M^4– A Metamodel for Data Preprocessing

Anca Vaduva
University of Zurich
Information Technology
CH-8057 Zurich,Switzerland
vaduva@ifi.unizh.ch

Jörg-Uwe Kietz
Swiss Life
IT Research & Development
CH-8022 Zurich, Switzerland
Uwe.Kietz@swisslife.ch

Regina Zücker
Swiss Life
IT Research & Development
CH-8022 Zurich, Switzerland
Regina.Zuecker@swisslife.ch

ABSTRACT

Metadata-driven tools store control information in repositories that are outside of programs and applications. At runtime, this control information (i.e., metadata) is read, interpreted and dynamically bound into software execution. If new requirements arise, metadata may be changed without affecting the programs sharing it and without requiring re-compilation of these programs. Repositories store metadata according to a metadata structure, called a *metamodel*. M^4 is the metamodel used by Mining Mart, a system for supporting data preprocessing for data mining. The aim of this paper is twofold. First, we introduce M^4 (the **M**eta**M**odel of **M**ining **M**art) and present some ideas underlying the design and implementation. Second, we discuss on the basis of M^4 issues related to metadata-driven software: advantages of building and using such software, its weaknesses and the role it plays for metadata management, especially within data warehousing environments.

1. INTRODUCTION

In information systems area, metadata (data about data) is a general notion that captures all kind of information necessary to support the management, query, consistent use and understanding of data. In particular, metadata may be any information related to schema definitions and configuration specifications, physical storage, and access rights, but also end-user-specific documentation, business concepts, terminology, and details about user reports. The easy access and the sharing of metadata may be crucial for facilitating the building, administration, and use of information systems, especially when considering complex environments with heterogeneus data sources and applications, like data warehouse environments. Metadata have to be *captured, stored* and *consistently managed* in order to be uniformly accessible by users and software components. However, the metadata extraction and capture may be a costly activity since metadata may exist everywhere, not only in data dictionaries and certain tool repositories but also hidden in scripts, programs, user manuals, and paper documents. In this context, the use of *metadata-driven* software for achieving certain tasks may enforce, as a side-effect, the metadata capture.

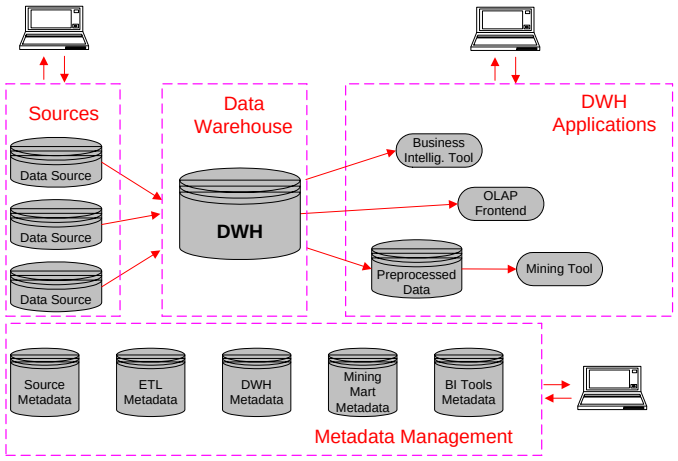

Figure 1: A data warehousing environment

Considering the data warehouse environment in Figure 1, various metadata-driven tools and components are used, for example for loading the data warehouse, for OLAP analysis and for data mining. The architecture of these tools enforces to automatically store some of their metadata in "open" repositories (i.e., with access interfaces). The repositories available may be integrated and metadata management may be consistently and uniformly performed across the entire environment. In this case, the costly step of metadata capture and extraction may be optimized since metadata is available "for free" in these repositories.

The use of metadata-driven software is additionally beneficial for enhancing reusability and flexibility of the system. The reason is that system behavior may be simply changed by updating the control information in the repository such that programs and applications sharing the repository (i.e., the clients) are not affected when software behavior has to be adapted to new requirements.

Repositories store metadata according to a metadata structure, called a *metamodel*. This paper introduces M^4 which is the metamodel used by Mining Mart, a system for supporting data preprocessing for data mining. Data mining algorithms and tools have specific input requirements which

inherently demand preparation of data before their use (e.g., construction of new features derived from existing ones, data segmentation, sampling and cleansing). Mining Mart proposes a case-based reasoning approach that enables both automatization of preprocessing and reusability of defined preprocessing cases for data mining applications. For fulfilling the purposes of Mining Mart, the special structure of the metamodel plays a crucial role. We address general aspects of M^4 only and give a brief overview of its classes and corresponding associations. For a detailed description of the Mining Mart system and metamodel see [5, 12, 16].

