the itemsets containing a level of depth. If the tree is a single path tree, all possible combinations of the items contained in it will be generated [PEl16]. Then, these will be added to the frequent itemsets.

If the tree has more than one path, then for the specific depth, a conditional pattern is constructed. In FP-Tree of Figure 2.3, for the depth level *a*, the conditional pattern base will consist of the following itemsets : $\langle b, e : 1\rangle$, $\langle b, d : 1\rangle$ and $\langle d, e : 1\rangle$. The item set is determined by traversing each path in an upward direction. The conditional pattern for all depth levels of the tree is shown in Table 2.5.

From the conditional pattern base, a conditional FP-Tree is built. This is done in the same way as the construction of the initial tree. The only difference is now that, the conditional pattern base is used instead of a transactional database. After having the conditional FP-Tree, the FP-Growth function is called recursively. This is done until the tree has only a single path or it is empty. At the end, all the items in the various condition FP-Tree are stored. Then, these items are returned as a list of frequent itemsets in the FP-Tree as well as in the database, respectively.

Algorithm 2.3 FP-Growth algorithm [HPY99]

procedure F(P)-Growth ($Tree, \alpha$)
  if Tree contains a single path P **then**
    for all combination (denoted as β) of the nodes in the path P **do**
      generate pattern $\beta \cup \alpha$ with support = minimum support of nodes in β ;
    end for
    for all a_i in the header of $Tree$ **do**
      generate pattern $\beta = a_i \cup \alpha$ with support=a_i . support ;
      construct β 's conditional pattern base and then β 's conditional FP-Tree $Tree_\beta$;
      if $Tree_\beta \neq \emptyset$ **then**
        call FP-growth($Tree_\beta$, β)
      end if
    end for
  end if
end procedure

2.2 FlexMash

In recent times, the data processing and integration is becoming complex because of an increasing size of IT systems used in enterprises and a growing connectivity between the corresponding data sources. This leads to high communication effort between domain-specific users, such as business persons and IT experts who implement the data processing. In most cases, this result in non-flexible solutions that work only for specific use cases. To solve this issues, a solution is required that allows users to define data processing and integration scenarios without defining any execution details.

Mashup plans that were introduced by Hirmer et al. [HRWM15], are graph-based models that enable domain-specific modeling of data mashups. Based on this Mashup plan, an approach named FlexMash is introduced by Hirmer et al. [HM16] that allows modeling and pattern based execution of data mashups. This tool transforms the Mashup Plans into an executable solution based on requirements defined by the use case scenario. Figure 2.4 shows a high-level view of the FlexMash tool.

First of all, a modeling of the Mashup Plan is required that describes the processing and integration of data. Then, a pattern is selected by the user that defines the requirements for mashup execution. Based on this selected pattern, the defined Mashup plan is transformed into an executable representation. Finally, the executable representation is executed in a suitable engine. It is possible to store and visualize the result of execution for later use.

This tool enables a flexible solution for data mashup execution that is specific to user requirements and uses transformation patterns selected by the user. Figure 2.4 shows a screen shot of the FlexMash application.

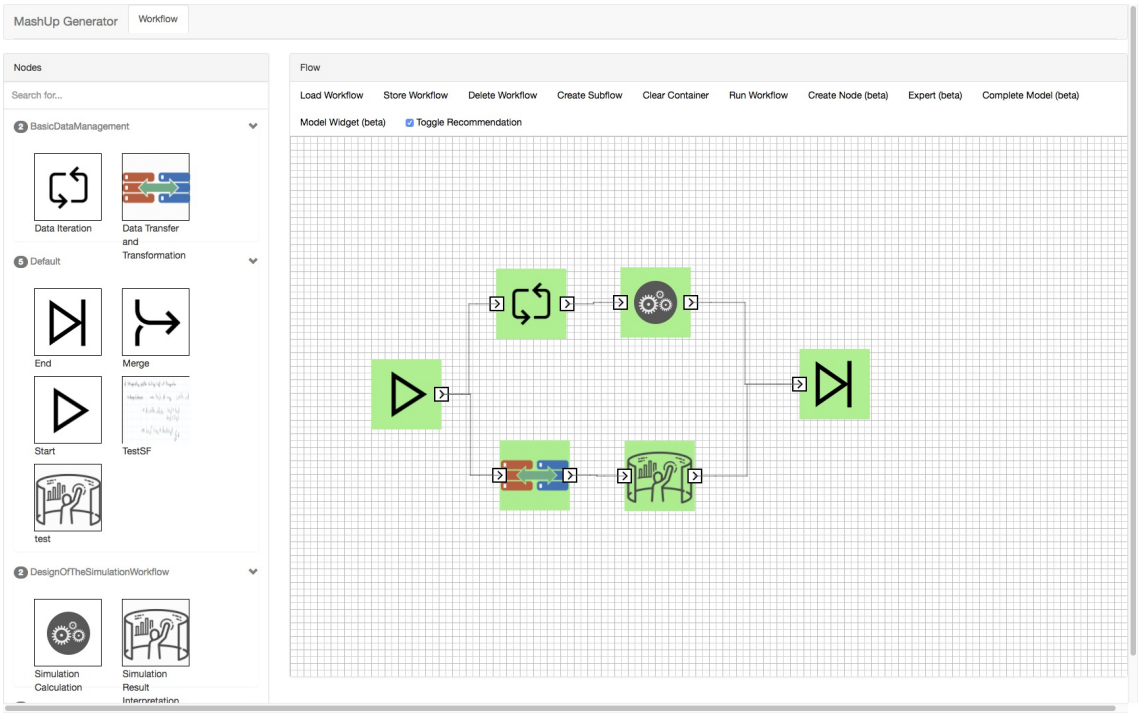

Figure 2.4: Screen Shot of FlexMash Application

3 Related Work

Rodríguez et al. [RCD+14] introduced an approach named Assisted Mashup Development. In this work, they use Yahoo! Pipes, a web-based mashup editor, as a mashup model development tool. They aim to assist users by recommending development knowledge. By mining existing mashup models, a set of composition patterns is discovered. These composition patterns are then interactively recommended to the users during mashup development. A canonical mashup model is described that is able to formalize different data flow mashup languages into a single modeling formalism. A set of composition patterns is then extracted out of this canonical mashup model. For each of the discovered patterns, a respective pattern mining algorithm is implemented that discovers the composition knowledge as reusable mashup patterns out of the stored mashup models. Finally, these patterns are recommended to the users based on the user actions on the modeling canvas. An automatic weaving of recommended patterns is also included. In this approach, they focus on how to automatically discover patterns from existing mashup models, recommend the discovered patterns fastly to the users, and automatically weave the recommended patterns into mashup models.

The goal of this thesis is to assist users to define a mashup plan for execution in the FlexMash environment. A set of transformation patterns is already introduced in FlexMash which users can select during the design of the model as a node. All data sources and operations are represented as nodes in FlexMash. The user can select data sources as well as operations from the node catalog. So pattern selection is out of the scope of this thesis. Instead of implementing a separate algorithm for each pattern, a common mining algorithm is used in this work which recommends the frequently used nodes together. It happen that, the Mashup modeler is a business user who does not have technical knowledge which nodes to model. In that case, the models that have been executed successfully in past can be analyzed to recommend the users which node or pattern to use. A canonical model is used to convert different models into a single modeling structure. Based on this canonical model, a data mining algorithm is used to generate the recommendation. The model designed in FlexMash is captured by the recommendation application during execution and converted into the canonical model. The Assisted Mashup Development approaches do focus on recommendation of composition patterns. They discover a set of composition patterns and design separate mining algorithms for each discovered pattern. However, the aim of this thesis is not only recommending the patterns but also the data sources that have been used previously. The work done in this thesis also provides an interactive recommendation feature. The difference is that, it is not only recommending nodes but also automatically completes the whole model using the top recommended nodes. So the user does not always need to have modeling knowledge. In addition, it also has a modeling

widget feature which can be used to design a partial model using the recommendation feature and weave this partial model into the main model in the modeling canvas. The recommendation is updated each time a mashup model is executed.

Roy Chowdhury at el. [CRDC12] introduced Baya, an extension of Yahoo! Pipes, for speeding up development by interactively recommending composition knowledge. It has two parts: one is the Baya recommendation server and the other is the Baya Firefox extension. The Baya recommendation server first takes the native models designed in the mashup tool and converts them into a canonical mashup model which is able to describe a different kind of similar mashup languages in a generic manner. Then a pattern miner runs a set of pattern mining algorithms on the canonical model to discover a set of patterns. Currently, Baya supports the following composition patterns: Parameter value pattern, Connector pattern, Connector co-occurrence pattern, Component co-occurrence pattern, Component embedding pattern, Multi-component pattern. These discovered patterns are stored in a canonical pattern database. A data transformer transforms and stores them into a persistent knowledge base. The Baya Firefox extension is composed of two main components: a recommendation engine and a pattern weaver. The recommendation engine communicates between client and recommendation server. The pattern weaver weaves the selected recommendation into a partial mashup model in the modeling canvas. After weaving of a pattern from recommendation, the knowledge base is updated. This updated metadata again is used for future recommendation.

In contrast to their work, the main focus of this thesis is modeling node recommendations in pattern-based mashup environments where a set of transformation patterns is already defined and represented as a node in a node catalog with other data source nodes. This thesis offers a recommendation of frequently co-occurred nodes together because both patterns and data sources are represented as a node in FlexMash. So there is no need to capture modeling actions that occurred in the modeling canvas. In addition, an interactive recommendation feature is offered in this thesis which is not present in the work of Roy Chowdhury at el. The system not only recommends the nodes but also places the node in the modeling canvas upon user selection. It also offers a modeling widget for designing a partial model and users can also toggle the recommendations.

Roy Chowdhury at el. [CTN+13] introduced an approach OMELETTE, a hybrid development assistance system. The system is developed on top of the open source mashup platform Apache Rave. It has two parts: the Automatic Composition Engine (ACE) which addresses users who have no or very little knowledge in mashup development and the Pattern Recommender (PR) which targets users who are already familiar with the composition environment. The ACE helps users to specify their goals using a dialog-based interface in an interactive manner. The dialog shows up in a question-answer manner out of which the system refines user goals. It helps users to choose and configure widgets out of a large collection of potentially incompatible components in an interactive way. The PR helps users by recommending existing composition knowledge stored in a knowledge base (KB). Currently, it supports two composition pattern types: widget co-occurrence and multi-widget patterns. Based on user modeling actions on widgets, the PR reacts during composition. The recommendation engine uses an event listener to capture the modeling

action and its object during each interaction. This information is then used to query the recommended patterns from the KB. The resulting patterns are filtered and ranked based on current composition context and are rendered in the recommendation panel. The user can select recommended patterns from the recommendation panel and upon selection of a pattern, the PR applies the pattern to the current workspace model automatically.

In this thesis, the main goal is to recommend possible nodes that are used frequently by the users for each node that is placed in the modeling canvas. For each placed node, users can see the recommendations by calling the recommendation dialog. The recommendation application recommends the nodes used frequently with the placed nodes based on association rules techniques. The recommendations are ranked based on the confidence level. Upon user selection, the selected node from the recommendation dialog is automatically placed in the modeling canvas and connected to the placed node. The main key difference with their approach is that they recommend composition patterns of a widget of Apache Rave whereas the approach, proposed in this thesis, recommends the frequently co-occurred nodes because patterns are represented as nodes in FlexMash. Another key difference is that a good interactive recommendation feature is provided which recommends nodes at each step of modeling based on user selection. It also automatically completes the whole model using top recommended nodes from the recommendation server.

Roy Chowdhury at el. [RDC11] proposed an approach for efficient and faster retrieval of a ranked list of development recommendations. They model the problem of interactively recommending composition knowledge as pattern matching and the retrieval problem in the context of data mashups and visual modeling tools. They propose a solution for the problem of matching a partial mashup model with a repository of composition patterns. In order to achieve this, they transform the graph-like data structure into an optimized structure which is directly mapped to the recommendations to be provided. They also have an efficient similarity search algorithm for complex pattern matching with the recommended pattern repositories.

