M^4 -The **M**ining **M**art **M**eta**M**odel

July 3, 2001

Deliverables D8, D9

Anca Vaduva[‡] Jörg-Uwe Kietz[†] Regina Zücker[†] Klaus R. Dittrich[‡]
Katharina Morik[±] Marco Botta[∓] Luigi Portinale[∓]

[‡] University of Zurich
Dept. of Information Technology
CH-8057 Zurich, Switzerland
vaduva@ifi.unizh.ch

[†] Swiss Life
IT Research & Development
CH-8022 Zurich, Switzerland
{Uwe.Kietz,Regina.Zuecker}@swisslife.ch

[±] University of Dortmund
Dept. of Computer Science
D-44221 Dortmund
morik@ls8.cs.uni-dortmund.de

[∓] Universita del Piemonte
Orientale
DISTA
I-16100 Alessandria
{botta,portinale}@unipmn.it

Contents

Abstract

Today's data mining algorithms and tools have specific input requirements which inherently demand preparation of data before their use. As a consequence, one of the most time-consuming steps in the process of knowledge discovery in databases (KDD) is data preprocessing, i.e., preparing data for data mining. Common preprocessing operations are: construction of new features derived from existing ones, adjustment of data formats, data segmentation, sampling and cleaning. The Mining Mart project proposes a case-based reasoning approach that enables both automatization of preprocessing and reusability of defined preprocessing cases for data mining applications. The system architecture follows a *metadata-driven* software approach. This technical report mainly deals with the structure of the metadata to be stored. This so called *metamodel* is the core of the system since all components have to be built in accordance with it.

M^4 has been developed as a collaboration between two projects, Mining Mart[1] and SMART[2]. The latter project deals with metadata management for data warehousing and considers metadata globally, with focus on metadata integration. The aim of SMART is an enterprise-wide metadata management system that consistenly and uniformly manages all metadata available in a company in order to provide better support for complex data warehousing processes. In this context, M^4 may be seen as part of the global metamodel behind the SMART metadata management system.

[1]http://www-ai.cs.uni-dortmund.de/FORSCHUNG/PROJEKTE/MININGMART/index.eng.html
[2]http://www.ifi.unizh.ch/dbtg/Projects/SMART/

Part I

Introduction

Chapter 1

Motivation

Extracting information and knowledge from data is the purpose of advanced technologies like data mining, data warehousing and information retrieval. Data mining combines statistical and mathematical techniques with machine learning algorithms and other artificial inteligence approaches and aims at detecting unknown patterns in data. This knowledge is then used for supporting business analysis and trend prediction. Even if a significant amount of algorithms and tools is available on the market, performing data mining is a complex task. It requires to be embedded in a whole process, the process of knowledge discovery in databases (KDD) [2]. Data has to be first collected, selected, integrated, cleaned, and further preprocessed in order to fullfil the input requirements of the chosen data mining tool or algorithm. Preprocessing operations include data transformations (e.g., data type conversion), aggregation, scaling, discretization, segmentation, sampling [5]. Practical experiences [11] showed that 50-80% of the efforts for knowledge discovery are spent for data preprocessing. However, data preprocessing is not only time-consuming but also requires profound business, data mining and database know-how.

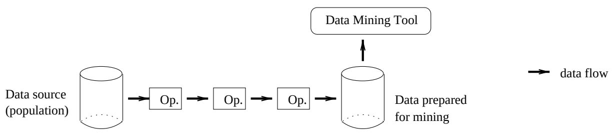

Figure 1.1: Preprocessing Chain

In this context, the aim of Mining Mart is to provide a user-friendly environment for performing preprocessing for data mining. To this end, a *case-based reasoning framework* has to be built [5]. The framework provides a collection of *cases* and tools to design these cases. A case consists of the specification of a mining task (e.g., selecting suitable addresses for a mailing action), the data to be mined, i.e. the population, and a chain of preprocessing operators to be applied to this population (see Figure 1.1). Each mining task deploys a certain mining tool or algorithm with special input requirements (see Figure 1.2) and thus the target of the chain is the data prepared in accordance with these input requirements.

A defined case may be either directly executed or reused for developing new ones: on the one hand, an end-user without any data mining and database knowledge may retrieve one of the prepared cases, make some simple adaption

- no 'unknown' (NULL) values are allowed for specific attributes
- scalar and ordinal attributes have to be numeric
- nominal attributes must have character values or be represented as sets of boolean values
- no numeric or no non-numeric attributes are admissible
- not more than N different values are allowed for nominal attributes
- always the same scale for numeric attributes is required
- no key attributes are considered
- input data must consist of a single flat table

Figure 1.2: Input restrictions of data mining tools [5]

if required (e.g., the selection of another population) and initiate the case execution. On the other hand, the highly skilled power-user (i.e., the KDD-expert) may use the framework for creating new cases. To this end, he reuses building blocks (i.e., operators) or parts of the chains available from the already defined cases.

One of the particularities of the Mining Mart approach is the realization of the framework as *metadata-driven* software (see next section). This solution enhances reusability and flexibility of the system.

The remainder of this report is organized as follows: the next section explains the notion of metadata and metadata-driven software in order to better understand Section 1.2 which generally discusses aspects of the system architecture. In Section 1.3 we present an overview of the metamodel of the repository. Parts II to III describe M^4 in more detail: each individual class with its attributes, associations and restrictions is presented. The model is structured into a data description and a case description. The data description begins with the relational model (Chapter 2), which is an excerpt or view of the data model as it exists at the data warehouse of the application. A more abstract part, the conceptual model (Chapter 3) describes the data in general business terms such as, e.g., *customer* or *product*. The meta-data characterizing cases of successful preprocessing are described in Part III. Also the case model consists of a conceptual part (Chapter 4) and an implementation part (Chapter 5). The conceptual part of the case model is the heart of the M^4 model. The implementation part is very short. It just states that operators are implemented as database operators (procedures, functions, or SQL-queries) or plug-ins of the KDD environment. Chapter 6 concludes the paper.

1.1 Metadata-Driven Software

Metadata (data about data) is a general notion that defines any information related to data in an information system. Metadata may be any information related to schema definitions and configuration specifications, physical storage, access rights, etc. Metadata may also represent end-user-specific documentation, dictionaries, business concepts and terminology, details about predefined queries, and user reports. Overviews of the state-of-the-art in metadata management with focus on data warehousing are provided in [12, 13].

In the case of a *metadata-driven software* package, metadata is stored in a repository and is used as *control information* for applications implemented with this software package. Examples of control information are static information (like structure definitions, configuration specifications, etc. as well as some part of application logic: conditions (e.g., for dynamic SQL), methods, parameters for stored procedures. At runtime, metadata is read by a tool engine, is dynamically bound into the engine software and the resulting application is then executed. In other words, application semantics is simply distributed between repository and engine and is pieced together just at runtime. Examples of metadata-driven software are the new generation tool packages for data warehousing, e.g., for building the data warehouse (like PowerMart[1], Ardent[2]) or for using it (like Cognos[3], Business Objects[4]).

To summarize, metadata-driven software provide a framework consisting of a repository structure and an engine which fits this structure. Users have to specify metadata instances (i.e., to fulfil this structure) in order to achieve executable task-oriented applications[5].

One of the main benefits expected from metadata-driven software is *software reusability and flexibility.* On the one hand, objects encapsulating control information are explicitely stored in the repository (i.e., outside scripts and programs) and may be reused in different contexts and applications. On the other hand, the engines running on top of the repository may be used for all metadata instances fitting the given metadata structure. This results in an improved flexibility. The system may be extended and adapted without difficulty. If new requirements arise, metadata instances may be easily changed without affecting the clients (i.e., engines) sharing it. Thus, *maintenance* is easier. Moreover, since operational metadata is liekly to be kept up-to-date, the documentation of the system is implicitly up-to-date as well.

Another advantage of metadata-driven software is the *automation of processes.* Since the repository may be shared by more than one programs performing certain tasks, they may pass control of execution to each other by means of metadata stored in the repository.

Nevertheless, the main advantage of using metadata-driven software is in view of enterprise-wide metadata integration [14]. Metadata stored in various repositories (e.g., from various tools like those for building a data warehouse resp. for using it) is integrated and linked with each other such that it is consistentenly and uniformly managed by an enterprise-wide metadata management system. In this way, links between metadata of various domains are established and exploited and thus up-to-date system information and documentation is

[1]http://www.informatica.com/

[2]Ardent Software was recently acquired by Informix, http://www.ardentsoftware.com

[3]http://www.cognos.com

[4]http://www.businessobjects.com

[5]Separating data and programs brings indisputable advantages, as we learned in database introduction courses. Metadata in the repository correspond to data in the database; database applications are, in this case, engines.

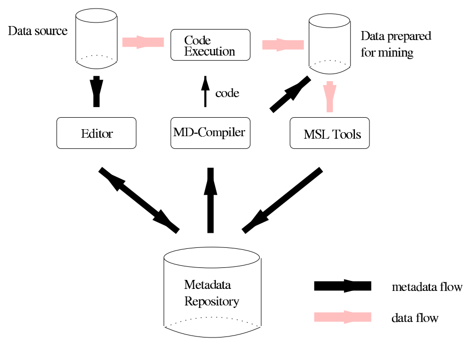

Figure 1.3: Architecture of Mining Mart System

available to all users and tools over the enterprise. Efforts are underway to establish metamodel standards for enterprise-wide metadata integration and exchange (for a comparison between two important standard proposals see [15]).

1.2 Architecture of the Mining Mart System

Mining Mart follows a typical metadata-driven software architecture, depicted in Figure 1.3. The core of the system is the *Repository* which is implemented on top of a DBMS. Case-specific informations are stored in the repository as metadata: the specification of the business problem to be solved by the case, specification of structures of the data to be mined, specification of processing operators to be applied on the data with corresponding parameters, the description of the data mining tool for which the data has to be prepared, etc. At runtime, a MD-*Compiler* reads the metadata and uses it in order to generate code. Through the execution of this code data is read from the data source (e.g., a data warehouse), is preprocessed and stored into the data target on which data mining will be applied later. The *Editor* is used for manipulating metadata (insert, delete, update) within the repository.