Metadata-driven software systems bring advantages both for achieving their special aims "locally" and for supporting the idea of metadata integration and management across enterprise. Starting with the example of Mining Mart, we discuss aspects related to metadata integration and interoperability, including the OMG efforts to the state of the art.

The remainder of this paper is organized as follows: the next section introduces background information for understanding the purpose of Mining Mart system. Section 3 explains the notion of metadata and metadata-driven software and presents the coarse architecture of Mining Mart system. In Section 4 we present an overview of the metamodel of the repository, M^4, while Section 5 introduces some of the classes of metamodel. Section 6 considers the metadata integration and the flexibility aspect of the metamodel and Section 7 concludes the paper.

2. CONTEXT SETTING

Extracting information and knowledge from data is the purpose of advanced technologies like data mining, data warehousing and information retrieval. Data mining aims at detecting unknown patterns in data which are then used for supporting business analysis and trend prediction. Even though a significant amount of algorithms and tools is available on the market, data mining is a complex task. It needs to be embedded in a comprehensive process, called knowledge discovery in databases (KDD) [2]. Data has to be first collected, selected, integrated, cleaned, and then 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 [10] have shown that 50-80% of the efforts for knowledge discovery are spent for data preprocessing which is not only time-consuming but also requires profound business, data mining and database know-how.

In this context, the aim of the Mining Mart project[1] 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 and adapt these cases. A case consists of the description 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. Each mining task involves a certain mining tool or algorithm with special input requirements and thus the target of the preprocessing chain is data prepared in accordance with these 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 if required (e.g., the selection of a different population) and initiate the case execution. On the other hand, the highly skilled power-user 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. Furthermore, the mining expert has the possibility to specify, if necessary, new operators that do not currently exist.

SPSS Clementine Server[2] and Oracle Darwin[3] are two commercial products for data preprocessing and data mining. Like Mining Mart, their purpose is the automatization of the KDD process. However, due to its special architecture, Mining Mart additionally enforces flexibility and reusability of software and allows the integration of additional tools that (partly) generate the metadata required by preprocessing operators. Other tool packages for data processing such as those for data warehousing (e.g., Oracle Warehouse Builder) are not suitable for preparing data for mining because they do not support domain specific operators. These tools can only perform general-purpose operators for data processing such as filter, aggregation and join.

3. METADATA-DRIVEN SOFTWARE

The particularity of *metadata-driven software* is the capture of (some) control information in a repository, outside of applications and programs. This control information determines the behavior of applications and programs accessing the repository. Examples of control information are static information (such as structure definitions, configuration specifications) as well as some parts of application logic: conditions (e.g., for dynamic SQL), methods, or 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 the repository and the engine and is pieced together at runtime only. Examples of metadata-driven software are the new generation tool packages for data warehousing, e.g., for building the data warehouse (such as PowerCenter[4], Ardent[5]) or for using it (such as Cognos[6], Business Objects[7]).

Metadata-driven software provides a framework consisting of a repository structure and an engine which fits this structure. Users have to specify metadata instances (i.e., to fill in the repository in accordance with this structure) in order to achieve executable task-oriented applications.

[1] http://www-ai.cs.uni-dortmund.de/FORSCHUNG/PROJEKTE/MININGMART/index.eng.html

[2] http://www.spss.com

[3] http://otn.oracle.com/products/datamining

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

[5] http://www.ardentsoftware.com (acquired by Informix)