In this thesis, the models designed in FlexMash are transformed into a canonical model. This canonical model stores the information about each connection of the model supplied by FlexMash. A data mining process is applied to this canonical model to generate association rules. Each association rule is a connection that occurred frequently in the repository of canonical models. In this approach, a unique id of each node is provided by FlexMash which is used to retrieve the recommended nodes from the repository of association rules.

4 Modeling Recommendations for Pattern-based Mashup Plans

In this chapter, the overall architecture of the master thesis, the different components that were developed in order to extend the FlexMash application with recommendation facilities is described. The architecture of the recommendation application and how to integrate it with the existing FlexMash application are also explained in detail.

Through the development of this thesis, an application for generating recommendations is developed which includes a set of services. This set contains services for processing and storing data that comes from FlexMash and for generating recommendations. It is important to highlight that the application is centralized and all models that are executed by any instance of FlexMash will be captured by the application in order to do the analysis for recommendation generation.

4.1 Overview of the approach

In this section, the overall approach to achieve the goal of the thesis is described. In order to assist users by recommending nodes during modeling, the FlexMash application needed to be extended to make it compatible with the recommendation application. Furthermore, for designing the recommendation application, the underlying canonical model and an appropriate mining algorithm have to be defined.

After developing the recommendation application, it needs to be adjusted with FlexMash to get recommendations during modeling. The work done in this thesis consists of:

- Creation of the training data, i.e., the flow models from FlexMash.
- Creation of the underlying canonical model for analysis.
- Analysis and selection of a suitable analytics algorithm to generate the recommendations.
- Design of the repository for storing the canonical model and the generated recommendations.
- Design of a data transformer that transforms the flow models supplied by FlexMash to the canonical model.
- Creation of the services to interact with the recommendation application.

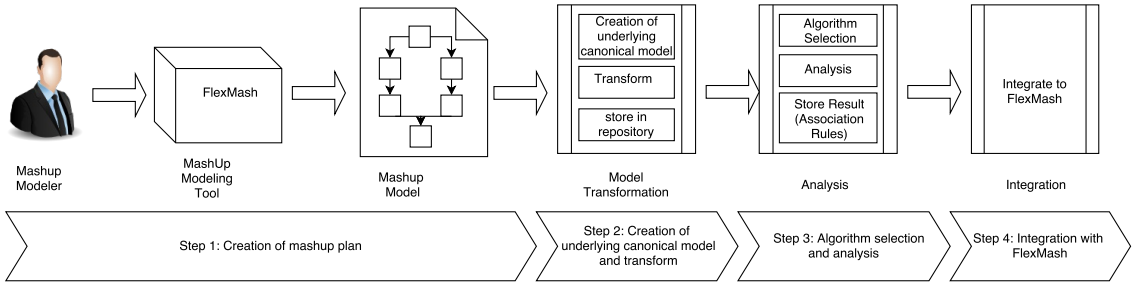

Figure 4.1: Overall Approach of the Thesis

- Integration into FlexMash in order to enable the interactive recommendation feature.

Figure 4.1 shows an overview of the approach. The whole approach is subdivided into four steps: (i) Creation of the mashup plan, (ii) Creation of an underlying canonical model, transformation of the mashup plan and storing, (iii) Algorithm selection, analysis and storage of results, and (iv) integration with FlexMash.

4.2 Step 1: Creation of the mashup plan

The first step, the creation of mashup plans is already done in the existing FlexMash application. Users can model a mashup plan in FlexMash that defines data as well as how it is processed and integrated. After that, the mashup plan is executed. The model is sent to the recommendation application during the execution. Each model that is executed in FlexMash will be captured by the recommendation application for analysis. A model designed in FlexMash is shown in Figure 4.2. In the next step, a canonical model needs to be defined and the model needs to be transformed into the defined canonical model.

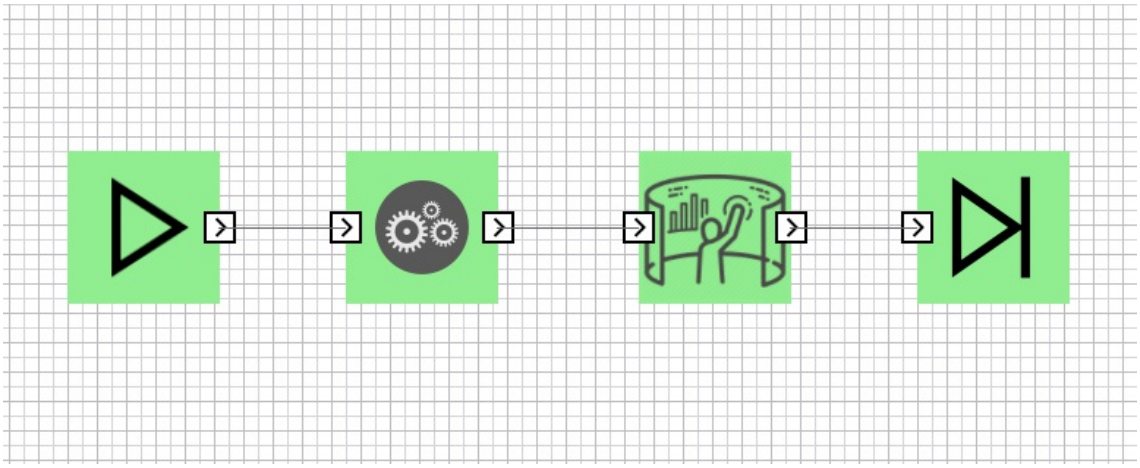

Figure 4.2: A Simple Model Designed in FlexMash

4.3 Step 2: Creation of underlying canonical model and transformation

To generate recommendations, a canonical model is needed which contains only basic and unique information about the models generated by FlexMash. In the model that is shown in Figure 4.2, each node is connected with others by a connection. So each connection has a source node and a target node. As already mentioned, in this thesis, association rules technique is used to generate recommendation. The basic idea behind association rules is to identify frequent patterns as already discussed in Section 2.1. How frequently one node is used with others can be found by storing the connections between nodes used in each model that is sent from FlexMash.

The canonical model used in this work for analysis can be expressed as a tuple $m = \langle id, sourceId, targetId \rangle$ where :

- **id** is the connection id which will be auto generated,
- **sourceId** is the id of the source node which is a unique id of the node in FlexMash.
- **targetId** is the unique id of the target node.

The mashup model sent by FlexMash needs to be transformed into this canonical model. Each tuple in the canonical model can be considered as a transaction for association rules analysis.

4.3.1 Transform

The transformation is done using the JSON format that is sent by the FlexMash application to the canonical model introduced in the previous section. The JSON format is depicted in Listing 4.1.

After executing the designed model, the JSON representation of the model is sent from the FlexMash application through HTTP. The JSON data sent by FlexMash contains the id of the nodes and the connections between nodes. The recommendation application extracts the relevant data from the file in order to transform it into the canonical model. For the transformation, the unique id of the nodes in the model and connections between the nodes are needed. The other data is ignored since it is not needed.

The model depicted in Listing 4.1 can be represented using the following mathematical construct:

$M = \langle N_x, N_y, N_z \rangle$

Where N_x, N_y and N_z are nodes that are contained in the model.

```
{
"Nodes":
[
      {
      "guiId":"4c7c775e-5922-463e-8f80-9719e9b8d5da",
      "serviceId":"849AFDC4-45A1-3700-AB70-2A32E650C9EA",
      "Properties":[],
      "Target":
      [
            "b7e3465d-f12d-490c-adef-d3a3036a22fa",
            "b868a562-f84c-414a-85f3-f359c4449c95"
      ]
      },
      {
      "guiId":"b7e3465d-f12d-490c-adef-d3a3036a22fa",
      "serviceId":"3601AFC0-13B3-2BA7-B428-747F02A8BD89",
      "Properties":[],
      "Target": []
      },
      {
      "guiId":"b868a562-f84c-414a-85f3-f359c4449c95",
      "serviceId":"5AE4C8CA-D491-224F-A78C-4578F69896AF",
      "Properties":[],
      "Target": []
      }
      ],
"Identifier":"6f2e2a9f-2b74-462c-9503-8608db791c4b"
}
```

Listing 4.1: JSON Representation of the Model

Each node of the model can be represented as:

$N_x = \langle g_x, s_x, P_x, T_x \rangle$

Where, g_x is the *guiId*, s_x is the *serviceId*, P_x represents the *Properties* and T_x is the list of *Target* nodes in the model for the node N_x. And

$T_x = \langle g_y, g_z \rangle$,

where, g_y and g_z is the *guiId* of target nodes of the source node N_x.

The other nodes in the model can be represented as follows:

$N_y = \langle g_y, s_y, P_y, T_y \rangle$ and $T_y = \emptyset$,
$N_z = \langle g_z, s_z, P_z, T_z \rangle$ and $T_z = \emptyset$.

To transform the FlexMash model into the canonical model we need only *guiId*, *serviceId* and *Target* of each node, the *Properties* can be discarded as it is not needed for our canonical model defined in Section 4.3.

Let, first consider the node N_x. The s_x is the *serviceId* of the node N_x which is the unique id of this node in FlexMash. The target nodes of the node N_x are g_y and g_z, where g_y and

Algorithm 4.1 Algorithm for Transformation

```
1: Nodes=Parse JSON array of Nodes;
2: for (i = 1; i ≥ Nodes.length; i++) do
3:     sourceID = Nodes[i]["serviceID"];
4:     Targets = Nodes[i]["target"];
5:     for (k = 1; k ≥ Targets.length; k++) do
6:         initialize connection object;
7:         set sourceID as connection source;
8:         map Targets[k]["guiID"] to Nodes[i]["serviceID"];
9:         set connection target;
10:    end for
11: end for
```

g_z are the *guiId* of the node N_y and N_z, respectively. So, there are two connections, one between node N_x and N_y, and the other between N_x and N_z.

The canonical model contains tuples of connections in the model and is represented as:

$m = \langle id, sourceID, targetID \rangle$

So, the above mentioned mathematical representation of the model can be transformed into the canonical model as follows:

$m_1 = \langle id, s_x, s_y \rangle$,
$m_2 = \langle id, s_x, s_z \rangle$

Where, *id* is the unique identifier for the connection m_1, s_x is the *serviceId* of the node N_x and s_y is the *serviceId* of the node N_y. For connection m_2, s_x is the *serviceId* of node N_x and s_z is the *serviceId* of the node N_z.

The nodes N_y and N_z have no target nodes, so these are the end nodes. There are no outgoing connections from these nodes, so we do not have to transform these nodes.

Algorithm 4.1 outlines the approach for model transformation. Here, first the JSON array is parsed and added into *Nodes*. Then, by iterating over all items of the *Nodes* array, the source and the target of a connection are extracted.

id	sourceId	targetId
1	849AFDC4-45A1-3700-AB70-2A32E650C9EA	3601AFC0-13B3-2BA7-B428-747F02A8BD89
2	849AFDC4-45A1-3700-AB70-2A32E650C9EA	5AE4C8CA-D491-224F-A78C-4578F69896AF

Table 4.1: Database Table Structure with Example Records to Store Canonical Models

4.3.2 Store

The above mentioned canonical model needs to be stored in a relational database. The design of the relational database used in this work is discussed in Section 5.3 in detail. All models coming from FlexMash are transformed into the canonical model and are stored in a table of a relational database. The analysis is done based on this table. Each time a model is executed in FlexMash, it is captured by the recommendation application, transformed into the canonical model and stored in the database. The analysis is run on all data stored in the canonical model table and updates the result of the analysis. Table 4.1 shows the structure of the database table with some example canonical model records to store the canonical models.

Next, a suitable algorithm needs to be selected for association rules analysis.