Note in Figure 1.3 that metadata may be produced and stored into the repository by means of other components as well. This component represents the *MSL tools* (multistrategy learning tools) [5] which are used for determining operator parameters when these cannot be manually specified. MSL tools accompany preprocessing operators and produce the metadata they require, i.e., the input parameters for them. An usual example is the discretization operator. One of the input parameters is a discretization table which specify for a certain attribute value intervals and corresponding values to substitute the intervals (i.e., when an attribute value is in the range of a certain interval, the corresponding value in the table has to be considered). Since the manual specification of an optimal discretization table is not always easy, a MSL tool is used for discovering the best discretization table by means of data analysis. Furthermore, optimal parameter settings usually depend on data, thus their discovering by means of MSL tools is the prerequisite for case reuse. For example, if discretization tables may be automatically rendered by a tool, the same case may be directly re-executed without designer intervention for different populations.

The software components accessing the repository (Editor, MD-Compiler, MSL-Tools) are “bound” to the given metadata structure and this structure

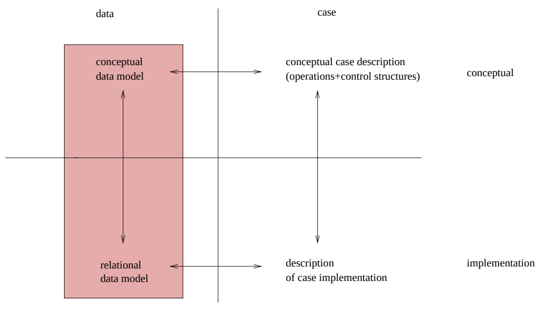
data
case
conceptual data model
conceptual case description (operations+control structures)
conceptual
relational data model
description of case implementation
implementation

Figure 1.4: Coarse Description of the Metamodel (the marked rectangle represents the data model of M^4)

represents the core of the system. The metadata structure is conceptually described by a *metamodel.* The domain-specific language for specifying applications is derived from the metamodel as well. Next section introduces the Mining Mart metamodel which is then described in more detail in the remainder of the paper.

1.3 Overview of M^4

In the following, we coarsely describe the metamodel of Mining Mart with focus on the data model part. Recall that data models are particularly used for data representation. Since a metamodel should "catch" the particularities that are relevant for a specific domain, our data model reflects the features of the data set that are important for preprocessing.

The Mining Mart Metamodel (M^4) is illustrated in the class diagram in Appendix. M^4 can be logically divided into the following two main groups: *data modelling* and *case modelling* information. Each group is again subdivided in accordance with the abstraction level into *conceptual and mining specific* information on the one hand and *implementation* information on the other hand. Figure 1.4 depicts the four parts resulting from this partition; parts are tightly coupled to each other.

- *Data modelling.* This group comprises classes for describing the *relational data model*, which corresponds to the implementation level and the *conceptual data model* which reflects, besides the known entity-relationship model, data mining specific aspects (as e.g., special data types - Time, Ordinal, Nominal, etc) and ontology knowledge.

- *Case modelling.* Describes preprocessing operators and the required controlling structures. The model is again divided into the mining-specific description of the case semantics (including for example operators as feature selection and discretization) and their implementation as e.g., function, stored procedure or SQL-query. We call the two groups *conceptual case description* respectively *description of the implemented case.*

On the one hand, partitioning M^4 in data and case modelling is necessary for ensuring reusability: the already specified operators may be used within cases having various parameter values represented in the data model. Furthermore, cases may be reused for different data sets (i.e., populations), represented within the data model as well. On the other hand, distinguishing between the two abstraction levels is required for enhancing:

- *user-friendliness.* End-users may manipulate familiar elements on the conceptual level in order to configure cases for execution. Regarding technical users, the two main categories are *case designer* and *case adapter.* The case designer accesses and manipulates only the elements of the upper, conceptual level during his work. That means, the implementation is transparent to him: he deals with mining-specific elements and constructs and has not to be aware of how they are implemented (which DBMS is used, how functions are implemented, etc.). In contrast, the case adapter is responsible for building the connection between the two levels when something in the structure of the database changes.

- (again) *reusability.* The conceptual, abstract level may be (re)used for any implementation behind (i.e., either database implementation or implementation of operators).

- *transportability* (is a sort of reusability as well). The idea is to be able to reuse (parts of) the cases not only in the same company (e.g., Swiss Life) and the same branch (life insurance) but in other branches as well. To this end, the representation of ontologies [3, 4, 10] has to be considered within the *conceptual data modelling* part. A common ontology basis has to exist which is specialized by all domain-specific ontologies.

Recall that the four parts of M^4 are linked to each other and connections between metadata instances are often in both directions navigable such that the required information may be rapidly accessed. Note that the consideration of case implementation submodel is optional. Since this submodel represents detailed information related to the implementation of operators and cases, it is necessary only if a step-by-step tracing of data transformations is intended. This could be desired for supporting understanding, debugging and maintenance of code. Otherwise, if implementation informations have to be stored with a coarse granularity only, the two submodels, conceptual case and implementation case description are merged.

M^4 combines ideas from two existing standards for metadata representation and exchange in the area of data warehousing (OIM and CWM) [15]. The ideas are drastically simplified but extended with data mining and preprocessing elements to make the metamodel domain-specific. Since both standards have UML as core, M^4 uses some of UML classes as foundation as well.

Foundation of the Metamodel: UML Classes

Figure 1.5 depicts the UML classes which are specialized for defining M^4. These are:

ModelElement is the base for all modeling metaclasses in the UML. All other modeling metaclasses are either directly or indirectly specialized from ModelElement.

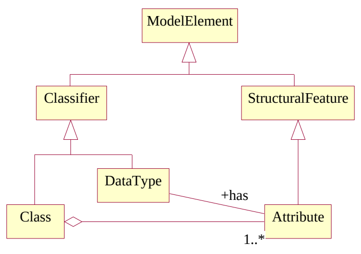

Figure 1.5: Simplified UML

Classifier A classifier is an element that describes behavioral and structural features; it appears in several specific forms, including class, data type, interface, component, and others.

Class A Class is a description of a set of objects that share the same attributes, operations, methods, relationships, and semantics. Specializations in M^4 of the UML class *Class* are ColumnSet and Concept (supporting conceptual and relational modelling).
Supertype Classifier
Subtypes (in M^4) Concept, ColumnSet
Associations *attributes.* Multiplicity: 0..n

Attribute An attribute is a named slot within a classifier that describes a range of values that can be assigned to instances of the classifier. Attribute is specialized to support different needs, and associated with a data type.
Supertype StructuralFeature
Subtypes(in M^4) Column, FeatureAttribute, Value, RoleRestriction
Associations *classes.* Ordered association to the class this attribute is in. Multiplicity:1..1
dataType. The data type of this attribute. Multiplicity: 1..1

DataType A data type is attached to an attribute. Data types include primitive built-in types (integer, strings etc.) as well as definable enumeration types (e.g. boolean, true and false).
Supertype Classifier
Subtypes(in M^4) Integer, String, DomainDataType etc.
Associations *has* (to Attribute). Multiplicity: 1..1

We assume a strict inheritance, where attributes of a subclass can only be the same or specializations of attributes of the superclasses. Multiplicity of an association denotes how many instances of the association a class may have. It indicates the minimal and maximal range of an association. If 0 is written as the lower bound then the association need not be present at all. Although this can be considered kind of a negation, we treat 0 in the same way as all other numbers. In particular, the specialization of multiplicity is simply increasing the lower and/or decreasing the upper bound.

In the following, we present M^4 in more detail. The next part deals with the two submodels obtained through vertical partition (data and case modelling). We present each class in turn with some of its particularities. Even if not always explicitly stated, each instance of the classes in the metamodel has a unique *ID* which identifies it, a *name*, and a *description* (which is simple text). Moreover, each class manages its *extension* which contains the set of all instances of this class.

Part II

Mining Mart Data Model

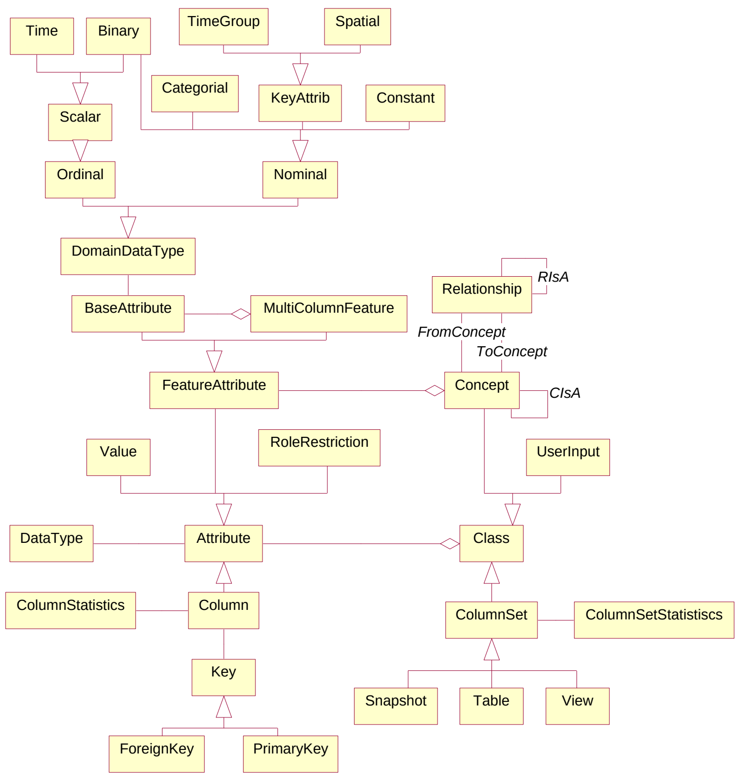

Figure 1.6: The class diagram of Mining Mart data model

The data model part of M^4 is illustrated in Figure 1.6. It consists of the conceptual and relational data model which are strongly coupled with each other. We address first the relational data model [6]which is shown in the lower part of the picture. It describes the data as they are at the application data warehouse. We could directly use the database schema – and actually do use it – but we must offer users the selection of relevant tables and columns. This selection determines the population of the data analysis task. It is up to further preprocessing steps to sample within this population or to determine relevant columns. The conceptual model as shown in the upper part of the picture declares the role that a table or column or set of columns is playing for a task. For instance, conceptual metadata characterize a particular set of columns (in the relational model) as the *concept* under investigation.

[6]This meta model is not new since it is rather standard and present in other meta models as well, e.g., [15].