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

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

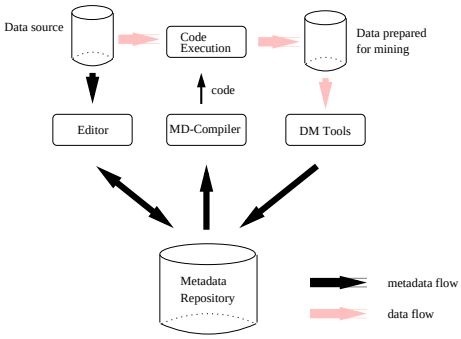

Figure 2: Architecture of Mining Mart System

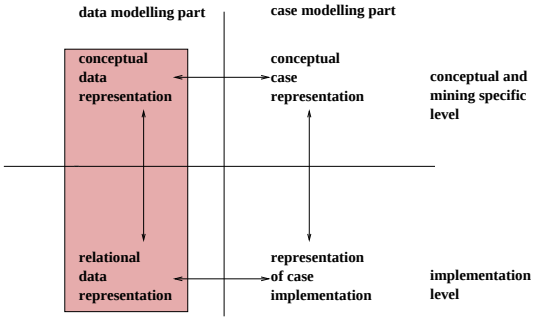

Figure 3: Coarse Description of the Metamodel (the marked rectangle denotes the part of M^4 that considers data representation)

One of the main benefits expected (locally) from metadata-driven software is *reusability* and *flexibility*. On the one hand, objects encapsulating control information are explicitely stored in the repository (instead of being hidden in 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 improved flexibility. The system may be extended and adapted without difficulty since metadata instances may be easily changed without affecting the clients (i.e., engines) sharing it. Thus, *maintenance* is easier.

Nevertheless, the main advantage of using metadata-driven software is when enterprise-wide metadata integration [11] is considered. Metadata stored in various repositories (e.g., from various tools like those for building a data warehouse respectively for using it) is integrated and linked with each other such that it is consistenly 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 available to all users and tools across the enterprise (see Section 6.1).

Mining Mart follows a typical metadata-driven software architecture, depicted in Figure 2. The core of the system is the *Repository* which is implemented on top of a DBMS. Case-specific information is stored in the repository, including: the specification of the business problem to be solved by the case, the specification of structures of the data to be mined, the 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. At runtime, a metadata *Compiler*[8] reads these metadata and uses them in order to generate code. When executing this code, data is read from the data source (e.g., a data warehouse), is preprocessed and stored into the target system on which data mining will be applied later. The *Editor* is used for manipulating metadata (insert, delete, update) within the repository.

Note in Figure 2 that metadata may be produced and stored into the repository by means of other components as well. These components represent *DM tools* (data mining tools) [5] which are used for determining operator parameters when these cannot be manually specified. DM tools accompany preprocessing operators and produce the metadata they require, i.e., the input parameters for them. A typical example is the discretization operator. One of the input parameters is a discretization table which specifies for a given attribute, value intervals and their corresponding discrete values. During preprocessing, the discretization operator checks whether an attribute value is in the range of given intervals and substitutes it with the corresponding discrete value of the input table. Since the manual specification of an optimal discretization table is not always feasible, a DM tool has to be used for discovering the best discretization table by means of data analysis. Nevertheless, when optimal parameter settings depend on data, their discovering by means of DM tools is the prerequisite for automatic case adaption and case reuse. For example, if discretization tables are 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, DM Tools) are "bound" to the given metadata structure which is conceptually described by a *metamodel*. The next section introduces M^4, the **M**ining **M**art **m**eta**m**odel.

4. PARTICULARITIES OF M^4

Before considering some of the classes of M^4 in the next section, we now give a coarse description of the metamodel and explain the reasons for its design. The whole metamodel is depicted in Figure 5 but we consider first the overview of Figure 3. M^4 can be logically divided into two main parts, one managing information with regard to *data modelling* and the other one regarding *case modelling*. Each part is again subdivided in accordance with the abstraction level into *conceptual and mining specific* representation on the one hand and *implementation* representation on the other hand. The four parts resulting from this partition are tightly coupled to each other:

- *Data modelling* part comprises classes for describing

[8]The implementation so far considers a code generator as engine and not an interpreter as in the typical case presented above in this section. However, the basic idea of the engine remains the same.

the *relational data representation*[9], which corresponds to the implementation level and the *conceptual data representation* which essentially deals with the entity-relationship model enhanced with data mining specific aspects (e.g., special data types, including Time, Ordinal, Nominal) and ontology knowledge of the application domain.

- *Case modelling* part describes preprocessing operators and the required controlling structures. This submodel is again divided into the mining-specific description of the case semantics (including for example operators like feature selection and discretization) and their implementation, e.g., function, stored procedure or SQL-query. We call the two metamodel parts *conceptual case representation* and *representation of the case implementation* respectively.