4.4 Step 3: Algorithm selection for analysis

In this section, the selection of a suitable algorithm for association rules analysis is discussed. The algorithms discussed in Section 2.1.4 are compared based on several parameters. Based on this performance comparison, a suitable algorithm is selected for analysis.

There are two disadvantages of the Apriori algorithm that has been discussed in Section 2.1.4.1. One is the complex process of candidate generation that uses most of the time, space, and memory. Another disadvantage are the multiple scans of the database. FP-Growth solves these two disadvantages of Apriori. FP-Growth generates frequent itemsets with only two passes over the database and without any candidate generation process. By doing so, it solves one disadvantage of the apriori algorithm. In FP-Growth, the frequent patterns generation consists of two sub-processes: construction of FP-Tree, and generation of frequent patterns from the FP-Tree.

The FP-Growth algorithm is efficient because of three reasons. First, the FP-Tree constructed by the algorithm is the compressed representation of the actual database because only frequent items are used to generate the tree, other irrelevant data are discarded. Also, the overlapping parts appear only once with different support count by ordering the items according to their supports. Secondly, this algorithm scans the database only twice. The

generated FP-Tree contains patterns with the specified suffix from which frequent patterns can be easily generated. Also, the cost of computation decreases. Third, The FP-Tree uses a divide and conquer method which reduces the size of the subsequent conditional FP-Tree.

In data mining, one critical aspect is the number of disk accesses because the I/O operation takes more time than the memory operation. So, minimizing the disk access can be a reason for faster execution. The Apriori algorithm scans the whole database k-times for finding k-frequent itemset. The number of times the algorithm reads the database is dependent on the size of the longest itemset. In contrast, the FP-Tree algorithm scans the database only twice.

However, FP-Growth is difficult to use in an interactive mining system. In the interactive mining process, users may change the threshold of the support according to the rules. In case of FP-Growth, if the user changes the threshold of the support, the whole mining process will be repeated. FP-Growth is also not suitable for incremental mining. Since

Parameter	**Apriori**	**FP-Growth**
Storage structure	Array based	Tree based
Search type	Breadth First Search	Divide and conquer
Technique	Join and prune	Constructs conditional frequency pattern tree which satisfy minimum Support
Number of Database scans	K+1 scans	2 scans
Memory utilization	Large memory (candidate generation)	Less memory (No candidate generation)
Database	Sparse/dense	Large and medium data sets
Run time	More time	Less time
Accuracy	Less	More Accurate
Data Structure and mining methods	Easy to use	More complicated
Generating Frequent itemsets	Fast	Comparatively Slow

Table 4.2: Comparison between Apriori and FP-Growth

databases keep growing, new datasets may be added into the database. The whole process may repeat because of those additions, in case the FP-Tree algorithm is used.

Table 4.2 summarizes the difference between Apriori and FP-Growth based on literature review [SS13].

4.4.1 Experimental Evaluation

I tested the performance of the Apriori and FP-Growth algorithms in Weka[1]. Weka is an open source software that consists of a collection of open source machine learning and data mining algorithms. It also includes preprocessing of data, classification, clustering, and association rule extraction. The performance evaluation of both algorithms is done based on execution time. I used supermarket data included in Weka as sample data. The execution time is measured for the different number of records, support and confidence level. For efficiency evaluation, I used GUI based WEKA application.

4.4.1.1 Result Discussion

In this section, the result of the experimental evaluation done for both Apriori and FP-Growth is discussed.

Table 4.3 shows the result of the execution time analysis of Apriori and FP-Growth for a different number of instances. It can be seen that the execution time of both algorithms decreases when the number of instances decreases. Apriori takes 48 seconds when the number of instances is 3629. For the same number of instances, FP-Growth needs only 4 seconds for constructing association rules.

In Figure 4.3, the performance of Apriori and FP-Growth is compared based on execution time. Each algorithm is executed on different data sets with sizes of 3629, 1688 and 942. Here, the number of instances in the dataset is represented by the x-axis and execution time is represented by the y-axis. The figure shows that for any number of instances the execution time of the FP-Growth algorithm is less than the Apriori. So, the FP-Growth

Number of instances	Execution Time(in seconds)	
	Apriori	FP-Growth
3629	48	4
1688	26	3
942	8	2

Table 4.3: Execution Time for Different Number Of Instances

[1] https://www.cs.waikato.ac.nz/ml/weka/

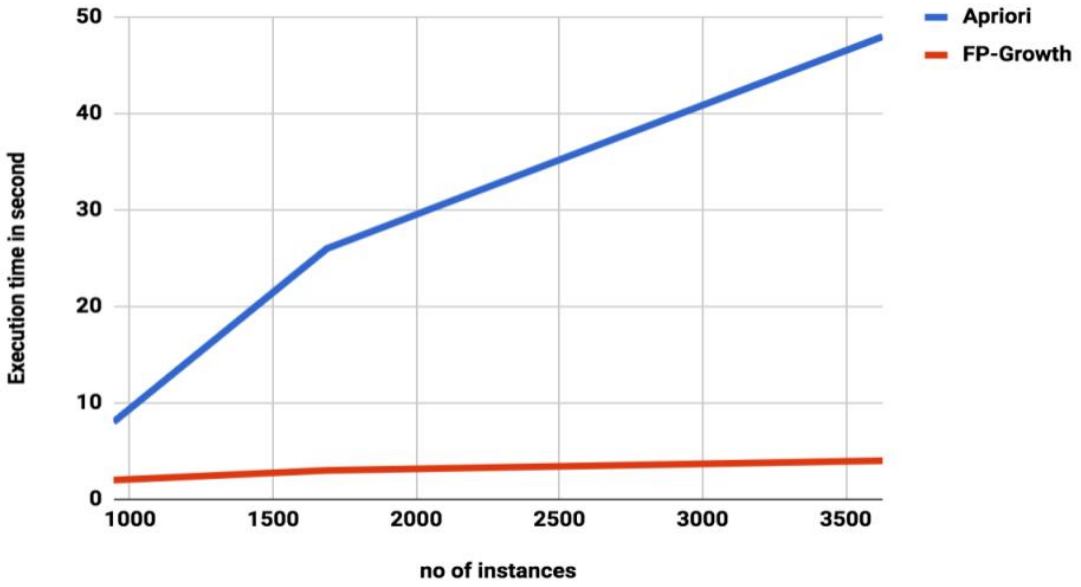

Figure 4.3: Comparison of Execution Time Based on Number of Instances

executes faster than Apriori.

Table 4.4 shows the execution time taken by both Apriori and FP-Growth for different confidence levels. The execution time of both algorithms is high when the confidence level

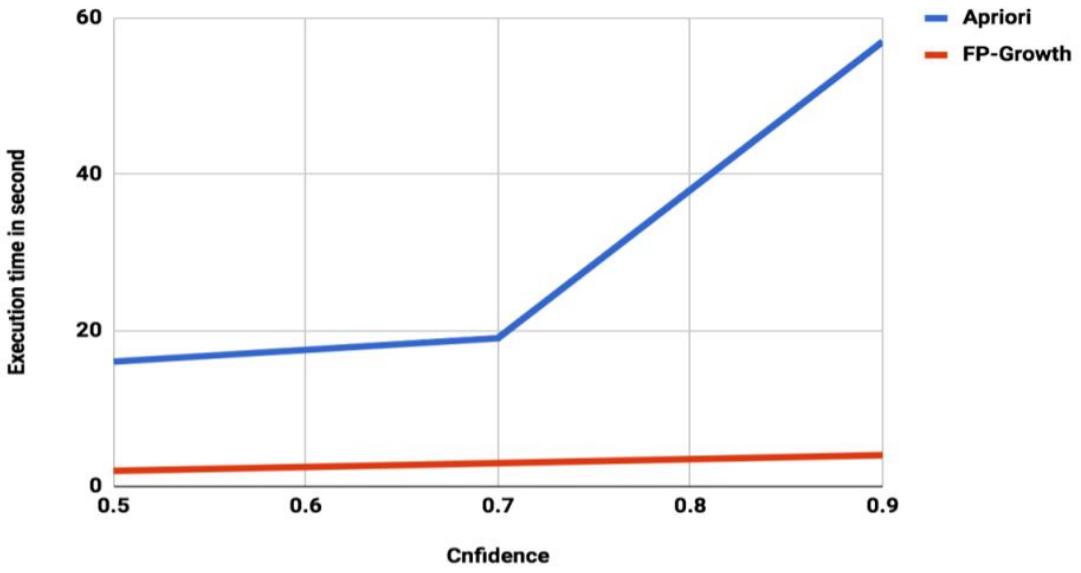

Figure 4.4: Comparison of Execution Time Based on Different Confidence Levels

Confidence	Execution Time(in seconds)	
	Apriori	FP-Growth
0.5	16	2
0.7	19	3
0.9	57	4

Table 4.4: Execution Time for Different Confidence Level

is high. When the confidence level is 0.9, Apriori takes 57 seconds and FP-Growth takes 4 seconds to generate association rules.

Figure 4.4 shows the graphical representation of the relationship between time and confidence. The confidence is represented by the x-axis and execution time is represented by the y-axis. It shows that the execution time of FP-Growth is less than the Apriori for any confidence.

From the above discussion and result analysis, it is proven that FP-Growth performs faster than Apriori. However, for providing a flexible solution, in this thesis, both algorithms are used for analysis and results of this analysis is stored separately. The users can select which algorithm to use according to their use case.

4.4.2 Analysis

In this section, the Apriori algorithm is used to generate association rules from the canonical models stored in the database. An example canonical model instance is shown in Table 4.5. This table represents a model designed in FlexMash.

As already discussed in Section 2.1.4.1, the Apriori algorithm has two steps: (i) Frequent itemsets generation, and (ii) Association rules generation using the frequent itemset.

id	**sourceId**	**targetId**
1	849AFDC4-45A1-3700-AB70-2A32E650C9EA	5AE4C8CA-D491-224F-A78C-4578F69896AF
2	5AE4C8CA-D491-224F-A78C-4578F69896AF	B4652E72-1F34-69D5-BC35-6AEDA218B0C5
3	B4652E72-1F34-69D5-BC35-6AEDA218B0C5	A203915A-39C0-7CF1-92F6-81B6A42C4BF4
4	A203915A-39C0-7CF1-92F6-81B6A42C4BF4	DDCB56F7-D526-E011-936C-177094DDA67B

Table 4.5: An Example Table Storing Canonical Models

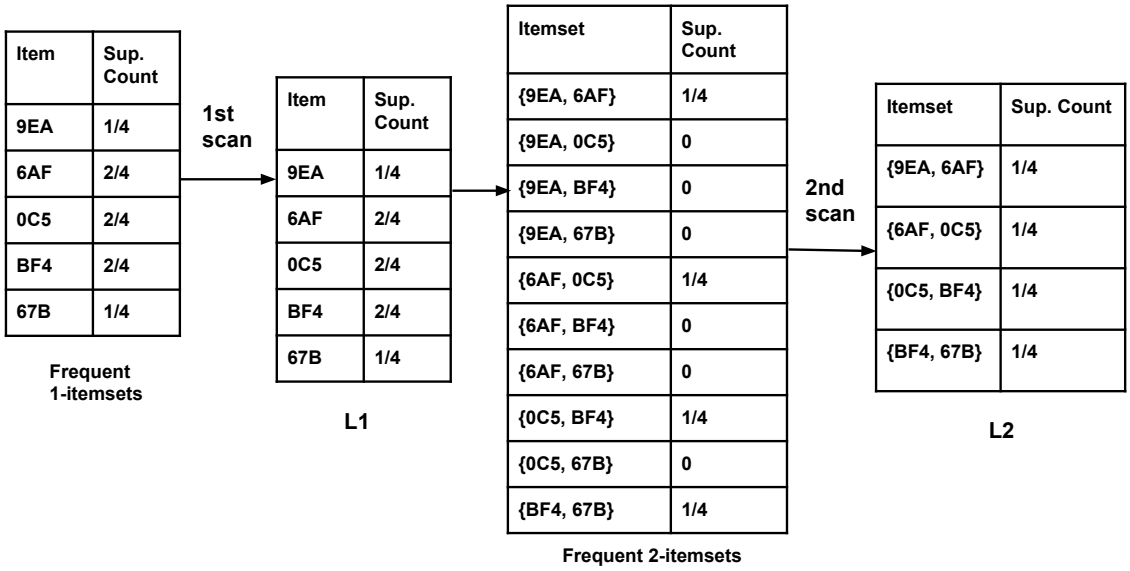

Figure 4.5: Frequent Itemset Generation

4.4.2.0.1 Frequent Itemsets Generation The data represented in Table 4.5 is considered as transactional data to generate frequent itemsets using Apriori. Here, four connections are represented. I assume that, the minimum support is 25% (i.e. 1/4) and the confidence is 40%.