Chapter 2

Relational Data Model

The relational model is what will be given by most applications. It is the data description as used for many purposes. It could well be This submodel comprises classes for representing data structures within the relational model. The main classes are: *Column*, *ColumnSet* (with its subclasses *Table*, *View*, *Snapshot*) and *Key*. *Column* and *ColumnSet* have each a class containing statistical information (*ColumnStatistics* and *ColumnSetStatistics*). A *ColumnSet* consists of a list of *Column*s.

2.1 Column

A Column defines a set of values, i.e, describes a column in a result set, a view, a table. It corresponds to a BaseAttribute at the conceptual level.

Supertype Attribute

Subtypes None

Attributes

- *name* - represents the name of the column in the database schema (e.g., PTANSCH, PTEPFI, etc.)
- *dataType* - represents the data type (in the implementation language) of the column (integer, string, etc)

Associations

- *belongsToColumnSet*. Is an aggregation (Column is part_ of ColumnSet). It points to the ColumnSet that contains this Column. Multiplicity: 1..1 (Aggregation will be implemented as an attribute TableIndentifier being foreign key to ColumnSet)
- *keys*. An association describing in which keys this column is part of. Multiplicity: 0..n
- *correspondsToBaseAttribute*. It points to the corresponding BaseAttribute at the conceptual level. Note that a FeatureAttribute may be either a BaseAttribute or a MultiColumnFeature. In the latter case there are more than one corresponding columns.

2.1.1 ColumnSet

A ColumnSet describes any general set of columns - typically a table, view or snapshot.

Supertype Class

Attributes

- *name* - represents the name of the table, view, etc.
- *number* - represents the number of columns
- *file* - name of the file containing the command creating the ColumnSet
- *dbConnectString* - name of the DB
- *user* - the name of the owner of the ColumnSet (e.g., for the access in Oracle User.Name@DBString is needed).

Associations

- *hasColumn.* Is an aggregation (ColumnSet has Column). It points to all the Column(s) that form this ColumnSet. Multiplicity: 1..n.
- *hasKeys.* It points to the corresponding primary and foreign keys that apply to the column set. Multiplicity 1..n.
- *correspondsToConcept.* It points to the corresponding concept at the conceptual level. Multiplicity 0..1;
- *correspondsToRelationship.* It points to the corresponding relationship at the conceptual level. Multiplicity 0..1;

Constraints ColumnSet points either to a Concept or a Relationship. There are ColumnSets which do not point to any concept but to a relationship (because when the Relationship has the multiplicity m:n, it will be implemented as a separate table);

2.1.2 ColumnStatistics

ColumnStatistics contains statistic information for columns necessary during data mining. On the one hand, statistics of the values for each attribute are stored (max and min value, average, etc). On the other hand, for each attribute, information about distribution blocks is stored (if such information is available). Distribution blocks are identified as follows: for a nominal attribute every value is counted and grouped. For an ordinal attribute the values are grouped in at most 1000 blocks (or intervals). For a time attribute the distribution depends on the months between the minimal and maximal value. If months_between > 600 the values are grouped into years. If months_between > 60 and <= 600 the values are grouped into quarters. If months_betweeen > 3 and <= 60 the values are grouped into months. If months_between <= 3 the values are grouped into days.

Supertype ModelElement

Attributes

- *unique* - number of diff. values of this column within the ColumnSet
- *missing* - number of missing value(s) within the ColumnSet
- *min* - minimal value of the column within the ColumnSet
- *max* - maximal value of the column within the ColumnSet
- *average* - average value of the column within the ColumnSet
- *standardDeviation* - standard deviation value of the column within the whole ColumnSet

The last two make sense for numeric (i.e., scalar) attributes only. The distribution information consist of:

- *distributionValue* - name of one distribution block, e.g 'YOUNG' for the attribute 'AGE'. If the attribute is of the type ORDINAL, the average value of the block is used.
- *distributionCount* - number of counted records for this distribution block.
- *distributionMin* - minimal value of the distribution block (makes sense for ordinal attributes only)
- *distributionMax* - maximal value of the distribution block (makes sense for ordinal attributes only)

• **Associations**

- *forColumn.* It relates to exactly one Column. Multiplicity: 1..1.

Commentar Statistics could be represented at the conceptual level as well.

2.1.3 ColumnSetStatistics

ColumnSetStatistics contains statistic information for ColumnSets necessary during data mining.

Supertype ModelElement

Attributes

- *allNumber* - total number of tuples within the ColumnSet
- *ordinalNumber* - number of ordinal attributes of the ColumnSet
- *nominalNumber* - number of nominal attributes of the ColumnSet
- *timeNumber* - number of time attributes of the ColumnSet

Associations

- *forColumnSet.* It relates to exactly a ColumnSet. Multiplicity: 1..1.

2.1.4 Table, View, Snapshot

No special features identified yet.

Supertype ColumnSet

Viev and Snapshot could possibly have an attribute containing the filtering condition (the WHERE part of the SQL- query used for view definition) and one association pointing to one or more instances of ColumnSet it has been applied on (FROM part of the View definition). Snapshots require also attributes for updating as *howRefresh* (i.e., either FAST or COMPLETE) and *refreshInterval.*

2.1.5 Key, PrimaryKey, ForeignKey

Supertype ModelElement

Attributes

– *isUsedForIndex* - may take two values, yes or no.

Associations

– *hasColumn.* It points to the Column(s) that form the Key. Multiplicity: 1..n.

– *isAssociatedToColumnSet.* It points to the ColumnSet where it is Key. It is a sort of redundancy to the Aggregation between Column and ColumnSet.

– *isConnectionTo.* It exists for ForeignKeys only - points to the Table where is key (usually a primary key). Multiplicity 0..1.

– *correspondsToRelationship.* Is valid for ForeignKey. It points to the corresponding relationship at the conceptual level. Multiplicity 0..1. Relationships 1:m or 1:1 will be implemented as ForeignKeys. A relationship m:n will be implemented as ColumnSet and 2 ForeignKeys. Multiplicity 1,2 or more;

Chapter 3

Conceptual Data Model

Due to its significant role for user-friendliness and reusability, the conceptual layer is the most important part of the data model. It is a combination between ER-modelling, description logic (DL) and ontology representation, extended with data-mining-specific classes. The main two classes are *Concept* and *Relationship*. A Concept (e.g., *Customer*, *Partner*) may have subconcepts (e.g., *Partner between 30-40 years* or *Mailed Person*), that means there are *IsA* relationships between Concepts (see Figure 3.2). Application-specific Relationships exist as well; they are binary Relationships (as in DL), e.g., Concept *Customer Buys* Concept *Product*, Concept *Partner* is in Relationship *Insurance Owner* with Concept *Insurance Contract*. Relationships may be bound to each other with *IsA* relationships as well. Figure 3.1 illustrates the metamodel representation of these semantics.

As visible in Figure 3.2, the three perspectives covered in the conceptual model are:

- ontology level: contains general business ontologies; is useful for reuse of cases in different companies; *Mailed Person IsA Partner* which in turn *IsA Customer*, *Tax-priviledged Contract IsA Insurance Contract* and *Insurance Contract IsA Product*; *IsA* applies for Relationships as well. *Customer* and *Product* are included in the basis ontology which should be common to many companies.

- database schema level: represents the conceptual schema of the database; this is mapped to the implemented schema. In Figure 3.2 this level correspond to *Partner, Insurance Contract and Insurance Owner* which have

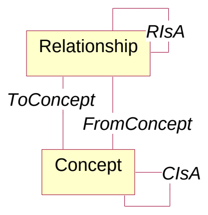

Figure 3.1: Two important classes of the conceptual data model and their UML associations

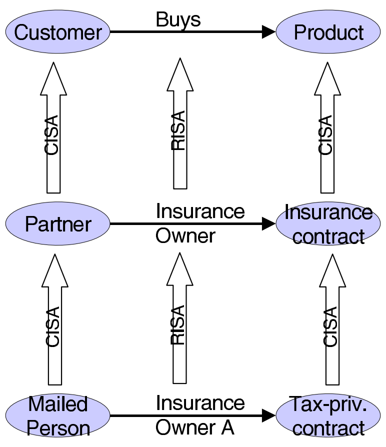

Figure 3.2: Instances of conceptual model elements

direct correspondents on the implementation model in terms of relational tables. So far, only the relational model is considered as implementation model in M^4 (see Chapter 2) but other data models may be considered as well. Note that the basis ontology (*Customer*, *Product*, *Buys*) has no direct correspondence on the implementation level.

- mining data level: describes data sets needed for and produced during preprocessing for mining; contains also mining specific data types. In Figure 3.2 these are the *Mailed Person*s, *Tax-priviledged contract*s, and the Relationship *Owns Insurance*. These are subconcepts and subrelationships of the elements beyond and are directly used for configuring or designing Mining Mart cases. They correspond on the implementation level to views or snapshots on tables.

We consider in detail the classes of the conceptual data model with their attributes.

3.1 Concept

Supertype Class

Attributes

- *name* - name of the concept (e.g., Partner, Product, Customer)
- *subConceptRestriction* - defines (in a machine processable language!) the characteristics of a subconcept (in relation with its superconcept), e.g., the specification of the fact that concept *Young&Powerful* represents the *Partner*(s) between 30-40 years while *Young* represents *Partner*(s) between 20-30 years. Both are subconcepts of *Partner*.

Associations

- *isA*. It points to its (super)concept e.g, *Young* isA *Partner*

- *correspondsToColumnSet.* It points to the corresponding ColumnSet that implements the concept. Multiplicity 0..1.
- *FromConcept.* It points to the Relationship the Concept is associated with. For example, Concept Partner is in Relationship PartnerRole to the Concept Contract. Then Partner is associated by means of *FromConcept* to PartnerRole. Multiplicity 1..1.
- *ToConcept.* It points to the Relationship the Concept is associated with. E.g., Contract is associated by means of *ToConcept* to Relationship PartnerRole. Multiplicity 1..1.

Constraints
Each ColumnSet has a Concept or Relationship but not each Concept or Relationship points to a ColumnSet (e.g., the basis ontology has no correspondant on the database side.
For each instance of *FromConcept* association an instance of *ToConcept* association has to exist and conversely.