On the one hand, partitioning M^4 in data vs. case modelling representation is necessary for ensuring reusability: the already specified operators may be used within cases which have parameter values represented in the data modelling part. Cases may be reused for different data sets (i.e., populations), represented within the data modeling part. On the other hand, distinguishing between the two abstraction levels (conceptual vs. implementation) 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. Thus, 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 (e.g., which DBMS is used, how functions are implemented). In contrast, the case adapter is responsible for building the connection between the two levels when the structure of the database changes.

- *reusability*: The conceptual, abstract level may be (re)used independently on the actual implementation of the database or of the operators. A relational data model has been chosen for the implementation level so far. That means, the input and output data will be stored in a relational database system (we considered Oracle). However, for the same specification on the conceptual level, also other data models may be used for logical representation.

- *transportability* (which 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] has to be considered within the *conceptual data representation* part. A common ontology basis has to exist which is specialized by domain-specific ontologies.

[9]We have chosen the relational model but other data models may be taken as well.

The four parts of M^4 are linked to each other and connections between metadata instances are often navigable in both directions such that the required information may be rapidly accessed. Note that the consideration of the case implementation submodel (see the right lower part of Figure 3) is optional. Since this part 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 information has to be stored with a coarse granularity only, the two case levels, conceptual case and implementation case representation are merged.

5. DESCRIPTION OF M^4

M^4 combines ideas from two existing standards for metadata representation and exchange in the area of data warehousing (OIM and CWM) [13]. They are drastically simplified but extended with data mining and preprocessing elements to make the metamodel domain-specific. Since both standards have the metamodel of UML[10] as their core, M^4 uses some UML classes as foundation as well. That means, UML is not only used as (graphical) language for describing M^4 in terms of class diagrams, but it is also used as core metamodel which is extended within M^4. In particular, the classes of UML specialized in M^4 are *Class*, *Attribute* and *DataType* (see Figure 5).

For space reasons, we only briefly present some of the classes of M^4 while a more detailed presentation is available elsewhere [12]. The specific impact and contribution of the Mining Mart system as well as a use case for M^4 have been already introduced and extensively discussed in previous work [5, 16]. In the following, we preserve the four part distinction (data/case modelling vs. conceptual/implementation representation) and address classes of each part in turn.

5.1 Conceptual Data Representation

Due to the significant role it plays for user-friendliness and reusability, the conceptual layer is the most important part for the data representation in M^4. It is essentially Entity-Relationship (ER) data representation enhanced with description logic (DL) [1] and ontology representation [3, 4], extended with data mining specific features. The main two classes are *Concept* and *Relationship*. A *Concept* expresses a "thing" in the application domain while a *Relationship* expresses the connection existing between two concepts. 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 4). Application-specific *Relationship*s exist as well (e.g., *Buys*, *Owns Insurance*, *Owns Insurance A*); they are binary Relationships such as in DL, e.g., Concept *Customer* *Buys* Concept *Product*, Concept *Partner* is in Relationship *Owns Insurance* with Concept *Insurance Contract*. Relationships may be bound to each other with *IsA* associations.

As shown in Figure 4, the three perspectives covered in the conceptual data representation are:

[10]http://www.omg.org/technology/documents/formal/unified_modeling_language.htm

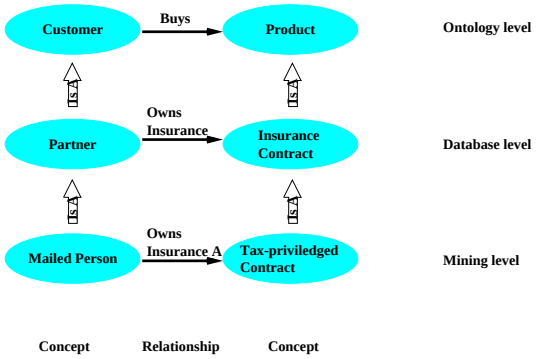

Figure 4: Instances of Concept and Relationship

- the *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 also for Relationships. *Customer* and *Product* are included in the basis ontology which should be common to many companies.
- the *database schema level* represents the conceptual schema of the database; this is mapped to the implemented schema. In Figure 4 this level corresponds to *Partner, Insurance Contract* and *Owns Insurance* which have direct correspondents on the implementation level in terms of relational tables. So far, only the relational model is considered on the implementation level in M^4 (see Section 5.2) 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.
- the *mining data level* describes data sets needed for and produced during preprocessing for mining; it contains also mining specific data types. In Figure 4 these are the *Mailed Persons*, *Tax-priviledged contracts*, and the Relationship *Owns Insurance A*. These are subconcepts and subrelationships of the elements above and are directly used for configuring or designing Mining Mart cases. They correspond on the implementation level to views or snapshots on tables.

The distinction between the three levels (ontology, database and mining specific) is implicitly represented in the instances of *Concept* and *Relationship*. The conceptual data representation of the metamodel in Figure 5 contains only the classes *Concept* and *Relationship* and the associations between them may be illustrated. There are two associations, *FromConcept* and *ToConcept*, which link two *Concepts* with a *Relationship*. For example, Concept *Partner* is in Relationship *HasPartnerRole* to the Concept *Contract*. That means, *Partner* is associated by means of *FromConcept* to *HasPartnerRole* and *Contract* is associated by means of *ToConcept* to Relationship *HasPartnerRole*. For each instance of *FromConcept* association an instance of *ToConcept* association has to exist and conversely. *RoleRestriction* is a special class related to the *FromConcept* and *ToConcept* associations. It expresses the constraints "all, atleast, atmost" known from description logic for the number of data instances for which a relationship is valid (for example, the customer "Maier" can own at most 3 insurance contracts and at least one).

Considering again Figure 5, another important class of the conceptual representation is *FeatureAttribute* which contain features of *Concepts*. It may be either a *BaseAttribute* or a *MultiColumnFeature*. A *MultiColumnFeature* consists of a set of *BaseAttributes*. Note that a *MultiColumnFeature* has no data type, only a *BaseAttribute* has one. As the name reflects, a *DomainDataType* class represents the domain specific data type of BaseAttribute. In particular, instances of DomainDataType are the data types relevant for mining: *Ordinal* (values are ordered), *Scalar* (distance between two values makes sense), *Binary* (has only two values, 0 and 1), *Time*, *KeyAttribute*, *Spatial*, etc. 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 only. Distance and operators such as +, - are applied to scalar attributes. Logical operators are applicable to binary attributes.

The class *Value* is needed within arithmetic expressions and conditions used in operators (which for example have parameters that are constants and these constants have to be expressed as *Values*). Value 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.

5.2 Relational Data Representation

This submodel comprises classes for representing data structures that use the relational data model. The main classes are: *Column*, *ColumnSet* (with its subclasses *Table*, *View*, *Snapshot*) and *Key*. A *ColumnSet* consists of a list of *Columns*. An instance of the class *Column* defines a set of values in a result set, e.g., a view or a table. All values of the same *Column* are of the same data type. A value from a *Column* is the smallest unit of data that can be selected from a table or view and the smallest unit of data to be updated. The *PrimaryKey* and *ForeignKey* classes represent the corresponding notions known from the relational model. The class *Key* is the superclass of *PrimaryKey* and *ForeignKey* and is an abstract class.

Table, View, Snapshot represent the analogous data structures known from relational database management systems (e.g., Oracle). Compared to their superclass *ColumnSet*, these classes have additional attributes containing information that caracterize them, e.g, the filtering condition (the WHERE part of the SQL- query used for view definition) or attributes for updating as *howRefresh* for Snapshots (i.e., either FAST or COMPLETE) and *refreshInterval*.

Preprocessing specific classes are those containing statistical information necessary during data mining (*ColumnStatistics* and *ColumnSetStatistics*). This information includes statistics of the values for each *Column* (e.g., maximal and min-

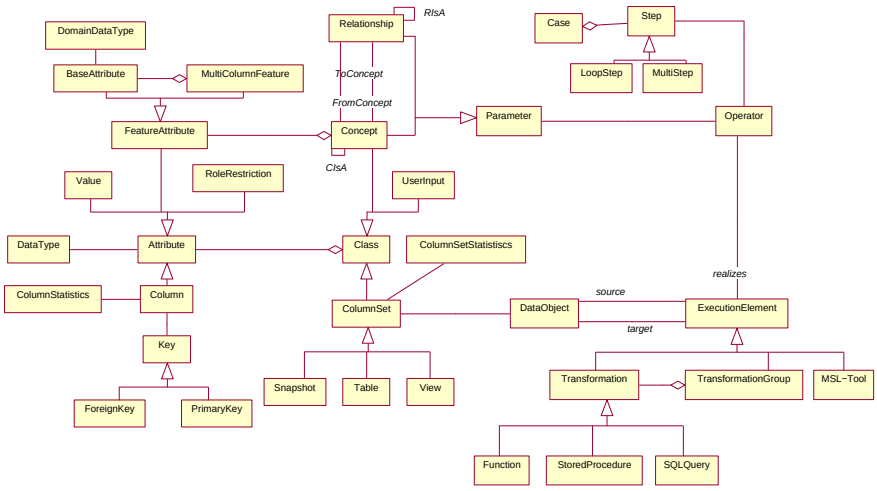

Figure 5: M^4- The MetaModel of Mining Mart

imal value, average) but also the distribution blocks necessary for further preprocessing (if such information is available). Distribution blocks contain values grouped in accordance with some criteria. Mining specific types play an important role when building distribution blocks: for *nominal* attributes, every value is counted and those with the same weight are grouped. For *ordinal* attributes, values are grouped according to certain intervals (e.g., 1000 intervals of equal width).

Regarding the connections between the two levels, conceptual and implementation representation, following constraints apply: each *ColumnSet* has a corresponding *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). There are *ColumnSet*s which do not point to any *Concept* but to a *Relationship* - this is the case when the *Relationship* has the multiplicity m:n, then it will be implemented as a separate *Table*) and 2 *ForeignKey*s. *Relationship*s with multiplicity 1:m or 1:1 will be implemented as *ForeignKey*s. Each *Column* corresponds to a *BaseAttribute* at the conceptual level.

5.3 Conceptual Case Representation

Classes in this submodel manage metadata for configuring cases, preprocessing operators and metadata for connecting these operators. A *Case* consists of a list of *Steps* and each step embeds an *Operator*. 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 when it has to be re-executed. *Step*s are part of cases and they make sense only in connection with a case. Each *Step* has to be linked to a set of predecessors and successors which are *Step*s as well. In this way, parallelisation of operator execution is possible because no fixed sequence is enforced, only dependency is specified; the *Operator* may be executed only if its predecessors have been executed. The output of a *Step* is needed as the input to the next one. Instances of class Operator are all the special preprocessing operators required for building a case [5], including *FeatureConstruction*, *FeatureSelection*, *RowSelection*, *Segmentation*. Each Operator instance points to an *ExecutionElement* on the implementation level.

*Operator*s have *Parameters* which are instances of *Concept*s, *Relationship*s, *FeatureAttribute*s or *Values* (for layout reasons, not all these semantics are represented in Figure 5). Cases have as parameters, among others, the population (i.e., the base concept) and the target attribute which need to be specified at the conceptual level. The result of the preprocessing chain is usually a (sub)concept (of the base concept) and one or some *FeatureAttributes* on which the actual data mining has to be finally performed. Parameter instances may be also updated at runtime (in case they are generated by DM-tools).

5.4 Representation of Case Implementation

The representation of case implementation contains informations needed for algorithms implementing the preprocessing operators. It also contains the links to the data elements which are the input and output for the *ExecutionElement*s implementing the operators. Recall that each operator instance on the conceptual level corresponds to an instance of an *ExecutionElement*. This submodel is necessary only if a detailed tracing of data transformations is intended. In particular, this submodel makes sense only if the aggregation between the *TransformationGroup* and *Transformation* is used (see Figure 5); in this case, the *TransformationGroup* corresponding to an *Operator* is broken in smaller pieces of code (e.g. functions or stored procedures) and each piece represents a *Transformation*. Each *Transformation* has as input and output a *DataObject* which is either a *ColumnSet*, *Column* or a *Value*. Note that the consideration of case implementation representation causes a proliferation of *ColumnSet*s and *Column*s because information for (temporary) *Column*s or *ColumnSet*s produced by any small *Transformation* has to be stored in the metadata repository. This information may be useful for maintenance but the developer has to be aware of what it means to collect and store it. In contrast, if each *Operator* is realized by a

single, atomic *ExecutionElement* (be it a StoredProcedure, Function or SQL-Query), the conceptual and implementation case levels may be merged; in this case, the *Operator* can be extended with attributes which store implementation information such as *functionToCall* containing the invokation of the function to be called.

6. DISCUSSION

We used the context of Mining Mart to present a typical example of a metadata-driven tool and the corresponding metamodel. In the following, we discuss some aspects related to metadata. First, the importance of metadata driven tools to enterprise-wide management and integration is addressed. Then, we point out some of the aspects related to the flexibility of metadata-driven software.

6.1 Metadata Management and Integration

Management of metadata has been identified already some years ago as a significant problem of both research [14, 15] and practice [6, 7] of data warehousing. Metadata is captured, generated and managed in order to be used in one of two ways

- as consistent *documentation* about data stores, processes and applications existing in an enterprise. It is needed by users (i.e., end-users, system administrators, and application developers) to achieve their tasks. That means, metadata is explicitly managed to be available to human beeings.
- as *control information* for metadata-driven tools. Metadata are transparently stored in the repository belonging to the tool and is thus implicitly managed for achieving the certain purpose of the tool (like data preprocessing in Mining Mart).

The separate management of metadata in various repositories with various aims (either for documentation purposes or for special tool purposes) brings advantages only locally. The maximal benefit is extracted from metadata if all repositories with metadata are integrated. Integration ensures consistency of metadata across the enterprise, promotes the wide access to all metadata available and supports their actuality. Metadata elements are inherently connected to each other allowing navigation, impact analysis (evaluation of consequences of potential changes before they are executed) and data tracking (reconstruction of the path followed by the data during processing). For example, data tracking is required in a data warehousing environment for figures in reports generated on the basis of data warehouse. Answers to questions like "which fields of which data sources have been used for calculating these figures" can be given only in an integrated metadata management system where all metadata (including those originating from extraction transformation, loading, data warehouse and OLAP) are consistently and uniformly managed and thus all links between related metadata elements are available as well.

Metadata consistency and actuality across the enterprise may be at least partially ensured if *interoperability* between repositories is supported. In this way, information update may follow automatically. For interoperability, bi-directional tool-specific interfaces and a common interchange representation format has to be adopted and used when data is imported or exported between repositories and/or tools. For both, interoperability and integration, the main problem is actually to agree on a common metamodel. In this case, repositories to be integrated or to interoperate have to map their metadata schema to this common metamodel. One step in this direction has been made with the standard for metadata representation and exchange recently proposed by OMG, called Common Warehouse Metamodel (CWM) [8].

If from the beginning repositories (belonging to commercial products or internal software) are designed with a schema compliant with a standard metamodel, then both integration and interoperability are straightforward. The metamodel of Mining Mart system is not entirely compliant with CWM but it has common parts with it (in particular the relational submodel and the transformation submodel of CWM). Also the fact that M^4 is based on fundamental classes of UML is a positive aspect for a potential integration based on OMG standards. Because of its application specific scope, M^4 contains specialized classes for data preprocessing but the general classes which are relevant for interoperability may be easily exchanged.

6.2 Flexibility Aspects

Adaptability and flexibility have ever been goals of software development which are hard to achieve. The idea is to design a system that is able to easily adapt to new business requirements and to easily extend (ideally at runtime) with new functionality. There are various levels in achieving these goals. The first level is to replace hard-coded logic with easy configurable software components which fetch their parameters from a storage place where these parameters may be edited at runtime. Metadata-driven tools satisfy these requirements.

To a higher extent, flexibility and adaptability may be achieved by means of *active (or adaptive) object-models*[11] They support the extension of the system model at runtime. This requires to persistently store the object model outside the application (e.g, in a database system) and interpret it at runtime only. In this way, the object model may be easily changed (usually by means of special purpose user interfaces) and the changes immediately result in a modified behavior. In this context, the long term vision of the proposed model-driven architecture [9] of OMG aims at developing software capable of automatic discovery of properties of its environment and adaption of that environment by various means, including dynamic modification of its own behaviour. However, these goals are not always easy to achieve. Systems become hard to understand and maintain and they can have poor performance also due to the steady access to a database to get the necessary information.