A high-level illustration of the frequent itemset generation using Apriori is depicted in Figure 4.5. Here, the last three digits of nodeID are used to represent the nodes. At first, each node is considered as a frequent 1-itemset. Then, the support is calculated for each item in the transactions. The support is calculated using the following formula :

$$\text{support(9EA)} = \frac{transactions\_containing\_node\_9EA}{total\_number\_of\_transactions}$$

In L1, the overall support count is above the minimum support (25% i.e. 1/4), so all items can be considered for the next iteration. In the next iteration, to generate frequent 2-itemsets, the Apriori algorithm uses L1 Join L1. Then, the support count for each itemset is calculated which is shown in the third table of Figure 4.5. After that, the set of frequent 2-itemsets, L2, is determined by discarding the items which have a support count below minimum support. The itemsets shown in table L2 of Figure 4.5 are the frequent itemsets which can be used to generate association rules.

4.4.2.0.2 Generating Association Rules To generate association rules from frequent itemsets, for each frequent itemset, the confidence is calculated. The formula for calculating the confidence is given below:

For example, if $A \rightarrow B$ is a frequent itemset then

Association Rules	Support(A,B)	Support(A)	Confidence
$9EA \rightarrow 6AF$	1/4	1/4	1
$6AF \rightarrow 0C5$	1/4	2/4	1/2
$0C5 \rightarrow BF4$	1/4	2/4	1/2
$BF4 \rightarrow 67B$	1/4	1/4	1

Table 4.6: Generated Association Rules out of Frequent itemsets

$$\text{confidence}(A \rightarrow B) = \frac{number\_of\_transactions\_containing\_both\_A\_and\_B}{number\_of\_transaction\_containing\_A}$$

The generated association rules are shown in Table 4.6. As all rules have confidence level above minimum confidence, all rules can be considered.

4.5 Step 4: Integration with FlexMash

One of the main goals of this thesis is to assist users in the FlexMash application by providing modeling recommendations. In order to achieve this goal, the recommendation service needs to be integrated into the existing FlexMash application. Figure 4.6 shows how the integration is done between the Recommendation service and the FlexMash application.

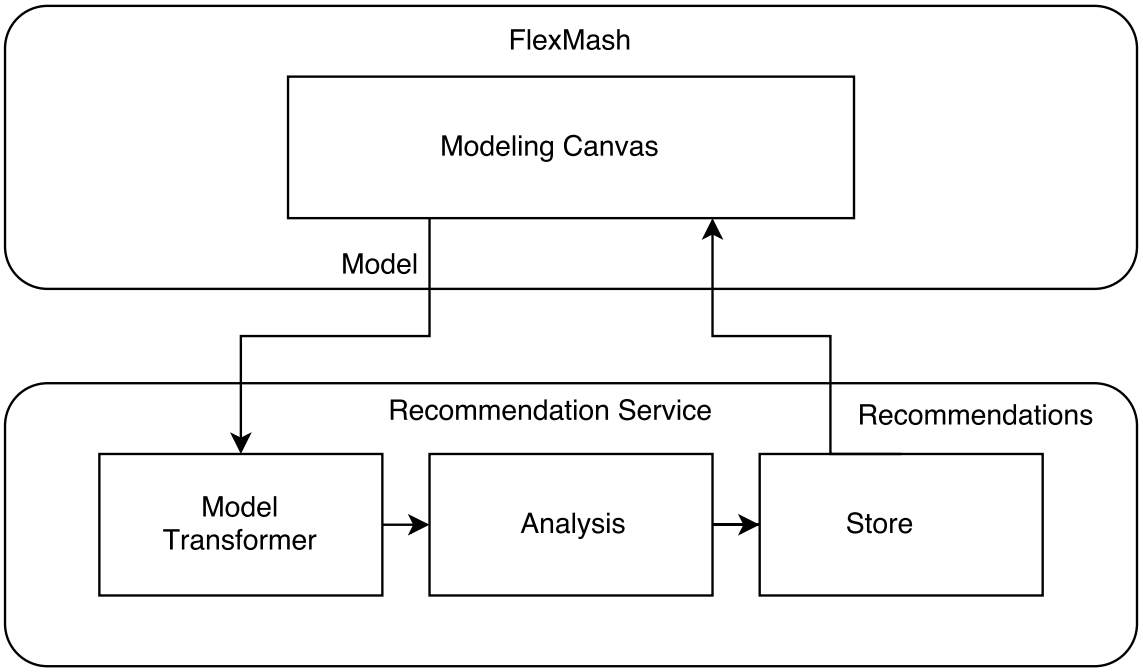

Figure 4.6: Integration into FlexMash

In the first step, the model designed in FlexMash is sent to the Recommendation Service during the execution of the model. The Recommendation Service has a component which transforms the model into a canonical model. This canonical model is then stored in the repository. The analysis process will use all the stored canonical models for generating association rules. Then, the result of the analysis are stored in the repository. This model transformation and analysis is done during each execution of the model in FlexMash. After that, FlexMash can query for recommendations for a specific node. The Recommendation Service sends all available association rules for this specific node to FlexMash.

A set of User Interfaces needs to be implemented inside FlexMash in order to provide an interactive recommendation feature during modeling.

5 Prototypical Implementation

The implementation details of this thesis work are discussed in this chapter which includes the approach discussed in the previous chapter. To generate recommendations by analyzing existing models, a web service named *Modeling Recommendation Service* is implemented. The chapter describes how to transform the models into a canonical model, how the analysis process is done and how the service is integrated into FlexMash. This chapter also outlines the required user interface implementation in the FlexMash frontend in order to achieve the interactive recommendation feature. FlexMash is an open source application so the work done in this thesis is also open source.

5.1 Adaptation of Technologies

The section states the technologies needed for the implementation of the recommendation service.

- **Node.js :** The Recommendation service implemented for this thesis is developed in Node.js which is a JavaScript runtime built on Chrome's V8 JavaScript engine.It is lightweight and efficient because it uses an event-driven, non-blocking I/O model. Node.js allows the creation of Web servers and networking tools using JavaScript and a collection of "modules" that handle various core functionalities. Node.js applications can run on Linux, MacOS, Microsoft Windows and Unix servers.

- **JAVA :** The data mining process in this work is implemented in JAVA which is a concurrent, class-based object-oriented programing language. JAVA is used to handle the open source library Weka which consists of a collection of machine learning algorithms for data mining tasks.

- **MySQL:** The repository used in this work for storing canonical models and analysis results is implemented in MySQL. It is one of the most popular relational database management systems.

- **TypeScript:** The FlexMash application is implemented in TypeScript which is an open source programing language developed and maintained by Microsoft. TypeScript is a superset of JavaScript.

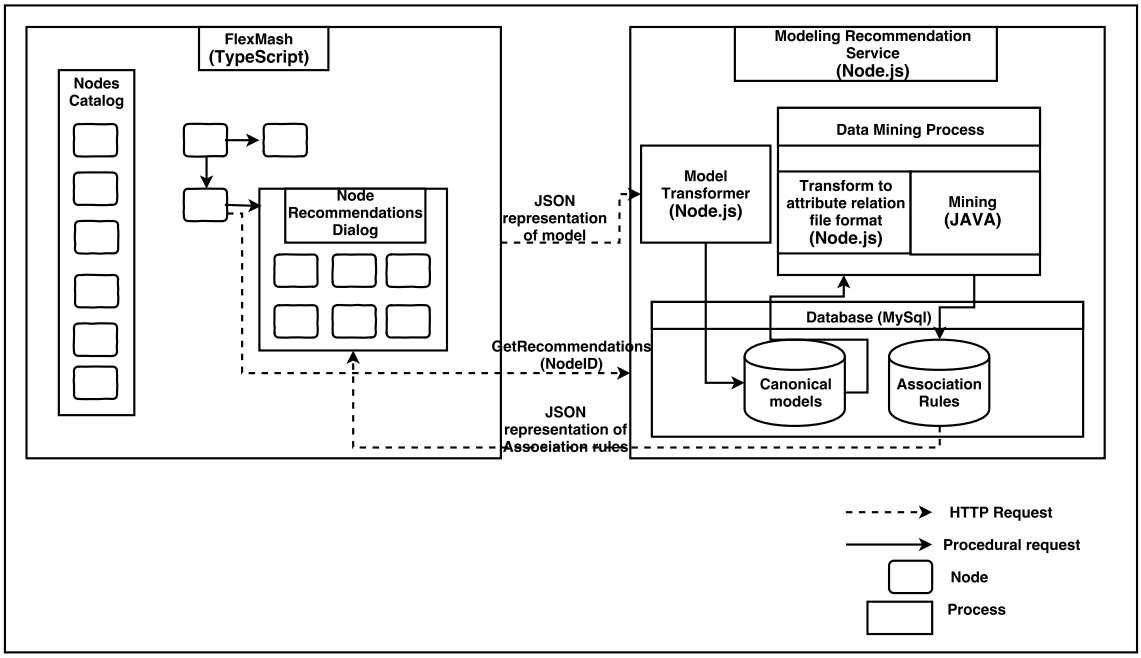

Figure 5.1: Architecture of the *Modeling Recommendation Service* Specific to the Implementation Scenario

- **HTML/CSS/jQuery :** This thesis uses HTML and CSS to implement additional user interfaces inside FlexMash for the interactive recommendation feature. Along with these, this thesis also uses libraries like Bootstrap, jQuery, etc.

5.2 Architecture

This section gives an insight into implementation details. An overview of the architecture specific to the implementation for the work done in this thesis is depicted in Figure 5.1. The *Modeling Recommendation Service* is hosted as a web service. It can be easily accessed, deployed, and scaled.

The Architecture of the *Modeling Recommendation Service* consists of two main processes. First, the Model Transformer, that transforms the model sent by the FlexMash application into a canonical model and stores it into the database. Second, the Data Mining Process which includes two subprocesses: (i) Transform canonical models into the Attribute-Relation File Format (ARFF) [wik], and (ii) the Mining process that mines the supplied ARFF dataset using the data mining algorithms Apriori and FP-Growth. The result of the data mining process (association rules) is stored in the database.

The database consists of two core models : (i) Canonical models, (ii) Association Rules. The database is described in more detail in Section 5.3.

During the execution of a model in FlexMash, the JSON representation of the model is sent to the *Modeling Recommendation Service* using an HTTP request. The service receives the request and extracts the data. Then, the transformer transforms the JSON data into a canonical model object and stores this object in the database. After that, the data mining process starts mining. The process first retrieves all the canonical models stored in the database and writes them to an ARFF file. The ARFF file is then used as the input dataset for running the mining algorithms. Finally, the generated association rules are stored in the database.

In FlexMash, when users request the recommendations for a specific node, the request is sent as an HTTP request to the *Modeling Recommendation Service* with the unique id of this node. The service retrieves all the association rules from the database associated with this node id. The service builds a JSON representation of resultant association rules and replies back to FlexMash using HTTP. FlexMash gets the reply, maps the JSON string, extracts the recommended nodes and shows them in a dialog.