3.2 Relationship

Supertype ModelElement

Attributes

- *name* - name of the relationship (e.g., PartnerRole, InsuredPerson, InsuranceHolder, InsuranceOwner)
- *subRelationshipRestriction* - defines (in a machine processable language!) the characteristics of a subrelationship in relation with its superrelationship, e.g., the specification of the fact that Relationship *InsuredPerson* resp. *InsuranceHolder* represents a subset of *PartnerRole* fulfilling some conditions. These are formulated using feature-attributes, concepts, and so on. A possible language could be DL role-terms [1]. InsuredPerson and InsuranceHolder are both subconcepts of *PartnerRole.*
- *defined* - with regard to the restriction above, it may be defined or primitive (sufficient condition or not)

Associations

- *isA.* It points to its (super)relationship e.g, *InsuranceHolder* isA *PartnerRole*
- *correspondsToForeignKey.* Multiplicity 1..n. It points to the corresponding ForeignKey(s) that implements the Relationship. Relationships 1:m or 1:1 will be implemented as ForeignKeys. A relationship m:n will be implemented as ColumnSet and 2 ForeignKeys. Moreover, a relationship may also have more than 2 ForeignKeys.
- *correspondsToColumnSet.* It points to the corresponding ColumnSet that implements the Relationhip. This is valid only when the relationship is m:n. Multiplicity 0..1.

- *FromConcept.* It points to one of the Concepts connected by the Relationship (Multiplicity 1..1) while
- *ToConcept.* Multiplicity 1..1. It points to the other Concept. E.g., PartnerRole is associated by means of *FromConcept* to Concept Partner and by means of *ToConcept* to Concept Contract.

Constraints
Each Relationship corresponds to either a ColumnSet or a ForeignKey on the implementation level. That means, the association *correspondsToForeignKey* may exist without the association *correspondsToColumnSet*. However, when emphcorrespondsToColumnSet exists, it requires two ForeignKeys as well (i.e.,*correspondsToForeignKey* exists as well).
A Relationship cannot exist without FromConcept and ToConcept.

3.3 FeatureAttribute

Contains attributes of Concepts. It may be either a *BaseAttribute* or a *MultiColumnFeature*. A FeatureAttribute may have various mining relevant data types (they follow below).

Supertype Attribute (from UML)

Subtypes *BaseAttribute*, *MultiColumnFeature*

Attributes

- *name* - name of the feature attribute. The name should be more comprehensive (e.g., postal address of partner) than usual names of columns (as. e.g, PTANSCH and PTEPFI)
- *relevanceForMining* - whether the FeatureAttribute is relevant for mining or not
- *attributeType* - defines whether it is a base attribute, a result or an intermediate attribute for a case.

Associations

- *belongsToConcept.* It points to the concept it belongs to. It is an aggregation. Multiplicity 1..1.
- *correspondsToColumns.* It points to the column (or columns) it represents. Multiplicity 1..n. Note that a feature attribute may have many columns if it is a MultiFeatureColumn.

3.4 BaseAttribute

A *BaseAttribute* may have various mining relevant data types (they follow below).

Supertype FeatureAttribute

Associations

- *domainDataType.* It points to the mining-specific data type, i.e., to the DomainDataType. Multiplicity 1..1.
- *isPartOfMultiColumnFeature.* Itis an aggregation and points to a MultiColumnFeature if it is part thereof. Multiplicity 0..1.

3.5 MultiColumnFeature

A *MultiColumnFeature* consists of a set of *BaseAttribute.* Note that a Multi-ColumnFeature has no data type, only a BaseAttribute has one.

Supertype FeatureAttribute

Subtypes *TimeInterval* (not depicted)

Associations

- *consistsOfBaseAttributes.* It points to the set of BaseAttributes that builds the MultiColumnFeature. It is an aggregation. Multiplicity 1..n.

Comments. *TimeInterval* has only two BaseAttributes it points to: startOfInterval and endOfInterval. Their data type is *Time.* Another example of an instance for MultiColumnFeature is "Money" which could be represented with two components (value, currency).

3.6 Value

Value is needed for specifying arithmetic expressions and conditions for e.g., segmentation operators. It is part_of *UserInput.* This aggregation is not additionally depicted in the figure since superclasses of *Value* and *UserInput* (i.e., *Attribute* and *Class*) are anyway linked by an aggregation.

Supertype Attribute (from UML)

Attributes

- *name* - name (or representation) of the value

Associations

- *domainDataType.* It points to the mining-specific data type, i.e., to the DomainDataType. Multiplicity 1..1.
- *belongsToUserInput.* It points to the UserInput it belongs to. It is an aggregation. Multiplicity 1..n., not depicted in the diagramm.

Comments. Values could be any complex structure as well, e.g., decision trees, regression trees, discretization tables, instance lists, aso. Complex structures have not been considered so far.

3.7 UserInput

Contains the set of *Value*s entered by users when specifying cases.

Supertype Class (from UML)

Attributes

– *name* - name of user input

Associations

– *ContainsValues.* It points to Values. It is an aggregation. Multiplicity 1..n. Not depicted in the diagram.

3.8 RoleRestriction

It is a special attribute. It is necessarily bound to a relationship and a concept. It actually expresses a constraint, the fact that if a Relationship is linked to a Concept by means of FromConcept, it has to be linked to another Concept by means of a ToConcept. RoleRestriction considers the constraint from another perspective, for THIS (i.e., the given) Concept for which the RoleRestriction attribute has been defined, there exists a Relationship and (at least) another Concept (such that the relationship exists between these two concepts).

Warning: the semantics may possibly be found in the Concept-Relationship modelling as well but it seems that various operators need the information explicitly available in form of "role restriction" and not hidden as Concept-Relationship representation. That's the reason RoleRestriction had to be introduced. It corresponds to the DL terms: all, atleast and atmost [1].

Supertype Attribute (from UML)

Attributes

– *name* - name of the role restriction

– *restrictionForRelationship* - pointer to the Relationship it is a restriction for

– *restrictionForConcept* - pointer to the (sub)Concept it applies to.

– *restrictionToConcept* - pointer to the Concept where all instances of the range of the relation will be member of (DL-all)

– *min* - minimum of number of Concept instances in Relationship with every instance of THIS Concept

– *max* - maximum of Concept instances in Relationship with every instance of THIS Concept. (Note that only a 2-Relationship is considered.)

Associations

- *belongsToConcept.* It points to THIS Concept for which roleRestriction is an attribute (in the diagramm is depicted as an association between Attribute and Class).

3.9 DomainDataType and its Subclasses

There are many domain-specific data types we only briefly mention below.

DomainDataType. Each attribute has a data type; this will be represented as association *hasDataType*, multiplicity 1..1.
Generally, note that constraints have to be implemented for these data types. In particular, operators making sense for each of the domain-specific data types have to be defined and processing information for them is required. For example, "<" and "<=" make sense for ordinal attributes only. In contrast, "=" makes sense for binary and categorial attributes. Distance is allowed for scalar attributes only. Operators like +, - are applied to scalar attributes. Logical operators are applicable to binary attributes.

Ordinal. All values of this attribute are ordered. Distance between values makes no sense.

Scalar. Dinstance makes sense. Scalar attributes are usually represented as numeric or date on the implementation level.

Time. It represents the absolute point in time. It may have the following attributes:

- value
- timeScale (second, minute, hour, day, month, year).

Binary. Has only two values, 0 or 1. "<" and distance makes sense (it is either 0 or 1). The two logical operations, "xor"(+) and "and" (·) are also applicable and result in the following arithmetic logic:

XOR:	AND:
$1 + 1 = 0$	$0 \cdot 0 = 0$
$1 + 0 = 1$	$0 \cdot 1 = 0$
$0 + 1 = 1$	$1 \cdot 0 = 0$
$0 + 0 = 0$	$1 \cdot 1 = 1$

Categorial. Has a fixed small number of values.

KeyAttribute. This kind of attributes is used for identification and is not suitable for mining (the number of different values is too high). Subclasses are *TimeGroup* and *Spatial.*

TimeGroup. It represents the identification of an individual for which *Time* data is collected. It makes sense only if it is paired with a set of *Time* attributes (representing the time series). It has as attribute *numberOfDifferentIndividualInstantiations* and *missing values.*

Spatial. This data type is used for GIS and mining visualisation.

Constant. It has only one possible value and thus it is not suitable for mining. Typically is the result of a selection.

Part III

Case Model

Chapter 4

Conceptual Case Modelling

The metadata that characterize sequences of preprocessing operators and their sequences is declared by the case model as depicted in Figure 4.1. A case consists of a list of steps and each step embedds an operator; the output of a step is the input for the next one. Operators have *Parameters* which should be *Concept*s, *Relationship*s, *FeatureAttribute*s or *Values* - to be found in the data model part (not all the semantics are represented in the diagram). Cases have as parameters the population (i.e., the base concept) and the target attribute which needs to be specified at the conceptual level (as *Concept*, *FeatureAttribute*, *Relationship*). Finally, the actual data mining will be then performed on a subconcept of the base concept which is constructed during preprocessing.

Steps can be carried out in loops 4.2.1 or as multiple steps 4.2.2. While *LoopStep* is an iteration over input elements, *MultiStep* is an iteration over output elements. Other control stuctures – for instance conditioned branching – are currently not included in the case model. The case model stores successful cases for further use. A certain case does not contain any conditioned branching.

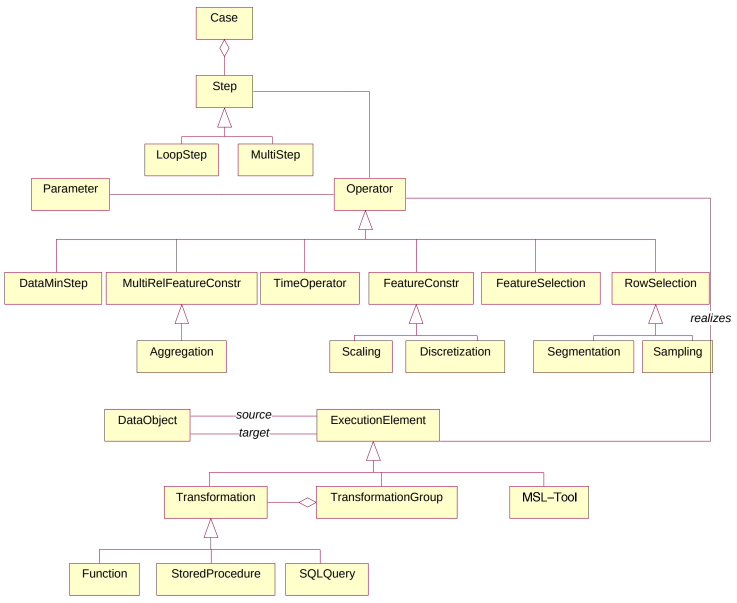

Figure 4.1: The class diagram of Mining Mart case model

Only if case adaptation would become easier by conditions, we shall include condistioned statemehnts into the case model.

The conceptual layer of the case model contains domain-specific classes which represent preprocessing on a higher abstraction level than the underlying implementation code behind (represented within the implementation case model in Chapter 5).

4.1 Case

A case consists of many steps. Note that a case has to contain an attribute *documentation* which describes by means of natural language what the case does. This documentation is important for retrieving the appropriate case from the set of already defined cases.

Supertype ModelElement

Attributes

– *name* - name of the case

– *case mode* - may be training or final

– *caseInput* - input for the case (is a heterogeneous list containing Concepts, FeatureAttributes, Values, Relationships)

– *caseOutput* - output of the case; is a Concept; it is actually the input required by the DataMiningStep.

– *documentation*

Associations

– *listOfSteps*, It points to the steps that build the case. It is an aggregation. Multiplicity 0..1.

– *population.* It points to *Concept* (which is an element of *caseInput*). The case has to be applied on this Concept; it will be taken as parameter for RowSelection operators

– *targetAttributes.* It points to the *FeatureAttribute*(s) for which the mining case is applied; It will be needed as parameter value for many of the operators.

4.2 Step

Steps are parts of cases (that means, they make sense only in connection with a case). Each step has to be linked to a set of predecessors and successors which are steps as well. In this way, parallelisation of operator execution is possible because no fixe sequence is enforced, only prioritisation is specified - that means, the operator may be executed only if its predecessors have been executed.

Each step embedds one operator. The output of the operator belonging to the predecessor step is needed as the input of the operator belonging to THIS step.

Supertype ModelElement(from UML)

Subtypes LoopStep, MultiStep

Attributes

– *name* - name of the step

Associations

– *belongsToCase*. It points to the corresponding Case. 1..n
– *embeddsOperator*. it points to the corresponding operator. 1..1
– *predecessor*. it points to the preceding steps. 0..n
– *successor*. it points to the succeding steps. 0..n

4.2.1 LoopStep

LoopStep is a special kind of *Step*. It allows the iteration over more than one input element. In a loop, for instance, one step consists of the discretization of several attributes of a table. The discretization then loops over the numerical attributes. Similarly, a missing value operator can be applied to more than one attribute during the same step. *LoopStep* may be applied only to*Operator*s having *loopable* = yes. These may be only instances of *FeatureConstruction* and *MultiRelationalFeatureConstruction*. Other operators like *RowSelection*, *FeatureSelection*, *TimeOperators* are not loopable. The output description is the same as for a single operator call.

Supertype Step

Attributes

– *iterationSet* - is a set of FeatureAttribute; for each element of the set the operator (embedded in step) is called.
– *outputSet* - is a set of FeatureAttribute

4.2.2 MultiStep

MultiStep is another special kind of *Step*. While *LoopStep* is an iteration over input elements, *MultiStep* is an iteration over output elements. That means, the rest of the case is applied for each of the output element in the list. In a *MultiStep*, for instance, several random samples are drawn from a population and for each of these, a certain operator transforms the sample into a data set for data mining. Multiple steps are also important when preprocessing multivariate time series into data sets for learning.

Supertype Step

Attributes

– *iterationCondition* - is the condition for looping over output elements

Comments. OutputSet is obtained from the Operator embedded in Step.

4.3 Operator

Operator is the superclass of all operators in MiningMart. Recall, that we adopt a strict inheritance approach. This means, that all descriptions of a superclass are inherited by its subclasses and can only be specialized. Usually, the inherited associations and attributes are not listed. If it enhances readability, however, the inherited features are repeated with the marker *inherited*. The notation for the operators is informal, but employs some conventions for the ease of reference within a description. Strings that are composed of the definite article and an element are variables of the operator metadata in the case model [1]. For instance, TheInputConcept is used in an operator class to refer to the input of the operator. Input and output are notions from the operator metadata. The type of the variables is given by reference to the data model. It is never written with the string "The" in front. For instance, TheInputConcept points to a Concept. This relates the operator description in the case model (TheInputConcept) with the data model (Concept).

Abstract

yes

Supertype ModelElement (from UML)

Attributes

- *loopable* - it may take two values, yes/no. FeatureConstruction and MultiFeatureConstruction could be loopable while the other ones (RowSelection, FeatureSelection, TimeOperators) are not loopable.
- *manual* - it may take two values, yes/no.

Associations

- *input*. It is an ordered list of parameters. Multiplicity 1..n.
- *output*. It is an ordered list of parameters. Multiplicity 1..n
- *realizes*. It points to ExecutionElement. Multiplicity 0..1
- *constraints*. Constraints of the operator that clarify the semantics of the operator. Their validity can be checked on the meta-data. Multiplicity 0..n
- *conditions*. Conditions of the operator which can be checked on the data only.
- *assertions*. Characterization of the operator relating input and output. Assertions are true by the nature of the operator. The information is supposed to support the sequencing of operators. Multiplicity 0..1

[1] Note, however, that the names of elements of the operator description need not start with the definite article "The".

4.3.1 MultiRelationalFeatureConstruction

Multi-relational feature construction takes as input several concepts (tables, relations) and produces a single concept.

Abstractyes

Supertype Operator

Attributes

loopable - yes

Associations

– *input*

TheConcept points to a Concept, Multiplicity 0..n.

4.3.1.1 Chaining

Chaining finds a feature of an object as a result from a chain of relationships. For instance, we can turn the characteristics of a living place into a characteristic of the people living there. We start with a customer in R_1 and use the relation R_2 between customers and areas in order to find the zip code. Let us suppose we have background knowledge about regions (e.g., the average rent of apartments). Then we use this relationship R_3 in order to find a feature F (the rent) for a region. Now, we ascribe this feature to the customer. At the implementation level this means adding an attribute to R_1. The new attribute is based on attributes in several tables.

Supertype MultiRelationalFeatureConstruction

Attributes

loopable - yes

Associations

– *input*

TheConcept points to a Concept, Multiplicity 1..1.
R_1,...,R_n point to n RelationShip, each with Multiplicity 1..1.
F points to a BaseAttribute, Multiplicity 1..1.

– *output*

TheOutputAttribute points to a BaseAttribute, Multiplicity 1..1.

– *Constraints*

Domain(R_1) = TheConcept
Range(R_i) = Domain(R_{i+1})
F is BaseAttribute of Range(R_n)

– *Assertions*

TheOutputAttribute belongs to TheConcept.

4.3.1.2 Propositionalisation

Propositionalisation is the process of transforming a multi-relational dataset, containing structured examples, into a propositional dataset with derived attribute-value features, describing the structural properties of the examples. The process can thus be thought of as summarising data stored in multiple tables in a single table containing one record per example. The operator uses aggregates to project information stored in several tables on one of these tables, essentially adding virtual attributes to this table. In the case where the information is projected on the target table, and structural information belonging to an example is summarised as a new feature of that example, aggregates can be thought of as a form of feature construction. The aggregates generated by this operator are *count*, *min*, *max*, *sum*, *avg* and *predominant value*.

Abstract no

Supertype MultiRelationalFeatureConstruction

Attributes

loopable - yes

Associations

- *input*

  theDataModel points to a set of Concepts, Multiplicity 0..n. *inherited*

  theRelationSet points to a set of Relationships, Multiplicity 0..m.

  theTargetConcept points to a Concept, Multiplicity 1..1.

- *output*

  thePropositionalConcept points to a Concept, Multiplicity 1..1.

- *Constraints*

  TheTargetConcept points to a Concept that is an element of theDataModel.

- *Assertions*

  TheOutputAttribute belongs to TheConcept.

4.3.2 RowSelection

RowSelection is the superclass of all instance selection operators. Of course, there are many different tools that implement the subclasses **Sampling** or **Segmentation**. Even for their subclasses, e.g. stratified random sampling or partitioning, there exist many different algorithms. At the level of the case model, we treat them all alike, do not differentiate among them (e.g.,with respect to performance). Hence, a case including a random sampling step can be executed applying whatever random sampling algorithm is available at the application site (see Chapter 5).

Supertype Operator

Attributes

loopable - no

multistepable - yes

Associations

– *input*

TheInputConcept points to a Concept, Multiplicity 1..1.
TheCondition points to a StructuralFeature, Multiplicity 1..1.

– *output*

TheOutputConcept points to a Concept, Multiplicity 1..1.

– *Assertions*

TheOutputConcept isA TheInputConcept.

4.3.2.1 Sampling

Abstract

yes

Supertype RowSelection

Associations

– *input*

HowMany points to a Value (dataType=Real domain=low-high (low=0 high=1)), Multiplicity 1..1.

4.3.2.1.1 RandomSampling

Abstract

no

Supertype Sampling

Attributes

manual - yes

loopable - yes

4.3.2.1.2 StratifiedRandomSampling

Abstract

no

Supertype Sampling

Attributes

manual - no

loopable - yes

Associations

– *input*

TheAttribute points to a BaseAttribute (domainDataType=Categorial), Multiplicity 1..1.

4.3.2.2 SelectCases

Abstract

yes

Supertype RowSelection

4.3.2.2.1 DeleteRecordsWithMissingValues

Abstract

no

Supertype SelectCases

Attributes

manual - yes

loopable - yes

Associations

– *input*

TheAttribute points to a BaseAttribute (hasMissingValues=Yes), Multiplicity 1..1.

– *Constraints*

TheAttribute is present in TheInputConcept

– *Assertions*

TheAttribute hasMissingValue=No

4.3.2.2.2 SelectByQuery

Abstract

no

Supertype SelectCases

Attributes

manual - yes

loopable - yes

Associations

– *input*

  TheCondition points to a Condition, Multiplicity 1..1.

– *Constraints*

  All attributes A_i $\in$TheCondition are present in TheInputConcept

4.3.2.3 Segmentation

Abstract

yes

Supertype RowSelection

Associations

– *output*

  TheOutputConcepts points to a set of Concept, Multiplicity 1..n.

4.3.2.3.1 StratifiedSegmentation

Abstract

no

Supertype Segmentation

Associations

– *input*

  TheAttribute points to a BaseAttribute (domainDataType=Categorial), Multiplicity 1..1.

4.3.2.3.2 NNSegmentation

Abstract

no

Supertype Segmentation

Associations

– *input*

  ClusterCenters points to a Concept, Multiplicity 1..1.

4.3.2.3.3 Partitioning

Abstract

no

Supertype Segmentation

Associations

– *input*

HowManyPartitions points to a Value (dataType=Integer domain=greater-than (value=1)), Multiplicity 1..1.

4.3.3 FeatureSelection

Some learning algorithms select the most relevant features automatically. However, the user might want to exclude a large set of irrelevant features early on.

Supertype Operator

Attributes

manual - yes

loopable - no

Associations

– *input*

TheConcept points to a Concept, Multiplicity 1..1.

TheAttributes points to a set of BaseAttribute, Multiplicity 1..n.

– *output*

TheOutputConcept points to a Concept, Multiplicity 1..1.

– *Assertions*

TheOutputConcept is an extensionally equivalent projection of TheConcept.

4.3.4 Feature Construction

The operator *FeatureConstruction* creates a new feature for a concept. This corresponds to a new attribute in a table or view at the implementation level. The new attribute is based on one or more base attributes. As opposed to multi-relational feature construction, the attributes that are combined to obtain the new one are all in the same table or view. The total number of data records is the same as before the operation. Examples of feature construction operators are: age computation for a person by using her date of birth, computation of the entry-age into an insurance contract, or the end-age of an insurance contract.

Supertype Operator

Attributes

loopable - yes

Associations

– *input*

TheConcept points to a Concept, Multiplicity 1..1.

TheAttributes points to a set of BaseAttribute, Multiplicity 1..n.

– *output*

TheOutputConcept points to a Concept, Multiplicity 1..1.

– *Assertions*

The OutputAttribute belongs to TheConcept.

4.3.4.1 Pivot

The operator *Pivot* merges all the records concerning a subject in only one record. It creates new features and the total number of data records is less than before the operation. One of the input parameters has to be categorial *Base Attribute.*

Supertype Feature Construction

Attributes

Associations

– *input*

UserIdentification points to BaseAttribute, Multiplicity 1..1.

FirstAttribute points to BaseAttribute, Multiplicity 1..1.

SecondAttribute points to BaseAttribute, Multiplicity 1..1.

– *output*

TheOutputConcept points to a Concept, Multiplicity 1..1.

NewAttribute(i) points to BaseAttribute, Multiplicity 1..1 (where FirstValueOfCategorial ¡= i ¡= LastValueOfCategorial).

– *Assertions*

The *name* of the column(NewAttribute(i)) = i.

The *datatype* of the columns NewAttribute = the *datatype* of SecondAttribute.

– *Constraints*

The PrimaryKey(TheConcept) *has column* UserIdentification, FirstAttribute, SecondAttribute.

DomainDataType(FirstAttribute) = Categorial (with n value).

The *number* of column of THeOutputConcept is equal to $n + 1$ (they are the NewAttributes + ThePrimaryKey).

The PrimaryKey(TheOutputConcept) *has column* UserIdentification.

4.3.4.2 Dual Pivot

The operator ıdual pivot splits a record and creates a *basket* of records; the input multi-features concept is transformed in a two features concept.

Supertype Feature Construction

Attributes

multistepable - no

Associations

– *input*

– *output*

TheOutputAttribute points to a BaseAttribute, Multiplicity 1..1 (it's the only attribute of TheOutputConcept).

– *Constraints*

ThePrimaryKey(TheConcept) ıhas column UserIdentification

ThePrimaryKey(TheOutputConcept) ıhas column UserIdentification, TheOutputAttribute

4.3.4.3 Scaling

Scaling is a specialization of the FeatureConstruction operator. The scale of numeric attributes is very important for distance-based mining algorithms (like e.g., clustering) because attributes with larger values are more influential on the result. To avoid this usually unintended weighting of attributes, all attributes have to be rescaled, i.e., the range of attribute values has to be changed in such a way that it fits into a specified new range. Input and output parameters have to be scalar *BaseAttribute*s.

Abstract

yes

Supertype FeatureConstruction

Association

– *input*

TheAttribute points to a BaseAttribute (domainDataType = Real), Multiplicity 1..1.

– *Constraints*

TheAttribute belongs to TheConcept

4.3.4.3.1 LinearScaling

Abstract

no

Supertype Scaling

Attributes

manual - yes

Association

– *input*

NewRangeMin points to a Value (dataType=Real), Multiplicity 1..1.

NewRangeMax points to a Value (dataType=Real), Multiplicity 1..1.

– *Conditions*

NewRangeMin $<$ NewRangeMax

4.3.4.3.2 LogScaling

Abstract

no

Supertype Scaling

Attributes

manual - yes

Association

– *input*

LogBase points to a Value (dataType=Integer domain=greater-than (value=2)), Multiplicity 1..1.

– *Conditions*

TheAttribute > 0

4.3.4.4 MissingValues

Supertype FeatureConstruction

Associations

– *input*

TheConcept *(inherited)*points to a Concept, Multiplicity 1..1.

TheTargetAttribute points to a BaseAttribute, Multiplicity 1..1.

ThePredictingAttributes *(inherited)* points to a set of BaseAttributes, Multiplicity 1..n.

- *output*

  FilledAttribute points to a BaseAttribute, Multiplicity 1..1.

- *Constraints*

  TheTargetAttribute and ThePredictingAttributes belong to TheConcept.

- *Conditions*

  TheTargetAttribute is a BaseAttribute with hasMissingValues=yes.

- *Assertions*

  FilledAttribute belongs to TheConcept. FilledAttribute is a BaseAttribute with hasMissingValues=no.

4.3.4.4.1 AssignDefault

Abstract

no

Supertype MissingValue

Associations

- *input*

  DefaultValue points to a Value (dataType=domainDataType(TheAttribute)), Multiplicity 1..1.

4.3.4.4.2 AssignModalValue Here, the most frequent value of an attribute is asserted, whenever the value is missing.

Abstract

no

Supertype MissingValue

Associations

- *input*

  ModalValue points to a Value (dataType=Categorial), Multiplicity 1..1.

- *Constraints*

  domainDataType(TheAttribute) = Categorial

4.3.4.4.3 AssignMedianValue

Abstract

no

Supertype MissingValue

Associations

– *input*

MedianValue points to a Value (dataType=Ordinal), Multiplicity 1..1.

– *Constraints*

domainDataType(TheAttribute) = Ordinal

4.3.4.4.4 AssignAverageValue

Abstract

no

Supertype MissingValue

Associations

– *input*

AverageValue points to a Value (dataType=Real), Multiplicity 1..1.

– *Constraints*

domainDataType(TheAttribute) = Real

4.3.4.4.5 AssignStochasticValue

Abstract

no

Supertype MissingValue

Attributes

manual - yes

Associations

– *input*

ProbabilityDistribution points to a Value (dataType=Distribution), Multiplicity 1..1.

– *Constraints*

domainDataType(TheAttribute) = Categorial

4.3.4.4.6 AssignPredictedValueCategorial

Abstract

no

Supertype MissingValue

Attributes

manual - no

Associations

- *input*

  UsePreviousTree points to a Value (dataType=Boolean default-Value=false), Multiplicity 1..1. DecisionTree points to a Value (dataType=DecisionTree dependency=UsePreviousTree (dependency-type=equals (value=true))), Multiplicity 1..1. PredictingAttributes points to a set of BaseAttribute (dependency=UsePreviousTree (dependency-type=equals (value=false))), Multiplicity 1..n.
- *Constraints*

  domainDataType(TheAttribute) = Categorial

Note

TheAttribute is used as Target by the decision tree building algorithm. The decision tree building algorithm must be run on a subset of TheConcept that contains no records with missing values for TheAttribute.

4.3.4.4.7 AssignPredictedValueContinuous

Abstract

no

Supertype MissingValue

Attributes

manual - no

Associations

- *input*

  UsePreviousTree points to a Value (dataType=Boolean default-Value=false), Multiplicity 1..1.

  RegressionTree points to a Value (dataType=RegressionTree dependency=UsePreviousTree (dependency-type=equals (value=true))), Multiplicity 1..1.

  PredictingAttributes points to a set of BaseAttribute (dependency=UsePreviousTree (dependency-type=equals (value=false))), Multiplicity 1..n.

– *Constraints*

domainDataType(TheAttribute) = ordinal

Note

TheAttribute is used as Target by the Regression Ttree building algorithm. The regression tree building algorithm must be run on a subset of TheConcept that contains no records with missing values for TheAttribute.

4.3.4.4.8 MissingValuesWithRegressionSVM

Supertype MissingValues

Attributes

manual - no

Associations

– *input*

The TargetAttribute *(specialized)* points to a BaseAttribute (Scalar), Multiplicity 1..1.

ThePredictingAttributes *(specialized)* points to a set of BaseAttributes (Scalar), Multiplicity 1..n.

C points to a BaseAttribute (Scalar), Multiplicity 1..1.

LossFunctionPos points to a BaseAttribute (Scalar), Multiplicity 1..1.

LossFunctionNeg points to a BaseAttribute (Scalar), Multiplicity 1..1.

Epsilon points to a BaseAttribute (Scalar), Multiplicity 1..1.

KernelType is a Constant.

– *output*

TheSVMModel points to a SVMModel, Multiplicity 1..1.

FilledAttribute *(specialized)* points to a BaseAttribute(Scalar).

4.3.4.4.9 MissingValuesWithDecisionTree

Supertype MissingValues

Attributes

manual - no

Associations

– *input*

TheInputDecisionTree points to a DecisionTree, Multiplicity 1..1.

– *output*

TheOutputDecisionTree points to a DecisionTree, Multiplicity 1..1.

4.3.4.5 Grouping

Abstract

yes

Supertype FeatureConstruction

Associations

– *input*

LabelPrefix points to a Value (dataType=Char), Multiplicity 1..1.

4.3.4.5.1 ECGrouping

Abstract

no

Supertype Grouping

Attributes

manual - yes

Associations

– *input*

IntervalCard points to a Value (dataType=Integer domain=greater-than (value=1)), Multiplicity 1..1.

– *Constraints*

domainDataType(TheAttribute) = Ordinal

– *Conditions*

IntervalCard < Max(TheAttribute)

4.3.4.5.2 UDGrouping

Abstract

no

Supertype Grouping

Associations

– *input*

TheGroupings points to a set of ValueSet, Multiplicity 1..m.

– *Constraints*

domainDataType(TheAttribute) = Categorial

– *Conditions*

Every value in TheGroupings must belong to Domain(TheAttribute)

4.3.4.6 Discretization

Abstract

yes

Supertype FeatureConstruction

Associations

- *input*

  LabelPrefix points to a Value (dataType=Char), Multiplicity 1..1.

- *Constraints*

  domainDataType(TheAttribute) = Ordinal

4.3.4.6.1 EWDiscretization1

Abstract

no

Supertype Discretization

Associations

- *input*

  Delta points to a Value (dataType=Real domain=greater-than (value=0)), Multiplicity 1..1.

4.3.4.6.2 EWDiscretization2

Abstract

no

Supertype Discretization

Associations

- *input*

  NumberOfIntervals points to a Value (dataType=Integer domain=greater-than (value=1)), Multiplicity 1..1.

4.3.4.6.3 ECDiscretization

Abstract

no

Supertype Discretization

Associations

- *input*

  IntervalCard points to a Value (dataType=Integer domain=greater-than (value=1)), Multiplicity 1..1.

4.3.4.6.4 UDDiscretization

Abstract

no

Supertype Discretization

Associations

– *input*

IntervalExtremes points to a set of Value (dataType=Real), Multiplicity 1..m.

4.3.5 DataMiningStep

The data mining step will most likely be executed by a plug-in tool or even by another system. Since the MiningMart project is about preprocessing, we do not model all the algorithms or tools that can be used for data mining, but characterise at a higher level the classes of learning methods. The data mining step is part of the case model as the ultimate target of the preprocessing chain. Input restrictions of the data mining step determine the necessary preprocessing steps that convert the given data into ones that fulfill the learning algorithm's needs.

Supertype Operator

Attributes

loopable - no

manual - no

Associations

– *input*

TheExamples points to a Concept, Multiplicity 1..1.

4.3.5.1 Classification

This is the standard task of learning from examples the recognition of class members. Classes are marked by nominal values.

Supertype DataMiningStep

Associations

– *input*

TheExamples *(inherited)* points to a Concept, Multiplicity 1..1

TheTargetAttribute is a BaseAttribute (Nominal), Multiplicity 1..1.

– *Constraints*

TheTargetAttribute belongs to TheExamples.

4.3.5.1.1 DecisionTree

Supertype Classification

Associations

– *input*

TheConcept points to a Concept, Multiplicity 1..1.

TheAttributes is a set BaseAttributes, Multiplicity 1..n.

– *output*

TheDecisionTree points to a DecisionTree, Multiplicity 1..1.

4.3.5.1.2 SupportVectorMachineForClassification

Supertype Classification

Associations

– *input*

ThePredictingAttributes are a set of BaseAttribute(Scalar), Multiplicity 1..n.

C points to a BaseAttribute (Scalar), Multiplicity 1..1.

LossFunctionPos points to a BaseAttribute (Scalar), Multiplicity 1..1.

LossFunctionNeg points to a BaseAttribute (Scalar), Multiplicity 1..1.

KernelType is a Constant.

– *output*

TheSVMModel is a SVMModel, Multiplicity 1..1

TheLearningResult points to a TextFile, Multiplicity 1..1.

4.3.5.2 Regression

This is the learning task where the target attribute is numerical.

Supertype DataMiningStep

Associations

– *input*

TheExamples *(inherited)* points to a Concept, Multiplicity 1..1.

TheTargetAttribute is a BaseAttribute (Scalar), Multiplicity 1..1.

– *Constraints*

TheTargetAttribute belongs to TheExamples.

4.3.5.2.1 SupportVectorMachineForRegression

Supertype Regression

Associations

– *input*

ThePredictingAttributes is a set of BaseAttribute(Scalar), Multiplicity 1..n.

C points to a BaseAttribute(Scalar), Multiplicity 1..1.

LossFunctionPos points to a BaseAttribute(Scalar), Multiplicity 1..1.

LossFunctionNeg points to a BaseAttribute(Scalar), Multiplicity 1..1.

Epsilon points to a BaseAttribute(Scalar), Multiplicity 1..1.

KernelType points to a Constant.

– *output*

TheSVMModel is a SVMModel, Multiplicity 1..1.

TheLearningResult points to a TextFile, Multiplicity 1..1.

4.3.5.3 Deviation Detection / Subgroupdiscovery

The subgroup discovery algorithms have been developed especially for knowledge discovery in databases. Many algorithms exist. We characterise here the abstract class and illustrate it by the simple discovery of subgroups, Sidos, and the multi-relational Midos.

Abstract

yes

Supertype DataMiningStep

Associations

– *input*

TheConcept points to a Concept, Multiplicity 1..1.
TheAttribute points to a BaseAttribute (domainDataType=Categorial), Multiplicity 1..1.
SolutionSize points to a Value (dataType=Integer, domain=greater-than (value=1)), Multiplicity 1..1.
MinimalSupport points to a Value (dataType=Real, domain=between (value=0,value=1)), Multiplicity 1..1.
SearchDepth points to a Value (dataType=Integer, domain=greater-than (value=1)), Multiplicity 1..1.

– *output*

TheRuleset points to a DeviationRules, Multiplicity 1..1.

– *Constraints*

TheAttribute is present in TheConcept

All attributes A_i of TheConcept have (domainDataType=Categorial)

4.3.5.3.1 Sidos

Abstract

no

Supertype Deviation Detection / Subgroupdiscovery

Associations

- *input*

4.3.5.3.2 Midos

The RelationChain is as described in the multi-relational feature construction.

Abstract

no

Supertype Deviation Detection / Subgroupdiscovery

Associations

- *input*

  BackgroundConcepts points to a set of Concept, Multiplicity 0..n.

  RelationChain points to a set of Relationships, Multiplicity 0..m.

- *Constraints*

  All Concepts in BackgroundConcepts are "reachable" via Relations in RelationChain from TheConcept

4.3.5.4 Clustering

Abstract

yes

Supertype DataMiningStep

Associations

- *input*

  MaxClusters points to a Value (dataType=Integer domain=greater-than (value=1)), Multiplicity 1..1.

- *output*

  ClusterCenters points to a Concept, Multiplicity 1..1.

4.3.5.4.1 K-means

Abstract

no

Supertype Clustering

Associations

– *input*

DistanceMeasure points to a Value (dataType=Function), Multiplicity 1..1.

MaxIterations points to a Value (dataType=Integer domain=greater-than (value=1)), Multiplicity 1..1.

4.3.6 TimeOperator

Handling time becomes more and more important in knowledge discovery. In addition to statistical time series methods which analyse the particular function of a series of measurements, we also consider sequences of events ore relations between time intervals [8]. Very often, time phenomena are not handeled using specialised data analysis techniques. Instead, the time stamped or continuous data are transformed such that the result of this preprocessing can be handled by standard tools for the data mining step. Hence, preprocessing operators are particularly important for data including time information [7].

Supertype Operator

Attributes

loopable - no

Associations

– *input*

TimeSeries is a Concept, Multiplicity 1..1.

TimeAttribute is a BaseAttribute (domainDataType = Time) of the Concept, Multiplicity 1..1 .

ValueAttributes points to n BaseAttribute of the Concept. Multiplicity 1..n.

– *output*

OutputTimeSeries points to a Concept, Multiplicity 1..1.

OutputValueAttributes points to BaseAttribute. Multiplicity 1..n.

– *Conditions*

TimeSeries is ascendingly sorted over TimeAttribute.

This superclass of time-specific preprocessing takes as input a table with n ValueAttributes and one attribute for time points. The output is another table with the same value attributes but usually much less rows, since the operator aggregates rows.

4.3.6.1 Windowing

The windowing operators split a time series into several time intervals of the size WindowSize. This window is moved over the time series. Most often, the window is moved from one time point to the next (WindowMovement 1).

Supertype TimeOperator

Associations

– *input*

WindowSize points to a BaseAttribute (Scalar), Multiplicity 1..1.

WindowMovement points to a BaseAttribute (Scalar), Multiplicity 1..1.

– *output*

OutputTimeAttribute points to a TimeInterval, Multiplicity 1..1.

4.3.6.2 SignalToSymbolProcessing

The signal to symbol method transforms a time series of numerical measurements into a sequence of time intervals with nominal values. The sequence can then be analysed by a data mining algorithm which is not oriented towards time handling. The method does not employ a fixed window size but reads one measurement after the other until the deviation from a summarizing function exceeds the user given tolerance parameter. Then the time interval is closed. A further parameter indicates whether alternating values are to be tolerated. In other words, outliers are either handled as important peaks of measurements (alternating values parameter set to false) or smoothed into the time interval (alternating values parameter set to true).

Supertype TimeOperator

Associations

– *input*

Tolerance points to a BaseAttribute(Scalar), Multplicity 1..1.

Alternating points to a BaseAttribute (Boolean), Multiplicity 1..1.

– *output*

OutputTimeAttribute points to a TimeInterval, Multiplicity 1..1.

IntervalSymbol points to BaseAttribute (Nominal), Multiplicity 1..1.

Gradient to BaseAttribute (Scalar), Multiplicity 1..1.

– *Assertions*

OutputTimeSeries is extensionally equivalent projection of TimeSeries

4.3.6.3 MovingFunction

The standard statistical approaches to time series are subsumed by the class *MovingFunction*. Each subclass characterises a different way of how to handle the attribute values within a window. Are they all of equal importance or weighted? Whereas in statistics, each particular function that summarizes the values within a time window (e.g., average, median, gradient) is considered a method in its own right (e.g., moving average, moving median), we consider the function a parameter of a general method. The function is denoted by its name.

Note, that here we only consider uni-variate times series. If m-variate time series are to be processed, they have to be split into m uni-variate time series, first.

Supertype TimeOperator

Associations

– *input*

ValueAttributes *(specialized)* points to a BaseAttribute (Scalar), Multiplicity 1..1.

InputFunction points to a BaseAttribute (nominal), Multiplicity 1..1.

– *output*

OutputTimeAttribute points to a BaseAttribute (Time), Multiplicity 1..1.

OutputValueAttribute points to a BaseAttribute (Scalar), Multiplicity 1..1.

4.3.6.4 WeightedMovingFunction

The user may weight the values within a time window. Most likely, the measurements of very past timepoints are less important than the more current values. However, the user can give any weighting for the values within a window. For instance, the Hanning weights for windows of the size 3 assign the first and last measurement the weight 0,25 and the middle value the weight 0,5. This weighting scheme worked out so well for many applications that there is a name of it. We allow the user to input a table with just one attribute for the weights and a row for each step within a window. Hence, the table has as many rows as the window size (e.g., 3 rows for the Hanning approach).

Supertype MovingFunction

Associations

– *input*

Weights is a set of BaseAttributes(Scalar), Multiplicity 1..n

– *Constraints*

The sum of the weights is 1.

The number of weights $n = WindowSize$.

4.3.6.5 ExponentialMovingFunction

Exponential moving functions are algorithmically different from weighted moving functions. The weights for the past values (tail) and current values (head) are recursively applied. Applying the user-given function to the $tail_i \times TailWeight$ and $head_j \times HeadWeight$ becomes the new $tail_{i+1}$. Then, the function and the weights are applied to the calculated $tail_{i+1}$ and the new $head_{i+1}$.

Supertype MovingFunction

Associations

– *input*

TailWeight points to a BaseAttribute (Scalar), Multiplicity 1..1.
HeadWeight points to a BaseAttribute (Scalar), Multiplicity 1..1.

– *Constraints*

The sum of TailWeight and HeadWeight is 1.

Chapter 5

Modelling of Case Implementation

The case implementation level mainly contains information needed for algorithms implementing the preprocessing operators. It also contains the links to the data elements which are the input and output for operators. Each operator instance on the conceptual level corresponds to an instance of an *ExecutionElement*. Note that this submodel is necessary only if a detailed tracing of data transformations is intended. In this case, any ExecutionElement (Function, Stored Procedure and SQL-Query) of any (small) granularity has to be linked to a DataObject instance. DataObjects are either ColumnSets or Columns. Note that, a proliferation of ColumnSets and Columns arise. Information for any small, temporary step inbetween is stored in the metadata repository. This information may be useful but the developer has to be aware of what it means to collect and store it.

5.1 ExecutionElement

Supertype ModelElement

Associations

- *source.* It points to the DataObject which is the "source" on which the operator is executed. This "source" may be a Column or a ColumnSet.
- *target.* It points to the DataObject which is the "target" of the operator execution. This "target" may be a Column or a ColumnSet.

The *DataObject* makes the connection between the two parts of the implementation level. It is a placeholder for either a Column or a Columnset which is input and/or output for an ExecutionElement.

5.2 Transformation

A Transformation may be either a Function, a StoredProcedure or the definition of a SQL-Query (not the result).

Function and *StoredProcedure* have as attribute *nameOf* which is the name of the function or StoredProcedure that contains the implemented code. *algorithm* contains the signature of the called function or stored procedure.

For the operator Feature Construction considered above (see Section 4.3.4), there is at least an instance of the class *StoredProcedure* named *Scaling*. Note that *StoredProcedure* contains only the call of the PL/SQL procedure. The code itself is managed by the DBMS and may look as follows:

```
PROCEDURE Scaling(IntervalLowRangeInput REAL,
IntervalHighRangeInput REAL,
IntervalLowRangeOutput REAL,
IntervalHighRangeOutput REAL,
NewColumn REAL, OldColumn REAL)
IS

ScalingFactor REAL;
BEGIN
ScalingFactor = (IntervalHighRangeOutput − IntervalLowRangeOutput) /
(IntervalHighRangeInput − IntervalLowRangeInput)

SET NewColumn AS IntervalLowRangeOutput +
ScalingFactor(OldColumn − IntervalLowRangeInput)
RETURN;
END Scaling
```

The *DataObject* instance as "source" of this *ExecutionElement* is the value of *OldColumn* and the target is *NewColumn*. Note that Transformations work with implementation data types like REAL, INTEGER and not with (conceptual) mining types (e.g., SCALAR, NOMINAL, BINARY). Moreover, note that the procedure above could be "broken" in smaller code modules, e.g., *Scaling* could call a function *ComputeScalarFactor* which would be then an instance of the class *Function*. In this way, the implementation of operators could be better tracked and maintained.

For the time handling operators, stored procedures are implemented. The function that is used for summarizing values within a time window is a user-given parameter. Its name is a call of a non-public function. This implementation is not documented by metadata. As stated above: we need not use the implementation level of cases as documentation of the implemented operators (external implementation, function, stored procedure, SQL query). Here, we show how it can be done using the example of the manual missing value operators. We show the possible variations in that these operators do not take the particular function that is used as a user-given parameter. Instead, for each function there exists an operator. Note, however, that $v_{function}$ could well be a user-given parameter.

AssignDefaultAlgorithm

Supertype ExecutionElement

Attributes

- *algorithm* – View T as Select *, v_{def} as A' From T_0 Where $(T_0.A)_{miss}$

AssignModalValueAlgorithm

Supertype ExecutionElement

Attributes

- *algorithm* – Create View T as Select *, v_{modal} as A' From T_0 Where $(T_0.A)_{miss}$

AssignMedianValueAlgorithm

Supertype ExecutionElement

Attributes

- *algorithm* – Create View T as Select *, v_{median} as A' From T_0 Where $(T_0.A)_{miss}$

AssignAverageValueAlgorithm

Supertype ExecutionElement

Attributes

- *algorithm* – Create View T as Select *, v_{avg} as A' From T_0 Where $(T_0.A)_{miss}$

AssignStochasticValueAlgorithm

Supertype ExecutionElement

Attributes

- *algorithm* – Create View T as Select *, v = extract(A,p) as A' From T_0 Where $(T_0.A)_{miss}$

where $extract(A, p)$ is a function which returns a value from the A domain according to probability distribution p.

Chapter 6

Conclusion

This technical report presents a metamodel design for a metadata-driven software package that performs preprocessing for data mining. In the course of software development, metamodel changes are expected. However, the main features derived from the four part distinction (data/case modelling vs. conceptual/implementation representation) should be preserved. Also the view of operators and their most relevant assertions and attributes should remain stable. They are the basis for building the human-computer interface as well as for further work on applicability conditions of operators and conditions for their most effective use. They ease the integration of new operators and the reference to external operators.

One can benefit the most from using metadata-driven software if company wide integration is planned. There are currently two groups proposing metamodel standards to store and exchange metadata within data warehousing area: Object Management Group (OMG)[1] and Meta Data Coalition (MDC)[2]. The OMG standard CWM (Common Warehouse Metamodel)[9] is restricted to (technical) metadata for data warehousing, whereas the MDC standard OIM (Open Information Model)[6] is much broader in scope, covering, besides data warehousing, also aspects like business engineering (business rules, business processes etc), organizational elements, object-oriented analysis and design etc. A unification of the two standards has been recently announced[3]. If a new release of CWM provides us with a unified standard, the metamodel of MiningMart can be adapted to it.

Acknowledgements: This technical report emerged from the collaboration between two projects, SMART (Supporting Metadata for Date Warehousing) and Mining Mart (Enabling End-User Data Warehouse Mining). SMART is partly funded by the Swiss committee of innovation and technology (project number KTI 3979.1). Mining Mart is a European research project (IST-1999-11993) with support of the Swiss federal office for education and science (project number BBW 99.0158).

[1]www.omg.org
[2]http://www.MDCinfo.com/
[3]http://www.cwmforum.org/

Bibliography

[1] Borgida A. Description logics in data management. *IEEE Transactions on Knowledge and Data Engineering*, 7(5):671–682, Oktober 1995.

[2] R. Brachman and T. Anand. The process of knowledge discovery in database. In *Advances in Knowledge Discovery and Data Mining*. AAAI Press, 1996.

[3] B. Chandrasekaran, J.R. Josephson, and V. R Benjamins. What are ontologies, and why do we need them? *IEEE Intelligent Systems*, 14(1):20–26, January/February 1999.

[4] N. Fridman Noy and C.D. Hafner. The state of the art in ontology design. *AI Magazine*, 18(3):53 – 74, Fall 1997.

[5] J.U. Kietz, R. Zücker, and A. Vaduva. MINING MART: Combining case-based-reasoning and multistrategy learning into a framework for reusing KDD-applications. In *Proc. of the 5th Intl. Workshop on Multistrategy Learning (MSL 2000)*, Guimaraes, Portugal, June 2000.

[6] Microsoft. *Open Information Model*. http://www.microsoft.com/sql/ techinfo/openinfo.htm.

[7] K. Morik and H. Liedtke. Learning about time. Technical report, Mining-Mart, Univ. Dortmund, 2000.

[8] Katharina Morik. The representation race – preprocessing for handling time phenomena. In *Machine Learning: ECML 2000*, 2000.

[9] Object Management Group (OMG). *Common Warehouse Metamodel (CWM) Specification*, 2000. OMG Document ad/00-01-01, ad/00-01-02, ad/00-01-03, ad/00-01-11, also see http://www.cwmforum.org/.

[10] D. O'Leary. Using AI in knowledge management: Knowledge bases and ontologies. *IEEE Intelligent Systems*, 13(3):34 – 39, May/June 1998.

[11] D. Pyle. *Data Preparation for Data Mining*. Morgan Kaufmann Publishers, San Francisco, California, 1999.

[12] M. Staudt, A. Vaduva, and T. Vetterli. Metadata management and data warehousing. Technical Report 21, Swiss Life, Information Systems Research, July 1999. ftp://ftp.ifi.unizh.ch/pub/techreports/TR-99/ifi-99.04.pdf.gz.

[13] M. Staudt, A. Vaduva, and T. Vetterli. The role of metadata for data warehousing. Technical Report 99.06., University of Zurich, Dept. of Information Technology, September 1999. ftp://ftp.ifi.unizh.ch/pub/techreports/TR-99/ifi-99.06.ps.gz.

[14] A. Vaduva and K.R. Dittrich. Metadata management for data warehousing: Between vision and reality. Technical report 2000.08, University of Zurich, Dept. of Information Technology, January 2000. ftp://ftp.ifi.unizh.ch/pub/techreports/TR-2000/ifi-2000.08.pdf.

[15] T. Vetterli, A. Vaduva, and M. Staudt. Metadata standards for data warehousing: Open Information Model vs. Common Warehouse Metamodel. *ACM Sigmod Record*, 29(3), September 2000.