In order to support extensibility of the system, preprocessing-specific operators are specified as instances of the general class *Operator* in the Mining Mart repository. *Operator* instances are the definition of preprocessing operators such as FeatureSelection, FeatureConstruction. The individual parameter values for each operator call are stored in the

[11] http://www.joeyoder.com/Research/metadata

class *Parameter*. This solution allows the easy manipulation (e.g., insert, update) of both operator definitions and their case specific settings. The definition of constraints and assertions applying to each of the operators, are "bound" to every Operator instance separately (for example, constraints on the data type of the input). Constraint checking is implemented in stored procedures. Procedures are called, that means, constraint checking is done before a parameter instance has to be stored during case editing. Constraints spanning over more than one operator (e.g., concerning the order of execution) have to be hard-coded into the editor respectively compiler and this brings inherently limitations to the flexibility. Note that flexibility and user-friendliness would be improved if a declarative language is used for specifying constraints, instead of "hiding" constraint checking in stored procedures and SQL functions.

To summarize, in the actual stage of software development, the consideration of improving flexibility through an adequate constraint and assertion handling is still in its incipient phase. On the other hand, the endeavor to maximal flexibility raises problems for performance and, depending on the purpose, the latter may be more important.

7. CONCLUSION

This paper presents a metamodel design for a metadata-driven software package performing preprocessing for data mining. Using metadata-driven software is particularly beneficial if enterprise-wide management of metadata is planned. In this way, metadata available in repositories may be easily used for integration and this is beneficial for consistency of metadata, uniform and easy access to all information, impact analysis and data tracking. Due to its special design, the metamodel of Mining Mart system fullfills the requirements of automatization of data preprocessing, user-friendliness and reusability. However, the problems of flexibility and adaptability could be only partially solved with the actual solution.

Acknowledgements: This paper has emerged from the collaboration between two projects, SMART[12] (Supporting Metadata for Date Warehousing) and Mining Mart[13] (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 Commission research project (IST-1999-11993) with support of the Swiss Federal Office for Education and Science (project number BBW 99.0158).

8. ADDITIONAL AUTHORS

Peter Brockhausen (Swiss Life, IT Research & Development, email: `peter.brockhausen@swisslife.ch`) and Klaus R. Dittrich (University of Zurich, Dept. of Information Technology, email: `dittrich@unizh.ch`).

9. REFERENCES

[1] A. Borgida. 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. 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. Hafner. The state of the art in ontology design. *AI Magazine*, 18(3):53 – 74, Fall 1997.

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

[6] R. Kimball, L. Reeves, M. Ross, and W. Thornthwaite. *The Data Warehouse Lifecycle Toolkit: Expert Methods for Designing, Developing, and Deploying Data Warehouses*. Wiley, 1998.

[7] D. Marco. *Building and Managing the Meta Data Repository: A Full Lifecycle Guide*. Wiley, 2000.

[8] Object Management Group (OMG). *Common Warehouse Metamodel (CWM) Specification*, 2000. OMG Document ad/2001-02-01, -02, -03, -04, -05, -06, -07, also see http://www.cwmforum.org/.

[9] J. Poole. Model-driven architecture: Vision, standards and emerging technologies. In *Workshop on Adaptive Object-Models and Metamodeling, ECOOP 2001*, April 2001. http://www.cwmforum.org.

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

[11] A. Vaduva and K. R. Dittrich. Metadata management for data warehousing: Between vision and reality. In *International Database Engineering & Applications Symposium, IDEAS'01*, Grenoble, France, July 2001.

[12] A. Vaduva, J.-U. Kietz, R. Zücker, and K. R. Dittrich. M^4 - The Mining Mart MetaModel. Technical report 2001.02, University of Zurich, Dept. of Information Technology, April 2001. ftp://ftp.ifi.unizh.ch/pub/techreports/TR-2001/ifi-2001.02.pdf.

[13] 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.

[14] J. Widom. Research problems in data warehousing. In *Proc of CIKM '95*, Baltimore, Maryland, USA, November 1995.

[15] M. Wu and P. Buchmann. Research issues in data warehousing. In *Datenbanksysteme in Büro, Technik und Wissenschaft (BTW '97)*, pages 61 –82, Ulm, Germany, 1997.

[16] R. Zücker, J.-U. Kietz, and A. Vaduva. Mining Mart: Metadata-driven preprocessing. In *Workshop on "Database Support for KDD"*, Freiburg, Germany, September 2001. http://www.informatik.uni-freiburg.de/~ ml/ecmlpkdd/WS-Proceedings/w09/zuecker.pdf.

[12] http://www.ifi.unizh.ch/dbtg/Projects/SMART/

[13] http://www-ai.cs.uni-dortmund.de/FORSCHUNG/PROJEKTE/ MININGMART/index.eng.html