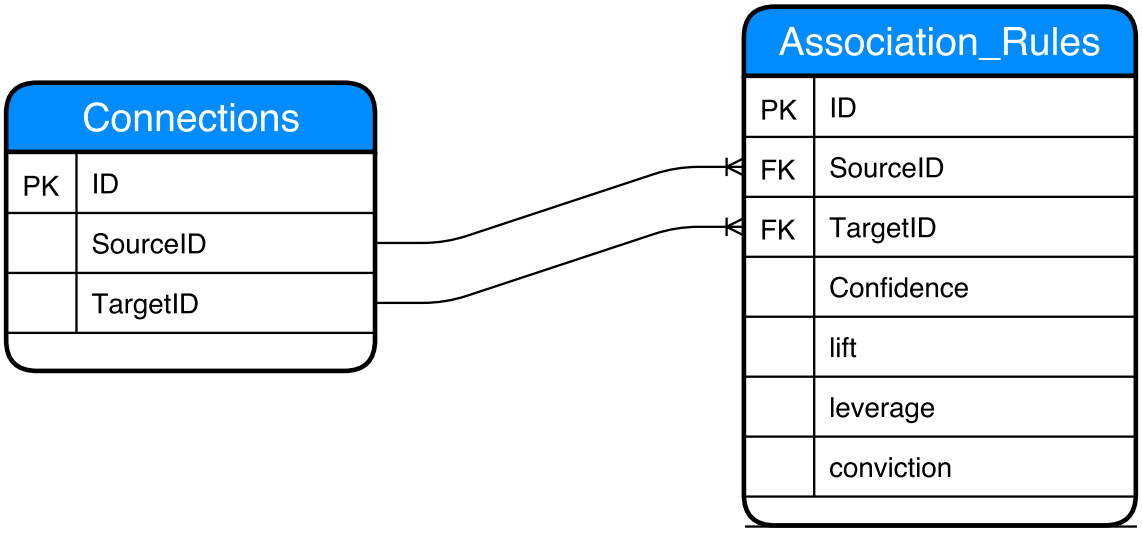

Figure 5.2: Entity Relationship Diagram of the *Modeling Recommendation Service*

5.3 Database Design

In this section, the design of the database used in this work is discussed in detail.

A relational database is used in this thesis to store canonical models and analysis results. The relational database is important because deletion of a node in FlexMash has impact on the recommendation generation. More precisely, if one node is deleted from the FlexMash database, the corresponding information associated with this node must be deleted from the recommendation database as well. Otherwise, the user will get recommendations for nodes which does not exist in the FlexMash database. The FlexMash application will fail to map these nodes. In order to enable consistency of data, a relational database is important in this work.

In this thesis, MySQL is used. Figure 5.2 shows the entity relationship diagram of the database used in this thesis. The database consists of two tables: (i) **Connections** which is used to store canonical models, and (ii) **Association_Rules** which is used to store the generated association rules by the data mining process.

As already mentioned, in this thesis, both Apriori and FP-Growth algorithms are used for data mining. The association rules generated by these algorithms are stored separately. So, in this work, two tables are used to store generated association rules by the Apriori and FP-Growth algorithms. For better understanding, one table "Association_Rules" is shown in the Figure 5.2. This same structure is used to store the association rules generated by two algorithms. Depending on the user selection, the association rules can be retrieved from the corresponding tables.

In the next sections, the tables shown in Figure 5.2 are explained with the field description.

5.3.1 Tables

In this section, the tables which are used to store information in this work are described.

5.3.1.1 Connections

This table is used to store the canonical model, introduced in Section 4.3. Each row represents a connection between two nodes. It has three fields :

- **id** is the primary key of the table and will be auto-generated,
- **sourceId** is a VARCHAR field, the unique id of a node which represents the source node of a connection.
- **targetId** is a VARCHAR field, which represents the target node id of a connection.

5.3.1.2 Association_Rules

This table is used to store the generated association rules by the data mining process. Each record in this table represents an association rule. It has six fields:

- **ID** is the primary key of the table and is auto generated.
- **sourceId** represents the left-hand-side (LHS) of the association rules which is source node id.
- **targetId** represents the right-hand-side (RHS) of the association rules which is target node id.
- **Confidence** defines the confidence of a rule.
- **Lift** indicates the lift of a rule. Given a rule $X \rightarrow Y$, lift is the ratio of the probability that X and Y occur together to the multiple of the two individual probabilities for X and Y, i.e.,
$$lift = \frac{Pr(X,Y)}{Pr(X).Pr(Y)}. \tag{5.1}$$
- **Leverage** indicates the leverage of the rule. The leverage is calculated by subtracting the independent probabilities of each X and Y(from the above example) from the probability of co-occurrences of X and Y , i.e.,
$$leverage = Pr(X,Y) - Pr(X).Pr(Y). \tag{5.2}$$
- **Conviction** is similar to lift, but it measures the ratio of the expected frequency that X occurs without Y. So, conviction is measured as:
$$conviction = \frac{Pr(X).Pr(notY)}{Pr(X,Y)}. \tag{5.3}$$

5.4 The Modeling Recommendation Service

In this section, the web service developed for generating modeling recommendations in this thesis is described in detail.

In order to implement the *Modeling Recommendation Service*, a RESTful API is developed using Node.js. The RESTful application usees HTTP requests to perform CRUD[1] operations. CRUD consists of four operations: (i) C:create, (ii) R:read, (iii) U:update, and (iv) D:delete. Create or update is done using the HTTP POST method, and read is done by the HTTP GET method.

The *Modeling Recommendation Service* offers a number of both POST and GET methods. The available methods offered by the service implemented in this thesis is described in Table 5.1.

The aim of having the *Modeling Recommendation Service* as RESTful application is to provide the recommendation service as an independent application. This RESTful application can be deployed in a server and can be accessed without any authentications. The FlexMash application can be easily integrated by calling the methods provided by the *Modeling Recommendation Service*.

Method Signature	**Method Type**	**Parameter/Data**	**Functionality**
addConnections	POST	JSON representation of models	Converts the JSON representation of models into the canonical model, stores the canonical model, and runs the data mining process
getAssociationRule	GET	sourceID	Returns all the association rules matched with the sourceID.
getCompleteModel	GET	sourceID	Returns a complete model which starts with sourceID.

Table 5.1: Description of Methods Provided by the *Modeling Recommendation Service*

[1] https://en.wikipedia.org/wiki/Crud

5.4.1 Model Transformer

In this section, the implementation details of the model transformer implemented in this thesis is discussed.

In the FlexMash frontend, when a model is executed, it generates a JSON representation of the model. This JSON representation of the model is sent to the *Modeling Recommendation Service* using a HTTP request by calling the method addConnections. *addConnections* is a POST method and the JSON data is included in the request body. The JSON representation of the model has the information about the nodes contained in the model. An example of the JSON representation of the model can be found in Listing 4.1

Listing 4.1, contains an array of nodes and an identifier of the model. Each node in the array of nodes has the following informations:

- **guiId** is the instance id of the node in the modeling canvas in FlexMash.
- **serviceId** is the unique identifier of the node.
- **Target** is a string array which contains the guiId of the nodes that are connected as a target with the node.
- **Properties** is an array representing properties of the node.

For the model transformation, the guiId, serviceId and Target are needed, other information can be discarded because the canonical model does not need the properties of the node and the identifier of the model.

As already mentioned in Section 4.3, the canonical model represents each connection in the model and is a tuple of, $m = \langle id, sourceId, targetId \rangle$. So, the list of connections occurred in the model has to be extracted from the JSON data.

The model transformer implemented in this work extracts a list of connections from the JSON data. An object-oriented approach is used to build a connection object. The connection object has three properties: id, sourceId, targetId.

In the nodes array, each node can be considered as a source of a connection and target can be extracted from the Target array. The serviceId of the node is the sourceId of the connection and each item of the Target array is the target of the connection. To make it more clear, for example, if there are two items in the Target array, then there will be two connections. The source of these connections is the same. In the JSON data, the Target array contains the guiId of nodes. The guiId in the Target array has to be mapped to the guiId of each node of the Nodes array. Then, the corresponding serviceId can be retrieved from the list of nodes in the JSON model. The node that does not have any target is the end node and the node which is not a target of any node is the start node. The transformation process is written in Node.js and shown in Listing 5.1.

id	sourceId	targetId
1	849AFDC4-45A1-3700-AB70-2A32E650C9EA	3601AFC0-13B3-2BA7-B428-747F02A8BD89
2	849AFDC4-45A1-3700-AB70-2A32E650C9EA	5AE4C8CA-D491-224F-A78C-4578F69896AF
3	5AE4C8CA-D491-224F-A78C-4578F69896AF	DDCB56F7-D526-E011-936C-177094DDA67B

Table 5.2: Table Storing Canonical Models

```
mapJSONStringtoObject : function (jsonString,callback)
{
      list=[];
      var obj=JSON.parse(jsonString.body["flow"]);
      for (var i=0; i<obj.nodes.length; i++)
      {
            let conn=new Connection();
            var sourceID = obj.nodes[i]["serviceId"];
            var keysArray = obj.nodes[i]["target"];
            for (var j = 0; j < keysArray.length; j++)
            {
                  var value1=(keysArray[j]);
                  conn.sourceID=sourceID;
                  var rta = obj.nodes.find(
                              (it) => {
                                    return it["guiId"] === value1;
                              }
                        );
                  conn.targetID=rta["serviceId"];
                  list.push(conn);
            }
      }
      callback(list);
},
```

Listing 5.1: Model Transformer Code Snippet

After mapping of JSON data into a list of connection objects, it is stored in the database. The table storing connections is shown in Table 5.2

5.4.2 Data Mining Process

In this section, the data mining process implemented in this thesis for generating association rules is described. This is the main part of the implementation. The canonical models implemented in the previous section is used as data source for the data mining process. Both the Apriori and FP-Growth algorithms have been used for the data mining process, as

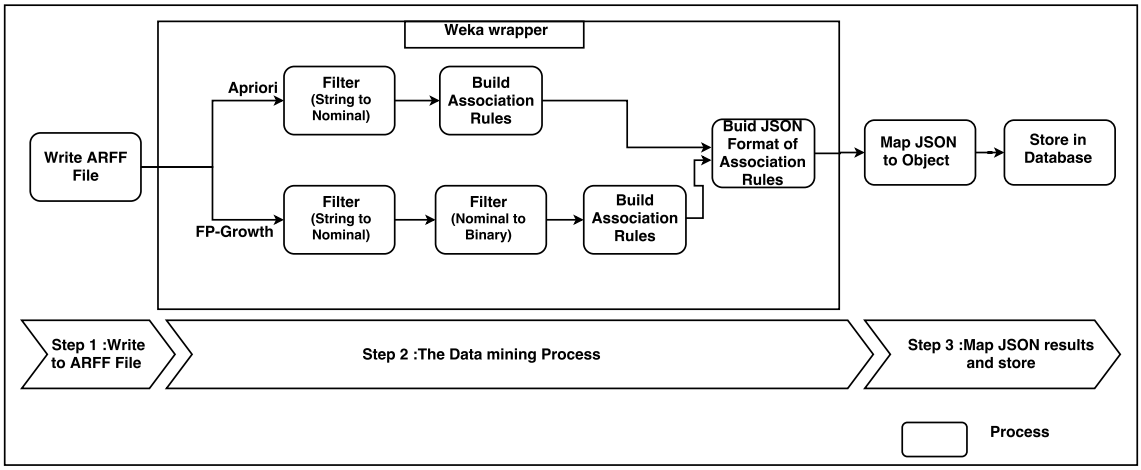

Figure 5.3: Processing Steps of the Data Mining Process

already mentioned. The users can select one of these two algorithms based on their use case.

For implementing the data mining algorithms, the open source library Weka[2] is used. Weka contains a collection of machine learning algorithms that are used for data mining tasks. Apriori and FP-Growth are used for the data mining process in this thesis.

