Interoperability and Integration of Processes of Knowledge Discovery in Databases

Diana Gorea
Department of Computer Science
University Alexandru Ioan Cuza Iasi

Scientific coordinator: prof. Toader Jucan, PhD

The 16th of January, 2009
I would like to dedicate this thesis to my family.
I would like to acknowledge the support offered by my professors.
Abstract

In the context of interoperability in knowledge discovery in databases, this thesis proposes an architecture of an online scoring system (DeVisa) that can be integrated easily in loosely coupled service oriented architectures. It manages a repository of PMML models as a native XML database and exploits their predictive or descriptive properties.

This thesis proposes a novel technique for online scoring based on web services and the specification of a specialized XML-based query language for PMML models called PMQL, used to enable communication with the data mining consumers and for processing the models in the repository. At the abstract level, the thesis presents a theoretical foundation that captures both structural and behavioral aspects of the system providing solutions to problems that arise. The structural aspects include the mining models/schemas, the data dictionaries etc. The behavioral aspects include the way the system interacts with consumers requests, namely scoring or composition requests. A theoretical framework for allowing prediction model composition is provided. Among others it uses the concept of semantic consequence in the functional dependencies theory. In the context of online scoring a novel hybrid technique for schema matching is provided. The technique is based on a modified version of the cycle canceling max-flow min-cost algorithm that allows integrating additional constraints such as derivability and validity. It also proposes an adaptive similarity measure based on string metrics, Jaccard index in the textual description, field statistics and lexical sense. In this context the work presents the global data dictionary architecture that alleviates the overhead of matching complexity within the scoring process and an algorithm for offline incremental update of the GDD.
## Contents

Nomenclature

1 Introduction

2 Knowledge Discovery in Databases
   2.1 General Presentation
   2.2 Knowledge Discovery Process Models
      2.2.1 Academic Models
      2.2.2 Industrial Models
      2.2.3 Hybrid Models
   2.3 Overview of the Methods for Constructing Predictive Models
      2.3.1 Unsupervised Learning
         2.3.1.1 Clustering
         2.3.1.2 Association Rules
      2.3.2 Supervised Learning
         2.3.2.1 Decision Trees
         2.3.2.2 Rule Algorithms
      2.3.3 Model Assessment and other Efficiency Considerations
   2.4 Representing Knowledge within the KDD
      2.4.1 Categories of Knowledge Representation
         2.4.1.1 Rules
         2.4.1.2 Graphs, Trees and Networks
         2.4.1.3 Predictive Models
   2.5 Interoperability in KDD
      2.5.1 Predictive Model Markup Language
8 Applications 128
8.1 DeVisa in a SOA 128
8.2 DeVisa as a Knowledge Repository for Learning Interface Agents 130
8.3 Application of DeVisa in the microRNA discovery problem 132
8.4 Other Applications 132
8.5 A Concrete Use Case 134
8.5.1 Description of the Data Sets 135
8.5.2 The Front End 137
8.5.3 Methods and Results 138
8.5.3.1 Model Selection Use Case 139
8.5.3.2 Schema Match with Global Data Dictionary Use Case 146

9 Conclusions and Further Work 149
9.1 The Contributions of this Work 149
9.2 Further Research Directions 153

A PMQL Specification 155
B Catalog Schema 164
C Use Cases 170
D List of Acronyms 174

References 191
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>The six-step hybrid model ([Cios &amp; Kurgan (2005)]).</td>
<td>11</td>
</tr>
<tr>
<td>3.1</td>
<td>The DeVisa main functional blocks.</td>
<td>31</td>
</tr>
<tr>
<td>3.2</td>
<td>The architecture of DeVisa.</td>
<td>34</td>
</tr>
<tr>
<td>3.3</td>
<td>The Concepts and Relations that are modeled by the Metadata Catalog.</td>
<td>35</td>
</tr>
<tr>
<td>3.4</td>
<td>The Global Data Dictionary as a mediator over the data dictionaries in DeVisa.</td>
<td>36</td>
</tr>
<tr>
<td>3.5</td>
<td>An example of matching between the Global Data Dictionary and the individual data dictionaries. Within a data dictionary mining schemas can overlap. For instance, in $DD_1$, field $C$ is part of two mining schemas. The field matches are annotated with the confidence factor. A field can match only one entry in the GDD with confidence factor 1.</td>
<td>37</td>
</tr>
<tr>
<td>3.6</td>
<td>Resolving consumers’ goals in DeVisa.</td>
<td>41</td>
</tr>
<tr>
<td>3.7</td>
<td>The internal mechanism of the PMQL engine.</td>
<td>44</td>
</tr>
<tr>
<td>4.1</td>
<td>An example of late composition of models. The <em>comb</em> function applies a voting procedure or an average on the outcomes of the individual models.</td>
<td>69</td>
</tr>
<tr>
<td>4.2</td>
<td>An example of implicit sequencing of models.</td>
<td>69</td>
</tr>
<tr>
<td>4.3</td>
<td>Explicit composition of models in DeVisa through a stacking approach.</td>
<td>73</td>
</tr>
<tr>
<td>5.1</td>
<td>A taxonomy of the dimensions that characterize the schema/ontology matching strategies ([Rahm &amp; Bernstein (2001)]).</td>
<td>82</td>
</tr>
</tbody>
</table>
5.2 An example of a valid matching using tripartite graph representation. The set of bold paths represents a valid local match function. The two data dictionaries are represented in squares and diamonds respectively. ................................................. 91

5.3 Examples of match functions (b) and the corresponding flow-network conform to algorithm 5.5.3 (a). The match function $\mu_3$ has maximum confidence and will be chosen. .................................................. 99

5.4 The sequence of max-flows and their corresponding residual networks. The negative cost cycles and augmenting paths are depicted in bold. ................................................................. 107

8.1 The General SOA Architecture. ............................... 129
8.2 Integration of DeVisa in a SOA architecture. .................. 130
8.3 The consumer’s metadata and univariate statistics for numeric fields. 140
8.4 A subset of the Hungarian metadata on which the J48 decision tree model (DeVisa ID 21) is defined and the univariate statistics for the numeric fields. .......................... 141
8.5 A subset of the Long Beach metadata on which the MLP model (DeVisa ID 10) is defined and the univariate statistics for the numeric fields. .......................... 141
8.6 A subset of the Cleveland metadata on which the Logistic Regression model (DeVisa ID 15) is defined and the univariate statistics for the numeric fields. .......................... 142
8.7 The accuracy, true positive rate, false positive rate and the area under the ROC curve of the three base classifiers measured through 10-fold cross validation. .......................... 143
8.8 A PDI workflow representing the scoring using heterogeneous model selection based on the heart disease data dictionary. .................. 143
8.9 The Prediction Voting step dialog. ............................... 144
8.10 The accuracy, true positive rate and false positive rate of the three base classifiers, the averages and of DeVisa selection, measured through testing them with the consumer’s data set. ............ 145
8.11 The ROC analysis of the predictions on the consumer’s data set of the three base classifiers and of DeVisa late composition, measured through thresholding the output probabilities. ........................................... 146
8.12 A PDI workflow representing the schema match over consumer’s metadata and the global data dictionary during scoring over a repository of heart-related prediction models. ................................. 147
C.1 The scoring use case. ................................................................. 170
C.2 The explicit model composition use case. ................................. 171
C.3 The model search use case. ......................................................... 172
C.4 The object model of the PMQL engine component in DeVisa. .... 173
List of Algorithms

3.2.1 The incremental update of the GDD when a new data dictionary is uploaded in DeVisa ........................................ 38
4.3.1 Deciding if a Horn clause is a logical consequence of a set of Horn clauses .................................................. 71
5.5.1 Finding a valid applicable local matching for the consumer’s schema ................................................................. 92
5.5.2 Transformation of the DeVisa weighted bipartite graph into a multiple source single sink flow network .................. 96
5.5.3 Applying the max-confidence max-flow algorithm for the individual data dictionaries and selecting the one that is both derivable and has maximum confidence .................................................. 98
5.5.4 $\text{getMaxFlows}(G^i_f)$ ................................................................................................................................. 101
5.5.5 $\text{getNegativeCycle}(G^i_f, t)$ ....................................................................................................................... 104
5.5.6 $\text{getAugmentingPath}(G^i_f)$ ......................................................................................................................... 105
5.5.7 $\text{getMaxFlow}(G^i_f)$ ..................................................................................................................................... 106
Chapter 1

Introduction

The general area of this thesis subscribes to the domain of managing and sharing the knowledge obtained through knowledge discovery in databases processes. The knowledge discovery in databases (KDD) is a vast domain, whose purpose is to extract high-level knowledge from data. It consists of a sequence of steps that can be roughly divided into understanding the domain, pre-processing the data, data mining and applying the generated knowledge in the domain.

There has been a lot of effort both in academia and industry to formalize the KDD processes into a common framework. Among these initiatives we mention Fayyad et al. (1996a), Cios & Kurgan (2005) and Wirth & Hipp (2000). The proposed frameworks are in many aspects very similar but differ slightly in the number of steps, applicability in certain application domains, specific objectives that are emphasized etc.

KDD is an expensive exercise in general and therefore limited to a certain category of organization. Some of the phases are extremely time consuming (e.g data pre-processing) and the infrastructure to deploy these processes requires high expertise and most of the times expensive hardware and software. The natural consequence is that knowledge has a broader purpose beyond its direct use in the system that has built it. Thus knowledge should be shared and transferred between different systems.

The use of popular industrial standards based on XML (XML) can add value along the KDD chain allowing knowledge interchange between applications in a platform and technology independent manner. During the past several years,
the Data Mining Group DMG has been working in this direction specifying the Predictive Model Mark-up Language or PMML PMML, a standard XML-based language for interchanging knowledge between applications.

The high-level knowledge that is the outcome of KDD can be represented in many forms, such as logical rules, decision trees, fuzzy rules, Bayesian belief networks, artificial neural networks etc. These are often called models and represent a relationship between the input and the output data. Most of the models are supported by the PMML standard.

The most common platform for data mining is a single workstation. More recently, data mining systems have been developed that use web services and service oriented architectures. Consequently, systems have been developed that provide knowledge as a service (Ali et al. (2005)) or even knowledge applications based on cloud infrastructures (Grossman & Gu (2008)).

Sometimes in practice it can occur that several available models that achieve more or less the same function and operate on the same concepts are available. In this situation the user should be offered the possibility of choosing the model that best suits its needs or even combine the results of the existing models. The latter approach is called model composition and has often proved to achieve better predictive performance than the individual models. In general this is not straightforward because the composition itself is only partially defined, covering a limited number of use cases. Furthermore, the combined models share the disadvantage of being difficult to analyze: they can comprise dozens or even hundreds of individual models, and although they perform well it is not easy to understand in intuitive terms what factors are contributing to the improved decisions. On the other side there are several interpretations of the concept of model composition, which are discussed further in this thesis.

Generally speaking the contribution of this work lies in extending the idea of reuse and sharing of knowledge obtained through the KDD processes in the context of KDD globalization. Therefore knowledge can be further stored, efficiently managed and provided via web services. The work presents a system called DeVisa (DeVisa) that is based on collecting knowledge in a domain and further providing it as a service to consumer applications. It is intended to provide feasible solutions for dynamic knowledge integration into applications that
do not necessarily have to deal with knowledge discovery processes. Nowadays the true value of KDD is not relying on its collection of complex algorithms, but moreover on the practical problems that the knowledge produced by these algorithms can solve. Therefore DeVisa focuses on collecting such knowledge and further providing it as a service in the context of service oriented architectures.

DeVisa provides several services, whose general description was presented in 
Gorea (2008a) and Gorea (2008b). The scoring service represents a mean to provide knowledge as a service in DeVisa. In this context Gorea (2008c) formalizes the main entities in the system and the online scoring problem. In addition the work provides a general context in which composition of data mining models can be achieved within the boundaries of this online scoring system(Gorea (2008d)).

A PMML query language, called PMQL, used both and for interacting with the consumer is defined.

Early work on this topic presents some general features of the system and potential uses cases (customer reliability in banking applications, knowledge bases for interface agents, bioinformatics applications). In Gorea & Felea (2005), Gorea (2005c) and Gorea & Buraga (2006) the idea of storing decision tree prediction models and applying online predictions that can be integrated into decentralized software architectures and in Semantic Web (SW) was depicted. Gorea (2007) gives a more detailed overview of the system, although the means through which the models should be processed were still not very clearly specified.

This work is organized as follows.

Chapter 1 provides a general overview of the knowledge discovery in databases, highlighting the efforts in the academic and industrial communities to unify the processes in common frameworks. In this context the necessity to interchange knowledge is concretized through the definition of a standard language named PMML for expression of high level knowledge obtained via KDD. Therefore the ability to store and exchange predictive models through the use of the PMML standard is presented.

Chapter 2 defines the notion of knowledge as a service and positions the DeVisa system in the field. It gives a functional and architectural perspective of DeVisa. The functional perspective highlights the main services that DeVisa exposes - scoring, composition, search, comparison, administration and the
mechanisms used in DeVisa for resolving consumer’s goals. The architectural perspective focuses on the main internal components - PMQL engine, PMML repository, metadata catalog - and the interaction between them. A language called PMQL (specified by DeVisa) designed especially for interaction between the DM consumers and DeVisa is presented. PMQL allows expressing consumer’s goals and the answers to the goals. The chapter ends with a highlight on the process of composition of prediction models. It first presents an overview of the existing approaches to model composition and the types of model composition supported by the PMML standard. Furthermore several possible approaches for the prediction models composition problem in DeVisa are presented.

Chapter 3 provides the formalization of the main entities DeVisa is composed of and of the main functionality. More exactly, the data dictionaries, the mining models/schemas, the repository and the metadata catalog are defined using formalisms from the relational model. Furthermore, some behavioral aspects are captured, such as the scoring process, matching schemas or composition process. The mining models/schemas are defined in terms of the functional dependency theory and a basic framework to derive new models is described. This approach is used further in the composition process when a new model needs to be derived from a set of models conforming to schemas in a certain data dictionary.

Chapter 4 is dedicated to the schema integration problem. First different flavors of the problem are overviewed and different strategies to solve the problem are investigated. Furthermore the chapter presents the projection of the schema integration problem in DeVisa and defines a new hybrid 1:1 schema matching strategy as well as a new adaptive similarity measure. The schema matching strategy is partly modeled as the max-confidence min-flow problem and an algorithm is provided. The similarity measure is multi-faceted, being able to adapt to available schema information.

Chapter 5 presents mostly related work in the field of scoring engines, knowledge as a service and online predictive models interchange in general and highlights DeVisa distinguishing features with respect to the existing approaches.

Chapter 6 presents a few implementation details, including an overview of the technologies that DeVisa is based on. It also presents the current status of the implementation, comprising the PMQL specification, the schema matching
technique implementation, the PMQL web service, the XQuery scoring functions, the Weka PMML export etc.

Chapter 7 presents several usage examples of the system, including: integration in a SOA as a decision support module, in particular in e-Health systems, integration in Semantic Web context as a knowledge base for agents, or aggregating the existing PMML predictive models that were published in the recent scientific work in the micro-RNA discovery problem into a common repository and providing further predictive support or deriving new potentially more powerful models from the existing ones. In addition the chapter presents a concrete example, built on top of the Pentaho Data Integration (PDI) tool, in which several DeVisa functions are developed as plug-ins. The example uses 3 data sets in the UCI machine learning repository, namely Heart disease, Arrhythmia, and Echocardiogram. The use cases are modeled as data flows in PDI and present the model selection (or implicit late composition) and schema matching scenarios.

Chapter 8 presents conclusions to the present work and draws roadmap with possible research directions or implementation plans.

The first appendix presents the PMQL specification document based on XML Schema. The second appendix presents the metadata catalog specification that needs to be implemented by any DeVisa instance. The third appendix presents the scoring, explicit composition, search and comparison use cases expressed as sequence diagrams.

Out of all the bibliographical citations a number of 12 represent original contributions of the author (either as unique author or as a co-author). The publications belong to well known international academic publishing houses. The current work was partially supported by the CNCSIS contract type PN-TD number 404/200, director Diana Gorea.
Chapter 2

Knowledge Discovery in Databases

2.1 General Presentation

Knowledge discovery in databases (KDD) is a non-trivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data (Fayyad et al. (1996b)). The goal is to seek new relevant knowledge about an application domain.

The domain concerns the entire process of knowledge extraction, including the way raw data is stored, understood, cleaned, transformed, the analysis and understanding of the domain, the algorithms that are applied, evaluation of the results, interpretation of the results, integration of the knowledge back into the domain etc. Thus the process consists of a sequence of steps, each of the steps being based upon the successful completion of the previous step.

KDD is a field that has evolved from the intersection of research fields such as machine learning, pattern recognition, databases, statistics, AI, knowledge acquisition from expert systems, data visualization and high performance computing. The unifying goal is to obtain high-level knowledge from low-level data. Knowledge can be represented in many forms, such as logical rules, decision trees, fuzzy rules, Bayesian belief networks, and artificial neural networks. The data mining component of KDD relies on techniques from machine learning, pattern
2.2 Knowledge Discovery Process Models

recognition or statistics to build patterns or models that are further used to extract knowledge. Thus data mining applies algorithms that-under acceptable computational efficiency limitations-produce the patterns or models.

In general, building data mining models is very expensive because it involves analyzing large amounts of data. It is universally acknowledged that the data pre-processing phase is the most consuming step in terms of computational time (Cios & Kurgan (2005)). Pre-processing the data involves collecting, cleaning, transforming, preparing the raw data. In the case in which the data resides in different heterogeneous sources an integration phase adds up to the process, making it even more expensive. In the modeling phase the complexity of the algorithms generally depends on various factors like the prior knowledge about the domain, the size of the schema and the number of data instances. A considerable focus in the literature has been on the algorithmic aspect of data mining. A systematic overview and comparative analysis of data mining algorithms as well as a comprehensive discussion on the computational complexity can be found in Lim & Shih (2000), Fasulo (1999), Hand et al. (2001) and Ian H. Witten (2005).

2.2 Knowledge Discovery Process Models

The knowledge discovery process can be formalized into a common framework using process models. A KD process model provides an overview of the life cycle of a data mining project, containing the corresponding phases of a project, their respective tasks, and relationships between these tasks. There are different process models originating either from academic or industrial environments. All the processes have in common the fact that they follow a sequence of steps which are more or less resembling between models. Although the steps are executed in a predefined order and depend on one another (in the sense that the outcome of a step can be the input of the next step), it is possible that, after execution of a step, some previous steps are re-executed taking into consideration the knowledge gained in the current step (feedback loops).

An interesting issue that arises in any KDD process model is choosing which algorithm or method should be used in each phase of the process to get the best results from a dataset. Since taking the best decision depends on the properties
of the datset, Popa et al. (2007) proposes a multi-agent architecture for recommendation and execution of knowledge discovery processes.

2.2.1 Academic Models

The most representative model in this category is the nine step model introduced in Fayyad et al. (1996a), which consists of the following steps:

1. Understanding the application domain. This step involves learning the prior knowledge about the domain and about the user’s goals.

2. Creating a target data set. A subset of the initial data needs to be filtered through means of selection and projection.

3. Data cleaning and preprocessing. This step involves dealing with outliers, noise and missing values etc.

4. Data reduction and projection. Methods for dimension reduction and transformation are applied in order to find an invariant representation of the data.

5. Choosing the data mining task. The user’s goals need to be matched against a known data mining task, such as classification, regression, clustering etc.

6. Choosing the data mining algorithm. The appropriate algorithm for the task identified in the previous step is chosen and its parameters are adjusted.

7. Data mining. The models or patterns are generated based on the chosen algorithm.

8. Interpreting mined patterns. This involves visualization of the extracted knowledge and data visualization based on this knowledge.

9. Consolidating discovered knowledge. This step involves incorporating the knowledge into the domain, documenting it and reporting it to the user. It is possible that the new knowledge conflicts with the previously obtained knowledge. In this case the appropriate measures needs to be taken.
2.2 Knowledge Discovery Process Models

2.2.2 Industrial Models

The academic models usually don’t take into consideration the industrial issues. Therefore in the industrial world some efforts were developed to adapt the KD process models to their business needs. In this category the most representative is the CRISP-DM (CRoss Industry Standard Process for Data mining) model, a tool-neutral process model initially developed by a consortium of companies. Nowadays the model has strong support in the industrial world. The process consists of six steps:

1. Business understanding. This step involves understanding the project objectives and requirements from a business perspective, and then converting this knowledge into a data mining problem definition.

2. Data understanding. This comprises initial data collection, getting familiar with the data, identifying data quality problems, detecting interesting subsets to form hypotheses for hidden information.

3. Data preparation. It covers all activities to construct the final dataset (data that will be fed into the modeling tool). Data preparation tasks are likely to be performed multiple times, and not in any prescribed order. Tasks include table, record, and attribute selection as well as transformation and cleaning of data for modeling tools.

4. Modeling. Various modeling techniques are selected and applied, and their parameters are calibrated to optimal values. Typically, there are several techniques for the same data mining problem type. Some techniques have specific requirements on the form of data. Therefore, stepping back to the data preparation phase is often needed.

5. Evaluation. It ensures that the obtained knowledge properly achieves the business objectives. A key objective is to determine if there is some important business issue that has not been sufficiently considered. At the end of this phase, a decision on the use of the data mining results should be reached.
6. Deployment. The knowledge gained will need to be organized and presented in a way that the customer can use it. This normally can range from generating reports or implementing a repeatable data mining process.

### 2.2.3 Hybrid Models

A hybrid model that combines aspects from both worlds is the six step model presented in Cios & Kurgan (2005). Although it is based on CRISP-DM, it provides a more general, research-oriented description of the steps and adding more explicit feedback loops. The model is depicted in figure 2.1

From the perspective of the current work this is the most interesting because it highlights an extension to the knowledge usage step, in which the knowledge discovered for a particular domain can be applied to other domains.

A feasible consequence is that some users or organizations, called consumers, can benefit from the knowledge acquired in a completely different context by the producers. The consumers do not need to have extensive background knowledge or manipulate the data. This goal involves storing and exchanging knowledge together with the metadata associated with the domain.

### 2.3 Overview of the Methods for Constructing Predictive Models

#### 2.3.1 Unsupervised Learning

Unsupervised Learning paradigm is a process that automatically reveals structure in a data set without any supervision.

##### 2.3.1.1 Clustering

Given $X = \{x_1, x_2, \ldots, x_m\}$ a set of $n$-dimensional tuples, the problem is to identify groups (clusters) of tuples which belong together (are similar). The similarity is defined in terms of different metrics, depending on the type of the features of the tuples. For continuous features one can use many distance functions such as Euclidean, Hamming, Tchebyshev, Minkowsky, Camberra, angular separation.
2.3 Overview of the Methods for Constructing Predictive Models

For binary features one can use similarity coefficients such as Russel, Jaccard or Czekanowski. There are several approaches to clustering:

1. *Hierarchical*, in which either clusters are gradually grown by agglomerating similar instances (bottom up) or are divided as dissimilar instances are separated (top down). The distance between clusters is calculated through various methods, as *single link*, *complete link* or *average link*. Hierarchical
2.3 Overview of the Methods for Constructing Predictive Models

clustering produces a graphical representation of the data (dendrogram).

2. *Partition-based*, in which an objective function (performance index) needs to be minimized. A good choice of the objective function reveals the true structure of the data. Depending on the formulation of the objective function and on the organization of the optimization activities, the most representative approaches are: *k-means*, growing a hierarchy of clusters, *kernel-based clustering*, *k-medoids*, *fuzzy c-means*.

3. *Model-based*, in which a probabilistic model is assumed and its parameters are estimated.

4. *Grid-based*, in which cluster structures are described in the language of generic geometric constructs like hyperboxes and their combinations.

2.3.1.2 Association Rules

Given a set of items \( I = \{i_1, \ldots, i_m\} \) and a set of transactions \( D \subseteq 2^I \), where each transaction \( T \in D \) has associated an unique identifier \( tid \). A transaction contains an *itemset* \( A \subset I \) if and only if \( A \subseteq T \). \( A \) is a *k-itemset* if \( |A| = k \). If \( A, B \subset I \), \( A \cap B = \emptyset \), we define an association rule as a dependency \( A \Rightarrow B \) [*support, confidence*] (Agrawal & ad A. Swami (1993)).

Support and confidence are used to measure the quality of a given rule in terms of its usefulness and certainty.

The *support* indicates the frequency (probability) of the entire rule with respect to \( D \), being defined as the fraction of transactions containing \( A \) and \( B \) to the total number of transactions, or the probability of co-occurrence of \( A \) and \( B \).

\[
\text{support}(A \Rightarrow B) = P(A \cup B) = \frac{|\{T \in D|A \cup B \subseteq T\}|}{|D|}
\]

The *confidence* indicates the strength of implication in the rule and is defined as the fraction of transactions containing \( A \) and \( B \) to the number of transactions containing \( A \).

\[
\text{confidence}(A \Rightarrow B) = P(B|A) = \frac{|\{T \in D|A \cup B \subseteq T\}|}{|\{T \in D|A \subseteq T\}|}
\]
Both measures are often represented in terms of a minimum threshold (let’s denote them as $s_{\text{min}}$ and $c_{\text{min}}$). Rules that satisfy both minimum thresholds are called strong association rules. The support count for an itemset $A$ is defined as:

$$\text{supCount}(A) = |\{T \in D | A \subseteq T\}|$$

$A$ is a frequent itemset if $\text{supCount}(A) \geq s_{\text{min}} \cdot |D|$, where $s_{\text{min}} \cdot |D|$ is called the minimum support count.

To generate single dimensional association rules the following four steps are used:

1. Prepare input data in transactional format
2. Choose items of interest, i.e itemsets
3. Compute support counts to evaluate whether selected items are frequent, i.e whether they satisfy minimum support
4. Generate strong association rules that satisfy the minimum confidence by computing the conditional probabilities.

Depending on the second and third step, there are different approaches depending on the computational complexity:

1. The Naive Algorithm, that explores the entire solution space ($2^m \cdot n$ itemsets), which makes it computationally infeasible for a large number of items;

2. The A priori Algorithm, (Agrawal & Srikant (1994)), which uses prior knowledge about the property that all nonempty subsets of an frequent itemset must also be frequent. This property is used to reduce the number of itemsets that must be searched to find frequent itemsets. Improvements on the a priori algorithm include hashing, removal of transactions, partitioning, sampling or mining itemsets without candidate generation.
2.3 Overview of the Methods for Constructing Predictive Models

2.3.2 Supervised Learning

There are three dominant groups in the domain of supervised learning, namely statistical methods, neural networks and inductive machine learning. Statistical methods include the Bayesian methods and regression models. The rest of this section overviews briefly some of the inductive machine learning algorithms.

Inductive machine learning is a technique used to learn from examples. The central concept is that of an hypothesis, that approximates a concept and is generated by the given algorithm (learner) based on a given data set. Let us consider that the training set is $S = \{(x_i \in C_j | i = 1, \ldots, N, j = 1, \ldots, c)\}$, where $x_1, \ldots x_N$ are the $N$-dimensional data instances and $C_1, \ldots, C_c$ are the classes (categories) data tuples, which are a priori known. The classes are often represented through decision attributes. $S$ represents information about a domain.

The hypotheses are expressed as production rules that cover the examples. In general the algorithm needs to provide a hypothesis for each of the classes.

Depending on the knowledge format used in representing the hypothesis the inductive machine learning algorithms can be split into three families: decision trees, decision rules or hybrid.

2.3.2.1 Decision Trees

The main objectives for designing decision tree classifiers are: to correctly classify as much of the training data instances as possible, to classify training instances with a maximum degree of accuracy, to have the simplest structure that can be achieved, and to be easy to update as more training data becomes available.

Resulting directly from the above, a good design of a decision tree must assure the balance between the accuracy and efficiency measures.

Tree construction can be divided into following tasks: (1) choosing the appropriate tree structure, (2) choosing the feature subsets to be used at each internal node and (3) choosing the decision rule to be used at each internal node.

Designing an optimal decision tree means maximizing the mutual information gain at each level of the tree. The problem of designing an optimal decision tree is a NP-complete problem (Hyafil & Rivest (1976)), motivating the necessity for finding efficient heuristics for constructing near optimal decision trees.
Various heuristics for constructing a decision tree may be categorized in: the bottom-up approach (Landeweerd (1983)), the top-down approach (Sethi & Sarvarayudu (1982)), the hybrid approach, the growing-pruning approach (Breiman (1984), Quinlan (1986), Quinlan (1993)), the entropy reduction approach. Some of the methods separate the tree structure construction task from the attribute or decision rule selection at each node, while others tend to combine these three tasks.

Some of the algorithms that assure the balance between the accuracy and efficiency measures were implemented by the majority of the decision support software and are briefly presented in the following paragraphs.

CHAID (Kass (1980)) and Exhaustive CHAID (Biggs et al. (1991)) are algorithms suitable for both classification and regression trees and are used especially for large datasets. CART (classification and regression trees) incorporates the growing-pruning method suggested in Breiman (1984) – it is considered computationally expensive, because in the pruning phase it generates a sequence of trees from which the one that minimizes the misclassification rate is chosen.

In Quinlan (1986) a growing-pruning algorithm, called ID3, is proposed, in which training data is initially placed in the root and then repeatedly split into partitions by the value of a selected feature at each node until no further splitting is possible or the class label is the same for all the instances in the current node. The order of attributes is selected by calculating the value of entropy function. ID3 involves a post-pruning process to eliminate branches that do not produce gain in accuracy. The ID3 algorithm achieves a complexity of $O(n)$ where $n$ is the number of instances.

C4.5 is a software extension to the basic ID3 algorithm proposed in Quinlan (1993) to address the following issues not dealt with by ID3: handling attributes with missing or continuous values, handling attribute with different costs and avoid over fitting of data by determining how deeply to grow a decision tree choosing an attribute selection measure. An improvement of C4.5 is C5 which provides more efficiency and a few additional facilities like variable misclassification costs, adding new data types, defining attributes as functions of other attributes, and giving support for sampling and cross-validation. The computational complexity of C5.0 is $O(n \log n)$. 
2.3 Overview of the Methods for Constructing Predictive Models

Regarding feature selection at a certain node, we distinguish two main approaches: an univariate test, when the decision rule at the node depends on a single feature (most of the presented methods), and multivariate test (Murthy et al. (1994)) – in this case, the decision rule depends on more features.

When training data arrives in continuous or periodic flow, it is more reasonable to update the existing tree than to rebuild it from scratch. Utgoff et al. (1997) presents a ID3-based method called ID5R for mapping a decision tree and a new training data set into a new tree. Transforming one tree into another requires the ability to efficiently restructure a decision tree. To do this, a transposition operator is defined that transforms the tree into another one that also incorporates the new set of instances. After the transposition operation is accomplished, the node decision rules have to be rearranged according to the feature selection metric that was used initially to construct the tree. Other approaches can be consulted in Crawford (1989), Fisher (1996) or Lovell & Bradley (1996).

2.3.2.2 Rule Algorithms

As it is stated in 2.4.1.2, there is a connection between decision trees and rules, in the sense that a decision tree can be easily translated into rules. Rules have several advantages over decision trees: are modular and independent, easy to comprehend, can be written in FOL format or directly used in a knowledge base in expert systems. The disadvantage would be that they do not show the relation between the component rules.

In the case of rules, it is important to establish a balance between generalization and specialization of the rule set. A general rule covers more training examples, while a specialized one covers a small one. A rule that covers the majority of the training examples is called a strong rule (to a certain extent the concept is similar to the correspondent of association rules).

Some representative algorithms for rule generation are SLIPPER and DataSqueezer. SLIPPER (Cohen & Singer (1999)) is a rule learner that generates rulesets by repeatedly boosting a simple, greedy, rule-builder, imposing appropriate constraints on the rule-builder and using confidence-rated boosting (a generalization of Adaboost). DataSqueezer (Kurgan et al. (2006)) is a simple, greedy, rule builder
that generates a set of production rules from labeled input data and exhibits log-linear complexity.

### 2.3.3 Model Assessment and other Efficiency Considerations

Within a KDD process, after generating one or several models, the data miner should assess the quality of the produced model before the deploying it at the user’s site (2.2). The user (or the data owner) has extensive knowledge about the domain and can evaluate the model against this knowledge. On the other side, the data miner does not possess this knowledge and needs to assess a model in terms of how well it describes the training data or how well it predicts the test data. These methods calculate some interestingness measures, use heuristics or resample (reuse) the data.

Each model has associated an error, which is calculated as the difference between the true value and the predicted value and is expressed either as an absolute value or squared value of the respective difference. A model needs to be evaluated both for goodness of fit (fitting the model to data) and for goodness of prediction. The goodness of prediction is expressed in terms of overfitting, when the model is too complex, or underfitting, when the model is too generic. Normally, when the data set is small, a simple model is a better choice.

The model assessment techniques depend strongly on the learning method, or, more exactly, on the nature of the data available. In the supervised learning, each data item has an associated output and it is straightforward to calculate the error of a prediction. In the unsupervised case, such data is not available, so the error cannot be calculated. According to Cios et al. (2007) the assessment methods can be classified as:

- Data re-use methods: simple split, cross-validation, bootstrap - for supervised learning methods;
- Heuristic: parsimonious model, Occam’s razor, very popular due to their simplicity;
2.4 Representing Knowledge within the KDD

- Analytical: Minimum Description Length (MDL), Bayesian Information Criterion, Aikake’s Information Criterion;

- Interestingness measures, which try to simulate the evaluation process made by the user.

To deal with large quantities of data, the data mining algorithms should be scalable. Another approach in dealing with scalability problem to achieve user support is integration of KDD operators in database management systems. In Geist & Sattler (2002) a uniform framework based on constraint database concepts as well as interestingness values of patterns is proposed. Different operators are defined uniformly in that model and DBMS-coupled implementations of some of the operators are discussed.

Another important property is robustness, i.e the algorithm should not be affected by imperfect/incomplete data (missing values, outliers, invalid values).

2.4 Representing Knowledge within the KDD

KDD in general deals with an enormous amount and diversity of data: continuous quantitative data, qualitative data (ordinal, nominal), structured data (hierarchies, relational data), unstructured data (text documents). The available domain knowledge and the knowledge acquired through the KDD processes can be organized in various ways. This process is called knowledge representation and has a high impact on the effectiveness, accuracy and interpretability of the KDD results.

Considering the aspects mentioned above the knowledge representation scheme should be carefully chosen. There are a number of factors (human centric or problem/algorithm centric) worth considering when representing knowledge in a specific model:

- expressive power of the model

- computational complexity, flexibility, scalability, as well as the trade-offs between them
2.4 Representing Knowledge within the KDD

- existing domain knowledge and experimental data
- the level of specificity (granularity) to be modeled

2.4.1 Categories of Knowledge Representation

2.4.1.1 Rules

A rule is a statement of the form:

```
if condition then
  conclusion (action)
```

Both condition and conclusion describe knowledge about the domain. The rule expresses a dependency between them, which is relevant to the problem. The condition and conclusion contain conceptual entities - or information granules that can be formalized using different abstraction frameworks.

The rules can have the condition expressed as a conjunction of other conditions.

```
if condition_1 \land condition_2 \land \ldots \land condition_n then
  conclusion (action)
```

In many cases a domain knowledge is formed by a set of rules, each of them assuming a similar format. The rules represent a collection of relationships within the domain. The collection should be extensible and consistent, i.e there are no conflicts in the collection and new rules can be added without affecting the existing ones.

As stated before, the conceptual entities contained in the conditions and conclusion can be expressed using different frameworks, which gives rise to a variety of architectures and extensions. Several of those are overviewed in the following paragraphs.

1. Gradual rules. Instead of stating a relationship between condition and conclusion, the gradual rules capture a trend within the conceptual entities contained in the condition and conclusion. Thus the rule will express a relationship between the trends in the condition and conclusion.

```
if \phi(condition) then
```
2.4 Representing Knowledge within the KDD

2. *Quantified rules* associate a degree of relevance or likelihood (quantification). The quantification can be expressed as confidence factors (numbers in a specific range) - in expert systems (Giarratano & Riley (1994)), or linguistic values - fuzzy logic (Zadeh (1996)). Also *association rules* (Han & Kamber (2001)) are usually expressed using quantification values, such as *support* and *confidence*.

3. *Analogical rules* focus on the similarity between pair of items in the condition and in the conclusion, aiming to form a framework for analogical reasoning.

4. *Rules with regression local models*. In such rules the conclusion comes in the form of regression models, whose scope is restricted to the condition part of the rule.

\[
\text{if condition then} \\
y = f(x, \text{condition})
\]

The rules are the building block in creating and maintaining highly modular reasoning frameworks. When representing generic entities of information in the condition and the conclusion of a rule, the granularity of information plays an important role, because it captures the essence of the problem and facilitates its handling. Granular computing (Bargiela & Pedrycz (2003)) enables coping with the necessary level of specificity/generality in the problem. Information granules can be expressed using a variety of formal systems, including interval analysis, fuzzy sets (Zadeh (1996)), shadowed sets (Pedrycz (1998)) or rough sets (Pawlak (1991)).

The relevance of a rule depends strongly on the correlation between the level of granularity in the condition and in the conclusion. In general, low granularity (high generality) of the condition correlated with a high granularity (high specificity) of the conclusion describes a rule of high relevance.
2.4 Representing Knowledge within the KDD

2.4.1.2 Graphs, Trees and Networks

The different flavors of graphs (Edwards (1995)), such as undirected, directed, weighted, can be used to express concepts (nodes) and the relationships between concepts (edges). When dealing with a significant number of nodes, a graph can be structured as an hierarchy of graphs, when nodes can be expanded to a lower level graph.

Ontologies can be usually represented using graph structures. Since the emergence of the Semantic web (SW), the graphs can be described in the Resource Description Framework (Herman et al. (2008)) line of languages by triples of the form (subject, predicate, object), as illustrated in the Notation 3 (Berners-Lee (2006)) syntax.

Networks are a generalization of the graph concept so that each node has local processing capabilities.

Decision trees are structures based on a special category of graphs - trees, in which the root and the terminal nodes are denoted. Each node of a decision tree represents an attribute with values ranging within a finite discrete set. The edges originating from the respective node are labeled with the specific values. So at each node an instance is mapped into a subset of class labels based on the value of the designated attribute.

In general decision trees can be easily translated into collection of rules. A rule corresponds in fact to a tree traversal from the root to a terminal node collecting the attributes and values on the way. This way the outcome of a global complex decision (the target function) can be approximated by the union of simpler local decisions (the decision rules) made at each level of the tree. The label of the leaf represents the output of the target function that takes a test instance as input. In a decision tree the attributes stored in nodes closer to the root tend to have higher weight (importance) in the decision process, depending on the tree construction method. While organizing the knowledge into rule sets this aspect is in general not captured.
2.4.1.3 Predictive Models

In general, models are some useful approximate representation of phenomena that generate data can be used for prediction, classification, compression or control design (Cios et al. (2007)). In KDD and data mining in particular a model is designed based on a set of data instances (samples from a population).

The high-level knowledge extracted from low-level data is increasingly seen as a key to provide a competitive edge and support the strategic decision making process within an organization. In KDD knowledge can be represented in many forms, such as logical rules, decision trees, fuzzy rules, Bayesian belief networks, artificial neural networks or mathematical equations. The ways in which this knowledge can be formulated are called models (and sometimes classifiers or estimators). According to Cios et al. (2007) a model can be defined as a description of a causal relationship between input and output variables.

2.5 Interoperability in KDD

2.5.1 Predictive Model Markup Language

As Cios et al. (2007) states, the future of KDD process models lies in achieving overall integration of its process through the use of standards based on XML, such as PMML. During the past several years, the Data Mining Group DMG has been working in this direction specifying the Predictive Model Mark-up Language or PMML PMML, a standard XML-based language for interchanging knowledge between applications such as data mining tools, database systems, spreadsheets and decision support systems. PMML can be also used for knowledge integration of results obtained by mining distributed data sets.

PMML is complementary to many other data mining standards. It’s XML interchange formats is supported by several other standards, such as XML for Analysis (*** (2008d)), JSR 73 (*** (2005)), and SQL/MM Part 6: Data Mining (Melton & Eisenberg (2001)).

At this moment PMML has reached the version 3.2 and the validity of a PMML document is defined with respect to both the reference XML Schema
2.5 Interoperability in KDD

(Sperberg-McQueen & Thompson (2008)) and the restrictions specified in PMML Schema (PMML).

Along the KDP framework PMML adds value starting from the data mining phase and continuing with knowledge utilization, which is in itself an iterative process. Through the use of PMML users can generate data mining models with one application, use another application to analyze these models, still another to evaluate them, and finally yet another to visualize the model. The application that generates a data mining model is called producer and the other applications that use the model in any other way are called consumers.

With PMML, statistical and data mining models can be thought of as first class objects described using XML. Applications or services can be thought of as producing PMML or consuming PMML. A PMML XML file contains enough information so that an application can process and score a data stream with a statistical or data mining model using only the information in the PMML file.

Broadly speaking most analytic applications consist of a learning phase that creates a (PMML) model and a scoring phase that employs the (PMML) model to score a data stream or batch of records. The learning phase usually consists of the following sub-stages: exploratory data analysis, data preparation, event shaping, data modeling and model validation (see 2.2). The scoring phase is typically simpler and either a stream or batch of data is scored using a model. PMML is designed so that different systems and applications can be used for producing models and for consuming models.

As mentioned in 2.3.3, data preparation is often the most time consuming part of the data mining process. PMML provides explicit support for many common data transformations and aggregations used when preparing data. Once encapsulated in this way, data preparation can more easily be re-used and leveraged by different components and applications.

The models expressed in PMML serve predictive and descriptive purposes. The predictive property means that models produced from historical data have the ability to predict future behavior of entities. The descriptive property is when the model itself is inspected to understand the essence of the knowledge found in data. For example, a decision tree can not only predict outcomes, but can also provide rules in a human understandable form. Clustering models are not
only able to assign a record to a cluster, but also provide a description of each cluster, either in the form of a representative point (the centroid), or as a rule that describes why a record is assigned to a cluster.

The PMML language uses an XML-based approach for describing the predictive models that are generated as the output of data mining processes. It provides a declarative approach for defining self-describing data mining models. In consequence it has emerged as an interchange format for prediction models. It provides an interface between producers of models, such as statistical or data mining systems, and consumers of models, such as scoring systems or applications that employ embedded analytics. A PMML document can describe one or more models. According to the specifications, a PMML document consists of the following components:

- Data Dictionary
- Mining Model
- Transformation Dictionary
- Model Statistics

In the following we will give more details about each of the components.

2.5.1.1 Data Dictionary

The data dictionary defines the fields that are the inputs to the models and specifies the type and value range for each field. These definitions are assumed to be independent of specific data sets as used for training or scoring a specific model. A data dictionary can be shared by multiple models, whereas statistics and other information related to the training set is stored within a model.

The data dictionary type system reuses names and semantics of atomic types in XML Schema, defined in XML (2008), and adds three new types that deal with time intervals.

Besides the data types for the fields the data dictionary also specifies additional type information used in the mining process such as: whether the field is categorical, ordinal or continuous, if it allows missing values or not etc.
The values of a categorical field can be organized in a hierarchy. Hierarchies are also known as *taxonomies* or *categorization* graphs. The representation of hierarchies in PMML is based on parent/child relationships (both *is-A* and *part-Of* relationships can be represented). A tabular format is used to provide the data for these relationships. A taxonomy is constructed from a sequence of one or more parent/child tables. The actual values can be stored in external tables or in the PMML document itself. The definitions of the content in PMML are at the moment implemented through the use of PMML extensions and are intended to be a framework which will be more specialized in future versions.

### 2.5.1.2 Mining Model

The mining model is the key concept in PMML’s approach to developing and deploying analytical applications. A mining model is a mapping between an input data instance and an output value, which is built on a set of training instances.

Each PMML document can contain a sequence of mining models such as: association model, clustering model, naive bayes model, neural network, regression model, rule set model, sequence model, support vector machine model, text model, tree model.

If the sequence of models is empty, the PMML document can be used to carry the initial metadata before an actual model is computed, so it is not useful for a consumer.

The structure of a model is expressed through a mining schema, which lists the fields used in the model. These fields are a subset of the fields in the Data Dictionary (a data dictionary is shared among several models). The mining schema contains information that is specific to a certain model, while the data dictionary contains data definitions that do not vary with the model. For example, the Mining Schema specifies the usage type of an attribute, which may be active (an input of the model), predicted (an output of the model), or supplementary (holding descriptive information and ignored by the model). The usage type is tightly correlated with the importance of the mining attribute. This indicator is typically used in predictive models in order to rank fields by their predictive contribution. A value of 1.0 suggests that the target field is directly correlated
to the this field. A value of 0.0 suggests that the field is completely irrelevant. This attribute is useful as it provides a mechanism for representing the results of feature selection. Other mining standards such as JDM (*** (2005)) include algorithms for computing the importance of input fields. The results can be represented by this attribute in PMML. Other aspects that mining schemas capture are the missing values treatment method (as is, as mean, as median, as modal value), invalid value treatment method (as extreme values, return invalid, as is, as missing), outlier treatment (as is, as missing values, as extreme values).

2.5.1.3 Transformation Dictionary

The Transformation Dictionary defines derived fields. Derived fields may be defined by normalization, discretization, value mapping, or aggregation.

Normalization maps continuous or discrete values to numbers. The elements for normalization provide a basic framework for mapping input values to specific value ranges, usually the numeric range $[0, 1]$. Normalization is used, for instance, in neural networks and clustering models.

Discretization maps continuous values to discrete values. This is achieved through specifying a set of intervals and a set of bin values, so that if the value of the field to be discretized lands in the interval $i$, then it is mapped to binValue$_i$.

The mapping is functional, in the sense that two intervals can map to the same bin value, but the intervals need to be disjoint, i.e., a specific field value cannot land in the same interval. The intervals should cover the complete range of input values.

Value mapping maps discrete values to discrete values. Any discrete value can be mapped to any possibly different discrete value by listing the pairs of values. This list is implemented by a table, so it can be given inline by a sequence of XML markups or by a reference to an external table. The same technique is used for a Taxonomy because the tables can become quite large. Different discrete values can be mapped to one value but it is an error if the table entries used for matching are not unique. The value mapping may be partial, i.e., if an input value does not match a value in the mapping table, then the result can be a missing value.
2.5 Interoperability in KDD

Aggregation summarizes or collects groups of values, for example by computing averages. These groups can be defined by an aggregation over sets of input records. The records are grouped together by one of the fields and the values in this grouping field partition the sets of records for an aggregation. This corresponds to the conventional aggregation in SQL with a GROUP BY clause. Input records with missing value in the groupField are simply ignored. This behavior is similar to the aggregate functions in the presence of NULL values in SQL.

The transformations in PMML do not cover the full set of preprocessing functions which may be needed to collect and prepare the data for mining. There are too many variations of preprocessing expressions. Instead, the PMML transformations represent expressions that are created automatically by a mining system.

2.5.1.4 Model Statistics

The Model Statistics component contains basic univariate statistics about the model, such as the minimum, maximum, mean, standard deviation, median, etc., of numerical attributes.

PMML provides a basic framework for representing univariate statistics, which contain statistical information about a single mining field. This information characterizes the data the models were built on and gives guiding information on the type of data the models can be applied on to the potential consumers of the model. Discrete and continuous statistics are possible simultaneously for numeric fields. This may be important if a numeric field has too many discrete values. The statistics can include the most frequent values and also a complete histogram distribution. For instance, it can include frequency of values with respect to their state of being missing, invalid, or valid. In the case of discrete fields, the modal value can be specified, that is the most frequent value, or a table that associates to each value, the number of occurrences. Other statistics include mean, minimum, maximum and standard deviation. In the case of continuous fields, an array defining intervals on the domain and specifying the frequencies, sum of values, and sum of squared values for each interval. Statistics for a subset of records (for example it can describe the population in a cluster) can be provided through the use of partitions. That is, each partition describes the distribution per values of
2.5 Interoperability in KDD

Table 2.1: The PMML support in the data mining and predictive analysis products

<table>
<thead>
<tr>
<th>Application</th>
<th>Import</th>
<th>Export</th>
<th>Scoring</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM DB2 Data Warehouse Edition</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>SAS Enterprise Miner</td>
<td>no</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>SPSS</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Zementis Adapa</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>R</td>
<td>no</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>Salford Systems</td>
<td>no</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>Weka (Pentaho)</td>
<td>yes</td>
<td>in progress</td>
<td>no</td>
</tr>
<tr>
<td>RapidMiner</td>
<td>in progress</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>MicroStrategy</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
</tbody>
</table>

the fields or clusters, i.e. information about frequencies, numeric moments, etc, reflecting the general univariate statistics.

In DeVissa the statistics information provided by the producers of the models add valuable to estimating the similarity of two schema elements which were produced in different contexts, as described in 5.6.

2.5.1.5 PMML Support in Predictive Analysis Products

PMML is becoming the lingua franca for data mining interoperability. There is an increasing number of data mining and predictive analysis packages that consider achieving PMML integration. The status of PMML support in the most important data mining and predictive analysis applications at the moment of writing this material is presented in table 2.1.

As we can see from the table 2.1 there are quite a number of applications that are able to generate PMML but a relatively small number of them are able to import or even less to score. The last two functions involve a full implementation of PMML specification resulting in a very complex system. Importing involves mapping the PMML concepts to the internal model representation of a specific application and therefore being an application and platform specific implementation. On the other side, scoring involves applying directly the PMML
representation of a predictive model to a set of data instances and producing the predictive results. This approach provides a system and platform independent consumption of predictive analysis, resulting in cheaper solutions for the potential consumers.

As a model consumer, it has the added benefit of separating the model definition from the model execution. Using PMML, a consumer can accept models from a wide variety of sources without having to build special drivers for each vendor. More importantly, they don’t have to worry about viruses, malware and other security issues that would prevent most IT organizations from blindly deploying alien code into their data infrastructure. The scoring engine simply parses the XML and scores the models.

PMML is constantly evolving to allow for the representation of more modeling techniques as well as being easier to produce and consume. It aims to cover the most widely used predictive techniques and so it is limited to these techniques. However, it does allow for users to represent data transformations, which provides great flexibility to the standard.
Chapter 3

Knowledge as a Service

In Xu & Zhang (2005) the concept of knowledge as a service is defined as the process in which a knowledge service provider, via its knowledge server, answers queries presented by some knowledge consumers. The knowledge server maintains a set of knowledge models and answers queries based on those knowledge models. For the consumers it may be expensive or impossible to obtain the answers otherwise.

3.1 DeVisa- Architectural Overview

3.1.1 Functional Perspective

The core functionality of DeVisa is provided via the PMQL Web Service (see 3.2) and includes scoring on the existing models, searching for models with given properties, composing models and model statistics. The messages exchanged with the consumer are represented in PMQL, which is described in 3.3. DeVisa also provides an interface to upload/download models through the Admin Web Service.

From the functional perspective, DeVisa presents several services in interaction with two main client roles: data mining producer application and data mining consumer application (Figure 3.1).

The DM (Data Mining) Consumer is a data mining application or a a Web Service client application that interacts with the PMML prediction models stored.
3.1 DeVisa - Architectural Overview

in DeVisa through a Web Service interface.

A DM Consumer also contains one of the following:

- a PMML import module: a module that transforms a PMML model into an internal model in order to further process it into decision or prediction processes (the DM consumer contains a scoring engine itself);

- a module that interprets a scoring result and integrates this result into its business logic (this is the case where the the DM consumer does not include a scoring engine);

The DM Producer is a web service client application - typically a data mining application, like Weka- which uploads models in the DeVisa Repository. If the model that the DM producer has uploaded already exists in the DeVisa repository and the model is newer than the existing one then it is replaced. It might need to be authenticated in order to execute the required function.

The DeVisa system provides its functionality via web services definitions. The core functionality is the PMQL Service. There is also an Admin Service that supports uploading/downloading of models in DeVisa.

The PMQL Service supports SOAP (***(2007b)***)) messaging. The web services are described using WSDL (**WSDL**), written in Java and deployed using Apache Axis framework (**Axis**). The PMQL Service deals with arbitrary XML of the incoming and outgoing SOAP Envelopes without any type mapping / data
binding. This type of web services is called a *message service*. The raw XML in a received SOAP envelope is passed to the PMQL engine, which attempts to interpret the XML as a PMQL query. This approach separates the expression of the consumer’s goal from the concrete service’s choreography.

The Admin Service supports SOAP, XMLRPC(*** (2003a)) and REST-style (Fielding (2000)) protocols.

In the remainder of the section we describe briefly the main functional capabilities available via DeVisa PMQL Service.

**Scoring.** The scoring function applies the stored PMML models in the DeVisa repository on new instances. Depending on the models, there are several types of DeVisa scoring procedures:

- **Classification Scoring.** The scoring method receives a set of instances and classification model specification as input and classifies the instances with respect to the DeVisa models that match the specification.

- **Cluster Scoring.** The scoring method receives a set of instances and a clustering model specification as input and assigns the instances to the most appropriate cluster in each of the matching models.

- **Association rules scoring.** The scoring method receives a set of items (instances) and a association rule model specification as input. For each of the input models it determines all the rules whose antecedent itemset is a subset of the input itemset and returns the consequents of these rules as the inferred itemsets.

**Model Search.** The searching functions allow inspecting the properties of the PMML models in the repository. The search functions conform and therefore are limited to the information that a PMML model can incorporate according to the PMML 3.2 specification. DeVisa provides model searching functions based on specified desired properties, certain schema conformance or full text search in the model repository.

**Model Composition.** It allows the combination of simple models into a single composite PMML model. DeVisa simple models can be combined into more complex ones forming new valid PMML documents. The two ways of achieving
model composition are model selection and model sequencing, both supported by the PMML specification. A client application can specify the models subject to composition and the combination method. DeVisa identifies the specified models and checks them for compatibility. If they are compatible a new valid model is returned to the user/stored in the repository.

*Model Comparison.* When a client application (DM Consumer) wants to compare two models it specifies the models to be compared, through exact references (model IDs) and the comparison type (syntactic or semantic). The syntactic comparison means that two PMML models are compared through a XML differencing approach. The semantic comparison involves a schema compatibility check, a function comparison or a performance comparison using different metrics.

### 3.2 The Main Components

The main architectural components of the DeVisa system are depicted in Figure 3.2.

The *PMML Model Service* is a web service that provides different specialized operations corresponding to the capabilities listed in 3.1.1. It is an abstract computational entity meant to provide access to the aforementioned concrete services. Thus the PMML Model Service receives and returns SOAP messages that contain queries expressed in PMQL (See 3.3). To solve the incoming requests the web service detaches the PMQL fragment to the PMQL Engine.

The *Admin PMML Service* is based on XMLRPC or SOAP protocols and consists of methods for storing and retrieving PMML models. DeVisa redefines the basic SOAP store / retrieve web service with customized PMML features. Therefore, when a model is uploaded in the repository, it is validated against the PMML Schema or by using the XSLT based PMML validation script provided by DMG. Then the model is distributed in the appropriate collection (based on the domain / producer) and the catalog is updated with the new model’s metadata. Also the service provides features for updating/replacing an existing model with a newer one via XUpdate (XUpdate) instructions.
3.2 The Main Components

The PMML Model Repository is a collection of models stored in PMML format that uses the native XML storage features provided by the underlying XML database system - DeVisa uses eXist (eXist) for this purpose. A PMML document contains one or more models that share the same schema. The models are organized in collections (corresponding to domains) and identified via XML namespace facilities (connecting to the producer application). The documents in the repository are indexed for fast retrieval (structured indexes, full-text indexes and range indexes).

The PMQL-LIB module is a collection of functions entirely written in XQuery for the purpose of PMML querying. The functions in the PMQL-LIB module are called by the PMQL engine during the query plan execution phase (See 3.3.2.4). A scoring function has a PMQL query plan as input and produces a PMQL query answer.

The PMQL Engine is a component of the DeVisa system that processes and executes a query expressed in PMQL (See 3.3). After syntactic and semantic validation, query rewriting, it executes the query plan by invoking functions in
3.2 The Main Components

The Metadata Catalog contains metadata about the PMML models stored in a specific XML format. The catalog XML Schema can be found in Appendix B. The catalog consists of the following type of information: available collections, model schema, model information (algorithm, producer application, upload date), statistics (e.g., univariate statistics: mean, minimum, maximum, standard deviation, different frequencies), model performance (e.g., precision, accuracy, sensitivity, misclassification rate, complexity), etc. This component is a materialized view on the PMML repository containing information on the PMML models in the repository. In DeVisa the Metadata Catalog is strongly dependent on the underlying XML database indexing system so that the performance of the retrieval process is influenced by the active configuration in a particular database instance. The relationships that the Metadata Catalog captures are depicted in Figure 3.3.

Each entity represented in the DeVisa Metadata Catalog can specify imported domain ontologies as the terminology to define the concepts used in the models. These semantic annotations are used in the model discovery phase (4.2.1) for resolving the ongoing request against the formal vocabulary used by the models. Shared domain ontologies is the most reliable and accurate mean to achieve schema mediation within the model discovery. The alternative mean is the schema mediation technique described in more detail in 5 and 4.2.1.2.

Because the PMML models can potentially originate from different PMML producer applications, the data dictionaries can be semantically heterogeneous, even if they are part of the same domain. Therefore the Catalog contains a
3.2 The Main Components

component called Global Data Dictionary (GDD), whose role is to provide unified view on the data dictionaries.

The GDD acts like a mediator for the existing data dictionaries in DeVisa. The mediator architecture conforms to a simplified GAV model, which is briefly reviewed in 5.2. The implementation of the Global as View (GAV) mediator architecture is depicted in Figure 3.4.

![Figure 3.4: The Global Data Dictionary as a mediator over the data dictionaries in DeVisa.](image)

The mediation occurs only at the semantic level, because both the global data dictionary and the local ones are represented in the same XML schema. Therefore the wrapper components are lightweight, their function resuming to seamlessly mapping an entity in the global dictionary to the corresponding local one during the PMQL query processing.

The GDD provides a matching between the fields in the data dictionaries, including the derived fields in the transformation dictionaries. For each attribute the matching is annotated with the confidence measure, meaning the degree of similarity between the respective fields. An example of such a matching is depicted in Figure 3.5.

It is incrementally updated at the model’s upload time, when the newly uploaded data dictionary is merged into the existing GDD. If in the matching process
3.2 The Main Components

Figure 3.5: An example of matching between the Global Data Dictionary and the individual data dictionaries. Within a data dictionary mining schemas can overlap. For instance, in $DD_1$, field $C$ is part of two mining schemas. The field matches are annotated with the confidence factor. A field can match only one entry in the GDD with confidence factor 1.

two or more attributes are found similar (match confidence is 1) then the GDD retains only one entry for all the similar attributes. If the match confidence is less than 1, GDD keeps all the attributes and annotates the matching with the confidence factor. From this we can deduce that for any attribute in any data dictionary there is exactly one GDD entry for which the match confidence is 1 and there can be more GDD entries for which the confidence factor is less than 1. The similarity of the fields is calculated using the technique described in 5.6 and memoization is used to store the results.

We observe from the algorithm 3.2.1 that two fields are considered to be identical if the similarity between them is exactly 1, which can occur in three different situations: when a field $A \in \mathcal{D}$ is copied in the $\mathcal{D}_g$; when the two fields refer the same term in a shared ontology; when the calculated similarity is higher that $c_{\text{max}}$, which is a tuning parameter that in general has a value close to 1. The last situation is necessary for very high similarity degrees that reflect irrelevant
Algorithm 3.2.1 The incremental update of the GDD when a new data dictionary is uploaded in DeVisa.

**Input:** $\mathcal{D}$ the data dictionary that is uploaded in DeVisa

$\mathcal{D}_g$ the existing global data dictionary as a view on the data dictionaries in DeVisa

**Output:** An updated $\mathcal{D}_g$ view to integrate $\mathcal{D}$

for all $A \in \mathcal{D}$ do

$added \leftarrow false$

for all $A_g \in \mathcal{D}_g$ do

if $ontologyTerm(A) = ontologyTerm(A_g)$ then

$added = true$

$sim[A, A_g] = 1$

else

$sim[A, A_g] = computeSimilarity(A, A_g)$

if $sim[A, A_g] \geq c_{max}$ then

$added = true$

$sim[A, A_g] = 1$

else

if $sim[A, A_g] \leq c$ then

$sim[A, A_g] \leftarrow 0$

if $added = false$ then

$\mathcal{D}_g \leftarrow \mathcal{D}_g \cup \{A\}'$

$sim[A, A'] = 1$
differences between the respective fields. For instance, if \( c_{\text{max}} = 0.99 \) and we consider the fields microRNA and miRNA, representing the same entity and having a similarity degree of 0.9939, then DeVisa memorizes the value 1. More details on the computation of the similarity are given in 5.6.

The incremental update of the GDD maintains an up to date view that can be queried during the online scoring, therefore decreasing the complexity of the matching algorithm. The GDD is used only when in the match schema case, when neither the schema nor the data dictionary are known. The GDD layer provides an uniform pool of matching candidates, along with the pointers to the actual fields.

The matching algorithm used in the incremental update of the GDD is fairly close to the one used in the model discovery phase of the scoring. In the first case the match is done between the existing GDD and the newly added data dictionary and in the second case the matching is done between the consumer’s schema and the GDD. On the contrary, the matching is used to produce a rather different outcome. In the first case the outcome is the modified GDD to reflect the new data dictionary, while in the second case the outcome is the transformation of the consumer’s schema into the schema corresponding to the best matching.

### 3.3 PMQL - A PMML Query Language

For the purpose of interacting with PMML documents DeVisa defines a language called PMQL (Predictive Model Query Language). PMQL is a query language with an XML-based syntax. A PMQL query is executed against the DeVisa repository of prediction models stored in PMML format. The result of a PMQL query is also expressed in PMQL syntax. The PMQL schema and specifications can be found in DeVisa. A consumer application wraps a PMQL query in a SOAP message that invokes a service method and receives the answer as a PMQL fragment wrapped in the SOAP answer.

DeVisa defines a XML-based language called PMQL (Predictive Model Query Language) PMQL that is used to realize both the communication with the consumer application and the internal communication between DeVisa components. The consumer application expresses its goal in PMQL and wraps it in a SOAP
A message that is sent to the PMQL Web Service. The PMQL Web Service forwards the PMQL goal to a DeVisa component called PMQL engine, which is responsible with processing PMQL. It transforms the goal so that it matches the existing DeVisa resources and, after successful matching, computes the answer - also expressed in PMQL - that is transferred back to the consumer. During processing the consumer’s goal is called query. The process is depicted in Figure 3.6.

Depending on the task, a PMQL query specifies the target models it is based on, such as the schema, the instances to predict etc. Below a simple PMQL query fragment containing a classification task is presented. The full example is listed in the PMQL use cases (DeVisa).

```xml
<pmql>
    <request>
        <score function="classification">
            <modelSpec>
                <schema>
                    <field id="01" name="MFE"
                        opttype="continuous"
                        datatype="xs:float" usage="active">
                        <description>
                            Minimal free energy
                        </description>
                    </field>
                    <field id="02" name="BN"
                        opttype="continuous"
                        datatype="xs:int" usage="active">
                        <description>
                            Number of Bulges
                        </description>
                    </field>
                </schema>
            </modelSpec>
        </score>
        <data>
            <instance>
                <field idref="#01" value="-20"/>
                <field idref="#02" value="5"/>
            </instance>
        </data>
    </request>
</pmql>
```

To resolve a scoring or composition goal, the PMQL engine performs the a sequence of steps shortly described below. The first step is annotation, which
3.3 PMQL - A PMML Query Language

refers to checking the PMQL syntax and the integrity of the references to entities in DeVisa. The rewriting identifies the desired task (scoring, composing etc) and retrieves all the models that satisfy the query requirements. Within the plan building phase an execution plan that involves specific models and XQuery (XQuery) functions (for scoring or retrieving the models) is constructed. If an implicit or explicit composition is necessary then an additional composing phase, which assembles the base models, is performed. The execution invokes the necessary XQuery functions against the repository of PMML models.

![Figure 3.6: Resolving consumers’ goals in DeVisa.]

3.3.1 The Structure of a PMQL Goal

PMQL can express queries that correspond to core DeVisa functionality (See 3.1.1). In the case of a scoring goal, the PMQL query specifies the target models
it is to be executed on and the instances to predict. The target models are
described in a query via a model specification element. A model specification is
a set of restrictions used to identify a subset of models stored in DeVisa.

There are three ways in which a DM consumer application specifies one or
more models in DeVisa.

1. **Exact Model.** The query refers a model in the DeVisa catalog by its unique
   name (through a XML reference). The DeVisa engine will use the specified
   model to execute the scoring task.

2. **Exact Schema.** The query refers a mining schema (and therefore a data dic-
   tionary) but does not refer a model. Nevertheless it can specify additional
   desired properties through the use of filters. DeVisa engine will select the
   appropriate model/models that conform to that particular schema. In its
   current version PMML uniquely associates a schema to a model, but we
   can foresee cases in which this association is not strict, i.e there are more
   models that apply to the same schema and therefore the applications can
   take advantage of this.

3. **Match Schema.** The query describes the required mining schema (the in-
   stances to be scored conform to the mining schema). It can also specify
   filters as in the previous case. In this case the DeVisa engine is responsible
   for identifying the matching schema and the appropriate models (should
   these exist). Moreover, the query can refer imported ontologies to explain
   the requested mining schema, which needs to relate with the ones in DeVisa
   Catalog. In this case the model is considered to be laxly specified. There
   are two possible degrees of specificity: the data dictionary is common to the
   goal and DeVisa or it needs to be matched. Hence we identify two possible
   sub-cases:

   - **Exact Data Dictionary.** In this case the goal specifies a data dictionary
     in DeVisa and specify a schema within that specific data dictionary.
     This approach simplifies the model discovery phase because the search
     is limited within a known vocabulary. The engine will try to match
     all the DeVisa schemas with the one specified by the consumer. If
composition is allowed, even if there is no schema that matches exactly the one specified by the consumer, there is a chance that a model conforming to the consumer’s schema can be built on the fly (see 4.3).

- **Match Data Dictionary.** This case adds another layer of complexity atop the previous case, in the sense that the engine first needs to find a data dictionary in Devisa that matches the one specified by the consumer. The names do not necessarily need to coincide, unless this is explicitly requested in the consumer’s goal. For each of the matching data dictionaries, the previous step is applied in order to identify the corresponding models.

As stated above, in the case of a scoring task, the PMQL query needs also to specify the instances to score. The set of instances need to conform to the specified mining schema.

### 3.3.2 PMQL Query Processing

The DM Consumer sends a SOAP message that wraps the request for a scoring task expressed in PMQL. The PMQL scoring query is transmitted to the component responsible with processing and resolving PMQL requests, which is called **PMQL engine**. The internal mechanism of the PMQL engine is presented in Figure 3.7.

The phases in which the PMQL engine executes a scoring query are:

#### 3.3.2.1 Annotation

This phase includes the PMQL syntactic validity check as well as the integrity of XML references. XML references occur in a PMQL query when the consumer specifies exactly the models to score on, or the mining schema of the model to score on. This phase involves querying the Metatada Catalog (Figure 3.7).

The semantic checking involves a look-up in the repository catalog for a model that satisfies the query constraints (function type, accuracy, complexity, freshness) and whose schema matches the query schema. The look-up can be strict
(exact matching of the schemas) or lax (compatible schemas). If the lax look-up is enabled then the schemas need only to be compatible with respect to the number of fields and field types. Should this be the case the subsequent phase performs the actual mapping between the fields of the two schemas.

If several models fulfill the query requirements, a sequence of model references is returned. If no model entry is complying with the requested properties found in the catalog, a null reference is returned and the execution of the current query stops.

### 3.3.2.2 Rewriting

The rewriting phase treats the case in which the models are laxly specified. In the exact schema case the model references applied to the schema are retrieved and replaced in the PMQL query. In the match schema case, the engine looks in the Metadata Catalog for a DeVisa mining schema that matches the specified one. As a general rule, rewriting refers to reinterpreting the PMQL requirements in terms of the DeVisa Metadata Catalog and replacing the model requirements in the
query with concrete model references in the PMML model repository. Annotation and rewriting are also called model discovery and the schema rewriting technique is described formally in 4.2.1.

This phase involves solving the schema differences between the query and the sequence of models returned in the first phase, provided that the look-up procedure is lax and there is one or more predictive models satisfying the request. Basically the goal is to rewrite the query in terms of the matching models. The possible dissimilarities can refer to the names, the types or the mining properties of the fields. A subset of type differences can be resolved by applying the allowed type conversions as specified in the XMLSchema.

DeVisa uses the following strategy in resolving the field dissimilarities.

In the first phase the engine establishes all the candidate matches between the fields in the consumer’s schema and the fields in the GDD. The matches are annotated with the degree of confidence. The notion of a match refers strictly to similarity, therefore the confidence factor measures how similar two fields are. The similarity is calculated based on name, textual description of the fields, field statistics, and the values in the consumer’s data (see 5.6).

In the second phase the matches are translated to the actual data dictionaries based on the existing GDD view, so it obtains the actual matches between the fields in the consumer’s schema and the ones in the dictionaries along with the confidence factors. For each field there are several potential matches, each of them annotated with a confidence factor.

In the third phase, it attempts to build a schema mapping by trying to find the best mapping (with the best confidence factor, calculated as the sum of the confidence factors of the individual field mappings) to a data dictionary and a mining schema. So the resulting matching needs to map consumer’s schema into one and only data dictionary. This cannot always be achieved, since it is possible that the individual field mappings span through several data dictionaries and a model that can be applied to the schema does not exist. If several such mappings exist, the one with the highest confidence factor is selected.

The detailed algorithm is presented in 4.2.1.2.

On the other side, the consumer’s goal can specify if the composition of models is allowed. In this is the case, if a mapping within the same data dictionary can
be found, then the composer component can attempt to build a new model based on existing ones in case no model exists on the respective schema.

The outcome of this phase is a rewritten PMQL query (or a sequence of queries) which is (are) reinterpreted and transformed with respect to the internal schema of the matching PMML model (or models) in the DeVisa repository.

3.3.2.3 Plan Building

This phase builds the PMQL query plan from a rewritten PMQL document. It reorganizes the document so that each model reference is associated with the appropriate instances to score. If the DeVisa schema specifies a transformation dictionary for certain attributes in a model, then the values of the attributes in the instances are mapped into another domain according to it. The transformations applied to the instances must be equivalent to the transformations applied when the model was created. DeVisa implements the transformations through the use of XQuery extension mechanisms available in eXist. The query plan is an XML document that can be directly passed to the DeVisa XQuery library in the execution phase.

3.3.2.4 Plan Execution

For each of the models in the PMML model repository that were identified in the rewriting phase it executes the appropriate XQuery scoring function stored in the DeVisa library. The scoring result is added to the PMQL document (that is, the scoring result is paired with the instance). Expressing the scoring algorithms on the PMML models in XQuery leverages the underlying XML database query optimization.
3.4 Predictive Models Composition

3.4.1 Methods and Challenges In Composing Data Mining models

In general the composition of prediction models can be realized in various ways. They all have in common the goal of making predictions more reliable. Composing several prediction models means merging the various outputs into a single prediction.

Several machine learning techniques do this by learning an ensemble of models and using them in combination. The most popular are bagging, boosting and stacking (a brief overview over the major methods can be found in Ian H. Witten (2005)).

Bagging (Breiman (1996)) applies an unweighted voting scheme on the outcomes of different classifiers built on possibly different data sets (which can be obtained by resampling the original data set). In the case of numeric prediction, instead of voting on the outcome, the individual predictions, being real numbers, are averaged. The component models are usually built separately.

Boosting (Freund & E. Schapire (1997)) also uses voting or averaging schemes to combine the outcomes, but, unlike bagging, it uses weighting to give more influence to the more successful models. Furthermore the process is iterative, each component model is built upon the previous model and therefore influenced by their performance.

Stacking (Wolpert (1992)) applies on heterogeneous classifiers and trains a new meta learner on the predictions of the component classifiers using a validation data set. The meta learner can be of various types depending on the set of attributes used for meta learning. Some meta-learners use only the class predictions of the component models for training, while others use both the class predictions and all the original input attributes. It applies both to categorical and numeric predictions.

Another approach which is mostly useful in distributed data mining is combining models with various levels of granularity. For instance, it might be the case that a model classifies an instance at a coarse level, while another model
does it with a finer granularity. One can use the first model to tag the instances with a more general class and then to use a specialized classifier for each of the resulted groups. This technique is also useful in classification when an algorithm cannot predict multi-class attributes, such as standard SVM. Another direct use of this technique is model selection in PMML, which is described in 3.4.2.

In the distributed data mining model there are situations when different models are built on vertically fragmented data (usually that reside at different sites). Each of the individual models is built on projections on the same relation, but unable to detect cross-site correlations. A meta-learning approach has been proposed in Prodromidis et al. (2000) that uses classifiers trained at different sites to develop a global classifier. DeVisa addresses vertically partitioned data only in certain configurations, as described in 3.4.3.

Another possibility to combine data mining models is to adjust a given model to a consumer’s specific needs. This is called model customization. This is the situation in which a model consumer repeatedly uses the model in its knowledge integration processes and may collect information on its performance. A new model can be constructed based on the specific needs for the consumer. The producer’s data can be combined with the consumer’s data to build a new customized model. The same principle applies to refreshing a model to reflect the new trends in the data; thus either a new model is built of the new data, or the new data is merged with the old one to adjust the model. This is related to the concept of incremental data mining, which was introduced in M.Harries et al. (1998). A related approach can be found in Kuncheva (2004), where strategies for building ensembles of classifiers in non-stationary environments in which even the classification task may change are presented.

DeVisa only stores the prediction models, not the original training data they were built on. Therefore we consider the training data as not being available. Subsequently the model composition in DeVisa is limited to certain techniques which are described in 3.4.3. However the consumer application can build new models based on the existing DeVisa models and its own validation set.
3.4 Predictive Models Composition

3.4.2 Model Composition in PMML

3.4.2.1 Model sequencing

Model sequencing is the process through which two or more models are combined into a sequence where the results of one model are used as input in another model. Model sequencing is supported partially by the PMML specification (PMML).

Examples of sequencing include:

- The missing values in a regression model can be replaced by a set of rules (or decision tree)
- Several classification models with the same target value can be merged via a voting scheme, i.e. the final classification result can be computed as an average of the results of the initial classifiers. The average can be computed by a regression model.
- Prediction results may have to be combined with a cost or profit matrix before a decision can be derived. A mailing campaign model may use tree classification to determine response probabilities per customer and channel. The cost matrix can be appended as a regression model that applies cost weighting factors to different channels, e.g., high cost for phone and low cost for email. The final decision is then based on the outcome of the regression model.

3.4.2.2 Model selection

Model selection is the process of combining multiple 'embedded models', such as model expressions, into the decision logic that selects one of the models depending on the current input values. For instance, a common method for optimizing prediction models is the combination of segmentation and regression. Data are grouped into segments and for each segment there may be different regression equations. If the segmentation can be expressed by decision rules then this kind of segment based regression can be implemented by a decision tree where any leaf node in the tree can contain an embedded regression model.
PMML version 3.2 supports the combination of decision trees and simple regression models. More general variants would be possible and may be defined in future versions of PMML.

In PMML model composition uses three syntactical concepts:

- The essential elements of a predictive model are captured in elements that can be included in other models,
- Embedded models can define new fields, similar to derived fields and
- The leaf nodes in a decision tree can contain another predictive model.

### 3.4.3 Model Composition in DeVisa

DeVisa supports the two types of composition methods described in the PMML specifications: *model sequencing* and *model selection*. As seen in 3.4.2 the current version of PMML supports the combination of decision trees or rules and simple regression models.

Model sequencing is the case in which two or more models are combined into a sequence where the results of one model are used as input in another model. In DeVisa it is very often an intrinsic part of a model, namely a transformation function, as formalized in 4.3. For instance, a supervised discretization algorithm is applied to a certain attribute and this algorithm is described within a transformation dictionary. Another common way to use sequencing is to fill missing values for an attribute if the model can be applied only on attributes without missing values. A transformation function made of decision rules is applied on the values of the respective attribute.

Model selection is when one of many models can be selected based on decision rules.

A common model selection method for optimizing prediction models is the combination of segmentation and regression. The data set is separated into segments and for each segment a regression equation is applied (segment based regression). It can be implemented in PMML by a decision tree (describing the segmentation rules) where any leaf node in the tree can contain an embedded
3.4 Predictive Models Composition

regression model. This scenario can be seen as an instance of composition of models with various levels of granularity.

The producer applications can upload composite models in DeVisa, as long as the models are expressed in well formatted PMML. In this section we focus moreover on the situations in which DeVisa is responsible for composing the models.

Depending on the moment when the composition process occurs, we can further classify the composition in DeVisa in *implicit* or *explicit* composition (Gorea (2008d)). All types of model composition occur within the same DeVisa data dictionary. There are multiple challenges and limitations in performing an accurate and flexible model composition. The most important of these is schema difference between the consumer’s goal and the DeVisa catalog. The difference manifests itself either at the level of vocabularies or at the data granularity level.

### 3.4.3.1 Implicit Composition

Implicit composition is the situation when the models are composed within the orchestration of the scoring or search service (see 3.1.1).

### 3.4.3.2 Explicit Composition

Explicit Composition is when a DM consumer explicitly specifies in its goal that a composition is desired. The composition query usually includes the models to be composed, the composition method and a validation data set. DeVisa identifies the specified models and checks them for compatibility. If all the prerequisites for the composition are fulfilled then a new valid model is returned to the user/stored in the repository.

An explicit composition scenario occurs when a consumer wants to make the best out of several heterogeneous models in DeVisa complying to the same mining schema.
Chapter 4

DeVisa - Formal Model

4.1 Overview of DeVisa Concepts

A DeVisa data dictionary is a tuple $\mathcal{D} = (\mathcal{U}, \mathcal{S}, \mathcal{I}, \mathcal{F})$ where:

1. $\mathcal{U}$ is a finite set of attributes $\mathcal{U} = \{A_1, A_2, \ldots, A_n\}, \forall A_i, i = 1, \ldots, n, \exists \text{dom}(A_i)$, called the domain of $A_i$. Within a data dictionary, the names of the attributes must be unique.

2. $\mathcal{S}$ is a finite multi-set of mining schemas $\mathcal{S} = \{S_1, S_2, \ldots, S_m\}$, where $S_i = (U_i, P_i), U_i \subseteq \mathcal{U}$, and $P_i$ is a set of properties related to the attributes in $U_i, i = 1, \ldots, m$.

Given a mining schema $S = (U, P)$ and $A \in U$, $P(A)$ signifies the subset of $P$ with the properties applicable to the attribute $A$.

The properties of an attribute refer to additional constraints besides the ones in the data dictionary that usually refer to the behavior of the data mining algorithm when handling the values of that particular attribute (e.g independent attribute, predicted attribute, or excluded). Examples of attribute properties are $\text{usage}(A) = \text{predicted}, \text{missingValueTreatment}(A) = \text{asIs}$ or $\text{outliers}(A) = \text{asExtremeValues}$.

We consider a multi-set of mining schemas to ensure compatibility with the PMML specification.
3. $\mathcal{T}$ is a sequence of transformation functions applicable to the attributes. $\mathcal{T} = (T_1, T_2, \ldots, T_m)$. The transformation functions are used in order to map user data to values that are easier to use in specific models. DeVisa allows various types of data transformations that are supported by the PMML specification: normalization, discretization, value mapping, function, aggregation. There are three types of transformations:

(a) Single attribute transformations. For each $A_i, i = 1, n$ we consider $T_i: \text{dom}(A_i) \rightarrow \text{dom}(B_i)$. $B_i$ is called a derived attribute and is further referred in the mining models. For convenience we assume the identity function for the attributes that do not define transformations. Examples of such transformations are: normalization, discretization, value mapping.

(b) Multiple attribute transformations. $T_i: \text{dom}(A_{i_1}) \times \cdots \times \text{dom}(A_{i_k}) \rightarrow \text{dom}(B)$. $B$ is also called derived attribute. This type of transformation includes functions and for each tuple in the relation the value of the function is calculated and a new relation is built considering the transformation.

(c) Multiple tuple transformations. This type of transformations are called aggregations and are similar to the group functions in SQL, such as: count, sum, average, min, max or multiset.

The transformations are applied to the data set prior to the model itself in the order they were presented, i.e. the single attribute transformations, followed by the multiple attribute transformations and the multiple tuple transformations.

4. $\mathcal{F}$ is a set of constraints. $\mathcal{F}$ corresponds to the structural relationships, constraints and rules expressed via ontologies or taxonomies.

The PMML specification includes a taxonomy element that can describe parent-child relationships between the values of categorical fields used in a data dictionary. In DeVisa the data dictionaries can refer domain ontologies through the use of PMML extension mechanism.
4.1 Overview of DeVisa Concepts

Definition 4.1 The Global Data Dictionary (GDD) is the tuple \( \mathcal{D}_g = (U_g, M) \), where \( U_g \) is the set of entries (attributes) in the GDD, \( M: U_g \rightarrow 2^{U_1 \cup U_2 \cup \ldots \cup U_k} \), \( A \in M(A_g) \) if \( \text{sim}(A_g, A) \geq c \), where \( \text{sim} \) is the similarity function defined in 5.6, and \( U_1, U_2, \ldots, U_k \) are the set of attributes in the DeVisa data dictionaries.

Intuitively, the GDD associates each field in each data dictionary or derived field in any transformation dictionary with one or more entries in the GDD. Each association is annotated with a similarity confidence factor in \([0, 1]\). It is not possible that a field is associated with two or more GDD entries with similarity confidence factor 1. An example of such a matching is presented in Figure 3.5. As it can be observed in the figure, the GDD can be modeled as a edge-weighted bipartite graph. The set of attributes includes the derived attributes in the data dictionaries as well, which are not explicitly mentioned for brevity.

A DeVisa Metadata Catalog is an tuple \( \mathcal{DS} = (\{D_1, D_2, \ldots, D_k\}, D_g) \) where \( D_1, D_2, \ldots, D_k \) are DeVisa data dictionaries and \( D_g \) is the global data dictionary.

Given a mining schema \( S = (U, P) \), with \( U = \{A_1 \ldots A_n\} \) we define a model on \( S \) as a function \( \varphi: \text{dom}(A_{i_1}) \times \text{dom}(A_{i_2}) \times \cdots \times \text{dom}(A_{i_r}) \rightarrow C \), where \( C \) is a set defined as:

1. if \( \varphi \) is a classification or regression model, then \( \exists A \in U \) so that \( C = \text{dom}(A) \)
   - in this case \( A \) is the class (predicted) attribute and \( r = n - 1 \);

2. if \( \varphi \) is a clustering model, then \( C \) is a finite set of cluster tags and \( r = n \).

According to the PMML specifications, each model needs to specify its own mining schema. Although from a theoretical perspective it is possible that several models apply to the same mining schema, we will assume that there is a one to one correspondence between the models and the mining schemas in DeVisa. Therefore the multi-set approach in the above definition.

Given a data dictionary \( \mathcal{D} = (\mathcal{U}, \mathcal{S}, \mathcal{F}) \) we consider a set \( \Psi = \{\varphi_1, \varphi_2, \ldots, \varphi_m\} \) of models applicable to the mining schemas in \( \mathcal{S} \). \( \bigcup_{\mathcal{D}_i \in \mathcal{D_S}} \Psi_i \) is called the DeVisa PMML Repository.

A tuple on \( U \) is a set \( \{A_1: x_1, A_2: x_2, \ldots, A_n: x_n\} \) so that \( x_i \in \text{dom}(A_i) \).

A relation on the data dictionary \( \mathcal{D} = (\mathcal{U}, \mathcal{S}, \mathcal{F}) \) is a set \( \mathcal{R} \) of tuples on \( \mathcal{U} \) that satisfy the restrictions in \( \mathcal{F} \).
A relation (or dataset) on a mining schema $S = (U, P)$ is a set of tuples $R$ that satisfy the restrictions in $F[U]$.

### 4.1.1 The connection between the mining models and functional dependencies

This section introduces the concept of mining functional dependencies and draws a framework in which new mining functional dependencies can be inferred from a data dictionary. We will focus only on classification and regression models.

Given a mining schema $S = (U, P)$, $U = \{A_1 \ldots A_n\}$ in a particular data dictionary and a classification/regression mining model $\varphi$ defined on $S$, we can draw a connection with the functional dependency, i.e $A_1 \ldots A_{n-1} \rightarrow A_n$, where $A_1 \ldots A_n$ are the active attributes and $A_n$ is the predicted attribute (class attribute). The information on whether an attribute is active or predicted is found in the set of properties $P$ associated with each mining schema.

Let us recall the definition of a functional dependency in the relational model. If $X, Y$ are sets of attributes, we say that a relation $R$ satisfies the functional dependency $X \rightarrow Y$ if for any tuple $t_1, t_2 \in R$ so that $t_1[X] = t_2[X]$, it results that $t_1[Y] = t_2[Y]$. The representation of mining schemas using functional dependencies is rather natural, considering that, for a given tuple, the class attribute values are calculated from the active attribute values based on the given model. Subsequently, if two tuples have equal values on the active attributes, the model will deterministically produce the same values for the class attributes - actually the functional dependency is self-fulfilling. We are dealing with a simplified form of functional dependencies, in which the right side is formed by a single attribute (actually any functional dependency can be reduced to this form using the decomposition rule of inference). We will name this type of functional dependency as mining functional dependency (MFD).

**Definition 4.2** Given a mining schema $S = (\{A_1 \ldots A_n\}, P)$ so that usage($A_n$) = predicted, a model $\varphi$: $\text{dom}(A_1) \times \text{dom}(A_2) \times \cdots \times \text{dom}(A_{n-1}) \rightarrow \text{dom}(A_n)$. We will denote a mining functional dependency as $A_1 \ldots A_{n-1} \rightarrow_m A_n$. 

55
If $R$ is a relation defined on $U$, then $R$ satisfies the mining functional dependency $A_1 \ldots A_{n-1} \rightarrow_m A_n$ if

$$\forall t \in R \text{ we have } \varphi(t[A_1 \ldots A_{n-1}]) = t[A_n]$$

**Proposition 4.1** If $R$ satisfies the MFD $A_1 \ldots A_{n-1} \rightarrow_m A_n$, then $R$ satisfies the FD $A_1 \ldots A_{n-1} \rightarrow A_n$.

**Proof.** Let’s consider $t_1, t_2 \in R$ so that $t_1[A_1 \ldots A_{n-1}] = t_2[A_1 \ldots A_{n-1}]$. $R$ satisfies the MFD $A_1 \ldots A_{n-1} \rightarrow_m A_n$, so $\varphi(t_1[A_1 \ldots A_{n-1}]) = t_1[A_n]$ and $\varphi(t_2[A_1 \ldots A_{n-1}]) = t_2[A_n]$. From $\varphi(t_1[A_1 \ldots A_{n-1}]) = \varphi(t_2[A_1 \ldots A_{n-1}])$, it results that $t_1[A_n] = t_2[A_n]$. So $R$ satisfies the FD $A_1 \ldots A_{n-1} \rightarrow A_n$. □

Let us note that the reverse is not necessarily true. From Proposition 4.1 it results that we can apply the functional dependency properties in the case of MFDs and use some of the formalisms present in the FD theory in DeVisa.

The set of properties that can be translated to our representation with mining functional dependency is:

1. Reflexivity. $\forall R$ relation, satisfies $A_1 \ldots A_n \rightarrow A_i$, $\forall i = 1, \ldots n$

2. Extension. If $R$ satisfies $X \rightarrow A$ then $R$ satisfies $X Y \rightarrow A$.

3. Extended Transitivity. If $R$ satisfies $X \rightarrow A$ and $AY \rightarrow B$, then $R$ satisfies $XY \rightarrow B$. If $Y = \Phi$, then we have the classical transitivity.

4. Reverse Projection. If $R[Z]$ satisfies $X \rightarrow A$, then $R$ satisfies $X \rightarrow A$.

We included only the properties that are relevant to our representation.

Using this representation, we can express a data dictionary $\mathcal{D}$ as a set of mining functional dependencies, where each of them is the translation of a mining schema. In practical cases the set of MFDs does not result in integrity constraints on given relations, as in the relational model case. As we mentioned before, the MFDs are self-fulfilling, i.e for a given relation (which is normally part of the consumer’s goal) the model itself will produce precise values for the attribute at the right side of the MFD. So most of the relations that the system interacts with will satisfy an individual MFD. Although we can consider a data dictionary
4.1 Overview of DeVISA Concepts

as a set of MFDs that hold simultaneously in given relations, the approach does not have too much practical relevance. Therefore a statement like a relation $R$ satisfies all the MFDs in the data dictionary $D$, which is very common in the relational theory, will not hold in our approach.

Moreover, DeVISA is a dynamic environment, in which new models with new mining schemas (and therefore new MFDs) can be added any time. The upload of new models would invalidate a potential (previously established) property of satisfiability of a set of MFDs on a given relation.

In somewhat the same category is the notion of consequence, which is central in the relational theory. A functional dependency $S_r$ expressed as $A_1 \ldots A_n \rightarrow B$ is a consequence of $D$ if every relation that satisfies the dependencies in $D$ satisfies also $S_r$. The concept of consequence has the notation $D \models S_r$. In DeVISA the concept of consequence also holds, although in practice it is rarely the case that a relation $R$ needs to satisfy simultaneously all the MFDs in $D$ (so the premise does not hold). Therefore we are moreover interested in weaker premises, such as conveniently chosen subsets of $D$.

In the composition of mining models (see 3.4.3, 4.3) we are interested in deducing a new mining schema that correspond to a given consumer’s goal from a DeVISA data dictionary (or set of MFDs). This problem can be solved using the inference rules in the functional dependency theory.

Armstrong’s axioms (Armstrong (1974)) form a sound and complete set of inference rules that can be also applied to MFDs.

A1 Reflexivity. If $Y \subseteq X$ then $X \rightarrow_m Y$

A2 Decomposition and union.

If $X \rightarrow_m YZ$ then $X \rightarrow_m Y$ and $X \rightarrow_m Z$.

Reversely, if $X \rightarrow_m Y$ and $X \rightarrow_m Z$ then $X \rightarrow_m YZ$.

A3 Transitivity. If $X \rightarrow_m Y$ and $Y \rightarrow_m Z$ then $X \rightarrow_m Z$.

If $\mathcal{A}$ is the Armstrong set of inference rules and $\mathcal{D}$ a set of MFDs, we say that the MFD $X \rightarrow A$ can be derived from $\mathcal{D}$ using rules from $\mathcal{A}$ if $X \rightarrow A$ is
obtainable by means of repeatedly applying the inference rules in \( A \) to functional dependencies in \( D \). We denote the derivation by \( D \vdash_A X \rightarrow A \).

If \( X \) is a set of attributes, then we denote \( X^+_A = \{ A | D \vdash_A X \rightarrow A \} \).

This approach becomes very useful in the query rewriting phase of the scoring process, when a consumer mining schema needs to be matched with a mining schema in the data dictionaries (see 4.2.1). Each potential matching needs to be validated using the functional mining schema approach, which can be reduced to verifying if a certain attribute \( A \) can be determined from a set of attributes \( X \) using the MFDs in \( D \).

### 4.2 The Scoring Process

From a theoretical perspective, the scoring procedure consists of:

1. identification of a subset of models in DeVisa that satisfy the model specifications in the query (filters and matching schemas);
2. scoring each of the models on the dataset specified in the query.

**Definition 4.3** A scoring query is a tuple \((M\text{Spec}, R)\), where

- \( M\text{Spec} \) is the model specification, defined as
  
  \[
  M\text{Spec} ::= M\text{Ref} \mid (\text{Filter}, S\text{Ref} \mid (D\text{Ref}, S \mid S))
  \]

  where \( M\text{Ref} \) is a reference to a model (corresponding to the Exact Model case), \( \text{Filters} \) is a set of filters corresponding to the additional properties that the model needs to conform to, \( S\text{Ref} \) is an reference to a mining schema (Exact Schema case), and \( S \) describes a mining schema (Match Schema case) that needs to be matched against one in the DeVisa Catalog. The cases are described in 3.3.1

- \( R \) is the dataset to score.
4.2 The Scoring Process

4.2.1 Model Discovery

The model discovery problem is that, given a scoring query, the engine needs to find the models that satisfy the query requirements and adjust the query in terms of the found models if it is the case.

Let’s assume for simplicity reasons that the scoring query does not allow model composition (the compose attribute is set to false). The model composition case is described in the subsequent section (4.3). The main challenge is identifying the appropriate models in the Match Schema lax model specification case (see 3.3.1). In this case it is not known if the schema given in the request conforms to a certain DeVisa data dictionary. The engine should check for an appropriate matching schema in all the data dictionaries. DeVisa allows that the fields in the data dictionary (and therefore mining schemas) to refer terms in domain ontologies via XML references. Therefore the match between schema elements can be achieved easily in the case in which the consumer’s goal refers terms in the same ontology.

We are going to describe the model discovery process taking into consideration the consumer’s model specification presented at the beginning of the current section.

4.2.1.1 Exact Model or Exact Schema Case

If the scoring query refers an exact model through $MRef$, then the model discovery resumes to finding the respective model $\varphi$ in the repository and scoring the data set $R$ on $\varphi$.

If the scoring query specifies an exact schema through $SRef$, then the the engine finds all the models complying to the specific mining schema and applies them on $R$. Since The PMML specification defines each mining schema within a specific model, that would reduce this case to the previous one. Our formal model also assumed a 1:1 correspondence between the mining schemas and the mining models. On the other side the data dictionaries contain multi-sets of mining schemas. Therefore DeVisa uses the mining schema equality to extend the search to equal mining schemas within the same data dictionary. Two mining
schematas $S_1 = (U_1, P_1)$ and $S_2 = (U_2, P_2)$, belonging to the same data dictionary, are equal if $U_1 = U_2$ and $P_1 = P_2$.

### 4.2.1.2 Match Schema Case

Here we further distinguish two sub-cases depending on whether a reference to a DeVisa data dictionary is provided in the consumer’s query.

First, if the data dictionary is referred by the query through $DRef$, the lookup is restricted within that specific data dictionary. Secondly, if the data dictionary is not specified in the consumer’s query, the lookup is slightly more complicated, in the sense that a matching mining schema in any of the DeVisa dictionaries needs to be found. The first case is a simplified configuration of the second case.

Therefore in the following we will consider the general case when the data dictionary is not specified. We will provide the general theoretical approach for the model discovery problem and then we will particularize for the first case. So the model discovery problem (without model composition) can be articulated as follows.

Given a set of DeVisa data dictionaries $D_1, D_2, \ldots, D_n$, where $D_i = (U_i, S_i, F_i)$ where $i = 1, n$, the corresponding set of models $\Psi_1, \Psi_2, \ldots, \Psi_n$, a mining schema $S_r$ and $R_r$ a relation on $S_r$, find $\{\varphi_1, \ldots, \varphi_k\}$ so that $\varphi_i, i = 1 \ldots k$ is best applicable on $R_r$. Furthermore, assuming the existence, the engine needs to determine all the models corresponding to the schemas equivalent with all $S_i, i = 1 \ldots k$ in their respective data dictionary.

In order to define the concept of best applicable, we need to provide a set of other definitions.

**Definition 4.4** Given two sets of attributes $U_r$ and $U$ ($U_r$ is the set of attributes to be transformed), $c, s \in [0, 1]$, a match function with support $s$ and confidence $c$ is a partial bijective function $\mu: U_r \to U$ so that $s \cdot |U_r| \leq |\mu(U_r)|$ defined as

\[
\text{if } \mu(A_r) = A \text{ then } \text{sim}(A_r, A) \geq c
\]

The values $c$ and $s$ are tuning parameters, $c$ represents how similar the fields need to be and $s$ is the fraction in which the consumer’s schema needs to be
covered by the matching. Intuitively, the more attributes are matched, the better the matching is.

We consider a partial function as a function in which not every element of the domain has to be associated with an element of the codomain. However, for each of the field elements of the codomain, there is exactly one field element in the domain that maps to it.

**Definition 4.5** A match function $\mu: U_r \rightarrow U$ is valid if $\exists D_i$ data dictionary having the set of attributes $U_i$ so that $U \subseteq U_i$.

**Definition 4.6** Given a consumer’s mining schema $S_r = (U_r, P_r)$, a DeVisa mining schema $S = (U, P)$ and $c, s \in [0, 1]$, we say that $S_r$ matches $S$ with support $s$ and confidence $c$ if both of the following are true:

1. $s|U_r| \leq |U| \leq |U_r|$ and
2. $\exists \mu: U_r \rightarrow U'$, match function with support $s$ and confidence $c$ so that $U \subseteq U'$ and
3. $\forall A_r \in U_r$ and $A \in U$ so that $\mu(A_r) = A$, we have $P_r(\mu(A_r)) \subseteq P(A)$

In another words, a consumer schema matches a DeVisa schema if, for a subset of the fields of size $s \ast U_r$, a similarity measure can be calculated, its value is lower than a given threshold and the requested mining constrains are lesser than the ones in the DeVisa repository. In DeVisa the $\text{sim}$ function between two fields has values in the interval $[0, 1]$ and is calculated using the technique described in 5.6.

**Definition 4.7** Given a model $\varphi$ defined on the mining schema $S$ in the data dictionary $D = (U, S, T, F)$, a consumer’s schema $S_r$, a relation $R_r$ defined on $S_r$ and $s, c \in [0, 1]$, we say that $\varphi$ is applicable on a dataset $R_r$ if:

1. $S_r$ matches $S$ with support $s$ and confidence $c$;
2. $I_S(R_r)$ satisfies the restrictions in $F$, where $I_S(R_r)$ is the interpretation of $R_r$ in terms of the mining schema $S$. 

61
As we see from the previous definitions, $s$ and $c$ are fixed thresholds for deciding if two schemas match or not. For two schemas there can be more match functions with support $s$ and confidence $c$. We are further interested to find the degree of actual confidence with which two schemas match.

**Definition 4.8** The actual confidence of a match function is calculated as the average of the similarity degree between any pair of fields that map to each other:

$$
conf(\mu) = \frac{1}{|U_r|} \sum_{A_r \in U_r'} \text{sim}(A_r, \mu(A_r)),
$$

(4.1)

where $U_r' = \{A_r \in U_r \text{ s.t } \exists A \in U \text{ s.t } \mu(A) = A\}$

To be observed that the actual confidence is proportional with the actual support, since the numerator grows with the number of attributes matched.

**Definition 4.9** Given two mining schemas $S_r$ and $S$ and $\mu_1, \ldots, \mu_t$ the match functions between $S_r$ and $S$, we say that $S_r$ matches $S$ with the actual confidence $conf(S_r, S)$ if

$$
conf(S_r, S) = \max_{i=1 \ldots t} conf(\mu_i)
$$

The match functions with the actual confidence $conf(S_r, S)$ are called optimal match functions.

**Definition 4.10** Given a model $\varphi$ defined on the mining schema $S$, a consumer’s schema $S_r$, a relation $R_r$ defined on $S_r$ and $s, c \in [0, 1]$, we say that $\varphi$ is best applicable on a dataset $R_r$ if:

1. $\varphi$ is applicable on $R_r$

2. $conf(S_r, S) = \max_{S_r \text{ matches } S'} conf(S_r, S')$

Otherwise stated, $S_r$ matches $S$ with the highest actual confidence.

Since we need to find one or more models which are best applicable on a schema, we are always interested in the optimal match functions, i.e the ones that map $S_r$ into schemas in DeVisa with the highest actual confidence.
4.2 The Scoring Process

Definition 4.11  Given $S_r$, a consumer’s schema, the set of optimal match functions for $S_r$ is

$$\{\mu_i | i = 1, \ldots, l\}, \text{ where } \mu_i: U_r \rightarrow U_i \text{ so that } \text{conf} (\mu_i) = \max_{\mu: U_r \rightarrow U} \text{conf} (\mu)$$

where $U$ is a set of attributes in DeVisa.

The optimal match function is to some extent analogous to the Match operator introduced in Rahm & Bernstein (2001) and recalled in 5.3. An overview of the state of the art in schema/ontology mediation and the general context in which this problem is modeled in DeVisa are presented in 5.3.1 and 5.4 respectively.

An algorithmic approach to calculate the optimal match function $\mu$ is described in 5.5. However in this general process of calculating the match function, a few functional considerations need to be taken into account. These are summarized in the isDerivable function called in the algorithm 5.5.3.

It is a decision function on whether a given mining schema obtained from the consumer’s schema through the match function $\mu$ can be derived from the mining schemas in a data dictionary. The result of the decision depends strongly on if composition is allowed by the consumer’s request. In this section we focus on the simple case when composition is not allowed.

We will use the mining functional dependency representation described in 4.1.1. We recall that in this representation a mining schema is abstracted to a functional dependency, whereas data dictionaries are abstracted to sets of functional dependencies.

Before giving the definition let’s denote by $\mu(S_r) = (\mu(U_r), \mu(P_r))$ the mining schema obtained from $S_r$, after applying $\mu$. $\mu(U_r)$ is the co-domain of the matching function as shown in algorithm 5.5.3 and $\mu(P_r)$ is obtained through translating the properties in $P_r$ in terms of the attributes in the co-domain.

Definition 4.12  Given a consumer’s schema $S_r = (U_r, P_r)$, a data dictionary $D_i$ having the set of attributes $U_i$, a valid match function with support $s$ and confidence $c \mu: U_r \rightarrow U'_i$, where $U'_i \subseteq U_i$. Let $\text{Let } X \rightarrow_m A$ be the MFD associated with the mining schema $\mu(S_r)$ and $D_i$ the set of MFDs associated with $D_i$.

We say that $\mu(S_r)$ is derivable without composition from $D_i$ if
4.2 The Scoring Process

1. \( \exists Y \rightarrow_m A \in D_i \) so that \( Y \subseteq X \) and

2. \( s \cdot |U_r| \leq |Y| + 1 \)

In another words, \( \mu(U_r) \) can be obtained by applying in one step the extension inference rule from \( Y \rightarrow_m A \), or using Armstrong’s axioms, in two steps, by applying reflexivity and transitivity. The condition 2 refers to maintaining the support factor \( s \) specified by the consumer, even if we cut off attributes from the rewritten consumer’s schema \( \mu(U_r) \). Checking if \( \mu(U_r) \) is derivable from \( D \) his can be achieved in \( O(n|U_r|) \), where \( n \) is the number of MFDs in \( D \).

**Proposition 4.2** Given \( S_r = (U_r, P_r) \) a consumer’s schema, \( D_i \) a data dictionary. It exists \( \mu: U_r \rightarrow U_i \) a valid match function with confidence \( c \) and support \( s \) so that \( \mu(S_r) \) is derivable without composition in \( D_i \) if and only if \( \exists S \in D_i \) so that \( S_r \) matches \( S \) with confidence \( c \) and support \( s \).

**Proof.** First let us assume that \( \exists \mu \) so that \( \mu(S_r) \) derivable without composition in \( D_i \) and prove that \( \exists S \in D_i \) so that \( S_r \) matches \( S \).

Let \( X \rightarrow_m A \) be the MFD associated with \( \mu(S_r) \). Then \( \exists Y \rightarrow_m A \in D_i \) so that \( Y \subseteq X \) and \( s \cdot |U_r| \leq |Y| + 1 \).

Let \( S = (U, P) \) be the mining schema corresponding to \( Y \rightarrow_m A \). We prove that \( S_r \) matches \( S \) with confidence \( c \) and support \( s \).

\[
U = Y \cup \{A\} \subseteq X \cup \{A\} = \mu(U_r). \mu \text{ match function, so } s|U_r| \leq |\mu(U_r)| \leq |U_r|.
\]
Then it follows that \( |U| \leq |U_r| \).

But \( s \cdot |U_r| \leq |Y| + 1 = |U| \). We proved the first condition in the definition 4.6.

The second condition in definition 4.6 results from \( \mu: U_r \rightarrow X \cup \{A\} \) and \( U = Y \cup \{A\} \subseteq X \cup \{A\} \).

Conversely, let us assume that \( \exists S = (U, P) \in D_i \) so that \( S_r \) matches \( S \) and prove that \( \exists \mu \) so that \( \mu(S_r) \) is derivable without composition in \( D_i \).

From the hypothesis we know that \( \exists \mu: U_r \rightarrow U' \), match function with support \( s \) and confidence \( c \) so that \( U \subseteq U' \). We prove that \( \mu(S_r) \) is derivable in \( D_i \).

Let \( Y \rightarrow_m A \) be the MFD associated with \( S \). Then \( U = Y \cup \{A\} \) and \( X \rightarrow_m A \) be the MFD associated with \( \mu(S_r) \).
From definition 4.6 we have $X \cup \{A\} = \mu(U_r) = U' \subseteq U = Y\{A\}$. It follows that $X \subseteq Y$.

$s|U_r| \leq |U| = |Y \cup \{A\}| = |Y| + 1$. □

From proposition 4.2 it results that in order to find the models that are best applicable on a data set, we need to find the optimal derivable match function (with the maximal actual confidence). The actual confidence of the match function $\mu$, is described in the formula 4.1 given earlier in this chapter. If there are more valid derivable match functions with the same confidence, they are all retained. Finding the optimal match function is described in detail in section 5.5.

**Definition 4.13** Given a data set $R_r$ defined on schema $S_r$, $S_r$ matches $S$ and an optimal match function $\mu: U_r \rightarrow U$, the interpretation of $R_r$ in terms of $S$, denoted $I_S(R_r)$ is obtained by renaming in $R_r$ the attributes with the corresponding names in $S$ and eliminating those attributes that do not have a correspondent in $S$ (projection).

Let’s assume that the previous steps retrieve the list $S_1, S_2, \ldots, S_l$ of DeVisa mining schemas that $S_r$ matches through maximum confidence match functions. In this case we have:

for all $S_i \in \{S_1, S_2, \ldots, S_l\}$ do

Build $I_{S_i}(R_r)$

The subsequent step is the transformation of each $I_{S_i}(R_r)$ according to the transformation dictionary $T_i$ corresponding to the data dictionary that $S_i$ belongs to. This means applying the transformation functions on the values of the affected attributes in the tuples in $I_{S_i}(R_r)$.

for all $S_i \in \{S_1, S_2, \ldots, S_l\}$ do

for all $T \in T_i$ do

$I_{S_i}(R_r) \leftarrow T(I_{S_i}(R_r))$

**Definition 4.14** The query plan is

$$Q = \{(S_1, \varphi_1, I_{S_1}(R)), (S_2, \varphi_2, I_{S_2}(R)), \ldots, (S_l, \varphi_l, I_{S_l}(R))\} \quad (4.2)$$

where $\forall i = 1, \ldots, l$, $S_i$ is a matching schema in DeVisa, $\varphi_i$ is a model defined on $S_i$ and $I_{S_i}(R)$ is the interpretation of the dataset $R$ in terms of the matching schema $S_i$. 

65
4.3 The Composition Process

In the case when the data dictionary is specified in the consumer’s goal, the lookup is restricted to the data dictionary. If \( D_i \) is the data dictionary that was specified, then the model discovery phase means finding a set of optimal match functions \( \mu : U_r \rightarrow U_i \) (they are automatically valid) so that \( \mu(U_i) \) derivable in \( D_i \). The schema matching technique presented in 5.5 is applicable in this case as well, but limiting it of course to only one data dictionary.

The model discovery covers the query rewriting (3.3.2.2) and the plan building (3.3.2.3) in the PMQL query processing.

4.2.2 Scoring

Given a query plan as described in (4.2), the scoring means computing an answer in which each tuple is tagged with the predicted value computed by the corresponding model. The query answer can be expressed as a set \( A = \{A_1, A_2, \ldots, A_l\} \), where:

\[
A_i = \{t : \varphi_i(t) | t \in I_{S_i}(R)\}, i = 1, \ldots, l
\]

Each individual answer \( A_i \) corresponds to one of the identified models.

In the case in which the consumer application wants for instance to verify the quality of prediction of different models, then it is not interested in each individual \( I_{S_i}(R) \). In this case the answer can be expressed as:

\[
A = \{t : \varphi_1(t) : \varphi_2(t), \ldots, \varphi_l(t) | t \in R\}, i = 1, \ldots, l
\]

This step coincides with the plan execution phase (3.3.2.4) in the PMQL processing.

4.3 The Composition Process

4.3.1 Implicit composition

Implicit composition is the situation in which the models are composed within the orchestration of the scoring or search service (see 3.1.1).

As described in 4.2, a scoring query is a tuple \((M_{Spec}, R)\), where
4.3 The Composition Process

- $M_{Spec}$ is the model specification, defined as

$$M_{Spec} ::= M_{Ref} | (Filter, S_{Ref} | (D_{Ref}, S | S))$$

The model specification has several instances:

1. Exact model case, in which exact references to one or more DeVisa model that the consumer wishes to score on is given via $M_{Ref}$.

2. Exact schema case, in which the consumer gives an exact reference to a mining schema $S_{Ref}$ and wishes to score on the models complying to that schema. However, an additional set of filters corresponding to the properties that the model needs to conform to can be specified via the $Filter$ element.

3. Match schema case, in which $S$ describes a mining schema that needs to be matched against one or more in the DeVisa Catalog. To restrict the search, an existing DeVisa data dictionary can be optionally referenced. A reference to an ontology in order to explain the terminology in can be included. Also an optional $Filter$ element can be specified.

- $R$ is the dataset to score.

Scoring with implicit composition follows more or less the same phases of processing as in the case of the simple scoring (see 4.2).

In the exact model case, there is no difference between scoring with model composition or without model composition, because the consumer schema refers an existing model. Allowing composition doesn’t affect the query answer.

The implicit model composition is applicable in two situations, given that the consumer allows scoring on composite models (i.e $compose = true$ is set in the consumer’s request):

1. Late composition. When more models complying to $M_{Spec}$ can be found;

2. Structural Composition. When no model complying to $M_{Spec}$ can be found.
4.3 The Composition Process

4.3.1.1 Late Composition

This situation can span all the model specification instances (exact model, exact schema or match schema). The composition occurs late in the scoring process (therefore the name), i.e. after the original models are found.

In the first two instances, given the models or schema reference, an existing data dictionary is implicit. In the match schema case a mining schema $S_r$ and optionally a reference to a data dictionary in DeVisa, $D$ is provided.

In the exact model case, all the referred models are retrieved. In the exact schema case the engine finds all the models complying to the specified schema. In the match schema case, the engine tries to find one or more models that match $S$. In the latter case the query rewriting phase is executed in a manner similar to the one described in the scoring process (4.2.1.2).

The composition method described below applies to the situation in which more models satisfy the requirements. The models are combined to give the best prediction as follows. The composer component of the engine (see 3.3.2) scores on all the models and then applies a voting procedure (similar to the bagging approach) and returns either the outcome that has the highest vote (in the case of categorical predictions), or the average (in the case of numeric predictions). An example of such an approach is given in Figure 4.1.

Note that if the model composition is not allowed ($compose = false$) the engine builds a query plan and executes it against the model (or models, if several are found) in the repository. Then it returns the predictions given by all the models. This is the classical scoring scenario.

4.3.1.2 Structural Composition

This situation applies only to the match schema case.

The names of the attributes in $S$ should either be among the attribute names in $D$, refer to the same terms in an ontology/taxonomy, or a match function is defined as specified in 4.2.1.2.

Thus we refer to the case when no model applicable on $S_r$ can be found. Then the engine is going to invoke the composer module that attempts to build a new
4.3 The Composition Process

Figure 4.1: An example of late composition of models. The $\text{comb}$ function applies a voting procedure or an average on the outcomes of the individual models.

model from the models complying to the mining schemas existing in the data dictionary $\mathcal{D}$ via composition.

An example of such a scenario can be seen in Figure 4.2, which depicts a sequencing composition. The two DeVisa classification models $\varphi_1, \varphi_2 \in \Phi$ are defined on the schemas $S_1(\{A, B, D, F\}, P_1), S_2(\{E, F, G\}, P_2)$ in the data dictionary $\mathcal{D}$. The scoring query (goal) specifies a mining schema $S_r = (\{A, B, C, D, E, G\}, P)$.

The composer attempts to build a new model by sequencing the existing $\varphi_1$ and $\varphi_2$ models.

Figure 4.2: An example of implicit sequencing of models.

The concept of derivability of a mining schema in a data dictionary changes as opposed to the one described in 5.5.1. The difference is that in this case a mining schema is applicable if a new model can be composed based on the ones in a data dictionary.

In order to formalize this situation, the mining functional dependency approach (see 4.1.1) is the most suitable.
Definition 4.15  Given a consumer’s schema \( S_r = (U_r, P_r) \), a data dictionary \( D_i \) having the set of attributes \( U_i \), a valid match function with support \( s \) and confidence \( c \mu \): \( U_r \rightarrow U'_i \), where \( U'_i \subseteq U_i \). Let \( X \rightarrow_m A \) be the MFD associated with the mining schema \( \mu(S_r) \) and \( D_i \) the set of MFDs associated with \( D_i \).

We say that \( \mu(S_r) \) is derivable with composition from \( D_i \) if \( D_i \vdash A \). 

So the problem of deciding if a mining schema is derivable on a given data dictionary \( D \) can be translated in the mining functional dependency representation as the problem of deciding if a functional dependency \( S_r \) can be inferred from \( D \) using Armstrong’s axioms for MFDs.

Given that the set of Armstrong’s axioms is sound and complete, this is equivalent to the problem of deciding if \( S_r \) is a consequence of a set of functional dependencies \( D \).

Using this representation we translate our example in figure 4.2. Thus the consumer’s schema \( S_r \) is expressed as the MFD \( ABCDEG \rightarrow_m G \) and the data dictionary as the set of MFDs \( D = \{ABD \rightarrow_m F, FE \rightarrow_m G\} \). We need to decide if \( D \vdash A \) \( S_r \), equivalent to \( D \models S_r \). In the example we can easily derive \( S_r \) from \( D \) using Armstrong’s rules in the following sequence:

\[
\begin{align*}
\{\} &\vdash_{A_1} ABDE \rightarrow_m E; \\
\{ABDE \rightarrow_m E,\} &\vdash_{A_2} ABDE \rightarrow_m FE; \\
\{ABDE \rightarrow_m FE, FE \rightarrow_m G\} &\vdash_{A_3} ABDE \rightarrow_m G; \\
\{\} &\vdash_{A_1} ABCDE \rightarrow_m ABDE; \\
\{ABCDE \rightarrow_m ABDE, ABDE \rightarrow_m G\} &\vdash_{A_3} ABCDE \rightarrow_m G, \text{ which is } S_r.
\end{align*}
\]

To resolve this decision problem DeVisa uses the well known Fagin’s result on the equivalence between functional dependencies in relational databases and propositional Horn clauses (Fagin (1977)). According to this theory the problem of deciding if a functional dependency is the consequence of a set of functional dependencies is translated to the problem of deciding if a Horn clause is a logical consequence of a set of Horn clauses in propositional logic.

This can be converted into the problem of satisfiability of propositional Horn clauses. This problem involves checking if a CNF Horn formula is satisfiable. It has been proven that this problem can be solved in linear time (Dowling & Gallier (1984)).
4.3 The Composition Process

The mapping is straightforward. For each attribute $A$ we assume a propositional variable $a$ and for each functional dependency $A_1 \ldots A_n \to B$ we build a Horn clause $a_1 \ldots a_n \Rightarrow b$. More details on how the mapping takes place and the proof of equivalence between the notion of consequence in the functional dependency theory and the logical consequence in propositional logic can be consulted in Felea (1996).

Considering the above notations, Chang’s algorithm (Chang (1976)) can be applied efficiently to solve the decision problem in the context of propositional logic. The approach is described shortly in Algorithm 4.3.1.

Algorithm 4.3.1 Deciding if a Horn clause is a logical consequence of a set of Horn clauses.

**Input:** $\bar{D}$ a set of Horn clauses corresponding to a DeVisa data dictionary formed with atoms in $\bar{U}$, $\bar{S}_r : c_1c_2\ldots c_m \Rightarrow d$ a Horn clause corresponding to the consumer’s schema formed with atoms in $\bar{U}$.

**Output:** true, if $\bar{S}_r$ is a consequence of $\bar{D}$ and False otherwise.

$W \leftarrow \Phi \{W$ is a set of words formed with atoms in $\bar{U}$ and the symbols $\sim$ (negation) and $*$ (concatenation)$\}$

**for all** $a_1a_2\ldots a_n \Rightarrow b \in \bar{D}$ **do**

$W \leftarrow W \cup \{ \sim a_1 * \sim a_2 * \cdots \sim a_n * b \}$

$W \leftarrow W \cup \{c_1, c_2, \ldots c_m, \sim d\}$

$continue \leftarrow true$

**while** continue and $\exists x \in \bar{U} \cap W$ **do**

$continue \leftarrow false$

**for all** $w \in W$ **do**

if $w$ starts with $\sim x$ **then**

Trim the prefix $\sim x*$ from $w$

if $w = \lambda$ **then**

return true \{ if the empty word $\lambda \in W$ \}

else

$continue \leftarrow true$

return false \{ if $\lambda$ was not obtained and there are no more symbols to test \}

It is easy to see that the algorithm must always terminate, and thereby give an
4.3 The Composition Process

answer. That this algorithm gives the correct answer is an immediate consequence of Chang’s theorem on Horn clauses (Chang & Lee (1973)) and Fagin’s equivalence theorem (Fagin (1977)). The algorithm gives a correct answer in linear time in the number of atoms in the clauses in the data dictionary.

In the implicit composition scenario the PMQL engine builds the sequenced model in order to score on the dataset provided in the scoring request. It then stores the model back in the repository for future use.

4.3.2 Explicit Composition

Explicit Composition is when a DM consumer explicitly specifies in its goal that a composition is desired. The composition query usually includes the models to be composed, the composition method and a validation data set. DeVisa identifies the specified models and checks them for compatibility. If all the prerequisites for the composition are fulfilled then a new valid model is returned to the user/stored in the repository.

An explicit composition scenario occurs when a consumer wants to make the best out of several heterogeneous models in DeVisa complying to the same mining schema. In this case it provides a composition goal (query) that be expressed as \( (M\ Spec, R) \), where \( M\ Spec \) has the same signification as in the scoring goal. \( R \) is a relation containing classified instances defined on the same schema as the models satisfying \( M\ Spec \). \( R \) is called a validation set.

DeVisa uses a stacking approach (see 3.4.1) to train a meta-learner \( \varphi \) based on the outcomes of the existing DeVisa base models, i.e the models that satisfy \( M\ Spec \) and the relation \( R \) provided by the consumer. Let’s assume that the base schema is \( S = (U, P) \), where \( U = \{A_1, A_2, \ldots, A_n\} \) and the base classifiers \( \varphi_1, \varphi_2, \ldots, \varphi_m : \text{dom}(A_1) \times \text{dom}(A_2) \times \cdots \times \text{dom}(A_n) \rightarrow \text{dom}(C) \), where \( C \in U \). The meta-learner is a simple decision tree \( \tau : \text{dom}(C)^m \rightarrow \text{dom}(C) \) (by default DeVisa uses ID3), since the main work is done by the base classifiers. The outcome is another classifier \( \varphi : \text{dom}(A_1) \times \text{dom}(A_2) \times \cdots \times \text{dom}(A_n) \rightarrow \text{dom}(C) \), as depicted in Figure 4.3.

This approach has the advantage of fitting the model to the consumer needs and the particularities of its own data. For this reason this method can be seen
4.3 The Composition Process

Figure 4.3: Explicit composition of models in DeVisa through a stacking approach.

as a model customization instance as well.

Note that it is possible that the base models are based on composition as well since DeVisa can store composed models. Nevertheless, they are treated atomically.

Model composition is allowed only within a common data dictionary. Intuitively, a data dictionary refers to a strict domain. In the absence of a reference to a data dictionary in the consumer’s goal, the same results can be achieved upon the availability of a common domain ontology, composed of concepts describing the domain in different abstraction levels, into which URLs are mapped. However, there are cases in which separate pre-existing ontologies are used. DeVisa does not address this aspect. In the Semantic Web SW community this is an intense matter of research and much valuable work is done in the domains of ontology alignment, meaning negotiation or ontology elicitation and application. These techniques are over-viewed in Hepp et al. (2008) and Davies et al. (2006).
Chapter 5

Semantic Integration and the Implications in DeVisa

5.1 Basic Terminology

Semantic integration has been a long-standing challenge for the database community and has nowadays become a prominent area of database research in multiple areas, such as artificial intelligence, e-commerce, bioinformatics, semantic web and ubiquitous computing. The volume of data available online and within organizations has been increasing exponentially in the latest years. The recent growth of the Web 2.0 has fueled the spread of data sharing web applications and created the need of semantic integration of the shared heterogeneous and independent data.

In the Semantic Web (SW), data is annotated using ontologies. Concepts (or classes) in ontologies give meaning to data on the Web. Because ontologies are shared specifications, the same ontologies can be used for the annotation of multiple data sources, like Web pages, collections of XML documents, relational databases, etc. As was argued in Uschold (2000), it is very hard to create standard ontologies because the standardization of a terminology is not feasible, especially in an inter-organizational context. As a result there are many different heterogeneous ontologies on the Semantic Web and in order to enable interoperability between applications on the Semantic Web, mediation is required between different representations (ontologies, schemas) of knowledge in the same domain.
5.1 Basic Terminology

So, schemas and ontologies have a couple of properties in common: provide a vocabulary of terms that describes a domain of interest and constrain the meaning of terms used in the vocabulary.

There are a number of terms that have been coined in the context of mediation: matching, mapping, alignment, relating, merging. With a few differences, they are trying to achieve the same main objective: bringing two or more ontologies/schemas into mutual agreement, by reconciling the differences between them.

In de Bruijn et al. (2006), a more exact hierarchical classification is made. At the high level we can distinguish two types of mediation: mapping and merging.

With ontology mapping, the correspondences between two ontologies are stored separately from the ontologies and thus are not part of the ontologies themselves. The correspondences can be used for, for example, querying heterogeneous knowledge bases using a common interface or transforming data between different representations. The (semi-)automated discovery of such correspondences is called ontology alignment.

More formally, a schema/ontology mapping $M$ is a declarative specification of the semantic overlap between two ontologies OS and OT. This mapping can be one-way (injective) or two-way (bijective). In an injective mapping we specify how to express terms in OT using terms from OS in a way that is not easily invertible.

When performing ontology merging, a new ontology is created which is the union of the source ontologies. The merged ontology captures all the knowledge from the original ontologies. The challenge in ontology merging is to ensure that all correspondences and differences between the ontologies are reflected in the merged ontology.

Both mapping and merging have as input a specification of similarities between the schemas/ontologies, which is called the matching process, detailed in 5.3.
5.2 Applications of Semantic Integration

There is a considerable number of specific problems that have driven research in the field of schema alignment (Doan & Halevy (2005)). One of them is *schema integration*, or integrating multiple schemas into a global one. The integration process requires establishing semantic correspondence - matches - between the component schemas, and then using the matches to merge schema elements (Pottinger & Bernstein (2003)).

In an artificial intelligence setting, this is the problem of integrating independently developed ontologies into a single ontology.

Another very popular topic is *data translation* in between database applications, which is a critical step in building data warehouses and performing data mining activities. Here data originating from different sources must be transformed to conform to a single target schema to enable further data analytical processing (Miller et al. (2000)). Given a data source, one approach (Bernstein & Rahm (2000)) to creating appropriate transformations is to start by finding those elements of the source that are also present in the warehouse -match operation. After an initial mapping is created, the detailed semantics of each source element are examined and transformations that reconcile those semantics with those of the target are created.

Another important class of applications is *data integration* ((Garcia-Molina et al. (1997))). Such an application translates a user query on a global schema, called *mediated schema*, to a set of queries applied on local underlying schemas where the data resides. The queries are executed using wrapper systems attached to the data sources, and the results are combined and translated to conform to the mediated schema.

There are two approaches to specify the mapping between the mediated schema and the source database schemas. One of them is to consider each of the data sources to be a view over the (nonexistent) mediated schema. Formally such an approach is called *Local As View* (LAV). The resulting architecture is very flexible and scalable because it can easily absorb new data sources in the system. Adding a new data source affects only the respective wrapper. In return, query processing is more difficult because the query undergoes a radical process
of rewriting. It is possible that a query on the global schema cannot always be translated on a specific local schema and incomplete answers to client queries can occur. This happens because the number of tuples that can be expressed on the global schema is larger than the one in the sources. On the other side, the Global As View (GAV) approach considers the global schema to be a view on each of the data source schemas. The latter approach has the advantage of the simplicity of query processing, since the queries are expressed in a common view on the sources. The drawback concerns scalability of such a system, because each time a new data source is added, the mediated schema might need to be modified to incorporate the new added source schema. This modification cascades to the potential clients of the system, since the queries that are normally issued on the mediated schema have to be rewritten. So this approach is suitable for systems in which the source databases are not likely to suffer changes.

Another class of applications is the peer management applications, that allow users (peers) to query and retrieve each other’s data. We encounter here an extension of a mediated schema (Aberer (2003)), allowing to create semantic correspondences between each pair of peers.

An important tool that benefits from semantic integration is the field of record linkage, proposed in Dunn (1946). It refers to the task of finding entries that refer to the same entity in two or more files. Record linkage is an appropriate technique when data sets that do not have a unique database key in common need to be joined. A data set that have been through record linkage is said to be linked. The most popular approach is Probabilistic Record Linkage (Jaro (1995)). It can be used to improve data holdings, data collection, quality assessment, dissemination of information, duplicate removal. Important areas of application are: medical practice and research and historical data archives (Winkler (1999), Jaro (1995)). Record linkage is a useful tool when performing data mining tasks, where the data originated from different sources or different organizations.

Recently an important application area that needs ontology mediation has developed - Semantic Web Services (Fensel et al. (2007)). In general, it cannot be assumed that the requester and the provider of a service use the same terminology in their communication and thus mediation is required in order to enable communication between heterogeneous business partners. It is subject to intense
5.3 Strategies in Achieving Schema and Ontology Mapping

Research in the context of achieving the Semantic Web desiderates. One of the promising fields of application of ontologies in Semantic Web (SW, Lee et al. (2001)) is the integration of information from differently organized sources, e.g. to interpret data from one source under the schema of another, or to realize unified querying and reasoning over both information sources.

5.3 Strategies in Achieving Schema and Ontology Mapping

We define matching as the process of discovering similarities between two source schemas or ontologies. The result of a matching operation is a specification of similarities between them. The matching is done through application of the Match operator - defined in Rahm & Bernstein (2001), which takes two schemas (or ontologies) as input and produces a mapping between elements of the two schemas that correspond semantically to each other. Any schema matching or ontology matching algorithm - called matcher - can be used to implement the Match operator.

There are many different algorithms which implement the Match operator. These algorithms can be generally classified along several dimensions.

- Schema types: relational, XML, entity-relationship (ER), OO, IDL etc;
- Metadata representation: XML schema, graph, ER, relational, OO etc;
- Syntactic or semantic matching. The syntactic matches take into account different string matches techniques, such as prefix, suffix, longest common substring, string metrics. One important syntactic method for ontology alignment is the comparison of class and property names of ontologies using string distance metrics. Today quite a number of such metrics exist in literature, such as the Levenshtein distance, LCS (longest common substring) distance, Needleman-Wunsch distance, etc. The semantic techniques take the word senses into account establishing lexical relationships between entities (Miller (1995), Wor (2006)).

78
5.3 Strategies in Achieving Schema and Ontology Mapping

- Rule or learning based techniques. The *rule-based matching* technique acts at the syntactic level and consists of applying rules created manually to realize the schema matching. In general, the rules exploit schema information such as element names, data types, structures, number of subelements, and integrity constraints. The source schema is compared with the target one, and each component in the source schema is matched with a corresponding component in the target schema.

The rule based matching techniques operate only on schemas and do not analyze the data instances. Therefore they are inexpensive from the time and space perspective and are domain independent, although they can be tailored to fit a certain domain.

The rule-based approach has two major drawbacks, both related to the lack of information on the data instances themselves. The data instances can encapsulate a lot of meta-information, such as format, frequencies of words in certain attributes, extreme values, mean, distribution of the values etc. The first drawback is the fact that such valuable information cannot be used in the matching process. Another drawback is that rule-based methods cannot exploit previous matching efforts to assist in the current ones. Thus we can say that systems that rely solely on rule-based techniques are statical and cannot learn from the past to improve the accuracy of the results.

The above reasons have motivated the development of *learning based schema matching*. The solutions have considered a variety of learning techniques and exploited both schema and data information. Most of the actual systems are hybrid in the sense that they analyze the schema from both syntactic and semantic angles and the data instances to establish a more accurate discovery of similarities.

- Match granularity: element-level, structure-level, relationship-level. etc. Match can be performed for individual schema elements, such as attributes, or for combinations of elements, such as complex schema structures. In the second case the schemas/ontologies are viewed as graph-like structures containing terms and their inter-relationships and techniques based on bounded path matching, super/sub-concept rules, children, leaves are applied.
5.3 Strategies in Achieving Schema and Ontology Mapping

- Match cardinality: 1:1, n:1, 1:n, n:m. The match result may connect one element of a schema to one or more elements of the other schema. This can reflect at the instance level as well.

- Schema level or instance level. Schema-level matchers only consider schema information, not instance data. The available information includes the usual properties of schema elements, such as name, description, data type, relationship types (part-of, is-a, etc.), constraints, and schema structure. In general, a matcher will find multiple match candidates. For each candidate, a degree of similarity in the range of \([0, 1]\) is usually calculated and the best candidates are those for whom the degree of similarity exceeds a certain threshold. Instance-level exploit the data instances, which give a precise characterization of the actual contents of schema elements and can give important insight into their meaning. This is especially true when useful schema information is limited.

- Need for auxiliary information. There are matchers that rely not only on the input schemas but also on other metadata, such as dictionaries, global schemas, previous matching decisions and user input.

- The level at which the alignment works: schema, data instances;

- Simple, hybrid or composite. Hybrid matchers directly combine several matching approaches to determine match candidates based on multiple criteria or information sources (e.g. syntactic and semantic). A composite matcher combines the results of several independently executed matchers, including hybrid matchers, which tends to be more flexible than the hybrid matchers.

- Need for manual input: human intensive ranging to fully automatic. Defines the extent in which the user is involved in the matching process. Ideally, any schema mapping tool should function with as less as possible input from the user and ask only for absolutely necessary feedback. In Yan et al. (2001) this issue is tackled by defining a framework that uses data examples as the basis for understanding and refining declarative schema mappings. The
higher the degree of autonomy of the matcher, the lower the number of user interactions and vice-versa. It is difficult to achieve a generic solution that is adaptable to different application domains (although the approach could be controlled by tuning parameters). Alternatively, a user can directly select the matchers to be executed, their execution order and how to combine their results. Such a manual approach is easier to implement and leaves more control to the user.

In Rahm & Bernstein (2001), Sun & Rose (2003), Doan & Halevy (2005), Predoiu et al. (2005) and Davies et al. (2006) surveys containing different strategies to schema/ontology matching and comparative analyses are performed based on a pre-established set of features. To summarize, the taxonomy of the distinguishing properties for the schema/ontology matching problem is presented in Figure 5.1.

5.3.1 Overview of the Existing Approaches

Early work such as TranScm system (Milo & Zohar (1998)) employed very simple rules such as: two schema elements map to each other if they have the same names (or synonyms) and the same number of sub-elements. The schemas are represented in labeled graphs, the edges being labeled with the relationships. The relationship properties (name, optionality, cardinality, transferability) are associated with the nodes. The algorithm performs an element level, 1:1 match, based on plugable rules.

In DIKE (Palopoli et al. (2000)) system the similarity between two schema elements is computed through the similarity of the characteristics of the elements and the similarity of the related elements. It attempts to automatically determine new synonym and inclusion (is-a, hypernym) relationships between objects of different entity-relationship schemas. The algorithms are based on a set of user specified synonym, homonym, and inclusion properties labeled with a numerical confidence factor (between 0 and 1). In order to (probabilistically) derive new synonyms and homonyms and the associated plausibility factors, the authors perform a pairwise comparison of objects in the input schemas by considering
5.3 Strategies in Achieving Schema and Ontology Mapping

Figure 5.1: A taxonomy of the dimensions that characterize the schema/ontology matching strategies (Rahm & Bernstein (2001)).

The similarity properties of their related objects (i.e., their attributes and the is-a and other relationships the objects participate in).

The ARTEMIS system (Castano & Antonellis (1999)) operates on a hybrid relational-OO model that includes the name, data types, and cardinalities of attributes and target object types of attributes that refer to other objects. First it computes affinities in the range 0 to 1 between attributes, which is a match-like step. It then completes the schema integration by clustering attributes based on those affinities and then constructing views based on the clusters. The algorithm computes matches by a weighted sum of name and data type affinity and structural affinity. Name affinity is based on generic and domain-specific thesauri, where each association of two names is a synonym, hypernym, or general relationship, with a fixed affinity for each type of association. Data type affinity is based on a generic table of data type compatibilities. Structural affinity
of two entities is based on the similarity of relationships emanating from those entities. ARTEMIS is used as a component within a heterogeneous database mediator, called MOMIS \cite{Bergamaschi2001}. MOMIS prescribes the use of a global merged ontology for the integration of data sources.

Cupid\cite{Madhavan2001} is a hybrid matcher based on both element- and structure-level matching. It is intended to be generic across data models and has been applied to XML and relational examples. It uses auxiliary information sources for synonyms, abbreviations, and acronyms. Like DIKE, each entry in these auxiliary sources include a plausibility factor in the [0, 1] range. The algorithm has three phases. The first phase does linguistic element-level matching and employs rules that categorize elements based on names, data types, and domains. It parses compound element names into tokens based on delimiters, categorizes them based on their data types and linguistic content, and then calculates a linguistic similarity coefficient between data type and linguistic content compatible pairs of names based on substring matching and auxiliary sources. The second phase transforms the original schema into a tree and then does a bottom-up structure matching, resulting in a structural similarity between pairs of element. This transformation encodes referential constraints into structures that can be matched just like other structures (making Cupid constraint-based). The similarity of two elements at the root of structures is based on their linguistic similarity and the similarity of their leaf sets. If the similarity exceeds a threshold, then their leaf set similarity is incremented. The focus on leaf sets is based on the assumption that much of the information content is represented in leaves and that leaves have less variation between schemas than internal structure. Phase two concludes by calculating a weighted mean of linguistic and structural similarity of pairs of elements. The third phase uses that weighted mean to decide on a mapping.

The LSD (Learning Source Descriptions) system uses machine-learning techniques to match a new data source against a previously determined global schema, producing a 1:1 element-level mapping \cite{Doan2001}. It both matches names at the schema level and trains several instance-based matchers to discover characteristic instance patterns and matching rules that can then be applied to match other data sources to the global schema. Given a new data source, each matcher
5.3 Strategies in Achieving Schema and Ontology Mapping

determines a list of match candidates and a global matcher is used to merge the lists into a combined list of match candidates for each schema element. LSD has also been extended to consider user-supplied domain constraints on the global schema to filter out the previously determined match candidates for improving match accuracy.

The SemInt system (Li et al. (2000)) is a hybrid matcher that uses a neural-network learning approach. For each schema element it builds a "matching signature" based on matching criterions, such as attribute specifications (e.g., data types, scale, the existence of constraints) and statistics of data content (e.g., maximum, minimum, average, and variance). Each signature can be seen as a point in the N-dimensional space, so the Euclidean distance can be used as a degree of similarity between elements and generating match candidates. Then it uses neural networks to determine match candidates. It first clusters the attributes in the first schema based on the Euclidean distance and then assigns each attribute in the second schema to a cluster based on a neural network trained with the already built cluster centers.

The iMAP (Dhamankar et al. (2004)) matches the schemas of two sources by analyzing the description of objects that are found in both sources. It takes a more domain-oriented approach by utilizing matchers that are designed to detect and deal with specific types of data, such as phone numbers. It also has an approach to searching for schema translations for numerical data using equation discovery.

Perkowitz & Etzioni (1995) investigates the problem of automatically learning declarative models of information sources available on the Internet and proposes a system named ILA that learns the meaning of external information by explaining it in terms of internal categories.

Ryutaro et al. (2001) proposes a concept alignment method for concept hierarchies as a solution to this problem, and construct a system called HICAL to evaluate the performance of the method. The proposed method can be used to induce align rules for concept hierarchies and classify information into appropriate categories within another concept hierarchy.

Giunchiglia et al. (2004) presents a 4 step semantic structural approach, called S-Match. It assumes that the conceptual models (schemas, ontologies) are repre-
5.3 Strategies in Achieving Schema and Ontology Mapping

sented as trees. It is based on the notions of concept of a label, which denotes the set of documents (data instances) that one would classify under a label it encodes, and concept at a node, which denotes the set of documents (data instances) that one would classify under a node, given that it has a certain label and that it is in a certain position in a tree. The algorithm takes as input two schemas and computes as output a set of mapping elements in four macro steps: (1) for all labels in the two trees, compute concepts of labels; (2) for all nodes in the two trees, compute concepts at nodes; (3) for all pairs of labels in the two trees, compute semantic relations (e.g., equivalence, more/less general) among the corresponding concepts at labels; (4) for all pairs of nodes in two trees, compute relations among the concepts at nodes.

The GLUE system, presented in Doan et al. (2004), applies machine learning techniques to semi-automatically create semantic mappings. Since taxonomies are the central components of ontologies, the system finds the correspondences among the taxonomies of two given ontologies. So for each concept in a taxonomy, it finds the most similar concept in the other. The similarity is defined in terms of joint probability distribution of the concepts involved. It also uses domain constraints and heuristics, such as ”Two nodes are likely to match if their neighbors also match”. GLUE uses a probabilistic model to combine results of different learners, which makes it a composite matcher.

The COMA++ system (Aumueller et al. (2005)) utilizes a composite approach to combine different match algorithms, supporting schemas and ontologies, e.g. W3C XML Schema (Sperberg-McQueen & Thompson (2008)) and OWL (OWL (2007)). COMA++ is characterized by: the utilization of shared taxonomies, the application of the different match strategies including various forms of reusing previously determined match results and the fragment-based matching, which decomposes a large match problem into smaller problems.

Curino et al. (2007) introduces an ontology mapping and integration tool called X-SOM, which combines several matching techniques by means of a neural network, performing semi-automatic resolution of inconsistencies.
5.4 Implications of the Schema Mapping in DeVisa

Schema matching problem also impacts the scoring process in DeVisa (4.2). Briefly, it occurs when a consumer application would like to score a given dataset having no knowledge if DeVisa contains a data dictionary (or schema) that matches the schema of its dataset. When solving the consumer’s goal containing a laxly specified target DeVisa model, the scoring engine (3.2) is performing a schema matching to identify an appropriate model or models to score on. During this process, it needs to iterate over the existing DeVisa data dictionaries and attempt to perform a mapping.

Furthermore, a query written in terms of the consumer’s application terminology needs to be rewritten using the terms in DeVisa data dictionary - query rewriting (3.3.2.2).

The final purpose of mapping in DeVisa is the consumer’s data instances transformation according to the DeVisa mining schema, so that a mining model can be applied. The mapping should be reversible, in the sense that the answer to a goal should reflect the consumer’s schema.

DeVisa is designed to serve random consumer applications through web services. Therefore the schema mapping process should function without manual intervention. In these circumstances the system should have the possibility to decide, under configurable parameters (degree of similarity, optimistic/pessimistic approach), if to apply a certain mapping or not. A manual validation process can be performed at consumer’s site after a scoring answer based on a set of matches is returned.

In DeVisa schemas correspond to data dictionaries. In a data dictionary one or more mining schemas can be defined, which are considered to be elements in a data dictionary. A mining schema corresponds to a single mining model and specifies additional properties for the attributes that are used in the mining model. Examples of such properties are usage type, importance, missing values treatment etc. A detailed formal description of the DeVisa data model can be consulted in 4.1 and an overview on mining schemas is presented in 2.5.1.2. These
properties are used in the schema matching resolution in DeVisa. For instance, the matching should give higher priority to higher importance attributes.

Data dictionaries are often correlated to taxonomies that describe hierarchical parent-child relationships between the categorical fields. In the context of a mining schema, additional properties for each field can be specified. These properties specify the manner in which the mining model manipulated the specific field: dealing with missing or inconsistent values etc.

Moreover, the transformation dictionary can specify rules and procedures to transform individual or group of fields into derived fields that are used in the mining model instead of the original fields. The transformations can be applied either at single row level or multiple row level (aggregation functions). Its role is to map user data to values that are easier to use in the mining model. The transformation dictionary broadens the spectrum of match candidates, since the consumer data might be expressed using also derived fields. Therefore the matching strategy can be considerably improved if it takes into account potential matches induced by the transformation functions.

On the other side one can imagine the case in only a subset consumer’s attributes are used in the prediction process, namely the maximal subset that can be mapped to a DeVisa data dictionary and for which an adequate mining schema can be deduced (i.e it either exists or can be composed as shown in 4.3). Nevertheless the degree of coverage of the consumer’s set of attributes is a configurable parameter (i.e support).

One of the main characteristics of the DeVisa system is that it does not store the data the models were built on, but it is meant to enable model and prediction interchange between applications. Therefore some of the schema matching strategies (especially instance based strategies) that employ learning techniques cannot be applied. Nevertheless the consumer’s data that needs to be scored on is available and can be analyzed. On the other side a model expressed in PMML can contain univariate statistics on each of the fields used in the mining model (see 2.5.1). Therefore a technique that builds a ”matching signature” in the manner presented in Li et al. (2000) could be applied for certain data dictionaries.

To improve the efficiency of the matching during the model discovery phase of the scoring procedure the preprocessing should be done offline. In this way, each
schema element has the internal matching representation stored along with the corresponding data dictionaries.

Furthermore, or at the moment the data dictionaries are uploaded in the system, a match process is triggered between the newly added data dictionary and the set of data dictionaries existing in the system. This process can identify similarities between the schema elements and sets up incrementally a integrated network of concepts. The algorithm is described in 3.2.1. This setup contributes to the efficiency of the online schema matching and also builds up the prerequisites for the model composition (4.3).

Another aspect worth mentioning is the type of relations that are established during the match process. In DeVisa we are not necessarily interested in mode complex relations like in most of the systems, such as: equivalence (=), more general (⊆), disjointness (⊥), but moreover in degrees of similarity. The question is: can a DeVisa model be applied on this specific attribute? So relations like generalization and specialization are likely to be mapped in lower degrees of similarity.

In summary, the distinguishing properties of the DeVisa conceptual model compared to other schema matching input configurations are:

- entities are not explicitly defined. Intuitively, a mining schema (and, respectively, a model) refers to a specific entity, but it is possible that multiple mining schemas refer to the same entity;

- lack of structural constraints which are normally present in a relational database schema (e.g primary or foreign keys);

- a data dictionary may refer a specific taxonomy; several data dictionaries may refer the same taxonomy;

- optional univariate statistics at the field level;

- optional textual description of the data dictionaries, or for the individual fields;

- optional transformation dictionary;
the type of relationships that are targeted reduces to degrees of similarity

Provided the context and the requirements presented above, the scoring process in DeVisa is a novel problem whose resolution involves strategies used in schema and ontology mapping, mediation and query rewriting. The schema matching strategy used in DeVisa is described formally in 5.5.

5.5 Schema Matching Approach in DeVisa

In DeVisa schema matching is necessary during the model discovery phase in the scoring or search process, namely in the match schema case (see 4.2.1.2). Given a consumer’s schema $S_r$, the matching process means identifying a set of optimal match functions. The optimal match functions for a consumer’s schema is defined in 4.11.

This section provides an algorithmic approach for calculating the set of optimal match functions for a given consumer’s schema. It uses the Global Data Dictionary, which is part of the Metadata Catalog (see 3.2).

Prior to describing the schema matching algorithm, we need to define several concepts.

**Definition 5.1** Given a consumer’s schema $S_r = (U_r, P_r)$ and $D_g$ a DeVisa GDD, $c, s \in [0, 1]$, we define a global matching as a partial function $m: U_r \rightarrow 2^{U_g}$ with support $s$ and confidence $c$ so that both of the following are true:

1. $A_g \in m(A_r)$ if $\text{sim}_{A_r, A_g} \geq c$

2. $|\{A_r | m(A_r) \neq \emptyset\}| \geq s|U_r|$

The maximum number of candidate global matches is:

$$
\left(\frac{|U_r|}{|s|U_r|}\right)^{2^{|U_g|}} + \left(\frac{|U_r|}{|s|U_r| + 1}\right)^{2^{|U_g|}} + \cdots + \left(\frac{|U_r|}{|U_r|}\right)^{2^{|U_g|}}
$$

$$
= 2^{|U_g|} \left(\left(\frac{|U_r|}{|s|U_r|}\right) + \left(\frac{|U_r|}{|s|U_r| + 1}\right) + \cdots + \left(\frac{|U_r|}{|U_r|}\right)\right)
$$

$$
= 2^{|U_r|} \cdot 0 \left(2^{|U_g|} - 1\right) \quad (5.1)
$$

$$
= 0 \left(2^{|U_r| + |U_g|} - 1\right) \quad (5.2)
$$
We estimated the result expressed in relation 5.1 because in practice the support $s$ tends to be close to 1 (however, we assumed the worst case when it is 0.5).

**Definition 5.2** Given a consumer’s schema $S_r = (U_r, P_r)$ and a DeVisa GDD $D_1$, a global matching $m$, we define a local matching as a function $m': U_r \rightarrow 2^{U_1 \cup U_2 \ldots U_n}$, with the following restrictions:

1. $m'(A_r) = \bigcup_{A_g \in m(A_r)} (M(A_g) - \{A | \text{sim}_{A,A} < c\})$
2. $|\{A_r | m(A_r) \neq \emptyset\}| \geq s|U_r|$

A local matching gives an upper bound to the search space of the match functions.

**Proposition 5.1** Given a local matching $m': U_r \rightarrow 2^{U_1 \cup U_2 \ldots U_n}$, any match function $\mu: U_r \rightarrow U$ has the property that $\forall A_r \in U'_r$, we have $\mu(A_r) \in m'(A_r)$ where $U'_r = \{A_r \in U_r \ s.t \ \exists A \in U \ s.t \ \mu(A_r) = A\}$.

**Proof.** Let $\mu: U_r \rightarrow U$ be an arbitrarily chosen match function. Then let’s consider $A_r \in U'_r$ so that $\mu(A_r) = A$, and therefore $\text{sim}_{A,A} \geq c$. We need to prove that $A \in m'(A_r)$.

But $m'(A_r) = \bigcup_{A_g \in m(A_r)} (M(A_g) - \{A | \text{sim}_{A,A} < c\})$, so we need to prove that $A \in \bigcup_{A_g \in m(A_r)} (M(A_g))$. That reduces to proving that $\exists A_g \in m(A_r)$ so that $A \in M(A_g)$. From definition 4.1 it follows that $A \in M(A_g)$ if $\text{sim}(A_g, A) \geq c$. So we need to prove that $\exists A_g \in m(A_r)$ so that $\text{sim}_{A_g,A} \geq c$.

From the algorithm of incremental construction of GDD (3.2.1), we know that $\forall A' \in U \exists A'_g \in U_g$ so that $\text{sim}_{A'_g,A'} = 1$ (two attributes with the similarity 1 are considered identical). Let $A_g \in U_g$ so that $\text{sim}_{A_g,A} = 1$. So $A_g$ and $A$ are identical. We know that $\text{sim}_{A,A} \geq c$. It results that $\text{sim}_{A_r,A_g} \geq c$, so $A_g \in m(A_r)$. \(\square\)

For a given local matching, the search space of match functions is:

$$\prod_{A_r \in U_r, m'(A_r) \neq \emptyset} |m'(A_r)| = \Theta(n'^{|U_r|})$$, where $n' = \max_{A_r \in U_r} |m'(A_r)|$. \hspace{1cm} (5.3)

In the worst case $n' = \Theta(|U_1| + |U_2| + \cdots + |U_n|)$, but in an usual configuration setup this case cannot occur.

We need to find the valid match function that maximizes the actual confidence.
Definition 5.3 A local matching $m'$ is valid if a valid match function can be obtained from it.

In another words, a local matching is valid if it can map the fields in the consumer’s schema into fields within the same DeVisa dictionary.

At a first glance the problem that we need to solve of finding the optimal valid match function can be reformulated using graphs. So, given a weighted tripartite graph $G = (U_r, U_g, U_1 \cup U_2 \ldots U_n, m' \cup M')$, where $m' = \{(A_r, A_g) \text{ so that } A_g \in m(A_r)\}$ and $M' = \{(A_g, A) \text{ so that } A \in M(A_g)\}$, each edge in $m' \cup M'$ being annotated with values from $sim$ we are interested finding a set of paths with exactly 2 edges having maximal overall confidence from vertices in in $U_r$ to vertices in only one of $U_1, U_2, \ldots, U_n$. An instance of this problem is depicted in figure 5.2.

![Tripartite Graph Example](image)

Figure 5.2: An example of a valid matching using tripartite graph representation. The set of bold paths represents a valid local match function. The two data dictionaries are represented in squares and diamonds respectively.

We are going to further reduce the problem to a simpler one that can be modeled using bipartite graphs. In order to do this we need to transform the global matching to a local matching.

The algorithm for determining a local matching in DeVisa given a consumer’s schema is depicted in 5.5.1.

In the first phase of the algorithm a global matching $m$ is calculated. $m$ is represented as an association table that maps each attribute $A_r$ in the consumer’s schema with a set of candidate match attributes in the global data dictionary $D_g$.

In the second phase $m$ is transformed into a local matching that maps $U_r$ to attributes in the local data dictionaries using the pre-calculated $M$ function, which is part of the global data dictionary. If necessary, the similarity values need to be recalculated ($getSim$ function). The GDD plays an important role because
Algorithm 5.5.1 Finding a valid applicable local matching for the consumer’s schema.

**Input:** $S_r = (U_r, P_r)$ the consumer’s schema

$D_g = (U_g, M)$ the global data dictionary as a view on the data dictionaries $D_1, D_2, \ldots, D_n$ in DeVisa

$sim$ a memoization table of the similarity measures between $D_g$ and $D_1, D_2, \ldots, D_n$ and between $S_r$ and $D_g$ respectively.

**Output:** An valid local match $m$, if it exists null, otherwise.

for all $A_r \in U_r$ do

$m(A_r) \leftarrow \emptyset$

for all $A_r \in U_r$ do

for all $A_g \in U_g$ do

if $sim_{A_r,A_g} > c$ then

$m(A_r) \leftarrow m(A_r) \cup \{A_g\}$

for all $A_r \in U_r$ do

for all $A_g \in m(A_r)$ do

for all $A \in M(A_g)$ do

if $sim_{A_r, A} = 1$ or $sim_{A_g, A} = 1$ then

$m(A_r) \leftarrow m(A_r) \cup \{A\}$

else

if $getSim(A_r, A) \geq c$ then

$m(A_r) \leftarrow m(A_r) \cup \{A\}$

15: $m(A_r) \leftarrow m(A_r) - \{A_g\}$

if $m(getPredictedAttribute(S_r)) = \emptyset$ then

return $\emptyset$

if $|\bigcup_{A_r \in U_r} \mu(A_r)| < s|U_r|$ then

return $\emptyset$

20: $support = |U_r|$

for $A_r \in U_r$ do

if $m(A_r) = \emptyset$ then

$support \leftarrow support - 1$

if $support < s|U_r|$ then

return $\emptyset$

25: return $m$
it reduces substantially the search for match candidates to a smaller number of attributes.

After computing $m$, the algorithm tests if the attribute that needs to be predicted has a match in DeVisa ($getPredictedAttribute$ function returns the attribute for which the predicted property is true). If not, a derivable match function can never be produced out of the local matching. Then it checks if the support is preserved (both in $U_r$ as in $m(U_r)$) and only if it is it validates the local matching.

So at the end of this phase $m$ maps each $A_r$ to a set of attributes in the DeVisa data dictionaries. The attributes are spread across all the data dictionaries in DeVisa. At this moment we don’t know if the local matching is valid, i.e if some valid match functions can be determined.

The two phases of the algorithm reduce in polynomial time the above mentioned problem to a constrained form of the assignment problem in weighted bipartite graphs.

In its classical form, the assignment problem requires finding a minimum weight subset of edges so that for any vertex in the graph, it is incident to at most one of the edges in the subset (such a subset is called matching). The assignment problem is tightly related to the bipartite matching problem, which requires finding the maximum cardinality independent set of edges in a bipartite unweighted graph (any two edges in the set don’t share any common vertex).

This class of problems can be solved efficiently using a more generic class of algorithms using flow networks. The flow network based algorithms are the preferred way to model different problems because there are straightforward implementations that are guaranteed to run in time proportional to a polynomial in the size of the network.

A flow network is a weighted network where we interpret edge weights as capacities. We refer to a network with a designated source $s$ and a designated sink $t$ as an st-network. The objective is to compute a second set of edge weights, bounded by the capacities, which called flow. The max-flow problem requires finding a flow in a st-network such that no other flow from $s$ to $t$ has larger value.

An effective approach to solving max-flow problems was developed by Ford and Fulkerson (Ford & Fulkerson (1956), Ford & Fulkerson (1962)). It is a generic
5.5 Schema Matching Approach in DeVisa

primal dual hill climbing method for increasing flows incrementally along augmenting paths from source to sink. If $C_{\text{max}}$ is the maximum edge capacity, the time required to find a max-flow is $O(|V||E|C_{\text{max}})$, which is $O(|V|^2M)$ in a sparse network (where $|E| = O(|V|)$). It serves as the basis for a family of algorithms, including Edmonds-Karp (Edmonds & Karp (1972)), that uses a breadth first approach to find the augmenting path, thereby ensuring that it always chooses the path with the smallest number of edges. The algorithm achieves a complexity of $O(|V|^2\sqrt{E})$, being fully polynomial, because it does not depend on the capacity values. The push-relabel algorithm is one of the most efficient algorithms to compute a maximum flow. The general algorithm has $O(V2E)$ time complexity, while the implementation with FIFO vertex selection rule has $O(V^3)$ running time, the highest active vertex selection rule provides $O(V^2\sqrt{E})$ complexity.

The bipartite matching problem can be reduced polynomially to the network flow problem, using a simple transformation to convert a bipartite graph into an equivalent flow graph, i.e augmenting-path maxflow algorithm. The proof of the correctness of this reduction can be found in Edmonds (2008).

We further investigate the most appropriate method to model our problem and solve it efficiently.

If we attempt to model the DeVisa matching as a max-flow problem as in the case of the bipartite matching problem, it is possible that an augmenting path method does not converge to the optimal solution. In the bipartite matching problem, all the edge capacities are considered 1 without any supplementary restrictions, and, therefore, the augmenting paths will always have residual capacities 0 or 1. This ensures that an edge is not considered twice. The max-flow is the flow with maximal cardinality. In our case, we additionally need to model the edge similarity values. If we replace the edge similarities in the bipartite matching flow network, the algorithms do not always converge to the optimal solution. The edge similarities are values in $[0, 1]$ and, when we find an augmenting path, the residual capacities are in $[0, 1]$ as well. Given that we need to include an edge only once, we need to erase the residual values.

As mentioned earlier, we obtain a problem that can be modeled as a constrained form of the assignment problem. Given a weighted bipartite graph $G = (U_r, m(U_r), m')$, where $m(U_r) = \{A \in U | \exists A_r \in U_r \text{ so that } A \in m(A_r)\}$,
$m' = \{(A_r, A), A \in m(A_r)\}$ is the directed edge set corresponding to the local matching $m$ obtained in the second phase of the algorithm 5.5.1 and $sim$ is the similarity table, containing the similarity values for the edges in $m'$. We need to find a maximal actual confidence matching $\mu$ so that all the vertices in $\mu(U_r)$ incident with edges in the matching are part one and only of $U_1, U_2, \ldots, U_n$ (validity condition) and form a derivable mining schema.

There are several standalone approaches to solve the assignment problem. Using the Kuhn-Munkres (Hungarian) combinatorial optimization algorithm (Munkres (1957)) it can be resolved in $O(|U|^3)$ ($U$ being the vertex set). The algorithm works by increasing the number of zeros in the matrix and searching for a set of starred zeros, one in every row and column. Zeros are primed, starred, or neither during the algorithm. If there are insufficient zeros a quick Gaussian elimination adds more. If there are not enough starred zeros, the primed zeros are starred and the starred zeros primed. Primed zeros are zeros in a column without any more zeros, which, because they are in the same row as another zero were not starred. The Hungarian algorithm cannot be used in DeVisa due to the fact that it is tightly coupled with the assignment problem configuration and therefore supplementary restrictions cannot be incorporated.

A more generalized flow network model is the min-cost – max-flow model, in which the edges have both capacities and costs. The flow cost of an edge in a flow network with edge costs is the product of that edge’s flow and cost. The cost of a flow is the sum of the flow costs of that flow’s edges. The problem is to find a maxflow such that no other maxflow has lower cost. In DeVisa we are moreover interested in a variation of the problem, in which the flow that needs to be calculated has maximum confidence (as opposed to cost) and maximum cardinality (flow). We will further call it max-flow max-confidence.

To model this problem as the max-flow max-confidence problem, we first need to do some prior transformation of the graph, which is described in algorithm 5.5.2. We will also observe some useful properties related to the particularity of the obtained network that can improve the performance of the provided algorithms.

Basically this transformation adds for each data dictionary $D_i$ a source vertex $s_i$ (and edges from $s_i$ to each attribute in $D_i$) and a sink node $t$ (with edges from
Algorithm 5.5.2 Transformation of the DeVisa weighted bipartite graph into a multiple source single sink flow network.

**Input:** \( G = (U_r, m(U_r), m') \) and \( \text{sim} \) defined earlier

**Output:** \( G_f = (U_f, m_f) \) a flow network with sources \( \{s_1, \ldots, s_n\} \) and sink \( t \)

\[
G_f \leftarrow G^T \\
U_f \leftarrow U_f \cup \{s_1, \ldots, s_n, t\} \\
\text{for all } A_r \in U_r \text{ do} \\
\quad \text{if } m(A_r) \neq \emptyset \text{ then} \\
\quad \quad m_f \leftarrow m_f \cup \{(A_r, t)\} \\
\quad \quad \text{sim}_{A_r,t} \leftarrow 0 \\
\quad \text{else} \\
\quad \quad U_f \leftarrow U_f - \{A_r\} \\
\text{for all } U_i \subseteq U_r \text{ do} \\
\quad \text{for all } A \in U_i \text{ do} \\
\quad \quad m_f \leftarrow m_f \cup \{(s_i, A)\} \\
\quad \quad \text{sim}_{s_i,A} \leftarrow 0 \\
\quad \text{for all } (u, v) \in m_f \text{ do} \\
\quad \quad \text{cap}_{uv} \leftarrow 1 \\
\text{for all } A_r \in U_r \text{ do} \\
\quad \text{if } \text{importance}(A_r) = \text{null} \text{ then} \\
\quad \quad \text{if } \text{usage}(A_r) = \text{active} \text{ or } \text{usage}(A_r) = \text{predicted} \text{ then} \\
\quad \quad \quad \text{importance}(A_r) \leftarrow 1 \\
\quad \quad \text{else} \\
\quad \quad \quad \text{importance}(A_r) \leftarrow 0 \\
\quad \text{for all } A \in m(A_r) \text{ do} \\
\quad \quad \text{sim}_{A_r,A} \leftarrow \text{sim}_{A_r,A} \times \text{importance}(A_r) \\
\text{return } G_f 
\]
each vertex in $U_r$ to $t$. For each of the new edges, the similarity value is considered 0 and the capacity is 1 for all the edges in the network. For each attribute in $U_r$ the importance of the attribute is taken into consideration. The importance measures the correlation between the respective attribute and the predictive attribute. In PMML the specification of the importance property is optional, whereas the usage type is active by default. Predicted fields and active fields are the most important, while supplementary fields have moreover explanatory purpose and are not used by the algorithm. Therefore they do not count in the matching process. In the transformation algorithm we adjusted the similarity measure so that higher importance fields count more in the computation of the confidence. All the attributes in $U_r$ that cannot match any attribute in $U$ are eliminated. The newly built network is called $G_f$.

Having this transformation we redefine our purpose as finding the max-flow with max-confidence that starts in one of the sources. If there are several such flows, we would like to keep them all. Furthermore, for each of the identified flows we need to check the derivability condition (defined in 4.12). In case the derivability is not satisfied, we want to have the option of trying the next ”best” flow, i.e the one with max-confidence from the remaining flows.

For each of the sources $s_i$ in $G_f$ we consider the subgraph $G_i^i$ induced by all the vertices accessible from $s_i$. The algorithm 5.5.3 builds a set of max-flows for each of $G_i^i$ $s_i$-t-networks and choses the ones with the highest confidence for which the derivability condition is fulfilled.

It uses the priority queue $PQMatch$ to store all the max-flows in $G_i^i$. The reason why we retain all the max-flows is that, if the top priority max-flow does not satisfy the derivability property, then it is eliminated from the queue and the next one is tested. When a max-flow that satisfies the derivability property as well is encountered, the others are dropped and the search continues with $G_i^{i+1}$.

Figure 5.3 depicts an example of such a subgraph $G_i^i$, where edges are annotated with $(capacity, similarity)$ and the match functions $\mu_1, \mu_2, \mu_3$ that are some of the potential solutions (match functions).
Algorithm 5.5.3 Applying the max-confidence max-flow algorithm for the individual data dictionaries and selecting the one that is both derivable and has maximum confidence.

**Input:** $G_f = (U_f, m_f)$, $\text{sim}$ and $\text{cap}$ as defined in algorithm 5.5.2

**Output:** A set of max-flows $F$ so that $\forall f \in F$ we have $\text{conf}(f) = \max_{f'} \text{conf}(f')$

where $\text{conf}(\mu) = \frac{1}{|U_r|} \sum_{(u,v) \in \mu} \text{sim}_{uv}$

$c_{\text{max}} \leftarrow 0$

$F \leftarrow \emptyset$

for all $s_i \in \{s_1, \ldots, s_n\}$ do

$U_r^i \leftarrow \{u_r \in U_r | \exists u \in U_i \text{ so that } (u_r, u) \in m_f\}$

$G_f^i \leftarrow (\{s_i, t\} \cup U_i \cup U_r^i, m_f^i)$

$PQMatch \leftarrow \text{getMaxFlows}(G_f^i)$

while $PQMatch \neq \emptyset$ do

$(f, c) \leftarrow \text{dequeue}(PQMatch)$

if $\text{isDerrivable}(f, D_i)$ then

if $c > c_{\text{max}}$ then

$c_{\text{max}} \leftarrow c$

$F \leftarrow \{f\}$

else

if $c = c_{\text{max}}$ then

$F \leftarrow F \cup \{f\}$

else

break

end if

end if

end while

return $F$
Figure 5.3: Examples of match functions (b) and the corresponding flow-network conform to algorithm 5.5.3 (a). The match function $\mu_3$ has maximum confidence and will be chosen.

**Proposition 5.2** In the residual network $G_r^i = (U_r^i, m_r^i)$ of any max-flow in the network $G_f^i$ built in algorithm 5.5.3, we have:

$$\forall u, v \in U_r^i \text{ if } (u, v) \in m_r^i \text{ then } (v, u) \notin m_r^i.$$ 

**Proof.** It results from the topology of the network $G_f^i$ and from the construction of the residual network $G_r^i$.  

Let $f$ be the max-flow from which $G_f^i$ was built. We have two situations. First $(u, v) \in f$. Then $flow_{uv} = 1$, $res_{uv} = 0$ and $res_{uv} = 1$ (results from the procedure augment). This means that $(u, v) \notin m_r^i$ and $(v, u) \in m_r^i$. Secondly, $(u, v) \notin f$. Then $flow_{uv} = 0$, $res_{uv} = 1$ and $res_{uv} = 0$ (results from the procedure augment). This means that $(u, v) \in m_r^i$ and $(v, u) \notin m_r^i$. □

**Proposition 5.3** In $G_r^i = (U_r^i, m_r^i)$ we have

1. $deg^+(t) = deg^-(s_i)$.
2. $\exists tAA_r s_i$ a path from $t$ to $s_i$. 

99
5.5 Schema Matching Approach in DeVisa

Proof. Both properties result directly from the flow conservation property and from the construction of $G_r^i$ based on a non-empty flow. □

**Proposition 5.4** In the residual network $G_r^i = (U_r^i, m_r^i)$ we have:

1. if $(A_r, A) \in m_r^i$, then $(t, A_r) \in m_r^i$ and $(A, s_i) \in m_r^i$.
2. if $(t, A_r) \in m_r^i$, then $\exists A \in U_r$ so that $(A_r, A) \in m_r^i$ and $(A, s_i) \in m_r^i$.
3. if $(A, s_i) \in m_r^i$, then $\exists A \in U_r$ so that $(A_r, A) \in m_r^i$ and $(t, A_r) \in m_r^i$.

Proof. If $(A_r, A) \in m_r^i$, it corresponds to the edge $(A, A_r) \in \mu$, the flow on which $G_r^i$ is based on. It follows that there is an entire corresponding path from $t$ to $s_i$ in $G_r^i$, so $(t, A_r) \in m_r^i$ and $(A, s_i) \in m_r^i$. The other two properties can be proven in a similar fashion. □

**Proposition 5.5**

if $(A, A_r) \in m_r^i$, then $deg^+(A_r) = 1$ and $deg^-(A) = 1$.

Proof. Let us prove that $deg^+(A_r) = 1$. We have two exclusive possibilities: $(t, A_r) \in m_r^i$ or $(A_r, t) \in m_r^i$.

If $(t, A_r) \in m_r^i$, then from property 5.4 it results that $A$ is the unique attribute so that $(A, A_r) \in m_r^i$, so $deg^+(A_r) = 1$.

If $(A_r, t) \in m_r^i$ then we know that $deg^+(A_r) \geq 1$. Let us assume that $deg^+(A_r) > 1$. Then $\exists A' \in U_r$ so that $(A_r, A') \in m_r^i$. Then from property 5.4 it follows that $(t, A_r) \in m_r^i$, which conflicts with the hypothesis. Therefore $deg^+(A_r) = 1$.

Using a similar rationale we prove that $deg^-(A) = 1$. □

The algorithm 5.5.4 builds a priority queue named $PQMatch$ containing the max-flows arranged in descending order of their confidence. The algorithm uses the cycle canceling approach and it is based on the property that a max-flow is a max-confidence maxflow if and only if its residual network contains no negative-cost directed cycle (Sedgewick (2003)).

The cost of the cycle in the residual network is calculated as

$$cost(cycle) = \sum_{(u,v) \in cycle} cost_{uv} \ast res_{uv}.$$
where the \( \text{cost}_{uv} = 1 - \text{sim}_{uv} \), for all the edges with strictly positive similarity. For the others, \( \text{cost} \) is 0.

**Algorithm 5.5.4 getMaxFlows(\( G_f^i \))**

**Input:** \( G_f^i = (U_f, m_f^i) \), a s/t network, \( \text{sim} \) and \( \text{cap} \) as defined in algorithm 5.5.3

**Output:** \( PQMatch = \{ \mu \text{ max-flow with } \max(|\mu|) \} \), a priority queue containing the max-flows in \( G_f^i \), with the priority set by the value of the actual confidence.

\[
PQMatch \leftarrow \emptyset \\
(f, c) \leftarrow \text{getMaxFlow}(G_f^i) \\
\text{if } \deg^+(s_i) < s \times |U_r| \text{ then} \\
\hspace{1cm} \text{return } \emptyset \\
\text{repeat} \\
\hspace{1cm} \text{enqueue}(PQMatch, (f, c)) \\
\hspace{1cm} \text{cycle} \leftarrow \text{getNegativeCycle}(G_f^i, t) \\
\hspace{1cm} \text{if } \text{cycle} \neq \emptyset \text{ then} \\
\hspace{2cm} \text{augment}(f, \text{cycle}) \\
\hspace{2cm} \text{path} \leftarrow \emptyset \\
\hspace{2cm} \text{while } \text{path} \neq \emptyset \text{ do} \\
\hspace{3cm} \text{path} \leftarrow \text{getAugmentingPath}(G_f^i) \\
\hspace{3cm} \text{augment}(G_f^i, \text{path}) \\
\hspace{1cm} \text{until } \text{cycle} = \emptyset \\
\text{return } PQMatch \\
\text{procedure augment}(G, P) \\
\hspace{1cm} \text{for all } (u, v) \in P \text{ do} \\
\hspace{2cm} \text{flow}_{uv} \leftarrow \text{flow}_{uv} + 1 \\
\hspace{2cm} \text{flow}_{vu} \leftarrow \text{flow}_{vu} - 1 \\
\hspace{2cm} \text{res}_{uv} \leftarrow \text{res}_{uv} - 1 \\
\hspace{2cm} \text{res}_{vu} \leftarrow \text{res}_{vu} + 1 \\
\]

The algorithm starts with identifying a max-flow and then attempts to identify a negative cost cycle in the residual network. If one is found the flow is saved in the priority queue and then it is augmented along the negative cycle. The iteration continues until no more negative cost cycles can be found. It is possible
that, after augmenting along a negative cost cycle, a new augmenting path is discovered. In this case the algorithm finds it by calling the function described in 5.5.6 and augments the flow along it.

**Proposition 5.6** The number of augmenting cycles needed in algorithm 5.5.4 is \( O(|m_f|) \).

**Proof.** We know that \( \forall (u, v) \in m_f, 0 < \text{cost}_{uv} \leq 1 - c \). Let us consider the worst case when in the initial max-flow each edge has the maximum possible cost, which is \( 1 - \min_{(u,v) \in m_f} \text{sim}_{uv} = 1 - c \), where \( c \) is the tuning parameter representing the confidence of the match function. So in the worst case the cost of the initial max-flow is at most \( |U_r| * (1 - c) \).

Each augmentation along a negative cycle replaces at least one edge with at least another edge with lower cost. We can only do \( O(|m_f|) \) such replacements. \( \square \)

**Proposition 5.7** In the residual network \( G^i_r = (U^i_r, m^i_r) \), if \( \deg^+(s_i) > 0 \) then there is at least one cycle that includes \( s_i \).

**Proof.** \( \deg^+(s_i) > 0 \), then \( \exists A \in U_i \) so that \( (s_i, A) \in m^i_i \). We constructed \( G_f \) in algorithm 5.5.2 such as all the attributes in \( U_f \) can be matched with at least another attribute. So, \( \exists A_r \in U_r \) so that either \( (A, A_r) \in m^i_r \) or \( (A_r, A) \in m^i_r \).

From property 5.2, it results that only one of the two holds.

If \( (A_r, A) \in m^i_r \), then, from property 5.4, it results that \( (A, s) \in m^i_r \), which conflicts with \( (s, A) \in m^i_i \), according to the proposition 5.2.

It results that \( (A, A_r) \in m^i_r \). From property 5.5 it results that \( \deg^+(A_r) = 1 \). Therefore we further have two mutual exclusive possibilities: \( (A_r, t) \in m^i_r \), or \( (t, A_r) \in m^i_r \).

If \( (A_r, t) \in m^i_r \), then \( s_iAA^r_t \) is a path from \( s_i \) to \( t \). Using the property 5.4, it results that it exists a cycle that includes \( s_i \).

If \( (t, A_r) \in m^i_r \), then, from property 5.4 it results that \( \exists A' \neq A \) so that \( (A_r, A') \in m^i_r \) and \( (A', s_i) \in m^i_r \). We obtained the cycle \( s_iAA^rA's_i \) that obviously includes \( s_i \). \( \square \)

**Proposition 5.8** Given \( G^i_f = (U^i_f, m^i_f) \) the network built in 5.5.3, for any cycle in the residual network, it is accessible from \( t \).
Proof. Let $C$ be a cycle in $G_f$. From property 5.2 it results that $C$ has a length of at least 4 edges.

If $C$ includes $s_i$ then, based on property 5.3, it follows immediately that $C$ is accessible from $t$.

If $C$ does not include $s_i$, then $C$ either includes $t$, or not. If it includes, then we are finished.

If not, let us consider $C = A_1^l A_1^l \ldots A_l^l A_1^l$, where $l \geq 4$. Because of $(A_j^l, A_j^l) \in m_i^l$ and property 5.4 it results that $(t, A_j^l) \in m_t^l$, $\forall j = 1 \ldots l$ so $C$ is accessible from $t$. □

Based on the result given by the property 5.8, we can apply the cycle identification algorithm depicted in 5.5.5 starting from $t$ to detect and identify a negative cost cycle. In the first phase it builds the shortest paths from $t$ to the other vertices. It maintains a FIFO queue of all vertices for which relaxing along an outgoing edge could be effective. The main loop takes a vertex off the queue, relaxes along all its edges, and updates the queue as appropriate. It uses a sentinel value to separate vertices that changed on the previous pass from those that change in the current one. The loop terminates after $|U^f_i|$ passes. In the second phase it tries to apply another relaxation. If this would be effective, this means that we discovered an edge in a negative cost cycle. Then the algorithm loops until it completely discovers the cycle and returns it. The second phase is done in $O(|U^f_i|)$. The cycle detection algorithm runs in $O(|U^f_i| \times |m^l_i|)$ computational time.

The cycle identification algorithm is based on an adapted version of Bellman-Ford algorithm (Bellman (1958)) Bellman-Ford is a label correcting algorithm that computes single-source shortest paths in a weighted digraph, where some of the edge weights may be negative.

The algorithm 5.5.7 uses an augmenting path approach to find a max-flow. It starts with zero flow everywhere and then increases the flow along any path from source $s_i$ to sink $t$ with no full forward edges or empty backward edges, continuing until there are no such paths in the network. The implementation uses the shortest augmenting path, and therefore the search for an augmenting path amounts to breadth-first search (BFS) in the residual network, as it can be seen in algorithm 5.5.6. The time needed to find an augmenting path is $O(|m_f^l|)$. 

103
5.5 Schema Matching Approach in DeVisa

Algorithm 5.5.5 $\text{getNegativeCycle}(G^i_f, t)$

for all $v \in U_i$ do
    $\text{pathcost}_{tv} \leftarrow \infty$
    $\text{pathcost}_t \leftarrow 0$
    $Q \leftarrow \{t, \text{sentinel}\}$
while $Q \neq \emptyset$ do
    $u \leftarrow \text{poll}(Q)$
    if $u = \text{sentinel}$ then
        level $\leftarrow$ level + 1
        if level $\geq |U^i_r| - 1$ then
            return $\emptyset$
        enqueue($Q$, $\text{sentinel}$)
    for $w \in U^f_i$ so that $\text{res}_{uw} \neq 0$ do
        if $\text{pathcost}_w \geq \text{pathcost}_u + \text{cost}_{uw} \cdot \text{res}_{uw}$ then
            $\text{pathcost}_w \leftarrow \text{pathcost}_u + \text{cost}_{uw} \cdot \text{res}_{uw}$
            $\text{parent}_w \leftarrow u$
    for all $v \in U^i_r$ do
        if $\text{pathcost}_v > \text{pathcost}_{\text{parent}_v} + \text{cost}_{\text{parent}_v, v} \cdot \text{res}_{\text{parent}_v, v}$ then
            $\text{cycle} \leftarrow \{(\text{parent}_v, v)\}$
            $u \leftarrow \text{parent}_v$
            while $u \neq v$ do
                $\text{cycle} \leftarrow \text{cycle} \cup \{(\text{parent}_u, u)\}$
                $u \leftarrow \text{parent}_u$
            return $\text{cycle}$
        return $\emptyset$
5.5 Schema Matching Approach in DeVisa

The maximum number of augmenting paths is $O(\min(|U_i|, |U_r|))$ (given that any path passes through both vertices in $U_i$ and in $U_r$, the flow is always 1 and any two paths don’t have common vertices except for $s_i$ and $t$). So we can deduce that the time needed to find a max-flow is $O(|m_f| \times \min(|U_i|, |U_r|))$

Algorithm 5.5.6 $getAugmentingPath(G^n_f)$

```

```

In figure 5.4 we continue the example in figure 5.3 and illustrate the algorithm to find a max-confidence max-flow.

The purpose of the schema matching algorithm is to compute a set of valid derivable match functions, as it is necessary in the scoring process, match schema case (see 4.2.1.2). The following proposition states the correctness of the schema matching algorithm.

**Proposition 5.9** (Correctness of the algorithm 5.5.3) The algorithm 5.5.3 produces a set of valid derivable optimal match functions.

**Proof.** We prove first that in any $s_it$-network $G^n_f$ a max-confidence max-flow corresponds to an max-confidence valid match function.
Algorithm 5.5.7 \textit{getMaxFlow}($G^i_f$)

\textbf{Input:} $G^i_f = (U^j, m^i_f)$, a $s,t$ network, \textit{sim} and \textit{cap} as defined in algorithm 5.5.3

\textbf{Output:} $\mu$ max-flow with $\max(|\mu|)$

\begin{verbatim}
\mu \leftarrow \emptyset
for all $(u, v) \in m^i_f$ do
  \textbf{flow}_{uv} \leftarrow 0
  \textbf{flow}_{vu} \leftarrow 0
  \textbf{res}_{vu} \leftarrow 1
  \textbf{res}_{uv} \leftarrow 0
  \textbf{if} \ \textit{sim}_{uv} > 0 \ \textbf{then}
    \textbf{cost}_{uv} \leftarrow 1 - \textit{sim}_{uv}
    \textbf{cost}_{vu} \leftarrow \textit{sim}_{uv} - 1
  \textbf{else}
    \textbf{cost}_{uv} \leftarrow 0
    \textbf{cost}_{vu} \leftarrow 0
\textbf{repeat}
\hspace{1em} P \leftarrow \textit{getAugmentingPath}(G^i_f)
\hspace{1em} \textbf{if} \ P \neq \emptyset \ \textbf{then}
\hspace{2em} \mu \leftarrow \mu \cup P
\hspace{2em} \textit{augment}(\mu, P)
\textbf{until} \ P \neq \emptyset
\\return \mu_{\max}
\end{verbatim}

\textbf{procedure} \textit{augment}($G^i_f, P$)

\begin{verbatim}
for all $(u, v) \in P$ do
  \textbf{flow}_{uv} \leftarrow \textbf{flow}_{uv} + 1
  \textbf{flow}_{vu} \leftarrow \textbf{flow}_{vu} - 1
  \textbf{res}_{uv} \leftarrow \textbf{res}_{uv} - 1
  \textbf{res}_{vu} \leftarrow \textbf{res}_{vu} + 1
\end{verbatim}
5.5.4 produces a set of max-flows organized in a heap where the max-flow with the maximum confidence has the maximum priority.

The algorithm 5.5.3 checks the derivability condition on the max-flows in descending order and stops at the max-flow with the max-confidence for which the condition holds.

Let $f$ be the top-priority max-flow for which the derivability holds. We define $\mu: U_r \rightarrow U_i$ so that $\mu(A_r) = A$ iff $(A, A_r) \in f$. $\mu$ is a bijection (results from the construction of the flow), is valid (because the flow only uses attributes in $U_i$) and it is optimal because the confidence is maximum. $s \cdot |U_r| \geq |\mu(U_r)|$, because algorithm 5.5.4 rejects all the flows for which $deg^+(s_i) < s \cdot |U_r|$. From here we deduce that $\mu$ is the optimal derivable valid match function.

In conclusion a set of optimal match functions across all the data dictionaries is built. □

**Proposition 5.10** The algorithm needs $O(|U_r|^2 \cdot |U|^3 + |U_r|^3 \cdot |U|^2)$ in the worst case to calculate a set of optimal derivable match functions.
5.6 The Similarity Measure

**Proof.** The algorithm needs:

\[
\sum_{i=1}^{n} (O(|m^i_f| \cdot \min(|U_r|, |U_i|)) + O(|m^i_f|^2)) = O(\sum_{i=1}^{n} |U^i_f| \cdot |m^i_f|^2)
\]

\[
= O(\sum_{i=1}^{n} (|U^i_r| + |U^i_i|)|U_r|^2|U_i|^2)
\]

\[
= O(\sum_{i=1}^{n} |U^i_r|^3|U^i_i|^2 + \sum_{i=1}^{n} |U^i_i|^3)
\]

\[
= O(|U^i_r|^2 \cdot |U^i_i|^3 + |U^i_i|^3 \cdot |U^i_r|^2)
\]

\[\square\]

5.5.1 The Validation of the Matching

Being part of a run-time process (scoring - see 4.2), the schema matching needs to be performed completely automatic.

However, there could be several match functions that are the outcome of the matcher and potentially not accurate. The PMQL engine will build the query plan that includes all the match functions and, subsequently, will score on all the resulting interpreted relations (see 4.2). The consumer is presented with a set of answers based on different match functions.

In this case a manual validation process by a human domain expert should be performed at consumer’s site. This validation process needs some additional explanation on the match functions, namely the DeVisa metadata information together with the similarity table based on which the match was computed. Future work will include a mechanism to store the match functions for future use, using alignment storage mechanisms described in Hepp et al. (2008).

5.6 The Similarity Measure

In general, a similarity measure is a function that associates a numeric value with (a pair of) sequences, with the idea that a higher value indicates greater
5.6 The Similarity Measure

similarity.

The similarity measure calculation algorithm in DeVisa is multi-faceted, being able to adapt to take into consideration the available schema information. As stated earlier much of the available metadata is optional and in the minimal case the similarity measure needs to be calculated based only on schema based means. DeVisa combines multiple available similarity criteria to achieve a final score, which makes it a hybrid adaptive approach.

For instance the similarity between the global data dictionary and the DeVisa data and transformation dictionaries utilizes merely schema based techniques, but it also takes into account the the univariate statistics when they are available (PMML standard specifies them as optional -see 2.5.1). Furthermore, during scoring schema matching, the statistics on the consumer’s data instances need to be calculated on the fly.

The only mandatory factor in calculating the similarity of elements in DeVisa is the syntactic analysis of the terminology in the data dictionaries. Therefore choosing an appropriate string metric is crucial in achieving good accuracy.

Today quite a lot of string metrics exist in literature and among others play an important role in schema/ ontology mediation (Cohen et al. (2003)). A string metric usually results in calculating a similarity or dissimilarity (distance) score between two pairs of text strings. The most well known are the edit - or Levenstein - distance (Levenstein (1966)), which counts the insertions and deletions needed to match two strings, the sub-string distance which searches for the largest common substring, the Needleman-Wunsch (Needleman & Wunsch (1970)) distance, which assigns a different cost on the edit operations, the Smith-Waterman (Smith & Waterman (1981)), which additionally uses an alphabet mapping to costs and the Monge-Elkan (Monge & Elkan (1996)), which uses variable costs depending on the substring gaps between the words, the Jaro-Winkler (Winkler (1999), Jaro (1995)) similarity, which counts the common characters between two strings even if they are misplaced by a short distance, the Q-Gram (Sutinen & Tarhio (1995)), which counts the number of tri-grams shared between the two strings.

Taking into consideration features like the threshold (the value above which two elements in a pair are considered identical), the cardinality of mappings,
Stoilos et al. (2005) identifies the main characteristics of a string metric capable of being used in schema/ontology mediation processes:

- **Fast**: Since ontologies are used in applications that demand processing in real-time, like the semantic web or intelligent retrieval tasks, the complexity of the string metric should be very low, leading to a fast matching procedure.

- **Stable**: This is the ability of a string metric to perform almost optimal even if small diverges from the optimal threshold take place. Most of the string metrics perform reasonably well if the threshold is optimized, but the performance rapidly decrease if the value of the threshold is slightly disturbed. There are methods that automatically adjust the threshold, as opposed to the ones that use an a priori established threshold.

- **Intelligent**: When an ontology is compared to an irrelevant one, but with which string resemblances occur, the metric should identify all the differences and provide correct results. It is very often that usual string metrics fail to identify cases where two strings represent completely different concepts but resemble a lot.

- **Discriminating**: In a 1:1 cardinality matching setup, which is a very common requirement, if a field in one ontology is mapped with the same similarity degree to several others in the other ontology, then it is very probable that the matching algorithm fails to choose the right matching. So it is desirable that a similarity metric rarely assigns the same similarity degree when a string is compared to several others.

Given two mining schemas or data dictionaries, the DeVisa approach in calculating the similarity performs the following steps:

**Constraint Similarity.** Two fields can map to each other only if the data types are compatible. The compatibility can be checked using the XML Schema types compatibility. A rougher data type categorization that is very relevant in the statistic measures matching is categorical, ordinal and continuous fields. The data type matching is realized by applying a set of a priori defined rules. So, in case of categorical fields, the similarity is calculated as the ratio of common
values of the total number of values in the domains of the two fields (Jaccard index).

\[
\text{if } \text{optype}(A_1) = \text{categorical and optype}(A_2) = \text{categorical then}
\]

\[
DT(A_1, A_2) = \frac{|values(A_1) \cap values(A_2)|}{|values(A_1) \cup values(A_2)|}
\]

**Terminology criteria** It uses the string metric described in Stoilos et al. (2005) to compute a table of similarities between any pair of field names. PMML allows both specification of a field name (an identifier relevant to PMML processing) and a display name (which is normally displayed in user interfaces, but irrelevant to PMML processing). The display name tends to have a higher descriptive value, therefore we apply I-Sub on both names. This approach, called Iterative-SubString (I-Sub), has been integrated in several popular ontology alignment and schema matching systems. It reveals good precision and recall when in this context compared to other general purpose string metrics. Given two attributes \(A_1\) and \(A_2\) the similarity between the two terms identifying the two attributes is calculated as follows:

\[
I\text{Sub}(A_1, A_2) = \text{Comm}(A_1, A_2) - \text{Diff}(A_1, A_2) + \text{Winkler}(A_1, A_2)
\]

where \(\text{Comm}(A_1, A_2)\) measures the commonality between the two fields and is defined as:

\[
\text{Comm}(A_1, A_2) = \frac{2 \sum_i \text{len}(\text{LongestCommSubstr}_i)}{\text{len}(A_1) + \text{len}(A_2)}
\]

\[A_1^u \leftarrow A_1\]
\[A_2^u \leftarrow A_2\]

**repeat**

\[\text{sub} \leftarrow \text{LCS}ub(A_1^u, A_2^u)\]

**if** \(\text{len}(\text{sub}) \geq 0\) **then**

\[A_1^u \leftarrow \text{remove}(A_1^u, \text{sub})\]
\[A_2^u \leftarrow \text{remove}(A_2^u, \text{sub})\]

**until** \(\text{len}(<\text{sub}>) \leq 0\)
Diff(A₁, A₂) measures the dissimilarities based on the unmatched substrings A₁¹ and A₂¹ resulted from the previous procedure. It uses the Hamacher parametric triangular norm (Hamacher et al. (1978)):

\[
Diff(A₁, A₂) = \frac{\text{len}(A₁¹)\text{len}(A₂²)}{p + (1-p)(\text{len}(A₁¹) + \text{len}(A₂²) - \text{len}(A₁¹)\text{len}(A₂²))}
\]

where \( p \in [0, \infty) \) is a parameter that can adjust the importance of the difference factor.

Winkler(A₁, A₂) quantifies the improvement of the result using the metric described in Winkler (1999) and is defined as:

\[
\text{Winkler}(A₁, A₂) = \text{Jaro}(A₁, A₂) + lp_w (1 - \text{Jaro}(A₁, A₂))
\]

\[
\text{Jaro}(A₁, A₂) = \frac{1}{3} \left( \frac{m}{\text{len}(A₁)} + \frac{m}{\text{len}(A₂)} + \frac{m - t}{m} \right)
\]

where \( l \) is the length of common prefix at the start of the attribute names (up to a maximum of 4 characters), \( p_w \) is a constant factor signifying how much of the score is adjusted up when the names have common prefixes (standard value is 0.1). \( m \) is the number of matching characters, and \( t \) is the number of transpositions (matching characters which occur on different positions in the two names).

The I-Sub string metric maps the similarity values in the interval \([-1, 1]\) and values above 0.4 signify high degree of similarity. For instance, the fields microRNAStart and miRNAStart, both meaning the same concept, achieve a degree of term similarity of 0.6607 when \( p = 0.6 \). In DeVisa we normalize the ISub string metric to return values in \([0, 1]\), i.e we consider:

\[
NISub(A₁, A₂) = \frac{ISub(A₁, A₂) + 1}{2}
\]

The complexity of the I-Sub string metric is polynomial to the size of the input strings, satisfying the property of a fast metric.

Refinement.

1. Linguistic: Similarity is measured in terms of WordNet synsets (or senses), namely structures containing sets of terms with synonymous meanings.
Synsets are connected to one another through explicit (lexical) semantic relations (hypernymy, hyponymy, troponymy, meronymy, holonymy, antonymy). These relations contribute to calculating the similarity factor between two fields. Here the display names are used, since the field names tend to be concatenate multiple strings in one and make extensive use of abbreviations. Display names on the other side use a longer name (potentially using multiple words) that better describe the given concept. For each pair of component words in the display name of the attributes the WordNet tree distance is calculated and the minimal tree distance is chosen. Then we average the component word similarities.

\[
WN(A_1, A_2) = \frac{1}{n} \left( \sum_{A_i^1 \in A_1} \max_{A_j^2 \in A_2} (1 - \text{treeDist}(A_i^1, A_j^2)) \right)
\]

where \(n\) is the number of pairs for which the WordNet tree distance is less than 1 and \(\text{treeDist}\) represents the normalized path distance in the WordNet tree between the synsets of the two words. The distance is calculated for every common speech part between the two words, but the minimal one is chosen.

2. Description: The basic idea behind this matcher is that the number of the same words occurring in the two field textual description increases the similarity value. If the number of shared words exceeds a given threshold then the fields are considered to be equivalent. This approach is resembling to the one in the WordNet glosses. DeVisa uses the Jacard index between the textual descriptions of the fields, where the two sets are the sets of words occurring in each of the descriptions. The Jacard index is calculated as a fraction of the number of shared words and the total number of words. To achieve better precision, all the words in the textual description are stemmed using a Paice/Husk stemming algorithm. Given two attributes \(A_1\) and \(A_2\), with the textual description \(Descr(A_1)\) and \(Descr(A_2)\), the Jacard index is calculated as:

\[
\text{Jacard}(A_1, A_2) = \frac{|Descr'(A_1) \cap Descr'(A_2)|}{|Descr'(A_1) \cup Descr'(A_2)|}
\]
where $Descr'(A)$ is the set of stemmed words in $A$’s textual description.

For instance, considering again the attributes `microRNAStart` and `miRNAStart`, with $Descr(microRNAStart) =$ "The start gene position in the chromosome" and $Descr(miRNAStart) =$ "The position in the chromosome where the miRNA starts", then

$$Jacard(microRNAStart, miRNAStart) = 0.6666.$$  

Further work will refine the textual description similarity to behave in a way similar to the gloss similarity in WordNet.

3. Statistical: The data instances statistical information (continuous or discrete statistics) gives additional useful information for calculating the similarity. It can be applied only if both data dictionaries provide this information or the information can be inferred from the data instances themselves. In the case of categorical/ordinal attributes we can use the modal value (the most frequent value in the data set used to build the model). In the consumer’s schema, this information can be easily deduced from the relation that needs to be scored. Rules like the one hereunder can be used:

$$\text{if } \text{optype}(A_1) = \text{categorical and optype}(A_2) = \text{categorical or optype}(A_1) = \text{ordinal and optype}(A_2) = \text{ordinal} \text{ and modalValue}(A_1) = \text{modalValue}(A_2) \text{ then}$$

$$Stat(A_1, A_2) \leftarrow 1$$

In the case of continuous attributes, for each of the attributes we build a statistical signature that consists of a vector of $N$ values corresponding to each of the available statistical information. The signatures correspond to points in the $N$-dimensional space. The signature for each of the attributes is given by two vectors:

(a) $u_A$, where $u_{Ai} \in \{0, 1\}, i = 1 \ldots N$, where $u_{Ai}$ marks if the respective statistical index is present for the attribute $A$

(b) $sign_A = (invalid(A), missing(A), min(A), max(A), mean(A), stdDev(A), median(A), interQuartileRange(A))$
5.6 The Similarity Measure

where \( \text{Invalid}(A) \), \( \text{missing}(A) \) calculates the percentage of invalid/missing values, \( \text{mean}(A) \) is the expected value, \( \text{median}(A) \) is the 50\% quantile and \( \text{interQuartileRange}(A) \) is calculated as 75\% quantile − 25\% quantile.

The statistical distance between two attributes \( A_1 \) and \( A_2 \) is calculated as the euclidean distance between the two signature vectors:

\[
\text{StatDist}(A_1, A_2) = \sqrt{\sum_{i=1}^{N} (\text{sign}_{A_1i} - \text{sign}_{A_2i})^2 u_{A_1i} u_{A_2i}}
\]

To get to the statistical similarity score, we use the Shepard’s law (Shepard (1987)):

\[
\text{Stat}(A_1, A_2) = e^{-\text{StatDist}(A_1, A_2)}
\]

**Structural.** This step is performed when both data dictionaries are explained through additional taxonomies. In this case a tree matching algorithm is performed, taking into account the context in which each field is defined within the taxonomy.

Steps 2 and 3 and 5.6 are optional, depending on the availability of the respective information. The values obtained in the performed steps are aggregated to produce the final similarity score.

The aggregation is achieved as follows. Three 6-dimensional vectors \( u \), \( w \) and \( s \) are calculated. The 6 dimensions represent the 6 match criteria presented above (in this order).

1. \( u \) is a binary vector representing if a certain match criteria is present or not in the matching strategy. The first two components of the vector will be always 1. So \( u_i \in \{0, 1\} \) and \( \sum_{i=1..6} u_i \geq 2 \).

2. \( w \) is a weight vector, where \( w_i \) represents the importance that a match criteria has in the matching strategy. Given that certain match criteria could not be present, \( w \) should be adaptive to the availability of the match criteria, so the following condition should hold:

\[
\sum_{i=1..6} u_i w_i = 1.
\]
We start by considering equal weights to each of the present match criteria. Based on experimental results, further improvement of the similarity measure will include probabilistic rule sets to adapt the weighting technique to achieve better matching results. These rules will adjust $w$ so that some of the match criteria are assigned higher importance.

3. $s$ is the vector of similarity scores achieved by applying each of the match criteria. $s_i \in [0, 1]$, the value of 1 corresponding to identical attributes. $s$ has the following structure:

$$s = (DT(A_1, A_2), NISub(A_1, A_2), WN(A_1, A_2),$$

$$Jacard(A_1, A_2), Stat(A_1, A_2), Struc(A_1, A_2))$$

Finally, the similarity between the fields is calculated as follows:

$$sim(A_1, A_2) = \sum_{i=1\ldots6} u_i w_is_i.$$
Chapter 6

Related Work

6.1 Online Scoring and Online Data Mining Services

Another term for infrastructures providing resources/services over the Internet is the concept of a cloud. There are storage clouds (block and file based services), data clouds (record-based, column-based, object-based services) and compute clouds (computational services). Often these are layered to create a computing platform for developing cloud-based applications. Examples include Amazon S3 (Ama (2008)), Google File System (Ghemawat et al. (2003)), BigTable and MapReduce infrastructure (Dean & Ghemawat (2008)) etc.

The concept of knowledge as a service is a specialized sort of computing cloud.

In the context of automated knowledge elicitation from data sources, Krishnaswamy et al. (2001) proposes two models of organization of the data mining service providers and the interaction with their clients. The paper also describes a XML based language for specifying data mining task requests and the services of data mining service providers.

Due to the increasing usage of the web space as a platform (one of the main goals of Web 2.0), there are many other proposals of data mining services available on the web such as Chieh-Yuan Tsai (2005).

Grigoris Tsoumakas (2007) describes a web-based system for classifier sharing and fusion named CSF/DC. It enables the sharing of classification models,
by allowing the upload and download of such models expressed in PMML in the
system’s online classifier repository. It also enables the online fusion of classifi-
cation models located at distributed sites, which should be a priori registered in
the system using a Web form.

In Ali et al. (2005) web services are provided to implement data mining func-
tions like classifiers, clustering algorithms and association rules, and a workflow
engine for model composition.

WekaWS (***(2007c), Talia et al. (2005)) is an initiative to provide the Weka
(Weka) data mining functions as WSRF - Web Service Resource Framework (***(2004b)) web services. In such a way, distributed data mining tasks can be con-
currently executed on decentralized Grid nodes by exploiting data distribution
and improving application performance. In Weka4WS, the data mining algo-
rithms for classification, clustering and association rules can be also executed on
remote Grid resources. To enable remote invocation, all the data mining algo-
rithms provided by the Weka library are exposed as a Web Service, which can be
easily deployed on the available Grid nodes.

Grossman et al. (2004) introduces Open DMIX, an open source collection
of web services for the mining, integration, and exploration of remote and distri-
buted data. Open DMIX is layered: the top layer provides templated data
mining and statistical algorithms, such as those defined by PMML. The mid-
dle layer provides access and integration of remote and distributed data. The
bottom layer provides specialized network protocols designed to work with large
distributed data sets over wide area networks. The clients interact with Open
DMIX servers using a version of web services designed for high performance ap-
lications (SOAP+).

Ferrucci et al. (2006) presents scenarios on how applications combining PMML
and UIMA framework can be built. The scenarios include using clustering or
probabilistic clustering PMML models to do text categorization (as opposed to
the bag of words approach), or processing XML tagged data to lower the classi-
fication level (from more specific to broader).

Grossman & Gu (2008) goes further and proposes a cloud-based infrastructure
(with storage and computing capabilities) that supports data mining applications.
To facilitate the development of data mining and scoring-enabled applications, APIs have been specified, such as Java Data Mining - JDM (** (2005)), which also includes web services extensions, i.e JDMWS.

6.2 PMML Scoring Engines

6.2.1 Augustus

Augustus (Chaves et al. (2006)) is an open source, PMML compliant system for building and scoring statistical models that is designed to work data sets that are too large to fit into memory. If data can fit into memory, then there are several open source statistical systems available, including the R System (** (2008b)).

Augustus is typically used to construct models and score data with models. Augustus includes a dedicated application for creating, or producing, predictive models rendered as PMML-compliant files. Scoring is accomplished by consuming PMML-compliant files describing an appropriate model. Augustus provides a dedicated application for scoring data with a class of models referred to as Baseline Models. The typical Baseline model development and use cycle with Augustus is as follows:

1. Identify suitable data with which to construct a new model.
2. Provide a model schema which proscribes the requirements for the model.
3. Run the Augustus producer to obtain a new model.
4. Run the Augustus consumer on new data to effect scoring.

In addition to the producer and consumer applications, Augustus is conceptually structured and provided with libraries which are relevant to the development and use of predictive models. Broadly speaking, these consist of components that address the use of PMML and components that are specific to Augustus. The current version of Augustus consists of the following main components.
6.2 PMML Scoring Engines

1. A data management component called UniTable is part of the Augustus kernel. A UniTable is broadly similar to a data frame in the R system. It contains data fields arranged in columns, that may be of different types, but all of which have the same number of rows.

2. Utilities for processing PMML files.

3. Components for shaping and transforming data. These components are also part of the Augustus kernel.

4. Model specific components. Currently, Augustus supports baseline models and tree-based models (DMG). These are collected together in the modellib module.

5. Run time support. Augustus provides a general mechanism for reading data into Augustus and writing data out of Augustus called anyreader and anywriter, respectively. For example, anyreader can read data from a stream or a file and put it into a common format for further processing by Augustus.

6. Miscellaneous tools. The Augustus distribution also includes a number of auxiliary utilities, such as configuration utilities, utilities for working with date and time etc. For example, there is a utility that will create a stub PMML data schema based upon an analysis of a data file.

ADAPA is intrinsically a decision engine. It combines the power of predictive analytics and business rules to facilitate the tasks of managing and designing automated decisions systems. This essentially allows a consumer to deploy PMML on the Amazon compute cloud (Ama (2008)). Rather than having to invest in significant infrastructure to put models to work in production small and medium sized organizations are now able to use the cloud and deploy them very cost effectively. It uses browser based deployment, is standards based - web services for run-time access, PMML for model design and JSR 73-JDM (*** (2005))) for run-time access to the engine. A consumer can use R, SPSS, SAS, or DB2 Intelligent Miner to build a model because these tools all support export into PMML.
It offers the computing cloud benefits - scalability, payment for the consumption, reliability, as well as security. All the instances are managed through a Control Center. One or more PMML models can be loaded in an instance, validated against the PMML schema, interactively fix the potential problems and deploy the model. Once deployed the model can be used as a web service (for which WSDL is generated on the fly) or interactively. This allows the consumer to upload and run data - either to score it or to match it against expected scores for testing. Scored data can be downloaded in CSV format.

6.3 Comparison to DeVisa’s approach

Compared to the related work described in 6.1, DeVisa is particular in the following aspects:

- DeVisa does not store the data the models were built on, but only the PMML models themselves;
- The DeVisa approach leverages the XML native storage and processing capabilities like indexing (structural, full text and range), query optimization, inter-operation with the XML based family of languages and technologies;
- DeVisa defines a XML based query language - PMQL - used for interaction with the DM consumers. PMQL is wrapped in a SOAP message, interpreted within DeVisa and executed against the PMML repository;
- It provides a XML-based language for expressing the Metadata Catalog;
- DeVisa deals with schema integration aspects in the scoring process and provides a 1 : 1 schema matching technique and an adaptive similarity measure;
- Uses a functional dependency approach in verifying if a consumer’s schema can be derived from the existing schemas in DeVisa;
- DeVisa allows online composition of prediction models either during the scoring process or explicitly;
6.3 Comparison to DeVisa’s approach

- The interoperability with other applications (e.g. consumers) is achieved exclusively through the use of web services;

- DeVisa integrates a native XQuery library for processing PMML documents.
Chapter 7

Implementation Details

7.1 Overview of the Underlying Technologies

7.1.1 Native XML Database Systems

A native XML database is a database that defines a logical model for an XML document and stores and retrieves documents according to that model. The model should comprise at least elements, attributes, PCDATA, and document order. Examples of such models are the XPath/XQuery data model (XDM (2007)), the XML Infoset, the logical structure of documents and the way a document is accessed and manipulated defined by DOM (Hors et al. (2004)) and the events in SAX (** (2004a)), or the pull parsing model implied by StAX (*** (2007a)). A XML database an XML document as its fundamental unit of (logical) storage, just as a relational database has a row in a table as its fundamental unit of (logical) storage. Is not required to have any particular underlying physical storage model. For example, it can be built on a relational, hierarchical, or object-oriented database, or use a proprietary storage format such as indexed, compressed files. An overview on the leading XML databases products can be consulted in Gorea (2005a), as well as potential uses on XML native databases in nowadays applications (Gorea (2005b)).
7.1 Overview of the Underlying Technologies

7.1.2 XQuery

XML (XML) is a versatile language, capable of describing the information content of diverse data sources including structured and semi-structured documents, relational databases, and object repositories. A query language that uses the structure of XML intelligently can express queries across all these kinds of data, whether physically stored in XML or viewed as XML via middleware.

XQuery (XQuery) is a functional language designed for querying XML documents and it has become a W3C recommendation as of January 2007. The language has the following distinctive properties: composition, closure, schema conformance, XPath compatibility, completeness, conciseness, and static analysis. XQuery complies to the requirement that an XML query language have both a human-readable syntax and an XML-based syntax. XQuery operates on the abstract, logical structure of an XML document, rather than its surface syntax. This logical structure is known as the data model (XDM (2007)). Currently at version 1.0, XQuery is an extension of XPath Version 2.0 (XPa (2007)).

It is flexible enough to query a broad spectrum of XML information sources, including both databases and documents, which makes it suitable for wide use in applications. DeVisa leverages the power of XQuery to implement efficient processing of the PMML prediction models in the repository.

7.1.3 eXist

eXist (eXist, Meier (2002)) is a native XML database engine featuring efficient, index-based XQuery processing, automatic indexing (structural, full-text, range), extensions for full-text search, XUpdate support, XQuery update extensions and tight integration with existing XML development tools. eXist provides basic database functions such as storing and retrieving XML or binary resources and XQuery/XPath querying via the XMLDB (** (2003b)), XMLRPC(** (2003a)) or SOAP (** (2007b)) interfaces.

eXist supports calls to arbitrary Java methods from within XQuery. The binding mechanism follows the short-cut technique introduced by Saxon - a collection of tools for processing XML documents (Kay (2008)).
eXist provides its own XQuery implementation, which is backed by an efficient indexing scheme at the database core to support quick identification of structural node relationships. The indexing scheme forms the very core of eXist and represents the basis of all efficient query processing in the database. The main purpose of the indexing scheme is to allow a quick identification of structural relationships between nodes. To determine the relationship between any pair of given nodes, the nodes themselves don’t need to be loaded into main memory. This is a central feature since loading nodes from persistent storage is in general an expensive operation. For eXist, the information contained in the structural index is sufficient to compute a wide-range of path expression.

This functionality is crucial for the implementation of DeVisa since it strongly influences the performance of the PMQL engine while solving PMQL consumer goals (3.3.2).

### 7.1.4 Web Services

Apache Axis2 [Axi] (2008) is the successor to the Apache Axis SOAP project, providing a major improvement of the Web services core engine. Apache Axis2 provides a complete object model and a highly flexible modular architecture that makes it easy to add functionality and support for new Web services-related specifications and recommendations.

Axis2 enables performing the following tasks: send SOAP messages with or without attachments, receive and process SOAP messages, retrieve the WSDL for a service, create or utilize a REST-based Web service, create or utilize services that take advantage of the WS-Security, WS-ReliableMessaging, WS-Addressing, WS-Coordination, and WS-Atomic Transaction recommendations etc.

Being a clean and extensible open source Web services platform, it aims to be the platform for the next generation of Web services and Service-Oriented Architecture - SOA [Erl] (2005).
7.2 Current DeVisa Implementation

The main challenge in developing DeVisa is obviously the PMQL engine, that, among others performs the PMML scoring. Compared to other scoring engines, it is more flexible providing extra functionality such as: schema matching within the scoring process and composition of prediction models. From a technology perspective, it is able to natively process the PMML models using the XQuery processing and indexing facilities offered by eXist.

At the time of writing this material, various parts of the DeVisa system are under development and the current status of the implementation can be checked out directly on the code repository on sourceforge.net (***(2008c)). The details of the implementation are presented hereunder.

From a functional design perspective, in Appendix C the main use cases of the system are presented: scoring, explicit composition, search. The specifications of the PMQL language expressed in XML Schema can be consulted in A. I also designed the architecture of the system, which is described in various parts of the present work. Also, the internal specification of the system is described, such as the Metadata catalog, which is also specified in XML Schema (Appendix B). The object model of the PMQL engine is also described in Appendix C.

Some of the horizontal cross-cutting aspects of the system are implemented: web service front-end, a skeletal implementation of PMQL engine and its components (annotator, rewriter, plan builder and plan executor) PMQL validation against its schema, schema matching strategy described in section 5.5, as well as the similarity measure described in 5.6.

The PMQL-LIB library includes an set of XQuery scoring functions - at this moment only for tree models-that operates directly on the PMML models. These functions are stored in the eXist XML database, like stored procedures.

DeVisa also includes a library that builds PMML based on Weka classifier model classes using the Java Reflection API. It represents work in progress and at the time of writing of this material it provides support only for several classification models (trees, naive bayes, rule set) and association rules. On the other side, in the Weka Pentaho project, support for PMML in Weka is in progress at
the time of writing this material. This module is likely to be more comprehensive and further supported along with future releases of the PMML specification.

In order to provide a proof of concept some of the DeVisa functions are developed also as Pentaho Data Integration plugins, as described in the concrete use case presented in 8.5. These are the prediction voting component, the global data dictionary or the schema matcher. These components are adapted to conform to the metadata constraints present in the PDI data flows and hence are compatible with the other PDI transformation steps. This offers the users the possibility of development of complex data integration projects that include prediction and decision support capabilities.
Chapter 8

Applications

8.1 DeVisa in a SOA

Service Oriented Architectures (or SOAs) are architectures meant to integrate autonomous software applications based on the business processes that they service. It is irrelevant whether the autonomous software applications are designed to interact with one another (loosely coupling). This is achieved through a typical property of a SOA, namely an Enterprise Server Bus (ESB). The ESB is in the most basic form a layer in the SOA that routes XML messages from one software component to another based on the content of the XML message. The routing of the message, thus the description of the recipient(s) based on the values of an xml schema, is defined in a message book, being part of the ESB. The ESB can also perform transformation of data offered by the sender of an XML, before it is offered to the intended recipient.

There are two ways of implementing a SOA: (1) an implementation of business logic, data layer and user interfaces designed to interoperate with one another; (2) an implementation that enables software applications to service business processes without them being designed to interoperate with one another.

For the implementation of business logic that is designed to operate with one another there is the pre-requisite that the interface of business logic and user interface is done through XML, for instance through the use of web services. The user interface will interact with one or more business logic components without the user being aware of the back-end that is servicing him or her. The back-end
processes will execute transactions on one database, either being one enterprise database, distributed databases or a federated database. The routing of the XML message from UI to a back-end component as well as between the back-end component and the UI and/or database is done via the ESB based on the content of the XML as mentioned above.

For the integration of applications that were not designed to interoperate with one another the only thing that needs to be done is making sure that there is an interface between the application and the ESB, either done through a generic web service that most ESBs offer, or by developing a connector with the ESB. This scenario is used most of the times when implementing SOAs and it is the textbook example of the integration of loosely coupled applications. It is called loosely coupled as the different applications have no interfaces with one another (thus making it possible to replace applications very easily) but together service a business process dictated by the way the XML schema through which they communicate is agreed upon.

As the services in DeVisa are published as web services it is by default able to service in a SOA as for instance a decision support component. The data mining producers or consumers can be nodes in the SOA as well. The producers should be able to communicate through PMML. The consumers should be able to communicate through PMQL.

In Alboia et al. (2008a) an SOA-based architecture of an e-health system is presented, in which DeVIsa is integrated as a module to provide medical prediction support. Additional details on the architecture on the mentioned e-Health
8.2 DeVisa as a Knowledge Repository for Learning Interface Agents

An interface agent is a semi-intelligent, semi-autonomous computational system which assists a user with a computer application. Such an agent is called a learning agent if it monitors and memorizes the user’s actions and preferences, discovers regularities in these, and learns to automate them. Other generic properties of an agent are: context-sensitivity, initiative, interaction with the environment or other agents.

The interactive user interfaces are multi-agent environments determining the following behavioral effects:

- The interface adapts to the requirements of different agents
- The interface learns new techniques and concepts
- The interface anticipates user’s intentions
- The interface can justify its actions in front of the user

In such an environment the interface agents assist the user in taking actions, compile user’s profile, infer information on user’s needs through communication and observation, translate user’s requests and selecting other agents capable of solving them, formating and presenting the results to the user. All these actions...
are transparent to the user, he/she is not aware of the structure of the system that is interacting with, of the agents’ learning process or the cooperation among agents.

The paper Gorea (2005c) presents a learning solution in a multi-agent system that offers to the user a friendly platform-independent customized web interface and which allows the integration of the web resources in an uniform manner. In general, one of the learning methods for agents is based on classification. The agent holds a minimum set of base knowledge and learns through observing and mimicking the user, adapting based on user’s feedback, consulting other agents, consulting its own experience or other agents’ experience. This way a high degree of customization can be achieved, comparable with the one obtained through man’s continuous intervention, but eliminating the necessity of explicit programming through rules or scripts. This approach has also disadvantages related to agents’ slow learning curve, who need a high number of examples before notable predictions can be made. In addition, there are issues when agents face completely new situations. Learning from the experience of other agents can overcome this problem, because it shortens the learning period and provides broader and more reliable knowledge.

Devisa can be used as a knowledge repository that stores agents’ experience and which can be consulted within the agents’ learning process. The agents consult the knowledge stored in DeVisa through web services (scoring, administration).

The main information that can contribute to building the repository is:

- information on user’s world: the environment, the type of relationships that the user has, his/her interests, the history regarding the access to a certain document etc;

- direct or indirect information on the external world;

- the available agents and their capabilities: the tasks that they can execute, the resources they need and their availability
8.3 Application of DeVisa in the microRNA discovery problem

In the last years, the information stored in biological sequence databases grew up exponentially, and new methods and tools have been proposed to get useful information from such data.

One of the most actively researched domains in bioinformatics is the microRNA discovery, i.e. discovering new types of micro-RNA in genomic sequences. There has been a great amount of published significant results concerning micro-RNA discovery with Decision Tree Classifiers Ritchie et al. (2007), Support Vector Machines as in Sung-Kyu (2006) and Sewer (2005) etc., Hidden Markov Models Nam (2005) etc., Association Rules, etc.

All these results are independent and, as far as known at this moment, there is no system that integrates all these results in a semantic manner. There is a large availability of prediction models in the scientific community - like the ones listed above, public databases of known micro-RNAs as miRBase, e.g Griffiths-Jones et al. (2006), public web ontologies as Gene Ontology GO, or online services for characterizing genomic sequences (for instance RNAFold that analyses RNA secondary structure).

DeVisa will be used to integrate all the knowledge resulting from sparse scientific work concerning the micro-RNA discovery problem into a web knowledge base platform that is easily integrable into the future work in the field. Gorea (2007) and Gorea (2008a) provide a few possible implementation ideas in this direction.

8.4 Other Applications

Gorea & Felea (2005) proposed a prototype system for predicting reliability of customers that can be exploited by a banking application or single users. The system integrates analytical data from multiple data sources by executing predefined queries over those data sources. The result sets can be processed to build prediction models, which are stored in a predictive model repository (DeVisa) in order to take advantage of the full database functionality. A banking application
can consult DeVisa services to map a set of values (customer profile) into a categorical class of an output attribute that represents a quantification of the customer reliability. A remote user can launch request to the system as well by entering personal information via a web form and obtain the classification attribute based on the values entered in the form. The banks will certainly benefit from embedding into their applications a method to predict reliability of a prospective customer when according a loan. As well as the analysis of the necessary documents of the customer, an additional and convenient criterion is statistics-based. The situation when the customer meets all the bank conditions and can be eligible for the credit is addressed. Although a straightforward decision can be taken in this case, nevertheless often in practice resources are limited and predicting the reliability of a potential customer is of considerable importance, especially when it is a matter of choosing the most competitive alternative. Furthermore, the bank could request for additional guaranty or decrease the value of the loan if the outcome of the prediction is not satisfactory.

An intranet is an enterprise-wide information distribution system that uses Internet tools and technology. The level of architecture or technology complexity varies depending on the number of users or the business logic. An intranet is used to give employees access to the documents, to distribute software, enable group scheduling, publish communication provide an easy front to the database (backend), let individuals or departments publish the information they need to communicate to the rest of the company.

Physically, an intranet is formed by linking the various pieces of information and communication technologies that an organization owns or uses, so that the any resource is available to anyone who needs it at any time. An Intranet encourages the members of the organization to make better and more informed decisions.

The core of an intranet system remains invariable, regardless of the rapid evolution of technical and software advancements. It is provided by the content, namely the internal knowledge asset (IKA). This is the enterprise intellectual property, its employees knowledge and expertise, being unavailable to the general public and is produced through the efforts of an internal community. IKA may include business strategies, market trend analyses, financial information,
internal document workflow information, internal rules and policies, operation manuals, inter-departmental communiques, or details regarding specific projects and contracts. Unlike external sourced information, IKA is highly focused and very specific to the enterprise logic, having the most added value, because it is produced internally by the specialists in the discipline who are aware of the enterprise requirements.

The works Gorea (2006b), Gorea & Moreau (2006), Gorea (2006a) and Gorea & Felea (2006) investigate dynamic intranet architectures based on the integration of disparate independent software components that communicate in a service-oriented manner, as well as different use cases. In this respect DeVisa can be seen as an independent knowledge module that collects predictive capabilities from internal producers. A predictive analysis application can turn the existing information within a corporation into a business advantage through data warehousing and data mining. Per implementation perspective, there are many synergies between intranet and data warehousing. The predictive application functions in conjunction with top-level strategies and the cooperation of the technical and other business entities within the corporation. An architecture to coordinate such an evolution is needed, one potential implementation solution being SOA-based (8.1).

8.5 A Concrete Use Case

The purpose of this concrete use case is to investigate potential situations in which enabling integration facilities within the scoring process broadens the applicability of the prediction models and provides the premises of getting better results than scoring on individual prediction models.

I considered several data sets available on the UCI Machine Learning Repository (Asuncion & Newman (2007)). The models built on the data sets form a knowledge base in the field of heart related issues (heart disease, heart attacks etc). The metadata of each data set corresponds to a data dictionary in DeVisa. There are several prediction models built on subsets of each of the data dictionaries, some of them obtained from the DMG web site (produced using data mining.
application like SPSS Clementine, IBM DB2) and the others are built using Weka for the purpose of this example.

Using the system involves performing domain-specific tasks and therefore the users should be domain experts. For instance, in the heart-related databases and prediction models the users should be doctors.

### 8.5.1 Description of the Data Sets

One of the data sets that are used in the example is the heart disease data set from the UCI machine learning repository, which was created by Andras Janosi, William Steinbrunn (University Hospital, Zurich, Switzerland), Matthias Pfisterer (University Hospital, Basel, Switzerland), and Robert Detrano (Cleveland Clinic Foundation).

This database contains 76 attributes, but all published experiments (such as Detrano et al. (1989), Gennari et al. (1989)) refer to using a subset of 14 of them. The attribute set includes: age, sex, chest pain type, chest pain location, cholesterol, resting blood pressure, smoking history, ECG results, exercise test results, fluoroscopy results etc. The predicted field refers to the presence of coronary artery disease (CAD) in the patient. Coronary artery disease (CAD) is a condition in which plaque (a combination of fat, cholesterol, calcium and other components of the blood) builds up inside the coronary arteries that supply the heart muscle with oxygen-rich blood. The predicted attribute expresses either the absence of the disease (< 50 % diameter narrowing) or the presence (> 50 % diameter narrowing). The database contains both numeric and categorical attributes. The presence can be expressed also as the degree in which the patient is affected (3 levels). Experiments with the Cleveland database have concentrated on simply attempting to distinguish presence from absence.

The heart disease data set is horizontally fragmented into patient data collected from patients in Cleveland, Switzerland, Hungary or Long Beach. There are different flavors of the Cleveland data set (the most used in the publications) in which values are expressed either as symbolic or as numeric.

The second data set that is integrated in the experiment is the arrhythmia data set (Guvenir et al. (1997)). This database contains 279 attributes, 206 of
which are linear valued and the rest are nominal. The active attributes represent ECG recordings and general attributes such as age, sex, weight or height.

The aim of this work is to distinguish between the presence and absence of cardiac arrhythmia and to classify it in one of the 16 groups. The first class refers to normal rhythm, the classes from second to fifteenth refer to different classes of arrhythmia (ischemic changes, myocardial infarction, tachycardia, bradycardia, premature contraction etc) and last class refers to the unclassified instances. The use of machine learning tools in this topic aims to minimize the differences between the cardiologist’s and the decision support software’s classification results using the cardiologist’s results as a gold standard.

The third data set which is taken into consideration is the Echocardiogram data set (Salzberg (1988)), that contains the patients that suffered heart attacks at some point in the past. Some are still alive and some are not. The survival and still-alive variables, when taken together, indicate whether a patient survived for at least one year following the heart attack. The problem addressed by past researchers was to predict from the other variables whether or not the patient will survive at least one year. The most difficult part of this problem is correctly predicting that the patient will not survive.

The aforementioned data sets originate from independent sources and therefore they might use different naming, value representation or degrees of granularity for the same attributes. On the other side relations of causality can occur between the predicted attributes of the models built on different data sets. For instance, it is known in the medical field that over time, CAD can weaken the heart muscle and lead to heart failure and arrhythmia. Conversely, there is a type of arrhythmia characterized by ischemic changes, that can lead to the conclusion that the patient suffers of CAD. A substantial problem in clinical cardiology is the gap in the ability to detect asymptomatic individuals at high risk for CAD for preventive and therapeutic interventions (Clarke & Seo (2008)). Up to 75 % of such individuals are designated as low to intermediate risk by standard CAD risk assessment models; however, a substantial number of such individuals who are actually at increased risk may not be identified. Over 50 % of individuals with sudden cardiac death have no prior symptoms of CAD. Therefore, it is likely that the traditional risk factors do not account fully for CAD risk. Recently the role
of role of genes and gene variants in the development of atherosclerosis is investigated and candidate genes with a strong statistical correlation with vascular atherosclerosis have been discovered.

8.5.2 The Front End

DeVisa is meant to provide integrated prediction services to consumer applications via web services. Nevertheless in order to illustrate the usage of DeVisa in integrated scoring of prediction models built on the data sets described in 8.5.1 we will use a front end component.

Pentaho Data Integration (PDI, also called Kettle) (***(2008a)*) is a software component for the Extract, Transform and Load (ETL) processes. Though ETL tools are most frequently used in data warehouses environments, PDI can also be used for other purposes: migrating data between applications or databases, exporting data from databases to flat files, loading data massively into databases, data cleansing, integrating applications. It consists of three distinct applications. For the purpose of this use case we are interested in a graphical end-user interface called Spoon that allows modeling the flow of data from input through transformation to output. It helps designing and testing of the integration processes through the use of jobs and transformations.

One of the interesting features of PDI is that it is model-driven. The transformations and the jobs are high level integration processes that are created declaratively. The models (transformations or jobs) can be interpreted and executed by different PDI tools, in fact executing the implied ETL processes.

PDI presents an extensible plug-in architecture in which new functionality can be added to the existing one. Therefore it becomes a suitable tool for combining data integration with scoring on prediction models. Besides the built-in PDI functions in our tests we use the following plug-ins: Weka Scoring, Univariate statistics. Moreover we extended the functionality with other DeVisa specific plug-ins, such as automatic schema matching, several procedures for implementing prediction output selection, global data dictionary mediator, predictive model repository access.
The DeVisa system itself does not have a user interface, all its functionality being available as web services. Therefore such an extensible plug-in based architecture specialized on integration and business intelligence can easily incorporate DeVisa functions and provide the same functionality to a human user. This approach has the advantage of being more flexible since the user can validate the results immediately and try alternative ways of achieving his/her goals. PDI also allows plug-ins that call web services, which makes it suitable as a remote client (consumer) for DeVisa services.

### 8.5.3 Methods and Results

The general idea is to provide to domain experts a set of visual methods of obtaining potentially better results by using tools for mediating model differences and for composition of various classifiers.

As mentioned 8.5.1, the heart disease data set is horizontally fragmented based on geographical localization of the patients. We make a set of basic assumptions:

1. The fields may be named differently in different partitions and it is possible that different data types were used to represent the values of the respective fields.

2. In each of the partitions various prediction models have been built that act on vertical fragments of the data set. From the total of 76 attributes, the models can use projections on some subsets of these attributes, which can be overlapping or not.

3. Even for common attributes the distribution of values differs per partition. As it will be seen later in this chapter, some of the attributes have a higher percent of missing values than the others, or data is sampled from different ranges of values.

One of the possible use cases is when we have a set of records belonging to patients in a different location for which prediction models have not been built. Based on the assumption that diseases are influenced by geographical locations, one can use a heuristic like choose the prediction model built on patient data
closest to the location of the test data. However this is not applicable when we
do not have the actual location of the patients. Furthermore we can encounter
schema heterogeneity under several forms: the new data has its own schema or
the schemas of the existing models and the consumer overlap partially.

What we try to do then is apply schema matching techniques and to combine
the results of the individual prediction models. Combining the results of various
prediction models has several facets:

1. We can apply different models that predict the same attribute on the new
data (each of the models can be applied on subsets of the original data) and
at the end combine the results through a voting procedure. Before sending
the data to a model a matching algorithm might need to be applied.

2. We can apply the models in a sequence to achieve the final prediction, i.e.
one model can predict an attribute that can be the input of another model
etc. This situation requires a more detailed domain database in which
correlation between various attributes can be established and represented
in the prediction models. In our use case we did not test this situation yet,
although it is possible to apply it in DeVisa and it is part of the future
work along with enriching the data set and building new prediction models
in the same field.

8.5.3.1 Model Selection Use Case

This use case operates only on the heart disease data dictionary, which contains in
total 76 attributes. Both consumer’s data and the considered models are defined
on the same data dictionary. The consumer’s data set is a vertical projection
on the Switzerland heart disease data set and contains 24 attributes, which are
shown in figure 8.3. We considered a number of 78 of data instances from the
Switzerland heart disease data set.

There are three classification models in the repository that can be applied
on overlapping subsets of the consumer’s metadata and have the same predicted
attribute as the consumer’s schema. We considered a classification tree model, a
neural network (multi layer perceptron) and a logistic regression model.
8.5 A Concrete Use Case

Figure 8.3: The consumer’s metadata and univariate statistics for numeric fields.

The J48 tree model is trained on 294 instances of the Hungarian heart data set. It is based on a subset of 18 attributes, the 19th attribute being the predicted one. The 14 attributes are a subset of the consumer’s metadata as well. Its metadata is depicted in figure 8.4.

The neural network model is built on a set of 200 data instances from the Long-Beach heart disease data set. It uses a subset of 22 attributes, plus the predicted attribute. The 22 attributes are a subset of the consumer’s metadata as well. Its metadata is depicted in figure 8.5.

The logistic regression model is built on a set of 304 data instances from the Cleveland heart disease data set. It uses the 14 attributes depicted in Figure 8.6.

As seen in the statistics, the values in similar fields on different data sets are distributed differently. The most obvious example in this sense is the field age, that differs from the consumer’s data set as opposed to the data set the models were built on. The patients in the consumer’s data set were drawn from a higher age range ([32 – 74]) compared to the others ([28 – 66]) and have a higher variance around the mean. Another example is the chol field, which is completely empty.

<table>
<thead>
<tr>
<th>FIELD NAME</th>
<th>COMMENTS</th>
<th>N</th>
<th>STD</th>
<th>MIN</th>
<th>MAX</th>
<th>MEAN</th>
<th>MEDIAN</th>
<th>10TH PERCENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>age</td>
<td>age in years</td>
<td>78</td>
<td>9.14</td>
<td>32</td>
<td>74</td>
<td>55.21</td>
<td>56</td>
<td>41</td>
</tr>
<tr>
<td>chol</td>
<td>serum cholesterol in mg/dl</td>
<td>78</td>
<td>10.69</td>
<td>20</td>
<td>40</td>
<td>28.57</td>
<td>26</td>
<td>20</td>
</tr>
<tr>
<td>cigs</td>
<td>cigarettes per day</td>
<td>7</td>
<td>3.01</td>
<td>1</td>
<td>12</td>
<td>3.06</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>cr</td>
<td>chest pain type (typical angina, atypical angina, non-anginal pain, asymptomatic)</td>
<td>78</td>
<td>1.24</td>
<td>4</td>
<td>0.78</td>
<td>1</td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td>cs</td>
<td>sex of the patient</td>
<td>78</td>
<td>2.37</td>
<td>3</td>
<td>12</td>
<td>7.06</td>
<td>9</td>
<td>5</td>
</tr>
<tr>
<td>cs</td>
<td>sex of the patient</td>
<td>78</td>
<td>12.29</td>
<td>46</td>
<td>100</td>
<td>71.03</td>
<td>71</td>
<td>55</td>
</tr>
<tr>
<td>thal</td>
<td>thal</td>
<td>78</td>
<td>4.87</td>
<td>0</td>
<td>18</td>
<td>6.70</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>thal</td>
<td>thal</td>
<td>78</td>
<td>11.16</td>
<td>55</td>
<td>110</td>
<td>84.74</td>
<td>82.5</td>
<td>70</td>
</tr>
<tr>
<td>thal</td>
<td>thal</td>
<td>78</td>
<td>24.62</td>
<td>120</td>
<td>230</td>
<td>164.67</td>
<td>160</td>
<td>135</td>
</tr>
<tr>
<td>thal</td>
<td>thal</td>
<td>78</td>
<td>20.51</td>
<td>80</td>
<td>200</td>
<td>123.44</td>
<td>120</td>
<td>100</td>
</tr>
<tr>
<td>years</td>
<td>number of years as a smoker</td>
<td>3</td>
<td>5.00</td>
<td>10</td>
<td>20</td>
<td>15.00</td>
<td>15</td>
<td>10</td>
</tr>
</tbody>
</table>
### 8.5 A Concrete Use Case

**Figure 8.4:** A subset of the Hungarian metadata on which the J48 decision tree model (DeVisa ID 21) is defined and the univariate statistics for the numeric fields.

<table>
<thead>
<tr>
<th>FIELD NAME</th>
<th>COMMENTS</th>
<th>N</th>
<th>STD DEV</th>
<th>MIN</th>
<th>MAX</th>
<th>MEAN</th>
<th>MEDIAN</th>
<th>10TH PERCENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>age</td>
<td>age in years</td>
<td>294</td>
<td>7.81</td>
<td>28</td>
<td>66</td>
<td>47.83</td>
<td>49</td>
<td>37</td>
</tr>
<tr>
<td>chol</td>
<td>serum cholesterol in mg/dl</td>
<td>271</td>
<td>67.66</td>
<td>85</td>
<td>603</td>
<td>250.85</td>
<td>243</td>
<td>182</td>
</tr>
<tr>
<td>emr</td>
<td>month of cardiac cath</td>
<td>294</td>
<td>3.21</td>
<td>1</td>
<td>12</td>
<td>6.24</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>cp</td>
<td>chest pain type (typical angina, atypical angina, non-anginal pain, asymptomatic)</td>
<td>292</td>
<td>4.64</td>
<td>1</td>
<td>24</td>
<td>10.63</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>famhist</td>
<td>family history of coronary artery disease</td>
<td>294</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>num</td>
<td>diagnosis of heart disease (angiographic disease status)</td>
<td>294</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>oldpeak</td>
<td>ST depression induced by exertion relative to rest</td>
<td>294</td>
<td>0.91</td>
<td>0</td>
<td>5</td>
<td>0.59</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>painex</td>
<td>provoked by exertion or not</td>
<td>294</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>painloc</td>
<td>chest pain location</td>
<td>294</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>restecg</td>
<td>resting electrocardiographic results</td>
<td>294</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sex</td>
<td>sex of the patient</td>
<td>294</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>slope</td>
<td>the slope of the peak exercise ST segment (upsloping, flat, downsloping)</td>
<td>294</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>smoke</td>
<td>is smoker</td>
<td>294</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>thal</td>
<td>thal</td>
<td>294</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>thalach</td>
<td>maximum heart rate achieved</td>
<td>293</td>
<td>23.59</td>
<td>82</td>
<td>190</td>
<td>139.13</td>
<td>140</td>
<td>108</td>
</tr>
<tr>
<td>thalrest</td>
<td>resting heart rate</td>
<td>293</td>
<td>15.53</td>
<td>46</td>
<td>134</td>
<td>81.03</td>
<td>80</td>
<td>63</td>
</tr>
<tr>
<td>trestbps</td>
<td>resting blood pressure (in mm Hg on admission to the hospital)</td>
<td>293</td>
<td>17.68</td>
<td>92</td>
<td>200</td>
<td>132.58</td>
<td>130</td>
<td>110</td>
</tr>
</tbody>
</table>

**Figure 8.5:** A subset of the Long Beach metadata on which the MLP model (DeVisa ID 10) is defined and the univariate statistics for the numeric fields.

<table>
<thead>
<tr>
<th>FIELD NAME</th>
<th>COMMENTS</th>
<th>N</th>
<th>STD DEV</th>
<th>MIN</th>
<th>MAX</th>
<th>MEAN</th>
<th>MEDIAN</th>
<th>10TH PERCENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>age</td>
<td>age in years</td>
<td>200</td>
<td>7.81</td>
<td>35</td>
<td>77</td>
<td>59.35</td>
<td>60</td>
<td>50</td>
</tr>
<tr>
<td>chol</td>
<td>serum cholesterol in mg/dl</td>
<td>200</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cigs</td>
<td>cigarettes per day</td>
<td>190</td>
<td>16.26</td>
<td>0</td>
<td>80</td>
<td>21.74</td>
<td>20</td>
<td>0</td>
</tr>
<tr>
<td>cp</td>
<td>chest pain type (typical angina, atypical angina, non-anginal pain, asymptomatic)</td>
<td>200</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>famhist</td>
<td>family history of coronary artery disease</td>
<td>200</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>htn</td>
<td>htn</td>
<td>200</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>num</td>
<td>diagnosis of heart disease (angiographic disease status)</td>
<td>200</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>painex</td>
<td>provoked by exertion or not</td>
<td>200</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>painloc</td>
<td>chest pain location</td>
<td>200</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>restecg</td>
<td>resting electrocardiographic results</td>
<td>200</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sex</td>
<td>sex of the patient</td>
<td>200</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>smoke</td>
<td>is smoker</td>
<td>200</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>thal</td>
<td>thal</td>
<td>200</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>thalach</td>
<td>maximum heart rate achieved</td>
<td>147</td>
<td>21.99</td>
<td>69</td>
<td>180</td>
<td>122.80</td>
<td>120</td>
<td>96</td>
</tr>
<tr>
<td>thalrest</td>
<td>resting heart rate</td>
<td>144</td>
<td>52.79</td>
<td>458</td>
<td>239.57</td>
<td>228</td>
<td>177</td>
<td></td>
</tr>
<tr>
<td>trestbps</td>
<td>resting blood pressure (in mm Hg on admission to the hospital)</td>
<td>141</td>
<td>14.39</td>
<td>11</td>
<td>120</td>
<td>88.17</td>
<td>90</td>
<td>70</td>
</tr>
<tr>
<td>years</td>
<td>number of years as a smoker</td>
<td>188</td>
<td>16.36</td>
<td>0</td>
<td>60</td>
<td>23.73</td>
<td>25</td>
<td>0</td>
</tr>
</tbody>
</table>
8.5 A Concrete Use Case

Figure 8.6: A subset of the Cleveland metadata on which the Logistic Regression model (DeVisa ID 15) is defined and the univariate statistics for the numeric fields.

in the consumer’s data.

The predicted attribute is \textit{num}, which represents the diagnosis of heart disease (angiographic disease status). In the original data sets it has 5 possible values, one of them meaning the absence of CAD (< 50% diameter narrowing), and the other 4 marking different degrees of CAD. This use case considers only models that distinguish between the presence and absence of CAD. The models produce only discrete class labels, which are combined through majority voting in a manner resembling to the bagging technique. The major difference is that while base classifiers are formed using bootstrap replicates of the learning data set, this technique uses base classifiers that are not necessarily defined on the same schema and formed independently with different learning sets.

The performance of the individual classifiers, tested using 10-fold cross validation is depicted in figure 8.7.

The use case is modeled as a PDI workflow and is depicted in the screenshot in Figure 8.8.
8.5 A Concrete Use Case

<table>
<thead>
<tr>
<th>NAME</th>
<th>ACCURACY</th>
<th>TP RATE</th>
<th>FP RATE</th>
<th>ROC AREA</th>
</tr>
</thead>
<tbody>
<tr>
<td>J48 Hungarian</td>
<td>80.270%</td>
<td>0.720</td>
<td>0.150</td>
<td>0.779</td>
</tr>
<tr>
<td>MLP Long Beach</td>
<td>68.000%</td>
<td>0.790</td>
<td>0.550</td>
<td>0.597</td>
</tr>
<tr>
<td>LogReg Cleveland</td>
<td>84.390%</td>
<td>0.784</td>
<td>0.109</td>
<td>0.895</td>
</tr>
</tbody>
</table>

Figure 8.7: The accuracy, true positive rate, false positive rate and the area under the ROC curve of the three base classifiers measured through 10-fold cross validation.

Figure 8.8: A PDI workflow representing the scoring using heterogeneous model selection based on the heart disease data dictionary.

The consumer sends a match schema classification request, allowing composition. After receiving the consumer’s data the ”Add DeVisa ID” component appends a local ID column (a surrogate primary key) to each row. Then the data is decomposed in two different relations using the ”Projection” components and scored on individual models ”Hungarian J48 ID 21”, ”Long-Beach MLP ID 10” and ”Cleveland LogReg ID 15”. After the individual predictions are appended to each row the resulting rows are joined using the primary key. After the ”Join rows by DeVisa ID” step each row contains the individual predictions and each row contains duplicate attributes originating from the two individual models. The ”DeVisa Prediction Voting” step applies a voting procedure that can be configured in the associated dialog window, as shown in the screenshot 8.9. It supports the simple method of Majority Voting and the Sum, Product,
Min, Max and Median rules. The rows are further projected to eliminate the individual predictions and the results are returned to consumer.

Running the PDI data flow in figure 8.8, I observed that the accuracy of the predictions after executing the "DeVisa Voting" step is 64.1%, which is slightly higher than the average of the accuracies of the individual classifiers tested on the consumer’s data set, which is 59.83% (these results are displayed in figure 8.10). The true positive rate is also higher than the average true positive rate, while the false positive rate situates at the maximum rate of the individual classifiers, which situates it towards the liberal side, making positive predictions with a weaker evidence.

At the moment the consumer issues its request the only available information on the performance of the models is the one in figure 8.7. The consumer can
8.5 A Concrete Use Case

opt to score on the classifier with the highest accuracy (which would be LogReg Cleveland - 84.39%), but, as the figure 8.10 shows, the classifiers would perform differently on the consumer’s data set. The LogReg achieves only 67.95% on the consumer’s data set. Of course, at the moment of its request this information is not available to the consumer, unless it uses the models solely for testing.

<table>
<thead>
<tr>
<th>NAME</th>
<th>ACCURACY</th>
<th>TP RATE</th>
<th>FP RATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>J48 Hungarian</td>
<td>30.77%</td>
<td>0.27</td>
<td>0.38</td>
</tr>
<tr>
<td>MLP Long Beach</td>
<td>80.77%</td>
<td>0.81</td>
<td>0.25</td>
</tr>
<tr>
<td>LogReg Cleveland</td>
<td>67.95%</td>
<td>0.7</td>
<td>0.5</td>
</tr>
<tr>
<td>AVERAGE</td>
<td>59.83%</td>
<td>0.593</td>
<td>0.377</td>
</tr>
<tr>
<td>DeViza Selection</td>
<td>64.10%</td>
<td>0.65</td>
<td>0.5</td>
</tr>
</tbody>
</table>

Figure 8.10: The accuracy, true positive rate and false positive rate of the three base classifiers, the averages and of DeViza selection, measured through testing them with the consumer’s data set.

If we consider the output probabilities of the individual classifiers, then we can choose the average voting method to combine the results. The performance of the resulting prediction is improved considerably and it is close to the performance of the best prediction of the individual classifiers. The ROC curves for all the 4 predictions obtained through thresholding the output probability (from 0.9 through 0) are depicted in figure 8.11. We considered 10 cutoff scores, decreasing with 0.1 at each step. As it can be observed from the figure, the MLP Long-Beach has the best predictions on the consumer’s data set (AUC 0.682), followed by DeViza, with AUC 0.673. Although it achieved good performance through cross-validation, the J48 Hungarian model performs worse that a random guess classifier on the consumer’s data.

The purpose of this use case is to illustrate the situation in which ad-hoc predictions for a completely new data set can be composed based on the existing models, which are not necessarily built with the new data set in mind. The different models in the repository are constructed on information about the patients from various geographical locations (e.g Hungary and Long-Beach). The consumer’s data originates from another location, potentially unknown (in our case Switzerland) that is not represented in the models in the repository. Considering that heart diseases tend to be influenced also by geographical trends, a
8.5 A Concrete Use Case

Figure 8.11: The ROC analysis of the predictions on the consumer’s data set of the three individual classifiers and of DeVisa late composition, measured through thresholding the output probabilities.

Prediction selection through voting is potentially the closest to the real value for the consumer’s data. As the example shows, the accuracy of the prediction on the consumer’s data is higher than the average of the individual classifiers.

8.5.3.2 Schema Match with Global Data Dictionary Use Case

This use case shows how the consumer’s schema can be matched against the schemas of the models in the DeVisa model repository during the scoring process. The schemas of the individual DeVisa models are integrated into a common schema, called the global data dictionary (GDD). The consumer’s metadata is extracted from the consumer’s request for scoring. The PDI workflow is depicted in figure 8.12.

GDD can be represented as a transformation step, having as input the metadata and optional univariate field statistics of the underlying models and the output as a common schema. In PDI every step has a unique type of input, i.e. every direct predecessor step needs to stream the same metadata to it. There-
Figure 8.12: A PDI workflow representing the schema match over consumer’s metadata and the global data dictionary during scoring over a repository of heart-related prediction models.

Before to prepare the input of the GDD step we first extract the metadata of each model and the optional univariate field statistics, then we join the obtained rows by the field name, so that the GDD step has a homogeneous set of input streams. Before entering the GDD step a model ID field is appended so that it can link back to the originating model. Then the metadata streams originating from different models are appended and further transferred to the GDD step. The GDD step itself computes the 1:1 similarities between the attributes and applies the 3.2.1 algorithm to produce the integrated schema that is used in the process of
schema matching during scoring. The integrated schema contains an association between virtual GDD fields and the original fields in the models. To each such association an marker field is added, indicating that it originates from DeVisa, so that the “DeVisa Schema Matcher” step distinguishes it from the metadata rows originating from the consumer.

On the other side, metadata is extracted and univariate field statistics are calculated from the consumer’s data stream. The rows are joined based on field names and some dummy fields are appended to achieve compatibility with the metadata stream originating from DeVisa GDD. The streams are appended to each other and the schema matcher step creates all possible valid associations between consumer’s fields and the original model fields by applying the algorithm proposed in 5.5.

The analysis of this scenario concludes that enabling integration facilities within the scoring process provides the premises of easing the KDD processes for the consumer and potentially getting better results than performing a standard KDD process. Exploiting mining models based on heterogeneous data sources can lead to discovering correlations between apparently independent facts that can lead to further discoveries in the domain.

In the future the data set will be further enriched with new data instances as well as new attributes and dependencies between attributes materialized in predictive and statistical models, so that more complex use case can be obtained.
Chapter 9

Conclusions and Further Work

In this thesis the conceptual architecture, the functionality and the formal model of an online scoring system (DeVisa) was presented. The system has been designed to provide unified access to different prediction models using standards like PMML and web services. From a theoretical perspective, the thesis uses formalisms from relational model to capture both the structural (statical) and the behavioral (dynamic) aspects of the system. The original contributions and future research directions are presented hereunder.

9.1 The Contributions of this Work

Chapter 3 introduces the architecture of a online scoring system called DeVisa in the context of knowledge as a service paradigm. The system has been designed to provide uniform and transparent access to heterogeneous prediction models using standard technologies like PMML and web services. The system assists domain experts in sharing their predictive capabilities based on their observations without disclosing the data itself. Besides basic functions such as uploading/downloading models (administration), the system provides functions such as scoring (a mean to provide knowledge capabilities), model comparison, model composition, searching on the stored models through a message style web service interface.

The chapter describes the components of the system (the service layers, the PMML repository, the metadata catalog, the PMQL library, the PMQL engine) and the interaction between the components. The predictive models in DeVisa
are provided by different producer applications, therefore the data dictionaries are potentially heterogeneous. One of the main components of DeVisa is the metadata catalog, which includes a component called global data dictionary, which behaves live a mediator between heterogeneous data dictionaries in the repositories. The section 3.2 proposes a technique of updating the global dictionary as new models are added to the repository (algorithm 3.2.1). The technique is based on re-calculating a similarity measure between the existing entities and the new ones, merging attributes if considered identical and maintaining a similarity matrix that is further used in the online request processing.

To achieve a clear separation between the consumer’s goal and the effective DeVisa services a specialized language (PMQL) is introduced. It is a XML-based language used to express communication both with the consumer applications (requests, answers) and between the internal DeVisa components. The section 3.3.1 describes the structure of a PMQL consumer request. The formal PMQL specification is presented in A.

The chapter emphasizes the main phases in which a PMQL scoring request is processed in DeVisa. The consumer sends a goal expressed in PMQL, containing the data to be scored and specifies a predictive model (exactly or laxly). The PMQL engine processes the request and uses a library of XQuery functions to effectively access the repository. The answer is expressed in PMQL as well. In the scoring process if a model satisfying the request is not found, then the engine may attempt to infer a new one (composition).

In section 3.4.3 an overview of the model composition problem is provided and the circumstances in which this problem is reflected in DeVisa.

The chapter 4 provides the theoretical foundation of the system that captures two different perspectives: structural and behavioral. The structural perspective focuses on the main entities that DeVisa is composed of: the data dictionaries, mining schemas, the metadata catalog, the model repository. The section 4.1.1 introduces the concept of mining functional dependency and the property 4.1 proves that it is stronger than the functional dependency in relational model. In consequence, the Armstrong’s set of axioms is sound and complete for MFDs. This result enables the definition of a formal framework in which new models can be derived from the ones in the DeVisa repository.
Besides the structural perspective the chapter provides a behavioral perspective, that includes the scoring process and the model composition process. The scoring process described in 4.2 is split into the component phases depending on the consumer’s request type and a formal model of each of the phases is provided. The first phase, called model discovery, is the most complex in the match schema case, in which the consumer has the possibility to describe its own schema that might not conform exactly to the ones in DeVisa repository (4.2.1.2). The property 4.2 proves that in order to find the models that are best applicable on a data set, we need to find the optimal derivable match function, which will be described further in 5.5.

In section 4.3 a theoretical foundation on the data mining models composition problem within DeVisa was presented. The work identifies the contexts in which model composition can occur (either implicit, during scoring or explicit, specified in the consumer’s goal) and analyzes the possible approaches. The formal inference framework based on the MFD concept introduced in 4.1.1 is used to achieve model composition in DeVisa. The section 4.3.1.2 that in the scoring process or implicit composition the derivability of a mining model from a data dictionary is similar with the notion of consequence of a functional dependency from a set of functional dependencies, and, therefore, similar techniques can be used for this decision problem in DeVisa.

The chapter 5 introduces a new 1:1 schema matching technique and a hybrid adaptive similarity function, both designed to be applied on mining schemas expressed in PMML.

The schema matching technique is customized to take advantage of the internal DeVisa configuration. It uses the global data dictionary as the front row mediated schema that filters out unfeasible match candidates and at the end produces a local matching between the consumer’s schema and the data dictionaries in DeVisa (5.5.1). To match the remaining attributes, a flow network modeling is used (5.5.2), but adapted to integrate additional constraints that are specific to the DeVisa configuration, such as validity and derivability. Furthermore the technique exploits the obtained network topology to further improve the efficiency of the matching technique (5.5.5, 5.5.6, 5.5.7). While applying the matching algorithm an optimal match needs to be calculated, the one with the
maximum support and confidence (5.5.3). However, the derivability restriction prevails to the optimality and therefore sub-optimal solutions are accepted provided that they satisfy the restriction. Some useful properties related to the underlying graph topology are stated and proven (such as 5.4, 5.5, 5.7,5.8). The thesis proves the correctness of the provided schema matching algorithm (property 5.5.3) and that the matching algorithm correctly solves the model discovery problem in polynomial time (property 5.10).

The similarity measure proposed in section 5.6 is calculated by aggregating several similarity criterions, such as short name, display name, data type, textual description, semantic sense, statistic information, structural information (taxonomies, ontologies) and data instances, using an adaptive technique that takes into consideration only the information available at runtime.

The chapter 7 presents the current status of implementation of the system: the PMQL specication, the metadata catalog specification, the schema matching technique implementation, similarity function the PMQL and admin web service, the XQuery scoring functions, the XQuery library, Weka PMML export.

The chapter 8 investigates the potential applications of the system in different contexts and application domains. It mainly can be integrated as a decision support module in a SOA. Application domains as e-Health systems, multi-agent interface agents systems, credit risk assessment, or bioinformatics applications are some usage examples. A concrete use case was presented using the heart-disease, arrhythmia and echocardiogram databases from the UCI machine learning repository, in which the DeVisa functionality is incorporated in a data integration visual tool (PDI) in which the scoring process is modeled through the use of data flows. In the section 8.5.3 a ROC analysis is provided for an implicit late composition use case based on the heart disease database, showing that incorporating integration facilities in the scoring process can lead to improved prediction capabilities with performance higher than the average performance of the individual predictions.
9.2 Further Research Directions

The research performed during this work has opened a suite of open research paths that can further enrich the DeVisa system. These directions are presented hereunder.

First of all, the current work has focused merely on classification. Although PMML and therefore DeVisa can store other type of predictive models, no formal way of dealing with those has been provided. An open direction is investigating the DeVisa behavioral aspects related to the clustering models.

The work introduced a formal framework for determining the derivability of a mining schema in a data dictionary using the concept of mining functional dependency. Further work will investigate the implications of other type of relational restrictions, such as multi-valuate dependencies or inclusion dependencies.

In the scoring process, if several match functions are found and no other constraint / recommendation is found, the system returns several answers to the consumer. The consumer needs to perform a manual validation process to choose the right mapping. If a feedback would be given back to DeVisa, it should be able to store a schema alignment (match functions, according to the terminology used in 5.5) for future use. Future work will include a match feedback mechanism and will update the schema matching algorithm to consult the stored alignments.

At this moment the similarity measure assigns equal weights to the available match criteria. In the future the technique is going to be refined so that employs probabilistic rule sets to adapt the weighting scheme to achieve better accuracy. Also the matching technique will be refined to seamlessly incorporate the additional constraints. Another research direction is improving the schema matching strategy to allow $n:1$ mappings.

The future work will include the investigation of a clustering technique for creating and maintaining the GDD, i.e it will behave like a partition matrix, in which entities are mapped to clusters with a certain degree of confidence. The schema matching technique will be likely to take advantage of such a representation.

The freshness of a model is one of the factors where the accuracy of a model strongly depends on. In the current implementation the freshness of a model is a feature that depends only on a model producer. DeVisa cannot control
that aspect, so models can get outdated very easily. However the consumer can specify the freshness and DeVisa can rank the models from that point of view. One possible solution is that DeVisa will trigger the upload of the models using web service interfaces of model producers.

In the DeVisa system the collection of models together with the related ontologies form a knowledge base, which aims to be integrated in the context of Semantic Web applications. A good research direction is designing a technique of ensuring consistency of the knowledge base. Each upload triggers a check-up against the existing knowledge base and a conflict might occur. There are a few ways of dealing with conflicts: reject the whole knowledge base (FOL approach, but unacceptable under the web setup), determine the maximal consistent subset of the knowledge base, or use a para-consistent logic approach. The knowledge base facilitates that new knowledge is derived from it, so that DeVisa can include in the future rule management and inference engines (RuleML).

Another item which is part of the DeVisa roadmap is extending the PDI tool to incorporate most of the DeVisa functionality so that interoperability and sharing of prediction capability can be manipulated by end users. One of the main benefits of incorporating Devisa functions in such a front end is the broad accessibility of the knowledge resulted from the application of prediction models. The front end can be shifted to a web setup, that can lead to evolution of user communities interested in modeling the same domain and provides the premises that the prediction models are refined and improved through the action of collective expertise.
Appendix A

PMQL Specification

```xml
<xs:schema targetNamespace="http://www.infoiasi.ro/dgorea/devisa/pmql"
  elementFormDefault="unqualified"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns="http://www.infoiasi.ro/dgorea/devisa/pmql"
  xmlns:dm="http://www.dmg.org/PMML-3_2">

  <xs:import namespace="http://www.dmg.org/PMML-3_2"
    schemaLocation="http://www.dmg.org/v3-2/pmml-3-2.xsd"/>

  <xs:complexType name="DataType">
    <xs:sequence>
      <xs:element name="instance" minOccurs="1" maxOccurs="unbounded">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="field" minOccurs="1"
              maxOccurs="unbounded">
              ...
            </xs:element>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```
<xs:complexType name="ModelSpecType">

  <xs:choice minOccurs="1" maxOccurs="1">
    <xs:element name="Model">
      <xs:annotation>
        <xs:documentation>
          A reference to an existing DeVisa Model
        </xs:documentation>
      </xs:annotation>
      <xs:complexType>
        <xs:attribute name="ref" type="xs:string" use="required"/>
      </xs:complexType>
    </xs:element>
  </xs:choice>
</xs:complexType>
<xs:element name="MiningSchemaRef">
  <xs:annotation>
    <xs:documentation>
      A reference to an existing DeVisa Mining Schema
    </xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:attribute name="ref" type="xs:string"/>
  </xs:complexType>
</xs:element>

<xs:sequence>
  <xs:annotation>
    <xs:documentation>Exact Data Dictionary Specification: A list of DeVisa data dictionaries where the schema is going to be looked up is provided.</xs:documentation>
  </xs:annotation>
  <xs:element name="DataDictionaryRef" minOccurs="0" maxOccurs="unbounded">
    <xs:complexType>
      <xs:attribute name="ref" type="xs:string"/>
    </xs:complexType>
  </xs:element>
</xs:element>

<xs:element ref="dm:MiningSchema"/>
Lax Schema Specification: A data dictionary and a mining schema is described. The engine needs to look for a matching data dictionary and, within the data dictionary, for corresponding matching mining schema.
<xs:enumeration value="clustering"/>
<xs:enumeration value="rules"/>
</xs:restriction>
</xs:simpleType>
</xs:attribute>
</xs:complexType>

<xs:complexType name="RequestType">
  <xs:choice>
    <xs:element name="score" type="ScoreRequestType"/>
    <xs:element name="search" type="SearchRequestType"/>
    <xs:element name="compare" type="CompareRequestType"/>
    <xs:element name="compose" type="ComposeRequestType"/>
    <xs:element name="statistics" type="xs:anySimpleType"/>
  </xs:choice>
</xs:complexType>

<xs:complexType name="ScoreRequestType">
  <xs:sequence>
    <xs:element ref="modelSpec" minOccurs="1" maxOccurs="unbounded"/>
    <xs:element ref="data" minOccurs="1" maxOccurs="1"/>
  </xs:sequence>
  <xs:attribute name="function" type="xs:string"/>
</xs:complexType>
<xs:complexType name="SearchRequestType">
  <xs:sequence>
    <xs:element ref="modelSpec"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="CompareRequestType">
  <xs:sequence>
    <xs:element ref="modelSpec" minOccurs="2" maxOccurs="2"/>
  </xs:sequence>
  <xs:attribute name="type" use="required">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="syntactic"/>
        <xs:enumeration value="semantic"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
</xs:complexType>

<xs:complexType name="ComposeRequestType">
  <xs:sequence>
    <xs:element ref="modelSpec" maxOccurs="1" minOccurs="1"/>
  </xs:sequence>
  <xs:element ref="data" maxOccurs="1" minOccurs="0"/>
  <xs:attribute name="type" use="required">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="syntactic"/>
        <xs:enumeration value="semantic"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
</xs:complexType>
<xs:restriction base="xs:string">
    <xs:enumeration value="selection"/>
    <xs:enumeration value="sequencing"/>
    <xs:enumeration value="stacking"/>
</xs:restriction>
</xs:simpleType>
</xs:attribute>
</xs:complexType>

<xs:complexType name="AnswerType">
    <xs:choice>
        <xs:element name="score" type="ScoreAnswerType"/>
        <xs:element name="search" type="SearchRequestType"/>
        <xs:element name="compare" type="CompareRequestType"/>
        <xs:element name="compose" type="ComposeRequestType"/>
        <xs:element name="statistics" type="xs:anySimpleType"/>
    </xs:choice>
</xs:complexType>

<xs:complexType name="ScoreAnswerType">
    <xs:sequence>
        <xs:element ref="data"/>
    </xs:sequence>
</xs:complexType>
</xs:complexType>

<xs:complexType name="CompareAnswerType">
    <xs:choice>
    </xs:choice>
</xs:complexType>
<xs:element name="sintactic">
    <xs:complexType>
        <xs:attribute name="identical" type="xs:boolean"/>
    </xs:complexType>
</xs:element>

<xs:element name="semantic">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="schemaCompatible" type="xs:boolean"/>
            <xs:element name="functionCompatible">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="function">
                            <xs:simpleType>
                                <xs:restriction base="xs:string">
                                    <xs:enumeration value="identical"/>
                                    <xs:enumeration value="different"/>
                                </xs:restriction>
                            </xs:simpleType>
                        </xs:element>
                        <xs:element name="algorithm">
                            <xs:simpleType>
                                <xs:restriction base="xs:string">
                                    <xs:enumeration value="identical"/>
                                    <xs:enumeration value="different"/>
                                </xs:restriction>
                            </xs:simpleType>
                        </xs:element>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
</xs:element>
</xs:element>
</xs:sequence>  </xs:complexType>
</xs:element>
</xs:sequence>  </xs:complexType>
</xs:element>
</xs:choice>
</xs:complexType>

<xs:element name="modelSpec" type="ModelSpecType"/>
<xs:element name="data" type="DataType"/>
<xs:element name="request" type="RequestType"/>
<xs:element name="answer" type="AnswerType"/>

<xs:element name="pmql" type="PmqlType"/>
</xs:complexType name="PmqlType">
  </xs:choice>
    <xs:element ref="request"/>
    <xs:element ref="answer"/>
  </xs:choice>
</xs:complexType>
</xs:complexType>
</xs:schema>
Appendix B

Catalog Schema

<?xml version="1.0" encoding="UTF-8"?>
<schema targetNamespace="http://www.infoiasi.ro/~dgorea/devisa/metadata"
  elementFormDefault="qualified" xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:dm="http://www.infoiasi.ro/~dgorea/devisa/metadata">
  <include schemaLocation="types.xsd"/>
  <element name="catalog">
    <complexType>
      <sequence>
        <element ref="dm:collections" minOccurs="1" maxOccurs="1"/>
        <element ref="dm:producers" minOccurs="1" maxOccurs="1"/>
        <element ref="dm:DataDictionaries" minOccurs="1" maxOccurs="1"/>
        <element ref="dm:models" minOccurs="1" maxOccurs="1"/>
      </sequence>
    </complexType>
  </element>
</schema>
<element name="collection">
  <complexType>
    <sequence>
      <element name="description" type="string"/>
    </sequence>
    <attribute name="id" type="ID" use="required"/>
    <attribute name="uri" type="anyURI" use="optional"/>
    <attribute name="localName" type="string" use="required"/>
  </complexType>
</element>

<element name="collections">
  <complexType>
    <sequence>
      <element ref="dm:collection" maxOccurs="unbounded" minOccurs="1"/>
    </sequence>
  </complexType>
</element>

<element name="modelRef">
  <complexType>
    <sequence>
      <element name="description" type="string" minOccurs="0" maxOccurs="1"/>
      <element ref="dm:modelStats" minOccurs="0"/>
    </sequence>
  </complexType>
</element>
maxOccurs="1" />
<element name="MiningSchema" type="dm:MiningSchemaType"
    minOccurs="0" maxOccurs="1" />
</sequence>
<attribute name="name" type="string"/>
<attribute name="function" type="IDREF" use="required"/>
<attribute name="algorithmType" type="IDREF" use="required"/>
<attribute name="uri" type="anyURI"/>
<attribute name="collection" type="IDREF"/>
<attribute name="dataDictionary" type="IDREF"/>
</complexType>
</element>

<element name="modelStats">
    <complexType>
        <sequence>
            <element name="instancesNumber" type="int"/>
        </sequence>
    </complexType>
</element>

<element name="dataDictionaries">
    <complexType>
        <sequence>
            <element ref="dm:dataDictionaryRef"/>
        </sequence>
    </complexType>
</element>
<element name="dataDictionaryRef">
  <complexType>
    <attribute name="idref" type="IDREF"/>
    <attribute name="uri" type="anyURI"/>
    <attribute name="localName" type="string"/>
  </complexType>
</element>

<element name="models">
  <complexType>
    <sequence>
      <element ref="dm:modelRef" minOccurs="0" maxOccurs="unbounded"/>
    </sequence>
  </complexType>
</element>

<element name="producers">
  <complexType>
    <sequence>
      <element ref="dm:producer" minOccurs="0" maxOccurs="unbounded"/>
    </sequence>
  </complexType>
</element>

<element name="producer">
  167
</element>
<complexType>
  <attribute name="id" type="ID"/>
  <attribute name="uri" type="anyURI"/>
  <attribute name="localName" type="string"/>
</complexType>
</element>
<element name="functions">
  <complexType>
    <sequence>
      <element ref="dm:function"/>
    </sequence>
  </complexType>
</element>
<element name="function">
  <complexType>
    <sequence>
      <element ref="dm:algorithm" minOccurs="1" maxOccurs="unbounded"/>
    </sequence>
    <attribute name="id" type="ID"/>
    <attribute name="uri" type="anyURI"/>
    <attribute name="localName" type="string"/>
  </complexType>
</element>
<element name="algorithm">
  <complexType>
  </complexType>
</element>
<sequence>
  <element ref="dm:algorithmType" minOccurs="1" maxOccurs="unbounded"/>
</sequence>

<attribute name="id" type="ID"/>
<attribute name="uri" type="anyURI"/>
<attribute name="localName" type="string"/>
</complexType>
</element>

<element name="algorithmType">
  <complexType>
    <attribute name="id" type="ID"/>
    <attribute name="uri" type="anyURI"/>
    <attribute name="localName" type="string"/>
  </complexType>
</element>
</schema>
Appendix C

Use Cases

Figure C.1: The scoring use case.
Figure C.2: The explicit model composition use case.
Figure C.3: The model search use case.
Figure C.4: The object model of the PMQL engine component in DeVisa.
## Appendix D

### List of Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAD</td>
<td>Coronary Artery Disease</td>
</tr>
<tr>
<td>CRISP-DM</td>
<td>Cross Industry Standard for Data Mining</td>
</tr>
<tr>
<td>DD</td>
<td>Data Dictionary</td>
</tr>
<tr>
<td>DM</td>
<td>Data Mining</td>
</tr>
<tr>
<td>DOM</td>
<td>Document Object Model</td>
</tr>
<tr>
<td>ER</td>
<td>Entity Relationship</td>
</tr>
<tr>
<td>ESB</td>
<td>Enterprise Service Bus</td>
</tr>
<tr>
<td>FOL</td>
<td>First Order Logic</td>
</tr>
<tr>
<td>GDD</td>
<td>Global Data Dictionary</td>
</tr>
<tr>
<td>IML</td>
<td>Inductive Machine Learning</td>
</tr>
<tr>
<td>JDM</td>
<td>Java Data Mining</td>
</tr>
<tr>
<td>KDD</td>
<td>Knowledge Discovery in Databases</td>
</tr>
<tr>
<td>MFD</td>
<td>Mining Functional Dependency</td>
</tr>
<tr>
<td>MS</td>
<td>Mining Schema</td>
</tr>
<tr>
<td>PCDATA</td>
<td>Parsed Character Data</td>
</tr>
<tr>
<td>PDI</td>
<td>Pentaho Data Integration</td>
</tr>
<tr>
<td>PMML</td>
<td>Predictive Model Markup Language</td>
</tr>
<tr>
<td>PMQL</td>
<td>Predictive Model Query Language</td>
</tr>
<tr>
<td>SAX</td>
<td>Simple API for XML</td>
</tr>
<tr>
<td>SOA</td>
<td>Service Oriented Architecture</td>
</tr>
<tr>
<td>SOAP</td>
<td>Simple Object Access Protocol</td>
</tr>
<tr>
<td>SVM</td>
<td>Support Vector Machine</td>
</tr>
<tr>
<td>REST</td>
<td>Representational State Transfer</td>
</tr>
<tr>
<td>UCI</td>
<td>University of California - Irvine</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language</td>
</tr>
<tr>
<td>XMLA</td>
<td>XML for Analysis</td>
</tr>
<tr>
<td>XMLDB</td>
<td>XML Database</td>
</tr>
<tr>
<td>XML-RPC</td>
<td>XML Remote Procedure Call</td>
</tr>
<tr>
<td>XSL</td>
<td>XML Stylesheet</td>
</tr>
<tr>
<td>XSLT</td>
<td>XSL Transformation</td>
</tr>
</tbody>
</table>
References


REFERENCES


*** (2008a). Pentaho data integration. 137


*** (2008d). XML for Analysis. 22


ABERER, K. (2003). Special topic section on peer to peer data management. 77


REFERENCES


REFERENCES


REFERENCES


REFERENCES


183
REFERENCES


184
REFERENCES