Weka is a JAVA library, so a wrapper class is implemented to use the functionalities of Weka. This wrapper class takes a data source and the algorithm to use as input, and gives JSON output of generated association rules. The structure of the processes implemented for the data mining process is shown in Figure 5.3. Weka uses an ARFF (Attribute-Relation File Format) [wik] file as a data source. An ARFF file is supplied to the wrapper class as input, the wrapper class processes the input, runs the data mining algorithms, and returns a JSON format of generated association rules as output. This output is then mapped onto an object using an object-oriented approach and inserted into the database.

The whole data mining process can be subdivided into three steps: (i) Step 1: Writing to ARFF file, (ii) Step 2: Running the data mining process (iii) Step 3: Mapping JSON results and storage.

In the next sections, these processing steps are described in detail.

5.4.2.1 Step 1 : Write ARFF File

An ARFF file is a text file based on the ASCII standard which describes a list of instances containing a set of attributes [wik]. ARFF files contain two parts: Header information and Data information. The ARFF Header section of the file is composed of relation declarations

[2]https://www.cs.waikato.ac.nz/ml/weka/

and attribute declarations. The name of the relation is defined at the first line of the ARFF file. The format is:

@relation <relation-name>

where <relation-name> is a string. If the relation name includes spaces, then it must be defined using quotes. Attribute declarations are done in an ordered sequence of **@attribute** statements. In the data set, each attribute has its own **@attribute** statement that uniquely defines the name and data type of that attribute. The order of the attribute declarations indicates the column position in the data section of the file. The **@attribute** statement is defined as:

@attribute <attribute-name> <datatype>

where <attribute-name> must start with a string. If the attribute name includes spaces then it must be defined using quotes. Weka supports four data types: numeric, string, nominal and date.

The **@data** declaration is a single line which denotes the start of the data segment in the file. Each instance in the file is defined in a single line and a carriage returns at the end of each line which denotes the end of instances. For each instance, the attribute values are delimited by commas or tabs. The attribute values must be defined in the order in which they were declared in the header section. The missing values are represented by a question mark. An example of an ARFF file is shown in Listing 5.2.

```
@relation LCCvsLCSH
@attribute LCC string
@attribute LCSH string

@data
      AG5, 'Encyclopedias and dictionaries.;Twentieth century.'
      AS262, 'Science -- Soviet Union -- History.'
      AE5, 'Encyclopedias and dictionaries.'
      AS281, 'Astronomy, Assyro-Babylonian.;Moon -- Phases.'
      AS281, 'Astronomy, Assyro-Babylonian.;Moon -- Tables.'
```

Listing 5.2: Example of Attribute-Relation File Format [wik]

In this work, all data in the table storing the canonical models are used to write the ARFF file. The node package "arff-utils" [3] is used to create ARFF files. The relation name is "connections" and the attribute names are "sourceID" and "targetID".

At first, all data from the table storing canonical models are retrieved. An ARFF file is created using the relation name and then attribute names are added. After that, the data has been added to the file by traversing the data fetched from the database. The process

[3] https://www.npmjs.com/package/arff-utils

implemented for writing ARFF files is shown in Listing 5.3.

```
var ArffUtils = require('arff-utils');
database.getAllConnections(connection,function(err,list){
            if(err) {
            console.log("Error in getting connections");
            } else
            {
                  var stream = ("myarff.arff");
                  var arffutils = new ArffUtils.ArffWriter("connection",
                      ArffUtils.MODE_OBJECT);
                  arffutils.addStringAttribute("sourceID");
                  arffutils.addStringAttribute("targetID");
                  var fite ='' ;
                  for(var i = 0; i < list.length; i++)
                  {
                        arffutils.addData({
                        sourceID: list[i].sourceID,
                        targetID: list[i].targetID
                  });
            }
            arffutils.writeToFile(stream,function (err){
            if (err)
                  return console.log(err);
            });
});
```

Listing 5.3: Code Snippet for Writting ARFF File

The output ARFF file for the data shown in Table 5.2 is shown in Listing 5.4.

```
@relation connections
@attribute sourceID string
@attribute targetID string
@data
849AFDC4-45A1-3700-AB70-2A32E650C9EA,3601AFC0-13B3-2BA7-B428-747F02A8BD89
849AFDC4-45A1-3700-AB70-2A32E650C9EA,5AE4C8CA-D491-224F-A78C-4578F69896AF
5AE4C8CA-D491-224F-A78C-4578F69896AF,DDCB56F7-D526-E011-936C-177094DDA67B
```

Listing 5.4: Example of Attribute-Relation File Format of the Canonical Models

5.4.2.2 Step 2 : The Data Mining Process

In this section, the details about the wrapper class implemented for the data mining process is discussed. The whole process is implemented in JAVA with the help of the Weka library.

In Figure 5.3, it can be seen that the process is done in three steps : (i) Filter, (ii) Building Association, and (iii) Building JSON representation of generated association rules.

5.4.2.2.1 Filter The data mining algorithms implemented in Weka support only specific types of attributes. In order to do mining using the algorithms provided by Weka, the attributes of the input data set should be converted to the supported types. In this thesis, the Apriori and FP-Growth algorithms are used from Weka.

The Apriori implementation of Weka supports nominal, binary and unary attributes. Weka also provides Filter to convert attributes of the data set. The ARFF file shown in the table contains only string attributes. Weka provides a filter "StringToNominal" which converts string attributes to nominal attributes. This filter is used in this work to convert the string attributes of the input ARFF file to the nominal attributes so the Apriori algorithm implemented by Weka can handle the data. This is done as shown in Listing 5.5

```
StringToNominal stringToNominal = new StringToNominal(); // new instance of filter
stringToNominal.setAttributeRange("first-last"); // range of the attributes
stringToNominal.setInputFormat(data);
Instances dataNominal = Filter.useFilter(data, stringToNominal);
```

Listing 5.5: Code Snippet for Applying StringToNominal Filter

The FP-Growth implementation of Weka supports binary and unary attributes. So the string attributes shown in Listing 5.4 must be converted to binary or unary attributes. However, Weka does not provide any filter to convert string attributes to binary or unary attributes. Weka provides a filter "NominalToBinary" which converts nominal attributes to binary attributes. In this work, first the "StringToNominal" filter is applied then the "NominalToBinary" filter is applied. This is done as shown in Listing 5.6

```
StringToNominal stringToNominal = new StringToNominal(); // new instance of filter
stringToNominal.setAttributeRange("first-last");
stringToNominal.setInputFormat(data);
Instances dataNominal = Filter.useFilter(data, stringToNominal);

NominalToBinary nominalToBinary = new NominalToBinary(); // new instance of filter
nominalToBinary.setInputFormat(dataNominal);
Instances dataBinary = Filter.useFilter(dataNominal, nominalToBinary);
```

Listing 5.6: Code Snippet for Applying NominalToBinary Filter

After applying the filter, the data is ready for mining. Now the Apriori and FP-Growth algorithms can be applied to the filtered data sets. In the next section, the process of applying the Apriori and FP-Growth algorithms on the filtered data set for generating association rules is discussed.

5.4.2.2.2 Build Association Both the Apriori and the FP-Growth implementation of Weka provide different attributes which can be used to adjust the outcome, such as the minimum support, the minimum confidence, etc. Each of the attributes has a default value. If the attributes are not set then the default value will be assigned during the generation of

association rules. In this thesis, for the simplicity of the implementation, the attributes are not set and default values are used. With the default value of confidence level (0.9) and minimum support level (0.1), both the Apriori and FP-Growth generates at least 10 rules. This is done as shown in Listing 5.7 and Listing 5.8

```
Apriori apriori = new Apriori();
apriori.buildAssociations(dataApriori);
```

Listing 5.7: Code Snippet for Build Association Rules Using Apriori

```
FPGrowth fpGrowth = new FPGrowth();
fpGrowth.buildAssociations(dataFPGrowth);
```

Listing 5.8: Code Snippet for Build Association Rules Using FP-Growth

The output of the Apriori and FP-Growth is shown in Listing 5.9 and 5.10

```
Apriori
=======
Minimum support: 0.1 (6 instances)
Minimum metric <confidence>: 0.9
Number of cycles performed: 18
Generated sets of large itemsets:
Size of set of large itemsets L(1): 11
Size of set of large itemsets L(2): 7
Best rules found:
1. targetID=5AE4C8CA-D491-224F-A78C-4578F69896AF 10 ==>
    sourceID=849AFDC4-45A1-3700-AB70-2A32E650C9EA 10 <conf:(1)> lift:(3.53) lev:(0.12) [7]
    conv:(7.17)
2. sourceID=6637B801-5673-61F5-A4C0-30C9D3812EB3 8 ==>
    targetID=A203915A-39C0-7CF1-92F6-81B6A42C4BF4 8 <conf:(1)> lift:(4.29) lev:(0.1) [6]
    conv:(6.13)
3. targetID=3601AFC0-13B3-2BA7-B428-747F02A8BD89 7 ==>
    sourceID=849AFDC4-45A1-3700-AB70-2A32E650C9EA 7 <conf:(1)> lift:(3.53) lev:(0.08) [5]
    conv:(5.02)
4. sourceID=2CC8355C-3CE5-43F7-82A8-196D33B81D8B 6 ==>
    targetID=DDCB56F7-D526-E011-936C-177094DDA67B 6 <conf:(1)> lift:(3.53) lev:(0.07) [4]
    conv:(4.3)
```

Listing 5.9: Output of Apriori

```
FPGrowth found 4 rules (displaying top 4)
1. [sourceID=2CC8355C-3CE5-43F7-82A8-196D33B81D8B=t]: 6 ==>
    [targetID=DDCB56F7-D526-E011-936C-177094DDA67B=t]: 6 <conf:(1)> lift:(3.53) lev:(0.07)
    conv:(4.3)
2. [targetID=5AE4C8CA-D491-224F-A78C-4578F69896AF=t]: 10 ==>
    [sourceID=849AFDC4-45A1-3700-AB70-2A32E650C9EA=t]: 10 <conf:(1)> lift:(3.53) lev:(0.12)
    conv:(7.17)
3. [targetID=3601AFC0-13B3-2BA7-B428-747F02A8BD89=t]: 7 ==>
    [sourceID=849AFDC4-45A1-3700-AB70-2A32E650C9EA=t]: 7 <conf:(1)> lift:(3.53) lev:(0.08)
    conv:(5.02)
4. [sourceID=6637B801-5673-61F5-A4C0-30C9D3812EB3=t]: 8 ==>
    [targetID=A203915A-39C0-7CF1-92F6-81B6A42C4BF4=t]: 8 <conf:(1)> lift:(4.29) lev:(0.1)
    conv:(6.13)
```

Listing 5.10: Output of FP-Growth

5.4.2.2.3 Build JSON of Generated Association Rules After the data mining process is done, the generated association rules are written into a JSON data structure and sent back to the *Modeling Recommendation Service*. This is done so that the *Modeling Recommendation Service* can easily parse the JSON data into an object-oriented structure and store it in the database. An example of a generated JSON output is shown in Listing 5.11

```
{
"rules":
      [
            {
            "sourceID":"6637B801-5673-61F5-A4C0-30C9D3812EB3",
            "leverage":"0.1",
            "targetID":"A203915A-39C0-7CF1-92F6-81B6A42C4BF4",
            "confidence":"1",
            "lift":"4.27",
            "conviction":"6.13"
            }
      ]
}
```

Listing 5.11: JSON Representation of Association Rules Generated by the Wrapper class

5.4.2.3 Step 3 : Map JSON and Store

The *Modeling Recommendation Service* maps the JSON string sent by the wrapper class into an object-oriented structure and stores it in the database. The table storing association rules is shown in Table 5.3

id	sourceId	targetId	Confidence	Lift	Leverage	Conviction
1	849AFDC4-45A1-3700-AB70-2A32E650C9EA	5AE4C8CA-D491-224F-A78C-4578F69896AF	1	3.53	0.12	7.17
2	6637B801-5673-61F5-A4C0-30C9D3812EB3	A203915A-39C0-7CF1-92F6-81B6A42C4BF4	1	4.29	0.1	6.13
3	849AFDC4-45A1-3700-AB70-2A32E650C9EA	3601AFC0-13B3-2BA7-B428-747F02A8BD89	1	3.53	0.08	5.02
4	2CC8355C-3CE5-43F7-82A8-196D33B81D8B	DDCB56F7-D526-E011-936C-177094DDA67B	1	3.53	0.07	4.3

Table 5.3: Table Storing Association Rules

5.5 Integration into FlexMash

In this section, it is discussed how the developed *Modeling Recommendation Service* is integrated into the FlexMash application. The user interfaces developed for providing an interactive recommendation feature are also discussed in this section.

As already mentioned, the *Modeling Recommendation Service* is a RESTful application and is hosted on a server, so it can be accessed by calling the methods provided by the service from the FlexMash application.

When a model is executed in FlexMash, it will send the JSON representation of the model to the *Modeling Recommendation Service* using an HTTP request by calling the method *addConnection*. The method *addConnection* transforms the JSON into the canonical model, stores the canonical model in the database, and runs the data mining process on existing canonical models. After that, the generated association rules are stored in the database. The whole process runs silently without any user interaction or acknowledgments.

To provide an interactive recommendation feature in the FlexMash application, a couple of user interfaces have been developed in this thesis. In the next sections, these user interfaces are discussed in detail.

5.5.1 Node Recommendation Dialog

A Node recommendation dialog is designed to show the list of recommended nodes for a particular node. When users click on the connection port of a node, the node recommendation dialog pops up. The dialog shows the list of recommended nodes for this particular node. This is done by calling the method *getAssociationRule* provided by the

Modelling Recommendation Service. The unique ID of the node is sent as a parament of this GET method. The *Modeling Recommendation Service* returns a JSON reply containing a list of association rules where this node ID appeared as source ID. An example of the JSON reply sent by the *Modeling Recommendation Service* is shown in Listing 5.12. Then FlexMash parses the JSON and extracts all target IDs. After that, the node information are extracted from the FlexMash database using the target ID extracted form the association rules. Finally, the list of recommended nodes is shown in the dialog. The user can select the node from the dialog. The selected node is automatically placed in the modeling canvas and connected with the node for which the recommendations were shown. A screenshot of the designed Node Recommendation dialog is shown in Figure 5.4

```
{
"AssociationRules":
      [
            {
            "sourceID": "849AFDC4-45A1-3700-AB70-2A32E650C9EA",
            "targetID": "5AE4C8CA-D491-224F-A78C-4578F69896AF",
            "confidence": "0.91",
            "lift": "3.23",
            "leverage": "0.11",
            "conviction": "3.95"
            }
      ]
}
```

Listing 5.12: JSON Representation of Association Rules Sent by the *Modeling Recommendation Service*

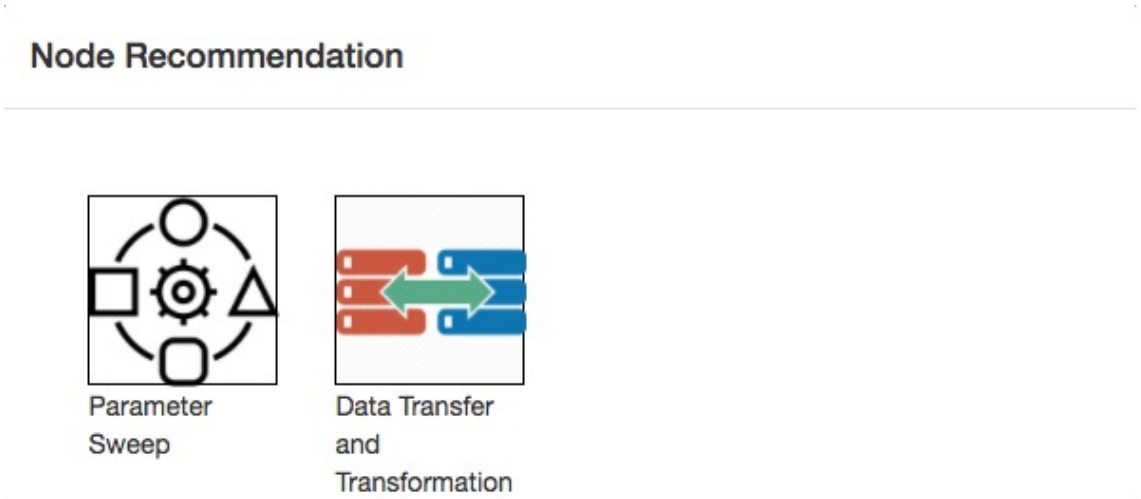

Figure 5.4: Node Recommendation Dialog

5.5.2 Model Completion

A model completion feature is also implemented in FlexMash. If the user is a business user who does not have enough technical knowledge to design a model, then this feature will complete the model using the top recommended nodes. To implement this feature, a new menu item named “Complete Model” is implemented. When users place a node in the modeling canvas and click on the “Complete Model” menu item, FlexMash will call the method *getCompleteMode*l provided by the *Modeling Recommendation Service* with the node ID (placed in the canvas) as parameter in the request. This method first retrieves the top one association rule ordered by the confidence level where the node ID that is sent in the request is source ID. Then the target ID of the retrieved association rule is considered again as source ID to retrieve the association rules. This process is recursively called until there is no association rule found for a node. The resultant list of association rules is then sent back to FlexMash in a JSON structure via the HTTP response. FlexMash then parses the JSON and extracts the source ID and target ID. Finally, using these node informations, a model is drawn in the modeling canvas. An example of a JSON response sent by the *Modeling Recommendation Service* that contains a complete model is shown in Listing 5.13.

```
[
  [
      RowDataPacket
      {
            id: 231,
            sourceID: '849AFDC4-45A1-3700-AB70-2A32E650C9EA',
            targetID: 'A203915A-39C0-7CF1-92F6-81B6A42C4BF4',
            confidence: '1',
            lift: '4',
            leverage: '0.19',
            conviction: '1.5'
      }
  ],
  [
      RowDataPacket
       {
            id: 237,
            sourceID: 'A203915A-39C0-7CF1-92F6-81B6A42C4BF4',
            targetID: 'DDCB56F7-D526-E011-936C-177094DDA67B',
            confidence: '1',
            lift: '4',
            leverage: '0.19',
            conviction: '1.5'
       }
   ]
]
```

Listing 5.13: JSON Representation of a Complete Model Sent by the *Modeling Recommendation Service*

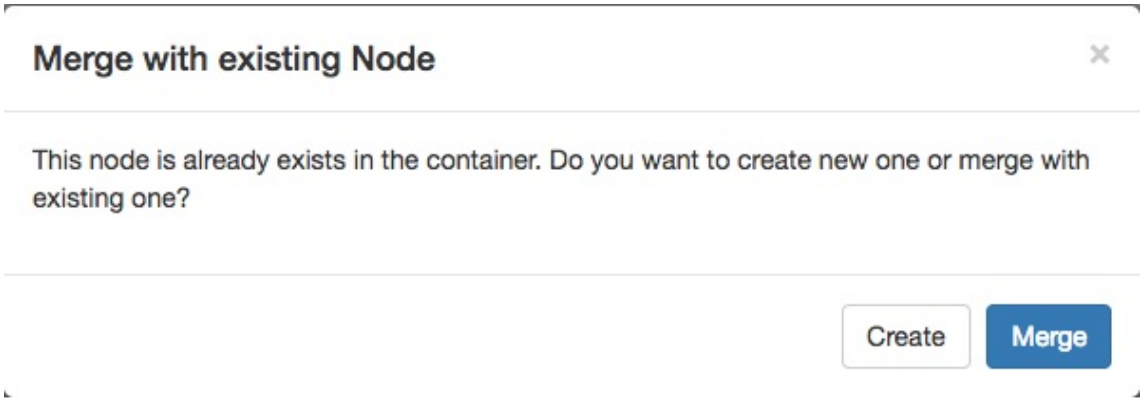

Figure 5.5: Merge Node Dialog

5.5.3 Merge Node Dialog

This dialog is designed to ask users whether a recommended node should be merged with an existing node or not. If a selected recommended node already exists in the modeling canvas then the Merge node dialog pops up asking the user whether the recommended node should be merged with the existing one or not. If users choose “Merge” then the source node is connected to the existing one instead of placing a new instance of the recommended node. If users choose “Create” then the recommended node will be placed in the modeling canvas and connected to the source node as usual. A screenshot of the merge node dialog is shown in Figure 5.5.

5.5.4 Toggle Recommendation

Sometimes, users do not want to see modeling recommendations. This feature is implemented to turn off the recommendation feature. A menu item "Toggle Recommendation" is implemented. The recommendation feature will work if the "Toggle Recommendation" is checked otherwise it will not work.

5.5.5 Model Widget

A model widget is developed where users can design partial models using modeling recommendation features. This widget has three parts. In the top panel of the widget, all the available nodes in FlexMash are shown. Users can select a node category, then the nodes corresponding to this category will be shown. A screenshot of the top panel of the model widget is shown in Figure 5.6.

In the middle panel, a modeling canvas is provided, where users can drag a node from the available nodes panel and drop it into the modeling canvas. This modeling canvas works

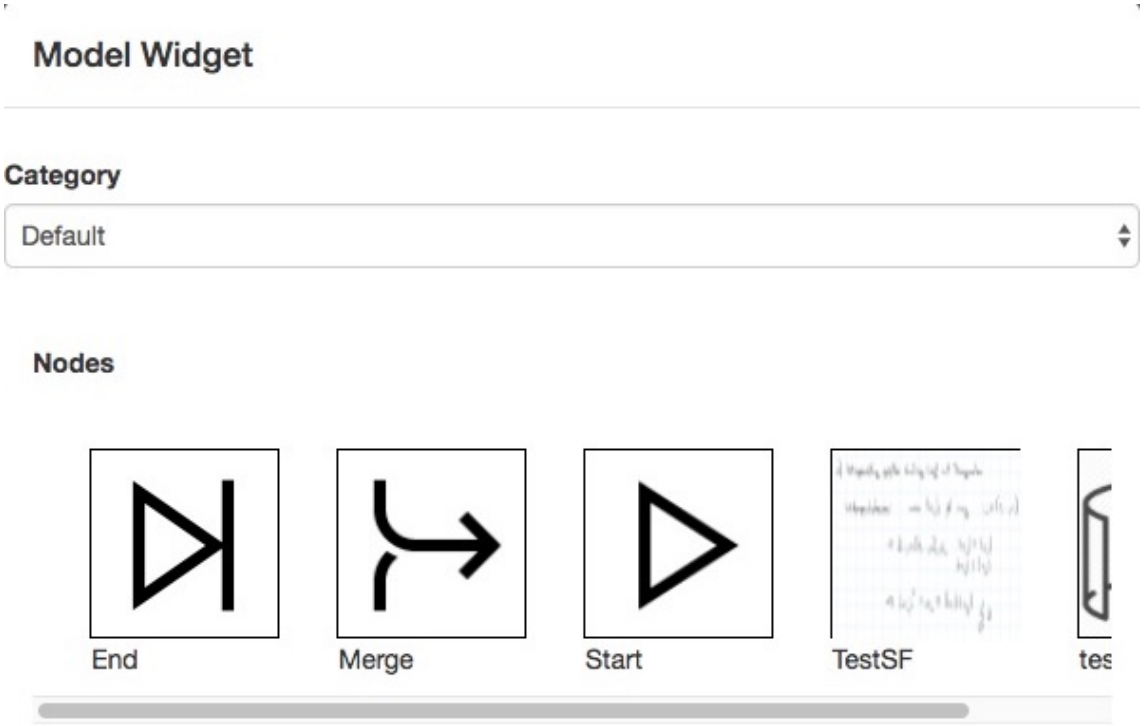

Figure 5.6: Top Panel of Model Widget Dialog

as the main modeling canvas of FlexMash. Users can delete nodes and can merge node with existing nodes. The middle panel is shown in Figure 5.7.

The bottom panel shows the recommended nodes. When users drag and drop a node from the available nodes then the recommended node panel is updated using the list of recommended nodes for the dropped node. The users can select a recommended node

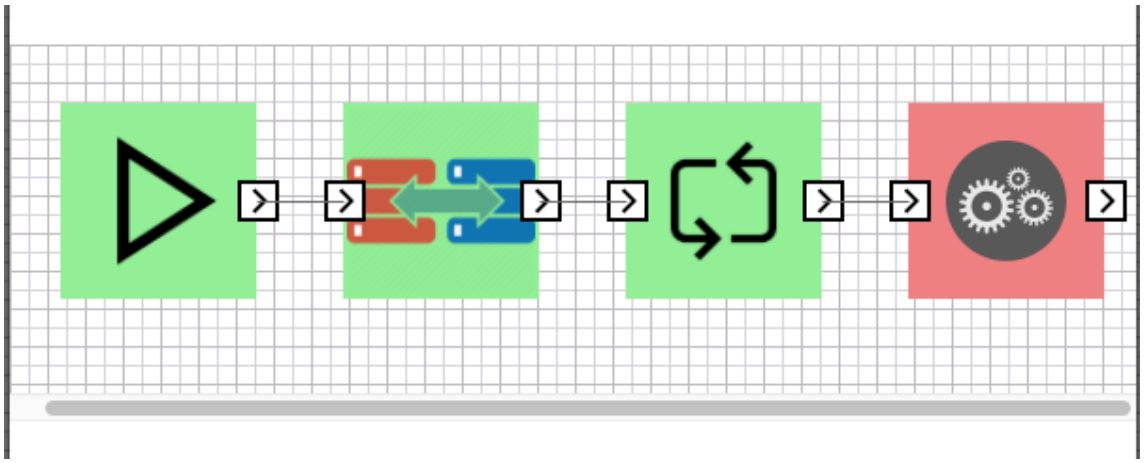

Figure 5.7: Middle Panel of Model Widget Dialog

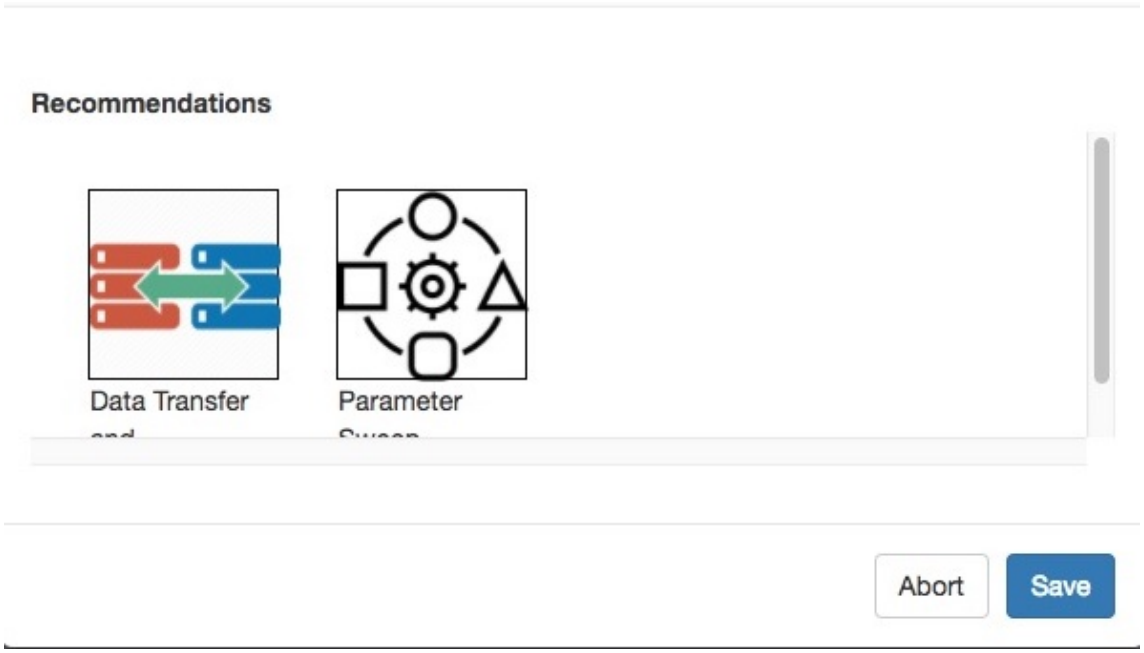

Figure 5.8: Bottom Panel of Model Widget Dialog

which is then automatically placed in the modeling canvas and connected with the source node. Figure 5.8 shows the screenshot of the bottom panel of the model widget.

Finally, if users click on the button “Save”, the partial model is placed into the main modeling canvas. After that, users can add more nodes or remove nodes from the partially designed model.

6 Conclusion and Future Work

The chapter summarizes the thesis work. Some possible future works are also discussed in this chapter.

6.1 Conclusion

The thesis work aims to assist users in mashup development environments like FlexMash. A generalized recommendation service is implemented in this work which can be used in multiple mashup platforms. A canonical model is described which can be used to represent different modeling constructs in a common modeling structure. The thesis also describes the recommendation generation strategy using the association rule mining techniques and how to present recommendations to the users interactively.

Although, the model generated using the recommended nodes does not always represents a valid model, but it shows how nodes have been composed in past. This helps a user with a limited technical knowledge to implement his own domain specific model. This improves the flexibility of the FlexMash application.

The algorithms used in this work to generate association rules uses the default values for the minimum support (0.1) and confidence (0.9) level which is a limitation of this thesis work. The expert users might want to change this values. So, a user interface needs to be implemented to set up the parameters of the algorithms.

The main contribution of this thesis is to assist users in pattern-based mashup development. The work is done to increase the flexibility of FlexMash. Users with no or limited technical knowledge can now develop their mashup model using the recommendation feature.

6.2 Future Work

The work done in this thesis can be enhanced further. Currently, there is no implementation for setting up the parameters of the data mining algorithms. Integrating a user interface for setting parameters, such as minimum support and confidence, could be helpful for skilled people to customize the association rule generation algorithms.

The addition of the recommended node to the modeling canvas works linearly one after another which sometimes goes out of the modeling canvas. An enhancement could be done in future on that issue by calculating the position of all nodes in the canvas.

One possible extension could also be checking for semantical errors in the model. After completion of model design, the model can be validated based on the existing models. Another extension could be, providing suggestions to improve the current model, e.g., replacing a node with a suggested node which is the best fit based on previous models.

Currently, we are considering only models designed in FlexMash to generate recommendations. In the future, we can also learn from the execution of models. Suggestions can be provided to users to improve current models based on execution details of existing models.

Bibliography

[AIS93] R. Agrawal, T. Imielinski, A. Swami. “Mining Association Rules between Sets of Items in Large Databases.” In: *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data* (1993) (cit. on pp. 17–19, 21).

[AS94] R. Agrawal, R. Srikant. “Fast Algorithms for Mining Association Rules.” In: *Proc. 20th Int. Conf. Very Large Data Bases, VLDB* (1994) (cit. on pp. 18, 22–24).

[Bor05] C. Borgelt. “An Implementation of the FP-growth Algorithm.” In: *ACM Press, New York, NY, USA* (2005) (cit. on p. 25).

[CRDC12] S. Chowdhury, C. Rodriguez, F. Daniel, F. Casati. “Baya: Assisted mashup development as a service.” In: (Apr. 2012) (cit. on p. 32).

[CTN+13] S. Chowdhury, A. Tschudnowsky, M. Niederhausen, S. Pietschmann, P. Sharples, F. Daniel, M. Gaedke. *Complementary Assistance Mechanisms for End User Mashup Composition*. May 2013 (cit. on p. 32).

[Hel07] B. L. Helm. “Fuzzy Association Rules An Implementation in R.” In: (Aug. 2007) (cit. on pp. 18, 19).

[HGN00] J. Hipp, U. Guentzer, G. Nakhaeizadeh. “Algorithms for Association Rule Mining - A General Survey and Comparison.” In: *ACM SIGKDD Explorations Newsletter* (2000) (cit. on pp. 22, 23, 26).

[HM16] P. Hirmer, B. Mitschang. “FlexMash - Flexible Data Mashups Based on Pattern-Based Model Transformation.” Englisch. In: Communications in Computer and Information Science 591 (Feb. 2016), pp. 12–30. DOI: 10.1007/978-3-319-28727-0_2 (cit. on pp. 15, 28).

[HPY99] J. Han, J. Pei, Y. Yin. “Mining Frequent Patterns without Candidate Generation.” In: *2000 ACM SIGMOD Intl. Conference on Management of Data, ACM Press* (1999) (cit. on pp. 25, 26, 28).

[HRWM15] P. Hirmer, P. Reimann, M. Wieland, B. Mitschang. “Extended Techniques for Flexible Modeling and Execution of Data Mashups.” Englisch. In: *Proceedings of the 4th International Conference on Data Management Technologies and Applications (DATA)*. Ed. by M. Helfert, A. Holzinger, O. Belo, C. Francalanci. Colmar: SciTePress, July 2015, pp. 111–122. ISBN: 978-989-758-103-8 (cit. on pp. 15, 28).

[Liu11] B. Liu. *Association Rules and Sequential Patterns*. Springer Nature, 2011. ISBN: 978-3-642-19459-7. DOI: https://doi.org/10.1007/978-3-642-19460-3_2 (cit. on p. 18).

[PEl16] D. K. P.Elango. "Fuzzy FP-Tree based Data Replication Management System in Cloud." In: (June 2016) (cit. on p. 27).

[RCD+14] C. Rodríguez, S. R. Chowdhury, F. Daniel, H. R. M. Nezhad, F. Casati. "Assisted Mashup Development: On the Discovery and Recommendation of Mashup Composition Knowledge." In: *Web Services Foundations*. Ed. by A. Bouguettaya, Q. Z. Sheng, F. Daniel. New York, NY: Springer New York, 2014, pp. 683–708. ISBN: 978-1-4614-7518-7. DOI: 10.1007/978-1-4614-7518-7_27. URL: https://doi.org/10.1007/978-1-4614-7518-7_27 (cit. on p. 31).

[RDC11] S. Roy Chowdhury, F. Daniel, F. Casati. "Efficient, Interactive Recommendation of Mashup Composition Knowledge." In: *Service-Oriented Computing: 9th International Conference, ICSOC 2011, Paphos, Cyprus, December 5-8, 2011 Proceedings*. Ed. by G. Kappel, Z. Maamar, H. R. Motahari-Nezhad. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 374–388. ISBN: 978-3-642-25535-9. DOI: 10.1007/978-3-642-25535-9_25. URL: https://doi.org/10.1007/978-3-642-25535-9_25 (cit. on p. 33).

[SA96] R. Srikant, R. Agrawal. "Mining Quantitative Association Rules in Large Relational Tables." In: *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data* (1996) (cit. on pp. 20, 21).

[SS13] M. S.Mythili, M. Shanavas. "Performance Evaluation of Apriori and FP-Growth Algorithms." In: 79 (Oct. 2013), pp. 34–37 (cit. on p. 42).

[wik] wikispaces. *wikispaces*. URL: https://weka.wikispaces.com/ARFF+%28book+version%29 (cit. on pp. 50, 57, 58).

[YKTZ11] Y. Yin, I. Kaku, J. Tang, J. Zhu. *Association Rules Mining in Inventory Database*. Springer, London, 2011. ISBN: 978-1-84996-338-1. DOI: 10.1007/978-1-84996-338-1 (cit. on p. 25).

All links were last followed on March 30, 2018.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature