KDDML System: Reference Guide

ANDREA ROMEI

University of Pisa
Department of Computer Science
`http://kdd.di.unipi.it/kddml`

May 5, 2006
KDDML System Version: 2.0.16 beta

Abstract

Knowledge discovery in databases (KDD) covers a wide range of applicative domains (retail, marketing, finance, e-commerce, biology, privacy, only to cite a few ones), several models of representing extracted patterns and rules (including classification models, association rules, sequential patterns, clusters) and a large number of algorithms for data preprocessing, model extraction and model reasoning.
KDDML is a middleware XML-based language (and system) needed to support the development of final applications or higher level systems which need a mixture of database access, data preprocessing, mining extraction and deployment.
As the name suggests, KDDML is heavily based on XML as a representation language for data, models and queries. The language is primarily intended as a middleware language on the basis of which higher abstraction levels can be built, such as vertical applications or more declarative languages. Also, the language tries to be as much as possible independent from lower level implementations of data mining algorithms, with the aim of confining the technicalities at the level of the implementation of the KDDML system.
This document describes in detail KDDML as KDD system whose design principles are motivated by requirements derived from recurring patterns in the KDD process.

Requirements
Data Mining, UML principles and the KDDML language specification.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

KDDML system overview

In this section the general architecture of the KDDML system is presented. It consists of three layers for data/model access and representation, operators implementation and query interpretation. Modularity and extensibility are basic requirements in the design of the system architecture. The KDDML system includes also a simple GUI for user friendly input of queries and for browsing extracted knowledge. However, strictly speaking, the GUI is not part of the core of the system. In fact, the general idea is that the KDDML queries can be generated by higher layers of abstractions or by other programs, such as a vertical applications that need performing some KDD steps. In other terms, programs can link the KDDML interpreter as an external library/driver, invoking the interpreter during their executions. This document did not treat of the GUI level.

1.1 System architecture

The overall system architecture is depicted in figure 1.1.

KDDML is implemented in Java, in order to be portable, and consists of more than 500 classes. The overall architecture is structured in layers: each layer implements a specific functionality and supplies an interface to the layer above. In the following, we give an overview of the design of each layer, commenting on how they address the required objectives.

1.1.1 Core layer

The bottom layer manages the read/write access to data and models repositories and the read access to data and models from external sources providing programmatic functionality to the higher layers.

A relational table is represented as a proprietary XML file, containing the schemata and the reference to the physical rows, which are stored in a Comma Separated Value (CSV) format. In a CSV file, each record takes one line, and each field is separated by a comma. By convention,

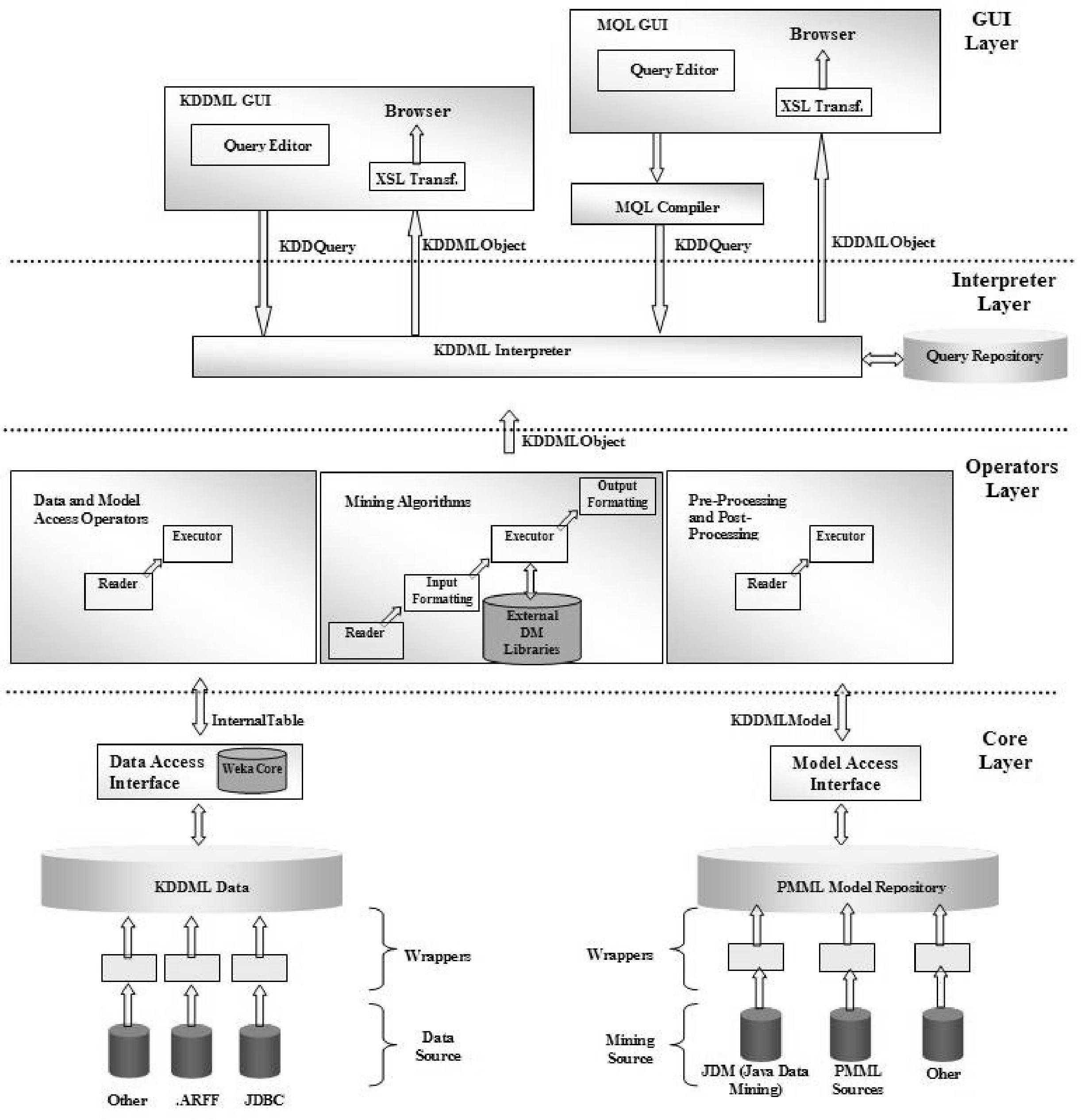


Figure 1.1: KDDML system architecture.

missing or null values are represented with the symbol "?". A preprocessing table (i.e. a table used in the preprocessing step of the KDD process) is similar to a relational table, but in addiction admits a reference to the preprocessing information, which are stored in CSV format too.

A mining model is represented as an extension of the PMML (Predictive Model Markup Language)[1].

Data and models manager

On the one side, the repository layer provides to higher levels a data/model access interface to manage tables and mining models.

Accessing a table yields a Java object satisfying an interface `InternalTableManager`, which abstracts sequential read/write access to (a portion of) the table rows, and provides metadata (such as column types and preprocessing history) and statistics on table columns.

Accessing a model yields a Java object satisfying an interface `AssociationModelManager`, `TreeModelManager`, `ClusteringModelManager`, `SequenceModelManager` or `HierarchyModelManager`. Such interfaces provide programmatic read/write access to the model contents, e.g. `AssociationModelManager` includes methods for adding and removing rules and frequent itemsets. In addition, all the interfaces above extends the interface `KDDMLModelManager`, which provides model metadata (PMML data dictionary and mining schema) read/write access.

Only manager interfaces methods are invoked by the classes belonging to upper level in order to implement the language core operators.

Data and models factory

On the other side, the repository layer includes wrapper modules for accessing and importing tables and models from external sources and to store, retrieve and delete objects in the repositories. Wrappers take care of translating the format of the data into the internal representation (`InternalTable` or `KDDMLModel`). Automatic conversions are performed by the system on data and metadata. Additional conversions can be forced by the preprocessing operators of the KDDML language.

As far as data is concerned, there are wrappers around ARFF text files, serialized Weka tables and RDBMS tables (accessed via JDBC). Concerning models, there is a wrapper importing from PMML files[2] and it is currently being defined a wrapper around serialized JDM (Java Data Mining) objects[3].

Wrappers are organized in factory classes, which provide objects for build data and models. All factory classes extends the main class `KDDMLObjectFactory`. The "factory" approach allows for separating the construction of data/model objects from the data/model object management: adding a new data or model wrapper will not demand in future any change to the manager section.

[1]In actual version, KDDML uses the PMML 2.0 definition.

[2]Since the internal representation is an extension of PMML, this amounts to add default values for tags beyond the standard.

[3]At the present, the system only incorporates some interfaces and empty classes for establishing a connection toward a JDM engine. This interfaces will be implemented in future releases of KDDML.

1.1.2 Operators and algorithms layer

The upper layer is composed of the implementations of language operators and it uses the manager section of the core layer in order to define the kernel of each operator.

Recall below the XML syntax of a generic operator:

```
<OPERATOR_NAME xml_dest="results.xml" att1="v1" ... attM="vM">
   <ARG1_NAME> .... </ARG1_NAME>
   ...
   <ARGn_NAME> .... </ARGn_NAME>
</OPERATOR_NAME>
```

XML tags correspond to operations to data and/or models, XML sub-elements define arguments passed to the operators and finally, XML attributes such as att1, ..., attM correspond to parameters of those operations. Also algorithm specifications admit parameters, as reported below:

```
<ALGORITHM algorithm_name="name">
   <PARAM name="param_name_1", value="param_value_1"/>
   ...
   <PARAM name="param_name_n", value="param_value_n"/>
</ALGORITHM>
```

This level is structured into two horizontal and autonomous sub-layers, called respectively the *operators settings sub-layer* and the *operators resolver sub-layer*.

The first one incorporates the definition of the parameters as used by an operator as well as by an algorithm. As instance, it contains the definition of the minimum support for a generic rules extraction algorithm. This sub-layer performs a correctness control about the types and the usage of input attributes values related to the operator/algorithm. As instances, it checks that the minimum support related to the Apriori algorithm is a real value between 0 and 1.

The operator resolver sub-layer contains the core implementation of the operator/algorithm. We distinguish several different implementations patterns about the operators, according to the KDD step supported and/or to their signature. As instance, mining operators have a typical behavior: they scan the passed data, transform it into the required input, call the appropriate algorithm (typically an external library) and, finally, transform its output into the appropriate `KDDMLModel` object. A set of interfaces and abstract classes are used in order to distinguish the different implementation patterns.

`OPERATOR_NAME` is implemented by means two Java classes placed in the package according to the KDD step supported. The first one incorporates the XML attributes related to the operator and their checking. The second one contains the physical implementation of the operator. Similar, each algorithm (both preprocessing and mining) implements two interfaces that encapsulate, respectively, the algorithm parameters checking and the algorithm core implementation. As for operators, interfaces are organized in a Java hierarchy structure, according to the type of algorithm (e.g. sampling, normalization, clustering) involved.

The separation of the resolver-layer from the settings-layer allow us to separate the definition of the parameters of the operator/algorithm from the real physical implementation.

1.1.3 Interpreter layer

The interpreter layers accepts a validated KDDML query (either in an XML format or as DOM tree), evaluates it, saves the final result into the repository and returns it as KDDMLObject. The result can be further processed by standard XML management tools and libraries.

The evaluation of a query fragment as reported above consist of:

1. recursive evaluation of fragments from `<ARG1_NAME> ... </ARG1_NAME>` to `<ARGn-_NAME> .... </ARGn_NAME>`; in case the ith argument of `<OPERATOR_NAME>` is expected of type `xml`, the element `<ARGi_NAME> ... </ARGi_NAME>` is itself the result of its evaluation;
2. evaluation of attributes `att1 ... attM` returning a set of scalar values;
3. a call to an operator $f_{\texttt{OPERATOR\_NAME}}$, accepting results from (1) and (2) and yielding the final result of the fragment.

Moreover, a copy of the final result (which may be an intermediate result of a possibly larger query) is stored in the (model or data) repository if the attribute `xml_dest` is specified.

1.1.4 User interface layer

The KDDML system includes a GUI for user friendly input of queries and for browsing of extracted knowledge. The GUI allows for:

- opening and modifying an existing query;
- creating a new query through a syntax driven editor, which builds valid queries against the KDDML DTD;
- executing a validated query;
- transforming query results into HTML browsable format via XSL style sheets.

Strictly speaking, the GUI is not part of the core of the system. In fact, KDDML queries can be generated by other programs, such as a vertical applications that need performing some KDD steps. In other words, programs can link the KDDML interpreter as an external library/driver, invoking the interpreter during their executions. This makes running KDDML queries as simple as running SQL queries over RDBMS. The result of the invoked interpreter is returned as a DOM object or an XML document, which can be further processed with standard tools.

1.2 Installing and configuring KDDML

1.2.1 Downloading

The latest version of KDDML is available on the web address:

`http://kdd.di.unipi.it/kddml`.

1.2.2 Installing the Java support

KDDML is written in Java, except for some DM algorithm, which makes it run on almost every platform. However C, C++ libraries are already incorporates inside the KDDML package distribution. Therefore it requires a Java Runtime Environment (JRE) or a Java Development Kit (JDK) version 1.5 or above to be installed properly. Both are available at `http://java.sun.com/`. KDDML can work on windows XP and linux platforms.

1.2.3 Installing and running KDDML

In order to install KDDML, choose an installation directory and uncompress the downloaded archive.

Under windows you can unzip the package using WinZIP or similar and then start the GUI application with the `run_gui.bat` file stored in the root directory.

Under linux you must unzip the package using the command:

```
> unzip kddml_XXX.zip
```

After this, to run the application you must give the right permission, for example, by typing:

```
> cd kddml
> chmod -R 777 *.
```

Finally, you can launch the KDDML low level gui typing

```
> ./run_gui.
```

1.2.4 Integrating KDDML into your application

KDDML can easily be invoked from other Java applications. The class to use is `kddml.Interpreter.QueryExecutor`, and the `Main` method takes the name of the query as unique input parameter[4].

1.2.5 Physical organization and external libraries

The installation of KDDML will create the home directory which contains the files and directories as listed in figure 1.2. More in details, sub-directories include:

- `[KDDML_HOME]/bin`: contains the Java executable bytecode;
- `[KDDML_HOME]/examples`: contains a set of testing queries;
- `[KDDML_HOME]/project`: contains the Java source code;

[4] See the `run_interpreter.bat` and `run_interpreter` executable files placed in the root installation directory.

- `[KDDML_HOME]/repository`: includes the tables/models system repositories;
- `[KDDML_HOME]/resources/DTD`: contains the DTD needed to parse all KDDML objects;
- `[KDDML_HOME]/resources/lib`: contains the external libraries and the data mining executable algorithms;
- `[KDDML_HOME]/resources/xsl`: contains the stylesheet needed to translate XML into HTML documents, for further visualization;

As reported in table 1.1 KDDML uses external libraries in order to implement specific functionality.

External library	Current version	Usage description
WEKA	3.4	weka.core package for instances representation
JASPER	hibernate	XML into HTML translation
JEP	2.3.0	parsing and evaluation of mathematical expressions
qizxopen	0.4	XQuery engine

Table 1.1: The system's external libraries

1.3 Packages design overview

This section is provided to show graphical relationships between the various high level components of the architecture. The figure 1.3 depicts the packages related to the core layer, operators layer and interpreter layers, and above all, it shows the dependencies between them. Notice that the interpreter package uses functionality of the core and operators packages, but not viceversa. In particular, there are not dependencies from the core package to the operator and interpreter packages and from the operator package to the interpreter package. This feature satisfies the architecture level design. Operators and core sub-packages are expanded in the figure 1.4 and 1.5 respectively.

Below, a brief description of the top-level packages is reported:

- `kddml.Utils`: contains utility classes used inside the other packages;
- `kddml.Core`: defines the kernel of the KDDML system; it contains the definition of KDDML objects, i.e. relational tables, preprocessing tables, mining models, KDD queries, scalar values;
- `kddml.Core.DataSources`: manages the data sources (i.e. relational tables and preprocessing tables);
- `kddml.Core.DataMining`: manages mining models;

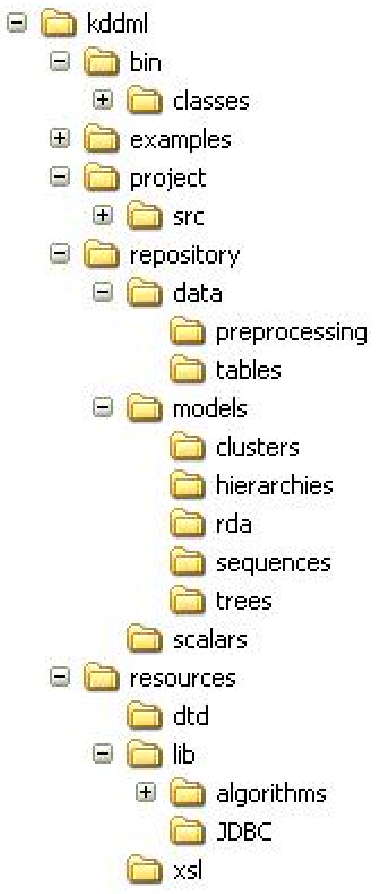


Figure 1.2: The system's home directory hierarchy.

- `kddml.Core.DataMining.AssociationRules`: manages association rules;
- `kddml.Core.DataMining.ClassificationTrees`: manages classification trees;
- `kddml.Core.DataMining.Clustering`: manages clusters;
- `kddml.Core.DataMining.SequentialPatterns`: manages sequential patterns;
- `kddml.Core.DataMining.Taxonomy`: manages hierarchies;
- `kddml.Core.QueryElement`: defines special elements as used inside a KDD query, such as algorithm specifications, conditions or expressions.
- `kddml.Core.Scalar`: defines a scalar value, such as a string or a number;
- `kddml.Operators`: contains the operators and algorithms of the KDDML language;
- `kddml.Operators.DataMining`: contains mining operators for extract models;

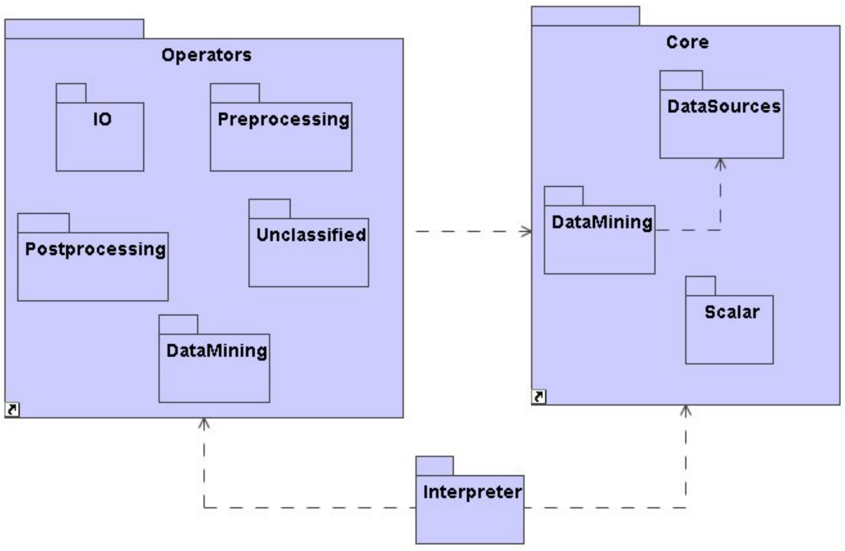


Figure 1.3: The dependencies between the top-level system packages.

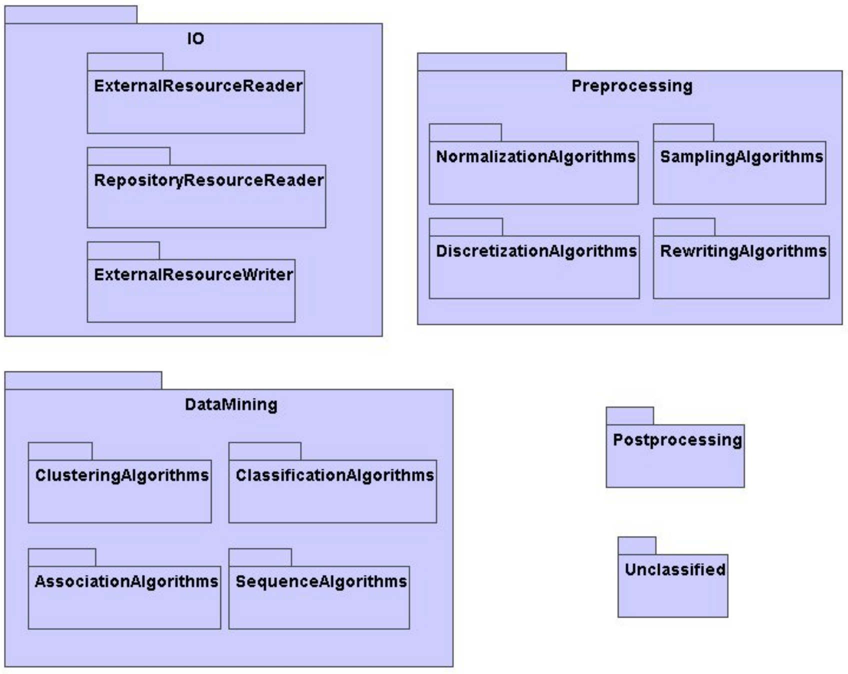


Figure 1.4: The dependencies between the operators layer packages.

- `kddml.Operators.DataMining.AssociationAlgorithms`: contains algorithms to extract association rules;
- `kddml.Operators.DataMining.ClassificationAlgorithms`: contains algo-

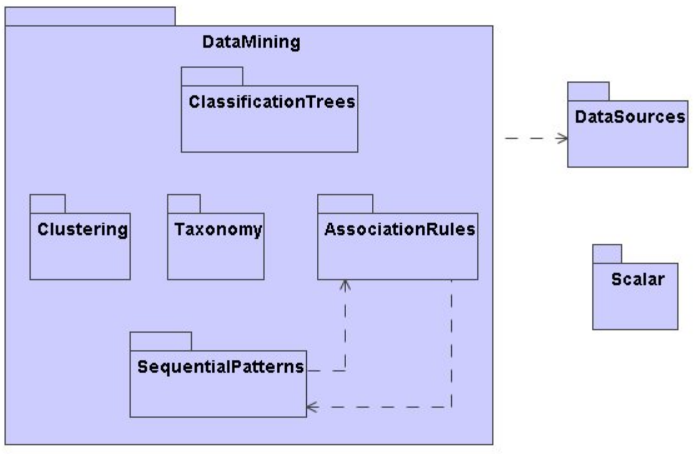


Figure 1.5: The dependencies between the core layer packages.

rithms to extract classification trees;

- `kddml.Operators.DataMining.ClusteringAlgorithms`: contains algorithms to extract clusters;
- `kddml.Operators.DataMining.SequenceAlgorithms`: contains algorithms to extract sequential patterns;
- `kddml.Operators.IO`: defines operators used in order to import/export mining model or data sources from the system;
- `kddml.Operators.IO.ExternalResourceReader`: contains operators that allow to read an external model (e.g. a PMML model) or an external data source (e.g. an ARFF file);
- `kddml.Operators.IO.ExternalResourceWriter`: contains operators that allow to export a proprietary model or a data source into an external representation (e.g. a PMML model or an ARFF file);
- `kddml.Operators.IO.RepositoryResourceReader`: contains operators that loads a model/table from the system repository;
- `kddml.Operators.Postprocessing`: defines operators used in the postprocessing step of the KDD process;
- `kddml.Operators.Preprocessing`: defines operators used in the preprocessing step of the KDD process;
- `kddml.Operators.Preprocessing.DiscretizationAlgorithms`: contains algorithms able to discretize the attribute of an input preprocessing table;

- `kddml.Operators.Preprocessing.NormalizationAlgorithms`: contains algorithms able to normalize a set attributes of an input preprocessing table;
- `kddml.Operators.Preprocessing.RewritingAlgorithms`: contains algorithms able to rewrite the values of an input preprocessing attribute;
- `kddml.Operators.Preprocessing.SamplingAlgorithms`: contains algorithms used for sampling of an input preprocessing table;
- `kddml.Operators.Unclassified`: defines unclassified operators (i.e operators that do not belong to a specific KDD step);
- `kddml.Interpreter`: defines the interpreter layer;
- `kddml.KDDMLGui`: defines the low level GUI.

1.4 Organization of this guide

This document focuses on the architecture of the KDDML system without details about each class and interfaces compounding the system.

In section 2 we present the KDDML object describing the core level KDDML packages and class diagrams. This allow us to illustrate the relationship between the various interfaces and classes. We always distinguish from manager and factory.

In section 3 we present how the KDDML operators and algorithms layer is structured and how the operator layer can be extended in order to add further operators and algorithms.

In section 4 we address the question on the KDDML query executor, showing and explaining the Java fragment related to the core KDDML interpreter.

finally, appendix A, B and C provide UML diagrams about the core layer, the operators layer and the interpreter layer, respectively.

CHAPTER 2

Core layer

This section describes the core level using UML class diagrams. For details on the interfaces and classes depicted below, refer to the accompanying Java documentation produced using Javadoc. In order to make this chapter readable, all UML core figures are placed in appendix A.

2.1 Package kddml.Core

The `kddml.Core` package manages the read/write access to data/models repositories and external resources. This feature is provided by the *factory sub-layer*. On the other side, the classes in the package manages a read access to data/models content, giving a programmatic functionality to the higher layers. This feature is provided by the *manager sub-layer*.

Core manager (see figure A.1)

`KDDMLObject` is the abstract root superclass of all KDDML objects. It contains the following public abstract methods:

- *public abstract KDDMLObjectType getType();*
  returns the type of this object. The class `KDDMLObjectType` contains an enumeration of all legal types and it strictly depends on the KDDML langauge specification.
- *public abstract void saveToRepository() throws KDDMLCoreException;*
  Saves the object into the system repository. The destination path is gathered by means a protected variable belonging to the class that is defined in the constructor. The method throws an exception if an I/O error occurs.
- *public abstract boolean isEmpty();*
  tests if the object is empty. This depends on the type of the object. As instance, if a relational table (resp. a RdA model) do not contain instances (resp. rules) then the method returns *true*.

As shown in the section 4.1, the KDDML query executor uses this methods during the interpreter cycle.

Directly sub-classes of `KDDMLObject` include:

- `kddml.Core.DataMining.MiningModel` used to implement a mining model;
- `kddml.Core.DataSources.InternalTable` that manages tables;
- `kddml.Core.KDDQuery` used to represent a KDD query as XML document;
- `kddml.Core.QueryElement.XMLSpecialElement`, that is a special common superclass containing XML elements as used inside a KDD query, such as conditions specification, algorithm settings and expressions;
- `kddml.Core.Scalar.KDDMLScalar` used to represent both XML attributes for language algorithms/operators specification and simple scalar values that can be returned by a KDD operator.

KDD query (see figure A.2)

A `KDDQuery` contains a KDD query as XML document.

Core exceptions (see figure A.4)

The class `KDDMLCoreException` extends the `java.lang.Exception` to indicate conditions that a reasonable application might want to catch. It has sub-classes to provide more sophisticated exception handling; i.e. a sub-classes for each sub-package defined in the core layer. This class implements the `KDDMLCoreErrorCodes` interface for enable error codes to upper layers.

Core factory (see figure A.5)

The factory classes allow to build a new KDDML object. Since all manager classes have protected constructors, the use of a factory class is required in order to create a new object. KDDML factory includes a set of classes able to build a new mining model, such as a set of association rules or a classification tree, a new data source or a condition/expression specification for a language operator.

The common superclass of factory sub-layer is `KDDMLObjectFactory`, whose main public method is reported below:

- *public abstract KDDMLObject newInstance() throws KDDMLCoreException;*
  returns a new instance of a KDDML object. It uses the manager sub-layer proprietary constructors in order to create the right parts composing the returned object. It throws an exception if an error occurs. The type of the returned object depends on witch class the method is implemented; Java class casting can be used in order to get the required type.

As shown in the section 4.1, the KDDML query executor uses this method during the interpreter cycle.

Concerning data and models, the factory sub-layer is divided into three main set of classes:

1. the *repository factory* is used in order to load a physical object from the system repository. Main sub-classes are `ModelRepositoryFactory` placed in the package `kddml.Core.DataMining` and `InternalTableRepositoryFactory` placed in the package `kddml.Core.DataSources`. Both classes extend the class `KDDMLObjectRepositoryFactory`.

2. the *external resource factory* is used to access an external physical resource, such as a PMML model or an ARFF file stored outside the repository. Main sub-classes are the `ModelExternalFactory` placed in the package `kddml.Core.DataMining` and the `InternalTableExternalFactory` placed in the package `kddml.Core.DataSources`; `KDDMLObjectResourceFactory` is the common superclass of both classes;

3. finally, the *proprietary factory* is used to build the table/model directly from the package classes themselves or from external Java libraries, but not using physical resources. Main classes are the `ModelProprietaryFactory` placed in the package `kddml.Core.DataMining` and `InternalTableProprietaryFactory` placed in the package `kddml.Core.DataSources`; `KDDMLObjectResourceFactory` is the common superclass of both classes.

2.2 Package kddml.Core.DataSources

The `DataSources` package contains classes and interfaces to create and manage relational and preprocessing tables. The available classes contain methods able to:

- read the data schemata of a table providing statistical information about attributes and instances;
- translate a table from an external format into the internal representation (as XML and CSV files)
- load in memory a sequential subset of the instances in order to read its content;
- store the table into the repository as XML and CSV format.

Relational tables manager (see figure A.6)

The class `InternalTable` is used to represent standard relational tables in which each column of the data corresponds to a logical attribute and each row corresponds to an individual case (transaction). It is composed by:

- the data schemata, that includes types of attributes and some simple statistics on instances values (class `DataStatistics`);
- the pointer to the physical data that is stored as a text file in a CSV format.

The class `TransactionalInternalTable` is used to represent data in a transactional format in which the table has an attribute `transaction` identifying the transaction and an attribute `event` containing the single item. Transactions are ordered with respect to the attribute `transaction`.

Finally, the class `TimestampedInternalTable` is similar to a transactional table, but with an extra attribute `timestamp`. This attribute defines a partial time order between transactions and items. Typically, this format is used for sequential patterns analysis.

Preprocessing tables manager (see figure A.7)

A `PPInternalTable` is a table used in the preprocessing task of the KDD process. It is composed by:

- the data schema, that includes attributes types and some simple statistics on attribute values (class `DataStatistics`);
- the pointer to the physical data that is stored as a text file in a CSV format;
- the pointer to the preprocessing data, including preprocessing information such as marks associated to a physical instance value belonging to the data section. Also the preprocessing section is in a CSV format, with the number of columns and rows coinciding with the number of attributes and rows of the data section;
- the statistics related to the preprocessing section (class `DataStatistics`);
- the preprocessing history (class `History`) used to list the set of preprocessing operations performed on the table.

Data statistics (see figure A.8)

`DataStatistics` provides a basic framework for representing univariate statistics. The statistics for a model is made of the collection of the statistics for a single field.

A single `AttributeStatistic` has a type and, optionally, a taxonomy related to the field. The statistic information depend on the type of the attribute. The taxonomy is not required and, it can be related to non-numeric attributes only. Attributes can be discrete (class `NominalStatistic`), numerical (class `NumericalStatistic`) or string (class `String-Statistic`). The `AttributeType` class contains an enumeration of all legal attribute types.

This classes cannot be instanced outside of the package. They are automatically created by using factory constructors.

Relational and preprocessing table factory (see figure A.9)

The factory classes belonging to this package allow to build a new table; in other terms, for each factory class that extends `KDDMLObjectFactory`, the main method *newInstance()* returns a new `InternalTable` or `PPInternalTable` or `TransactionalInternalTable` or a `TimestampInternalTable` instance as output.

Concerning relational tables (both `InternalTable`, `TransactionalInternalTable` and `TimestampInternalTable`), the type of returned object depends on the attributes belonging to the data schemata[1]. In this case, the table format is automatically recognized by the method *newInstance()* when the schemata is parsed. A relational table can be built by using one of the following factory classes:

- `InternalTableRepositoryFactory`: it creates a new relational table porting it directly from the system data repository;
- `WekaFactory`: it creates a new relational table given an instance of the `weka.core.Instances` class;
- `ARFFFactory`: it creates a new relational table from a given ARFF file;
- `DataBaseFactory`: it create a table from gathered from a relational DBMS; the connection uses the JDBC bridge.

A preprocessing table (i.e. `PPInternalTable`) can be built by using one of the following factory classes:

- `PPInternalTableRepositoryFactory`: it creates a new preprocessing table porting it directly form the system data preprocessing repository.
- `PPWekaFactory`: it creates a new `PPInternalTable` given two instances of the class `weka.core.Instances`. The first one contains the real physical instances; the second one contains the preprocessing information related to the physical instances.

2.3 Package kddml.Core.DataMining

Model manager (see figure A.10)

The abstract class `MiningModel` is used to represent a generic PMML document. A mining model is composed by:

- an header containing information on application name and application version;
- a data dictionary, containing definitions for fields as used in mining models.
- a mining schema, containing the field of data dictionary that a user has to provide in order to apply the model.
- the content of the model, varying from a model to another in respect of the type of model extracted (e.g. association rules, clusters, etc.).

[1] Remember that a `TransactionalTable` has an attribute `transaction` identifying the transaction and an attribute `event` containing the single item. A `TimestampInternalTable` is similar to a transactional table, but with an extra attribute `timestamp`.

Data Dictionary (see figure A.11)

The `DataDictionary` contains definitions for fields as used in mining models. It specifies the types and value ranges. These definitions are assumed to be independent of specific data sets as used for training or scoring a specific model. It is composed by a list of `DataField` each of them containing the description of the data field.

The `DataField` is composed by the name, the displayed name (that is a string which may be used by applications to refer to that field), a field type and a vector of enumerated values containing its values. The last one is used for categorical attributes only. The type can be *categorical*, for discrete attributes, *ordinal* for string attributes or *continuous* for numeric attributes.

Mining Schema (see figure A.12)

The `MiningSchema` lists the fields used in the model. These fields are a subset of the fields in the data dictionary. The mining schema contains information that is specific to a certain model, while the data dictionary contains data definitions that do not vary with the model. A `MiningSchema` is composed by a list of `MiningField` each of them containing the description of the mining field.

The `MiningField` is composed by the name and the usage type. The last one can be *active*, an input of the model, *predicted*, an output of the model, or *supplementary*, holding only descriptive information and ignored by the model.

Matrixes (see figure A.13)

In actual version, KDDML implements four types of matrixes:

1. a `CategoryMatrix` is the common superclass that represents a matrix whose axes are categories;
2. a `SymmetricMatrix` is a category matrix in which the value at (j, k) position is the same as the value at (k, j) position;
3. a `DiagonalMatrix` is a symmetric matrix in witch the non-diagonal values have zero value;
4. a `ConfusionMatrix` is a matrix that specifies the statistics of the correct predictions and mispredictions. The rows represent the predicted classes whereas the columns indicate the correct class values. The value at entry (j, k) is the number of predictions for class j when k is the correct prediction. On the rows we assume the actual values. On the columns we assume the predicted values. The class `ConfusionMatrix` is used in the classification model and it is placed in the `kddml.Core.DataMining.ClassificationTrees` package.

Model Factory (see figure A.14)

The `ModelRepositoryFactory` is the common abstract superclass for the repository factory classes managing models.

The `ModelProprietaryFactory` is the common abstract superclass for the proprietary factory classes managing models.

Finally, the `ModelExternalFactory` is the common abstract superclass for the external resource factory classes managing models. It defines a set of proprietary abstract methods to be implemented under the subclasses in order to add a new model external resource (such as a text file containing the model or another physical representation) into the system (see section 2.6.1). Both `ModelExternalFactory` and `ModelProprietaryFactory` extend the most generic abstract class `ModelResourceFactory`.

2.3.1 Package kddml.Core.DataMining.AssociationRules

The `AssociationRules` package contains classes and interfaces to generate and manage association rules and frequent itemsets. The available classes contain methods able to:

- construct a predefined set of association rules or import them from an external resource (e.g. a PMML model);
- add or remove association rules or frequent itemsets;
- read the content of the model;
- translate the set of association rules into a table representation;
- store the model into the repository as PMML or HTML document.

Association rules manager (see figure A.15)

KDDML association model deals with two types of association rules (itemsets):

- *inter-attribute association rules* than have the form *"outlook=sunny AND windy=false* $\rightarrow$ *play=yes"*. This association rules are obtained from inter-attribute transaction.
- *intra-attribute association rules* such as *"spaghetti AND tomato* $\rightarrow$ *parmesan"*; this association rules are generated by using boolean transactions.

An `AssociationModel` consists of a set of attributes (such as the minimum support and the minimum confidence) and three major parts:

1. a set of items;
2. a set of frequent itemsets;
3. a set of association rules.

An association model can contain any number (eventually empty) of items, itemsets and association rules witch are ordered following a precise criterium. They are stored into a `java.Utils.TreeSet` Java structure containing a comparator used to compare the objects (see `CompareItems, CompareItemsets, CompareAssociationRules` classes).

An `Itemset` consists of a set of items with a related support.

An `AssociationRule` defines a relationship between two itemsets where the antecedent implies the consequent. An association rule, $A \rightarrow C$, consists of the support, the confidence, the antecedent `Itemset` (A) and the consequent `Itemset` (C).

Data transactions (see figure A.19)

The class `Transaction` represents a data transaction that is composed by one or more items. `Item` can be *intra-attribute* (or *boolean*), i.e. with no value related, or *inter-attribute*, and it consists of three major parts:

1. the item name;
2. the item value for inter-attribute items only;
3. a mapped value to which the original item name is mapped; this field is not required.

Each transaction only contains boolean items or inter-attribute items. In the first one case, the transaction is said *boolean*; otherwise, the transaction is said *inter-attribute*. No hybrid transactions are allowed inside a model. As an instance, the inter-attribute transaction T_1=$\{$*carType=racing, homeInsurance=low, married=false*$\}$ is composed by three items with names *carType, homeInsurance* and *married* respectively and values *racing, low* and *false* respectively. On the contrary, the boolean transaction T_2=$\{$*milk, bread*$\}$ stores only two boolean items with names *bread* and *milk* and without values associated.

The class `Item` admits additional features, such as the price or the quantity of a purchased product. Each feature is stored in a distinct Java object named `ItemFeature`.

Finally, the `TimestampedTransaction` class is used in the sequential patterns analysis in order to add a timestamp to the related transaction (see sect. 2.3.2).

Association rules factory (see figure A.20)

The factory classes allow to build a new association model; in other terms, for each factory class belonging to the package, the method *newInstance()* returns a new `AssociationModel` as output.

An association model can be built by using one of the following factory classes:

- `EmptyAssociationFactory`: it generates an empty association model, i.e. without itemsets or rules inside; further frequent itemsets or rules can be added later, by using the related methods and constructors defined into the manager sub-layer.
- `ItemsetAssociationFactory`: it constructs a new association model from a given set of `Itemset` classes. A minimum confidence must be also provided in the constructor. The main method *newInstance()* first finds the list of items from the given itemsets, and then it computes the set of association rules that satisfy the specified minimum confidence. A public method allows to fix the maximum number of association rules to be generate.
- `RulesAssociationFactory`: it constructs a new association model given a set of `AssociationRule` classes. A minimum support and a minimum confidence must be

also provided in the constructor. The main method *newInstance()* then computes the set of frequent itemsets from the specified association rules by using the given minimum support. A protected method allows to fix the maximum number of itemsets to be generate.

- `AssociationRepositoryFactory`: it generates an association model directly from the system model repository. In other terms, the main method *newInstance()* finds the physical XML model in the directory repository and then, it parses and transforms the loaded model into an abstract object representation.
- `PMML2_0AssociationFactory`: it constructs a new association model from an external PMML model. This class is PMML 2.0 compatible.

2.3.2 Package kddml.Core.DataMining.SequentialPatterns

The `SequentialPatterns` package contains classes and interfaces to generate and manage sequential patterns. The available classes contain methods able to:

- construct a predefined set of sequential patterns or import them from an external resource (e.g. a PMML model);
- add or remove sequential patterns or rule sequences;
- read the content of the model;
- translate the set of sequential patterns into a table representation;
- store the model into the repository as PMML or HTML document.

Sequential patterns manager (see figure A.21)

A `SequenceModel` consists of a set of attributes (such as the minimum support, the minimum confidence, the number of transaction in the data, etc.) and four major parts:

1. a list of items;
2. a list of sequence elements (i.e. itemsets), composing the sequences;
3. a list of frequent sequences;
4. a list (eventually empty) of sequence rules.

Any object is ordered following a precise criterium and stored into a `java.Utils.TreeSet` Java structure, containing a comparator used to compare the objects (see `CompareSequence-Elements, CompareSequence, CompareSequenceRules` classes).

Items are defined as in the `kddml.Core.DataMining.AssociationRules` package. Sequence elements, sequences and sequence rules are described below.

Sequence element (see figure A.22)

In PMML, the elements that compound a sequence can be either itemsets or set predicates. In current version, KDDML implements itemsets, but it do not support the set predicates. However, the abstract class `SequenceElement` contains the encapsulation of a generic sequence element and can include both itemsets and set predicates as subclasses.

The subclass `ItemSetElement` implements an itemset as used inside a sequence. It is composed by a list of items as in `kddml.Core.DataMining.AssociationRules.Item` and a support related to the sequence.

Sequence (see figure A.23)

A `Sequence` manages sequential patterns, i.e. an ordered collection of itemsets. A delimiter is used as separator between two itemsets inside the sequence (see later). Thus, a `Sequence` consists of:

- the support, i.e. the ratio of the number of objects in the data for which this sequence holds true, to the total number of objects in the data. This attribute is optional.
- the number of occurrences, i.e. the number of objects in the data for which this sequence holds true. This attribute is optional.
- a list of `SequenceElement`, each of them separated by one that follows with a `Delimiter`, that specify the kind of relation between elements.

A sequence must contain at least one `SequenceElement`.

Sequence rule (see figure A.24)

A `SequenceRule` consists of an antecedent `Sequence` and a consequent `Sequence`, separated by a `Delimiter`.

The `Time` between the antecedent sequence and the consequent sequence gives statistics on the elapsed time between the antecedent and the consequence, while the `Time` between the antecedent (resp. consequent) itemsets gives statistics on the total elapsed time from the first to the last itemsets in the antecedent (resp. consequent) sequence rule. The elements `Time` are optional inside a `SequenceRule`: it is only statistics for information, not implying any constraints.

Delimiter (see figure A.25)

A `Delimiter` is the separation between two sets in a sequence, or between two sequences in a sequence rule. It is composed by a `DelimiterType` and a `GapType`.

A `DelimiterType` states whether or not this set predicate occurred within the same event or time period, as defined by a time window, (e.g. session) as the previous one. E.g., if items are purchased during the same visit, delimiter would be *“same time window”*. If items are purchased in separate visits, the value for delimiter would be *“across time window”*. In current version, KDDML manages only *“across time window”* delimiter types.

A `GapType` is the possible existence of set predicates between this and the previous set or sequence. In PMML it can be *"true"*, *"false"* or *"unknown"*. In the first case, it represents an open sequence, which allows for gaps between sequences (as does unknown). In a closed sequence the gap is set to *"false"*, indicating that the two sequences being described are consecutive sets in the data. *"Unknown"* expresses that may exists elements/sequences between two consecutive sequences/rules. In current version, KDDML manages only *"unknown"* gap types.

Data sequence transaction (see figure A.26)

A `DataSequence` is a list of data transactions ordered by means a timestamp.

Each single transaction is represented by means an instance of `TimestampedTransaction` class. It extends the class `Transaction` placed in the `kddml.Core.DataMining.AssociationRules` package and offers a timestamp attribute in addiction.

Sequential patterns factory (see figure A.20)

The factory classes allow to build a new sequence model; in other terms, for each factory class belonging to the package, the method *newInstance()* returns a new `SequenceModel` as output.

A sequence model can be built by using one of the following factory classes:

- `EmptySequenceFactory`: it generates an empty sequence model, i.e. without sequences or sequence rules inside; further elements can be added later, by using the related methods and constructors defined into the manager sub-layer.
- `SequenceFactory`: it constructs a new sequence model from a given set of `Sequence` classes. A minimum support and a minimum confidence must be also provided in the constructor. The main method *newInstance()* do not compute sequence rules. A public method allows to fix the maximum number of sequences to be generate; extra sequences are cut-off from the model.
- `RulesSequenceFactory`: it constructs a new sequence model given a set of `Sequence` classes and a set of `SequenceRule` classes. A minimum support and a minimum confidence must be also provided in the constructor. Two public methods allow to fix the maximum number of sequences and the maximum number of sequence rules to be generate; extra sequences and sequence rules are cut-off from the model.
- `SequenceRepositoryFactory`: it generates a sequence model directly from the system model repository. In other terms, the main method *newInstance()* finds the physical XML model in the directory repository and then, it parses and transforms the loaded model into an abstract object representation.
- `PMML2_0SequenceFactory`: it constructs a new sequence model from an external PMML model. This class is PMML 2.0 compatible.

2.3.3 Package kddml.Core.DataMining.ClassificationTrees

The `ClassificationTrees` package contains classes and interfaces to generate and manage tree models. The available classes contain methods able to:

- construct a predefined classification tree or import it from an external resource (e.g. a PMML model);
- read the content of the model;
- classify a set of given instances;
- store the model into the repository as PMML or HTML document.

The standard PMML model has been extended by KDDML in two cases.

In the first one, it has been used in order to add the notion of confusion matrix to a decision tree model. A confusion matrix can be related both to the training set, used to build the model, and to the test set, used to test the model. The definition is similar in both cases.

The second extension concerns meta-classifiers. We allow for classification models that exploit predictions of two or more decision trees. In actual implementation, KDDML supports three voting strategies: *committe*, *and*, *or*.

Tree model manager (see figure A.28)

The class `TreeModel` manages a classification tree. It consists of a reference to the node root. Each nodes holds a logical predicate expression that defines the rule for choosing the node or any of the branching nodes.

Optionally, a classification model can have two confusion matrixes related to the training and test set respectively. This depends on the algorithm used to build the model. A confusion matrix is a two-dimensional table that indicates the number of correct and incorrect predictions a tree model made on specific data. It provides a measure of accuracy of the model. It has been described in section 2.3.

Tree nodes (see figure A.29)

The main abstract class `Node` is an encapsulation for either defining a split or a leaf in a tree model or a compound tree (using for *and*, *or*, *committee* operations).

A `SimpleNode` represent an internal or a leaf node in a simple classification tree. Every `SimpleNode` contains a predicate that identifies a rule for choosing itself or any of its siblings. A predicate may be an expression composed of other nested predicates. In addition, a `SimpleNode` may contain a score distribution which characterizes the distribution of data at that node with respect to a reference dataset.

Summarizing, a `SimpleNode` consists of:

- the list of children if the node is an internal node. Children are `SimpleNode` themselves;
- the reference to the father node if the node is not the root. The father is a `SimpleNode` itself;

- a not null predicate;
- the score distribution of data;
- the category for the target attribute assigned by the algorithm and, optionally, the number of cases assigned to this tree node.

A `CompoundTree` represents a combination of classification trees. It is composed by a list of one or more `Node` (this means that compound operators can be nested). Combination can be boolean or *committee*.

A `CompoundBooleanTree` implements a combination between two or more classification trees with a boolean operator *and* or *or*. They can be applied only on decision trees, in witch the target attribute is binary and contains a positive class (e.g. *true*, *yes*) and a negative class (e.g. *false*, *no*).

A `CompoundBaggingsTree` implements a combination between two or more classification trees with a voting operator.

Predicates (see figure A.30)

A `Predicate` serves as the common representation for various types of predicates. A predicate can be a simple comparison predicate (e.g. *"*$age < 20$*"*) or a compound predicate (e.g. *"*$age < 20$ *and* $salary < 5000$*"*).

The class `SimplePredicate` represents a simple predicate that consists of defining a rule in the form of a simple boolean expression. The rule consists of a field, a binary comparison operator, and a value. A `SimplePredicate` can be either a `SimpleBooleanPredicate`, a `SimpleComparisonPredicate` or a `SimpleSetPredicate`, depending by the comparison operator.

A `SimpleBooleanPredicate` represents a simple boolean value, such as *true* or *false*.

A `SimpleComparisonPredicate` consists of a single comparison between a logical attribute value and a constant. The constant can be either numeric or categorical. A `SimpleComparisonPredicate` consists of:

- a comparison operator (*equal*, *not equal*, *less than*, *less or equal*, *greater than*, *greater or equal*);
- the name entry of one of the mining field elements at the mining schema;
- the information to evaluate or compare against.

A `SimpleSetPredicate` checks whether a field value is element of a set. The attribute associated with this object can take one of following boolean operators:

- "Is in" which becomes *true* if the value of the attribute is one of the specified set (e.g., *"color is in* {*red, green*}*"*);
- 'Is not in' which becomes *true* if the value of the attribute is not one of the specified set (e.g., *"color is not in* {*white, black*}*"*).

A `CompoundBooleanPredicate` is a set of predicates connected by a logical or relational operators. For example, "*age* $<$ *30 and salary* $>$ *50K*" is a compound predicate which is connected by a boolean *and* operator. Allowed boolean operators are *and*, *or*, *xor* and *surrogate*.

Score Distribution (see figure A.31)

The `ScoreDistribution` implements a method to list predicted values in a classification trees structure. If the node holds an enumeration, each entry of the enumeration is stored in an hashtable structure where:

- the key is the category of the target attribute;
- the value is the size (i.e., the number of records) associated with the category.

When a `SimpleNode` is selected as the final node and if this node has no "score" attribute, then the highest record count in the `ScoreDistribution` determines which value is selected as predicted class.

Tree model factory (see figure A.32)

The factory classes allow to build a new classification tree model; in other terms, for each factory class belonging to the package, the method *newInstance()* returns a new `TreeModel` as output.

A classification tree can be built by using one of the following factory classes:

- `ClassificationProprietaryFactory`: it generates a classification model given a `Node` as root of the tree and, optionally, the test and training confusion matrixes. The input tree can be generated by using the methods and the constructors belonging to the manager sub-layer.
- `ClassificationRepositoryFactory`: it generates a classification model directly from the system model repository. In other terms, the main method *newInstance()* finds the physical XML model in the directory repository and then, it parses and transforms the loaded model into an abstract object representation.
- `PMML2_0ClassificationFactory`: it constructs a new tree model from an external PMML model. This class is PMML 2.0 compatible.

2.3.4 Package kddml.Core.DataMining.Clustering

Clustering methods may be classified into three groups: *distance-based*, *distribution-based* (or *model-based*), *density-based methods*.

Distance-based clustering needs a distance or dissimilarity measurement based on which they try to group those most similar objects into one cluster. K-Means is a distance-based partitioning method.

Model-based or *distribution-based* clustering methods assume the data of each cluster conforms to a specific statistical distribution (e.g. the Gaussian distribution) and the whole dataset is a mixture of several distribution models. EM is an example of distribution-based partitioning clustering that do not require the specification of distance measures.

Density-based approaches regard a cluster as a dense region of data objects.

In actual version, KDDML manages both center-based clustering and distribution-based clustering. For each cluster, a center vector can be given. In center-based models a cluster is defined by a vector of center coordinates. Some distance measure is used to determine the nearest center, that is the nearest cluster for a given input record. For distribution-based models, the clusters are defined by their statistics. Some similarity measure is used to determine the best matching cluster for a given record. The center vectors then only approximate the clusters (i.e. the set of cases belonging them).

The `Clustering` package contains classes and interfaces to generate and manage clusters. The available classes contain methods able to:

- construct a predefined set of clusters or import them from an external resource (e.g. a PMML model);
- read the content of the model;
- get the cluster containing a given input instance;
- store the model into the repository as PMML or HTML document.

Clustering model manager (see figure A.33)

A `ClusteringModel` basically consists of three parts:

1. a cluster description containing the fields as used in the center vectors for each cluster;
2. a comparison measure used to compare a record with a cluster seed that represents the cluster;
3. a set (eventually empty) of clusters. Each cluster is defined by its center vector or by statistics, depending on the type of clustering (center-based or distribution-based respectively) performed; this information is stored in the cluster description.

Cluster Description (see figure A.34)

The `ClusterDescription` contains the fields as used in the center vectors for each cluster. The fields which are used in the center vectors are normalized, in particular this allows to map categorical input fields into numeric values in center vectors. A `ClusterDescription` is composed by the description of one or more `ClusteringField`.

Each `ClusteringField` represents a single clustering field and consists of four major parts:

1. the name referring a mining schema field;
2. a field weight that is the importance factor for the field. This field is used in the comparison functions in order to compute the comparison measure. The default value is 1.0;

3. the similarity scale that is the distance such that similarity becomes 0.5;
4. the attribute comparison measure that is a function of taking two field values and a similarity scale to define similarity/distance. It can be `null`, for example when a distribution-based function is used.

Other features depend on the type of clustering field that can be discrete or continuous.

For a `DiscreteClusteringField` a list of categories is given.

For a `ContinuousClusteringField` a normalization matrix is given. This matrix represents the linear norm for this numeric field. The matrix is 2 X n. The first row contains the origin values. The second row contains the normalized values.

Comparison Measures (see figure A.35)

A `ComparisonMeasure` is used to compare a record with a cluster seed that represents the cluster. This is important in order to determine the nearest cluster for a given input instance. Comparison measures are defined both for centroid-based clustering and distribution-based clustering.

A `DistributionBasedComparisonMeasure` is allowed only for distribution-based clustering. If this is the case, the distribution probability (i.e. the distribution probability that the instance belongs to the set of clusters) is of interest and the comparison measure is computed on cluster statistics information only. In actual implementation, KDDML supports the `EM` (Expectation Maximization) comparison function.

In `CentroidBasedComparisonMeasure`, the distance between the input instance and the seed of the cluster is of interest. A centroid-based measure can be used both for distribution-based clustering and for centroid-based clustering. In the first one case, the seed of the cluster is calculated from the cluster statistics. In the second case, the cluster seed coincides with the cluster centroid. When two records are compared then either the distance or the similarity is of interest. In both cases the measures can be computed by a combination of an *"inner function"* and an *"outer function"*. The "inner function" compares two single field values and the "outer function" computes an aggregation over all fields. The `CentroidBasedComparisonMeasure` describes the aggregation function to determine the similarity between two single cases (outer comparison function). It is composed by the function name and a list of one or more attribute comparison measures ("inner functions"). Depending on the attribute kind, the aggregated value is optimal if it is 0 (for distance measure) or greater values indicate optimal fit (for similarity measure). In actual implementation, the "outer function" can be one of:

- `EUCLIDEAN`;
- `SQUARED_EUCLIDEAN`;
- `TANIMOTO`;
- `MINKOWKI`;
- `CITY_BLOCK`;
- `JACCARD`;

- `SIMPLE_MATCHING`;
- `BINARY_SIMILARITY`;
- `CHEBYCHEV`.

Attribute Comparison Measure (see figure A.36)

The `AttributeComparisonMeasure` class describes the comparison function to be used to determine the similarity between two values of an attribute (*"inner comparison function"*) in a centroid-based comparison measure. In actual implementation, the inner function can be one of:

- `ABSOLUTE_DIFFERENCE`;
- `GAUSSIAN_SIMILARITY`;
- `DELTA`;
- `EQUAL`;
- `SIMILARITY_MATRIX`.

Clusters (see figure A.37)

A `Cluster` represents a single cluster inside the model and it is composed by five major parts:

1. a unique identifier;
2. the cluster name;
3. the number of elements (of the original population) belonging to the cluster;
4. a cluster distribution that holds information about the instances belonging to the cluster. This field is required for distribution-based clustering only and can be omitted otherwise;
5. a covariances matrix that contains information on overall data distribution. A covariance matrix stores coordinate-by-coordinate variances (diagonal cells) and covariances (non-diagonal cells). The covariance matrix is optional.

A cluster can be centroid-based or distribution-based, depending on the type of clustering performed.

In a `CentroidBasedCluster`, a vector of center coordinates is used in order to represents the seed of the cluster.

For a `DistributionBasedCluster`, the seed is defined by the statistics related to this cluster.

Cluster Statistics (see figure A.38)

A `ClusterStatistic` holds statistical information about the overall background population. It is required for distribution-based clustering only.

It is composed by one or more `StatisticField` which stores information about a single mining field belonging to the model. This includes discrete statistics or continuous statistics which include possible field values and interval boundaries.

For discrete fields, a `DiscreteStatisticField` is used. It contains the frequencies of the categories computed on instances belonging to the cluster. This information is stored in a `java.util.Hashtable` structure.

For continuous fields, a `ContinuousStatisticField` is used. It contains the mean and the standard deviation of instances belonging to the cluster.

A `ClusterStatistic` may also contain a cluster priors, that is the prior probability (computed on background population) that an instance belongs to the cluster.

In center-based models, a cluster is defined by a vector of center coordinates. Some distance measure is used to determine the nearest center, that is the nearest cluster for a given input record. For distribution-based models, the clusters are always defined by their statistics. However, the definition of a cluster may contain a center vector as well as statistics.

Clustering factory (see figure A.39)

The factory classes allow to build a new clustering model; in other terms, for each factory class belonging to the package, the method *newInstance()* returns a new `ClusteringModel` as output.

A clustering tree can be built by using one of the following factory classes:

- `EmptyClusteringFactory`: it generates an empty set of clusters. For the clustering, a comparison function and a cluster description are required. They can be generated by using the methods and the constructors belonging to the clustering manager sub-layer. The type of clustering depends on the `ClusterDescription` provided.

- `ClusteringRepositoryFactory`: it generates a clustering model directly from the system model repository. In other terms, the main method *newInstance()* finds the physical XML model in the directory repository and then, it parses and transforms the loaded model into an abstract object representation.

- `PMML2_0ClusteringFactory`: it constructs a new clustering model from an external PMML model. This class is PMML 2.0 compatible.

2.3.5 Package kddml.Core.DataMining.Taxonomy

The values of a categorical field can be organized in a hierarchy. The representation of hierarchies in KDDML is based on parent/child relationships; a tabular format is used to provide the data for these relationships. The actual values are stored in the hierarchy object. So, the tabular data can also be part of the PMML document itself. The table would be recursive in the sense that a value in the parent column can also appear in the child column.

The `Taxonomy` package contains classes and interfaces to generate and manage hierarchies. The available classes contain methods able to:

- construct a predefined hierarchy or import them from an external resource (e.g. a PMML model);
- remove nodes from the hierarchy;
- read the content of the hierarchy;
- store the model into the repository as PMML or HTML document.

Taxonomy manager (see figure A.40)

A `HierarchyModel` contains an item hierarchy. It consists of the hierarchy name and the pointer to the hierarchy root. Root node, leaf nodes and internal nodes of the hierarchy are managed by using the `HierarchyNode` class.

A HierarchyNode is composed by the node name, the pointers to the children (eventually empty) and the pointer to the father node (eventually empty). Father node and children nodes are `HierarchyNode` themselves.

Taxonomy factory (see figure A.41)

The factory classes allow to build a new hierarchy; in other terms, for each factory class belonging to the package, the method *newInstance()* returns a new `HierarchyModel` as output.

A hierarchy can be built by using one of the following factory classes:

- `HierachyHashtableFactory`: it generates a new taxonomy model given a `java.utils.Hashtable` containing the association child/parent of the hierarchy nodes. In the hash-table, the key is the node children name and the value is the parent node name.
- `HierarchyRepositoryFactory`: it generates a hierarchy model directly from the system model repository. In other terms, the main method *newInstance()* finds the physical XML model in the directory repository and then, it parses and transforms the loaded model into an abstract object representation.

2.4 Package kddml.Core.QueryElement

This package includes classes that define an XML special elements to be used inside a KDD query, such as conditions and algorithm specifications, mathematic expressions or generic XML elements.

Query element manager (see figure A.42)

`XMLSpecialElement` is the common abstract class for a KDD query elements. It includes as subclasses:

- a condition specification;
- an expression;
- an algorithm specification;
- a generic XML element.

Algorithm specification (see figure B.2)

The class `KDDMLAlgorithm` manages parameters associated with a particular preprocessing or mining algorithm. It is composed by the algorithm name and a list of parameter settings containing the parameter name (*formal parameter*) and the parameter value (*actual parameter*).

Generic XML element (see figure ??)

The class `XML` incorporates a generic XML object belonging to the KDD query. It is stored as XML element that is evaluated directly from the language operator.

2.4.1 Package kddml.Core.QueryElement.Condition

This package implements a condition specification as used in a KDD query. A condition is useful to evaluate boolean operators on table attributes and/or constants (e.g. *"give me all rows in which the attribute* `temperature` *is less than 80"*). Recall its DTD reported in figure 2.1.

```
<!ELEMENT CONDITION (TRUE|FALSE|OR_COND|NOT_COND|AND_COND|BASE_COND)>
<!ELEMENT TRUE EMPTY>
<!ELEMENT FALSE EMPTY>
<!ELEMENT OR_COND ((OR_COND|NOT_COND|AND_COND|BASE_COND),
                   (OR_COND|NOT_COND|AND_COND|BASE_COND)+)>
<!ELEMENT AND_COND ((OR_COND|NOT_COND|AND_COND|BASE_COND),
                    (OR_COND|NOT_COND|AND_COND|BASE_COND)+)>
<!ELEMENT NOT_COND ((OR_COND|NOT_COND|AND_COND|BASE_COND))>
<!ELEMENT BASE_COND EMPTY>
<!ATTLIST BASE_COND op_type %string; #REQUIRED
                    term1 %any_type; #REQUIRED
                    term2 %any_type; #IMPLIED
                    term3 %any_type; #IMPLIED>
```

Figure 2.1: The DTD of the element `CONDITION` as used in a KDD operator.

Condition Manager (see figure A.45)

A `Condition` can be a boolean condition, a base condition or a compound condition.

Boolean condition (see figure A.46)

The class `BooleanCondition` represents a basic boolean constant, such as *true* or *false*.

Base condition (see figure A.48)

The abstract class `BaseCondition` encapsulates the primitive `BASE_COND` element, as shown in figure 2.1. A primitive condition is used in order to evaluate boolean operators (such as *"not equal"*) on table attributes and/or constants. Here, table attributes stand both for relational (or preprocessing) table columns and model properties (e.g. the support of an association rule). Table attributes can be referred by using the special symbol "`@`" in the value attribute. Operators can be unary, binary or ternary, as expressed by the `term1`, `term2`, `term3` fields in the related DTD.

A `BaseCondition` class is composed by:

- the type of the operator (e.g. `not_equal`, `less_or_equal`);
- a ternary list containing string values that encapsulates the terms. The second (resp. third) element of the list is missing in case of unary (resp. binary) operator type;
- a ternary list containing boolean values. Each boolean value is *false* if the related term value is a table attribute instead of a constant.

The `BaseCondition` class already implements some basic operators, such as `equal`, `not_equal`, `greater`, `greater_or_equal`, `less`, `less_or_equal`, with usual significate. A specific language operator condition extends the abstract class `BaseCondition` and it can define further proprietary base condition operators.

Each primitive condition is accomplished with a description condition class. The abstract class `BaseConditionDescr` contains an enumeration of all operators related to the primitive condition, i.e. it lists the legal condition operators. Moreover, it defines methods performing a correctness checking about terms values, in respect to the operator specification.

Compound condition (see figure A.47)

A `CompoundCondition` realizes a combination between one or more conditions through logical operators, such as `NOT`, `AND`, `OR`. It is composed by a list one or more `Condition` that can be, recursively, both `BaseCondition`, `BooleanCondition` and `CompoundCondition`. In the case of `OrCondition`, the list size is exactly one.

Condition Factory (see figure A.49)

The class `ConditionFactory` allows to build a new condition specification given the DOM representation of the related XML element.

2.4.2 Package kddml.Core.ProprietaryElement.Expression

This package implements an expression specification as used in a KDD operator. An expression is defined by means the combination of mathematical operators (such as `sum`, `division`, etc.) on table attributes and/or constants (e.g. *"sum the content of the attribute* `temperature` *with the content of the attribute* `humidity`*"*). An expression is similar to a condition, but it returns a scalar value instead of a boolean value. The DTD is reported in figure 2.2.

```
<!ELEMENT EXPRESSION (BASE_TERM|SEQ_TERM|IF_TERM)>
<!ELEMENT SEQ_TERM ((BASE_TERM|SEQ_TERM|IF_TERM),
                    (BASE_TERM|SEQ_TERM|IF_TERM)+)>
<!ATTLIST SEQ_TERM op_type
        (concat|equal|sum|multiply|subtract|divide) #REQUIRED>
<!ELEMENT BASE_TERM EMPTY>
<!ATTLIST BASE_TERM value %any_type; #REQUIRED>
<!ELEMENT IF_TERM (CONDITION, (BASE_TERM|SEQ_TERM|IF_TERM),
                              (BASE_TERM|SEQ_TERM|IF_TERM)?)>
```

Figure 2.2: The element `EXPRESSION` as used in a KDD operator.

Expression Manager (see figure A.50)

An `Expression` is composed by a base statement, a sequential statement or a conditional statement.

The class `BaseTerm` provides primitive numerical/string constants or table attribute names. We can distinguish the last one from the constants by using the special symbol "`@`" before the attribute name.

The `SeqTerm` class includes basic operations (*addition*, *multiplication*, *subtraction*, *division*, *comparison* of numbers and *concatenation*, *comparison* of strings) on terms. It is composed by a pointer to the operator type and a list of one or more `Expression`.

Finally, the `IfTerm` class manages the conditional statement used to evaluate a *then* statement or an (optional) *else* statement according to a condition. It is composed by three elements:

1. a required `Expression` representing the *then* statement;
2. an optional `Expression` representing the *else* statement;
3. a condition specification, as defined in the `IF_EXPRESSION_CONDITION` class (see sect. 2.4.1).

Expression Factory (see figure A.51)

The class `ExpressionFactory` allows to build a new expression specification given the DOM representation of the XML element.

2.5 Package kddml.Core.Scalar

This package is used by KDDML to represent XML attributes as used inside a KDD operator or an algorithm specification. It also serves in order to represent the result of language operators whose output is a scalar value.

Scalar Manager (see figure A.52)

A `KDDMLScalar` is composed by a value and a pointer to the type of the object.

The value is a Java math expression (e.g. "3+4*2") or a constant (e.g. "12.3", "foo") that is parsed and evaluated by KDDML when required.

The type is important in order to check the correctness of the input scalar value. Checking is performed, for example, by the KDDML interpreter during the parsing of the KDD query.

Scalar Types (see figure A.53)

The `KDDMLScalarType` incorporates a type related to a scalar value. It can be one of the following:

- `StringType` (see figure A.54): a generic string;
- `FileReaderType` (see figure A.56): a file reader, used in order to read the content of a physical file. In this case, the scalar value contains the pointer to the physical file; KDDML checks whether the specified path exist when checking is performed.
- `FileWriterType` (see figure A.56): a file writer, used in order to write on a physical file. In this case, the scalar value contains the pointer to the physical file; KDDML checks whether the specified path exist when checking is performed.
- `EnumerationType` (see figure A.55): an enumeration in which scalar value can be one of a list of acceptable values.
- `ListType` (see figure A.57): a list of values in a comma separated form. Values can be of various type: `java.lang.String` (e.g. "sunny, overcast, rainy"), `java.lang.Integer` (e.g. "-1, 2, 6"), `java.lang.Double` (e.g. "-3.4, 0.0, 0.2").
- `RangeType` (see figure A.58): range of values used to contain all the values between the minimum and maximum values, where the minimum/maximum value can be considered either included or excluded from the range. Range can be both `java.lang.Integer` and `java.lang.Double` type. KDDML checks that the scalar value is included in the specified range when checking is performed.

The interface `KDDMLScalarTypeEnum` lists all the accepted scalar types. When not specified, `KDDMLScalar` assumes that the type of the scalar is a `StringType` by default.

2.6 Extending the core layer

The design of the KDDML system had to take into special account the requirements of extensibility. The extension related to the core layer can be distinguished into:

- *external data resources extensibility:* means adding a new physical data resource similar to the ARFF format, already included in the system. Typical examples include the `C4.5` data format, files in binary or XML format or proprietary data representation.
- *external model resources extensibility:* means adding a new physical resource to an existent mining model. Typical examples include further versions of PMML, text files or proprietary model representation.
- *condition extensibility*: means adding a new condition specification for a given language operator.
- *models extensibility*: means adding a new form of extracted knowledge to the system. This means also to add a new type to the the core layer.

2.6.1 Adding a new external resource

Adding a new data resource

To add a new external data resource, the abstract class `kddml.Core.DataSources.InternalTableExternalFactory` must be extended. This requires the implementation of some abstract protected methods that are used inside the method `newInstance()` (see the accompanying KDDML and WEKA[2] API Java documentation for details).

Adding a new model resource

To add a new model resource, an appropriate abstract class must be extended. This requires the implementation of some abstract protected methods that are used inside the method `newInstance()`. The table 2.1 specifies which class must be extended according to the type of model involved (see the accompanying API Java documentation for details).

2.6.2 Adding a new base condition specification

To add a new base condition, two appropriate abstract class must be extended:

- `kddml.Core.ProprietaryElement.Condition.BaseCondition`;
- `kddml.Core.ProprietaryElement.Condition.BaseConditionDescr`.

This requires the implementation of some abstract protected methods (see the accompanying API Java documentation for details).

The name of the classes created must coincide with the name of the parent operator containing the condition, followed by the extensions `_CONDITION` and `_CONDITION_DESCR` respectively.

[2]Remember that the package `kddml.Core.DataSources` uses the `weka.core` package in order to represent tables.

Model involved	Class to extend
Association rules	kddml.Core.DataMining.AssociationRules.AssociationExternalFactory
Sequential patterns	kddml.Core.DataMining.SequentialPatterns.SequenceExternalFactory
Tree	kddml.Core.DataMining.ClassificationTree.ClassificationExternalFactory
Clustering	kddml.Core.DataMining.Clustering.ClusteringExternalFactory
Hierarchy	kddml.Core.DataMining.Taxonomy.HierarchyExternalFactory

Table 2.1: Correspondence between models and external resources manager

2.6.3 Adding a new mining model

Adding a new mining model means add a new type to the core level and implement a new package of appropriate classes and interfaces.

Adding a new type

The `kddml.Core.KDDMLObjectType` lists all legal object types as used in the KDDML language specification. A new type (i.e. a new mining model) can be easily added to the system adding to this class the following Java code:

```
...
public final static KDDMLObjectType MY_NEW_MODEL = new KDDMLObjectType(
                          "MyNewModel",
                          "my new model description",
                          RepositoryResolver.MODELS_PATH + "myNewModel" + File.separator,
                          "kdd_query_my_new_model",
                          "MY_NEW_MODEL",
                          "pmml_v2_0.dtd",
                          "MyNewModel.xsl");
...
```

The protected constructor of the class `KDDMLObjectType` accepts as input the following arguments:

1. a name identifying the new model;
2. a model description;
3. the pointer to the physical repository containing the new mining models;
4. the XML entity name as defined in the KDDML DTD definition. It is required for mining models;
5. the operator core name as in KDDML language operator definition (i.e. the prefix used by the KDDML operators that use the model);
6. the name referring the DTD file related to this model (`pmml_v2_0.dtd` if the PMML representation is used);

7. the name referring the XSL file related to this model. The XSL stylesheet file is used in order to translate the XML model representation into an HTML model representation for further model visualization.

Adding a new package

A new package, named as instance `MyNewModel`, must be located in the `kddml.Core.DataMining` sub-package. This package will contain all classes and interfaces able to manage the new model.

The next step is to insert inside this package a set of minimal classes and interfaces and integrate them into the system. In order to do this, the distinction between factory sub-layer and manager sub-layer must be preserved. The table 2.2 shows the minimum set of classes and interfaces to implement and the relative classes and interfaces that must be extended.

New class/interface	**Class/interface to extend**
Factory sub-layer	
MyNewModelProprietaryFactory	kddml.Core.DataMining.ModelProprietaryFactory
MyNewModelExternalFactory	kddml.Core.DataMining.ModelExternalFactory
MyNewModelRepositoryFactory	kddml.Core.DataMining.ModelRepositoryFactory
Manager sub-layer	
MyNewModel	kddml.Core.DataMining.MiningModel
MyNewModelManager	kddml.Core.DataMining.MiningModelManager

Table 2.2: Minimal set of classes/interfaces to define when adding a new model

Further classes and interfaces must be implemented in the factory and manager sub-layers. Concrete classes will extend the external and proprietary factory in order to import the model from various resources. Further classes and interfaces will manage single components of the model providing to the upper layer a programmatic functionality of the content model. Their implementation depends on the structure of the mining model involved and on the functionality provided. However, the general organization of the core layer would not have to be changed.

CHAPTER 3

Operators and algorithms layer

This section describes the operators and algorithms layer using UML class diagrams. For details on the interfaces and classes depicted below, refer to the accompanying Java documentation produced using Javadoc. In order to make this chapter readable, all UML figures are placed in appendix B.

3.1 Package kddml.Operators

The package `kddml.Operators` encapsulates the definition of KDDML language operators and algorithms, both about their settings proprieties and physical concrete realization.

Operator settings and operator resolver (see figure B.1)

Operators settings and operators resolver sub-layers are composed by a set of classes structured in two hierarchies.

The `OperatorSettings` is the root abstract class that incorporates XML attributes specification related to a language operator. Attributes-list declarations specify the name, the usage, the data type, and default value (if any) of each attribute. The type of an attribute can be one of the specified by the class `kddml.Core.KDDMLScalarType`. The usage can be *required* (the attribute must have a value every time this algorithm is listed), *fixed* (it is not required, but if it occurs, it must have the specified default value), or *implied* (it is not required, and no default value is provided). A sub-class that extends `OperatorSettings` requires the implementation of the following abstract methods:

- *public abstract String[] listAttributes()*;
  returns the list of attributes belonging to the operator specification.

- *public abstract Boolean isRequired(String attribute_name)*;
  returns `Boolean.TRUE` if a given attribute is required in the operator specification; returns `Boolean.FALSE` if the attribute is fixed or implied; returns `null` if `attribute_name` do not belong to the list of attributes returned by the method *listAttributes()*.

- *public abstract Boolean isFixed(String attribute_name)*;
  returns `Boolean.TRUE` if a given attribute is fixed (i.e. with a constant value) in the operator specification; returns `null` if `attribute_name` do not belong to the list of attributes returned by the method *listAttributes()*; returns `Boolean.FALSE` otherwise.

- *public abstract String getDefaultAttributeValue (String attribute_name)*;
  returns the default value related to a given input attribute. Returns `null` if the attribute do not admit a default value or if `attribute_name` do not belong to the list of attributes returned by the method *listAttributes()*. An attribute can be given any legal value as a default. The attribute value is not required on each element in the document, but if it is not present, it will appear to be the specified default. If the attribute is implied and it is not included in the element, the operator assumes that this is the attribute value.

- *public abstract KDDMLScalarType getAttributeType(String attribute_name);*
  returns the type related to a given input attribute. KDDML supports the attribute types as expressed by the class `kddml.Core.Scalar.KDDMLScalarType`. Returns `null` if the attribute name do no belong to the list of attributes returned by the method *listAttributes()*.

- *public void checkCorrelationBetweenAttributes(Hashtable<String, KDDMLScalarManager> attributes) throws SettingsException*;
  checks the correctness of the correlation between operator attributes. This method is empty for a specific operator if and only if all attributes are independent. As instance, the `user` attribute and `password` attribute in the `DATABASE_LOADER` operator are independent. On the contrary, the attribute `old_attribute_names` and the attribute `new_attribute_names` in the `PP_RENAME_ATTRIBUTES` operator are not independent because they must share the same number of elements. The method can incorporate further controls between input attributes: e.g., that some particular events are true when the attribute value is used. By default, this method is empty and it can be override under subclasses (i.e. operators) when necessary.

The main purpose of the `OperatorSettings` class is to make a correctness control about the input attributes of `OPERATOR_NAME`. In order to do this, the class contains some methods that can be invoked by the interpreter of the system:

- *public void checkAttributes(Hashtable<String, KDDMLScalarManager> attributes) throws SettingsException*;
  performs a correctness control about the input attributes of the operator. Attributes are given as hash-table where the key is the XML attribute name and the value of the hash-table is the XML attribute value as instance of the interface `kddml.Core.Scalar.KDDMLScalarManager`. More precisely, for each attribute returned by *listAttributes()*, the method checks if the hash-table contains or do not contain the attribute. In the first one case, a

type checking is performed on attribute value comparing the type returned by the method *getAttributeType(attr_name)* with the type of the `KDDMLScalar` representing the value of the attribute. The method throws an exception if a type error occurs for a given attribute. In the second one case, if the attribute is required (i.e. the method *isRequired()* returns `Boolean.TRUE`), then the method throws an exception.

- *public Hashtable<String, KDDMLScalarManager> refreshAttributes(Hashtable<String,-KDDMLScalarManager> attributes)*;
  sets the types and the default values of all attributes returned by the *listAttributes()* method. The type is obtained by means the method *getAttributeType(attr_name)* that takes the attribute name as input. If the attribute value is fixed (i.e. the method *isFixed(attr_name)* returns `Boolean.TRUE`), then its value is replaced with the default value in the input hash-table that is obtained by using the method *getDefaultValue(attr_name)*. Finally, if an optional attribute do not belong to the hash-table and a default value is provided for that attribute, then its value is added to the input hash-table. The method returns the input hash-table updated as output.

The `OperatorResolver` is the root abstract class that contains the physical implementation of the operator. A class that extends the `OperatorResolver` class requires the implementation of the following abstract methods:

- *public abstract KDDMLObjectType getArgumentType(int i)*;
  returns the expected type of the ith argument of the operator. Returns an instance of the `kddml.Core.KDDMLObjectType` class or `null` if the operator do not admit argument as ith child;

- *public abstract boolean runTimeCheckNeeded()*;
  returns `true` if the type of the result of the *execute(settings, arguments)* method is not fixed at compile time, but need to be checked at run-time against the one required by the operator calling. This is required for operators with no fixed output types, such as `EXT_CALL` or `SEQ_QUERY`. By default, this method returns `false`;

- *public abstract boolean abortIfIsEmpty()*;
  returns `true` if the execution of the query must be interrupted by the interpreter when an empty result is returned as output (e.g. a table with an empty set of instances). By default, this method returns `false`.

- *public boolean saveXMLOutput()*;
  returns `true` if the result of the *execute(settings, arguments)* method must be stored in the system repository. If the method returns `false`, then the interpreter will not store the result in the system repository. This is required, as instance, by the repository loading operators, such as `RDA_LOADER` or `TABLE_LOADER`. By default, the operator returns `true`.

- *public abstract int estimatedRAMUsage()*;
  returns the estimated RAM usage for the operator. Must be a value between 0 and $1024MB$. By default, the method returns 256;

- *public abstract boolean highDiskAccessIsNeeded(),;*
  returns `true` if the operator performs an high number of I/O operation to the hard-disk. By default, the method returns `false`;

- *public abstract KDDMLObject execute(Hashtable<String, KDDMLScalarManager> settings, Vector<KDDMLObject> arguments) throws ResolverException, CoreException*;
  returns the result of evaluating the operator over the passed arguments and attributes. Attributes are given as hash-table, where the key is the name of the XML attribute related to the operator and the value is a `kddml.Core.Scalar.KDDMLScalar` object containing the attribute value. Attribute value is checked by the interpreter layer and it is correct when the method is invoked. Arguments are given as vector of `kddml.Core.KDDMLOjbect`. The different implementations of the execute method can be distinguished according to the type of operator.

Another purpose of the `OperatorResolver` class is to perform a correctness control about the types of the input object of the operator, as reported by the method below:

- *public void checkArgumentTypes(Vector<KDDMLObject> arguments) throws ResolverException*;
  checks that input arguments have type as required by the *getArgumentType(i)* method. Throws an exception if an incompatible type is found for an input argument.

Algorithms settings task and algorithms resolver task (see figure B.2)

Algorithms settings task and algorithms resolver task are composed by a set of interfaces structured in two hierarchies.

The `AlgorithmSettingsTask` is the root interface that incorporates XML parameters specification related to an algorithm (both preprocessing and mining). As for the XML attributes related to an operator, parameters-list declarations specify the name, the usage of the parameter, the data type, and default value (if any) of each parameter. Similar to the `KDDMLOperator-Settings` class, methods contained in this interfaces are used to perform a correctness control about algorithm parameters (see the `AlgorithmSettingsTask` interface API for details).

The `AlgorithmResolverTask` is the root interface that incorporates the physical implementation of an algorithm. We distinguished different implementation patterns for each algorithm, according to the type of knowledge extracted.

Typically, a mining algorithm is an external libraries (such as `YaDT`, `DCI` or Weka algorithms) which requires its own input format and provides its own output format. As a consequence, the implementation of the algorithm normally scans passed data, transforms it into the required input, call the actual algorithm, and finally interprets the output to return an appropriate KDDMLObject (see the `DMAlgorithmResolverTask` interface API for details).

Differently from data mining algorithms, preprocessing algorithm are mainly implemented in the KDDML system (i.e., not calling external programs) and they work directly on proprietary preprocessing tables (see the `PPAlgorithmResolverTask` interface API for details).

Operators and algorithms factory (see figure B.3)

Factory classes at operators layer allow to create a new resolver/settings instance of a given KDDML operator/algorithm.

The class `kddml.Operators.KDDMLOperator` manages the creation of operators. KDDML uses the Java reflection in order to load a new operator instance: checking is by name, comparing the XML tag related to the operator with the classes name implementing it. More in detail, an operator is supported if there are two classes in the sub-packages list of the `kddml.Operators` package. The first one extends the abstract class `kddml.Operators.KDDMLOperatorSettings` and contains the attributes specification of the operator; its name ends with the tag `_SETTINGS`. The second one must extend the abstract class `kddml.Operators.KDDMLOperatorResolver` and contains the core implementation of the operator; its name ends with the tag `_RESOLVER`.

The class `kddml.Operators.AlgorithmFactory` produces language algorithms instances. Also for operators, KDDML uses the Java reflection in order to load a new algorithm instance.

Operator exceptions (see figure B.4)

The classes `SettingsException` and `ResolverException` manage exception that can be throw in the settings sub-layer and in the resolver sub-layer respectively.

3.2 Package kddml.Operators.IO

This package includes operators that populate the KDDML data/models repository from external resources or that export a table/model in the repository to external format. The package is structured in three sub-packages:

1. `kddml.Operators.IO.RepositoryResourceReader`: it contains the operators that load a table/model from the system repository. A repository resource reader operator has a fixed signature, with no children and only one attribute, named `xml_source`. This attribute identifies the object in the repository.

2. `kddml.Operators.IO.ExternalResourceReader`: it includes operator that access to an external table/model resource (e.g. an ARFF file or a PMML model) and transform it into the internal proprietary representation. An operator belonging to this package do not admit input arguments, but it uses attributes in order to locate the resource.

3. `kddml.Operators.IO.ExternalResourceWriter`: it contains operators that get a data/model as input and save it into an external representation (e.g. a table into an ARFF file or a model into a PMML model).

Settings (see figure B.5)

RepositoryResourceReaderSettings, ExternalResourceReaderSettings and ExternalResourceWriterSettings are the main settings super-classes of the I/O package. All this classes extend the abstract class kddml.Operators.IO.IOSettings.

Resolver (see figure B.6)

RepositoryResourceReaderResolver, ExternalResourceReaderResolver and ExternalResourceWriterResolver are the main resolver super-classes of the I/O package. All this classes extend the abstract class kddml.Operators.IO.IOResolver.

The *execute(settings, arguments)* method is overrides both in the class RepositoryResourceReaderResolver and in the class ExternalResourceReaderResolver, according to the fixed operator signature.

3.3 Package kddml.Operators.Preprocessing

The package kddml.Operators.Preprocessing contains operators as used in the preprocessing step of the KDD process. Classes are structure in two main groups in order to distinguished operators that load a preprocessing algorithm (such as a sampling or a discretization algorithm) from all other preprocessing operators.

The first one group includes PP_SAMPLING, PP_NUMERIC_DISCRETIZATION, PP_NORMALIZATION and PP_REWRITING operators. This operators admit a fixed signature, taking a preprocessing table as first argument, the algorithm specification as second argument and returning a preprocessing table as output.

The second one group includes all other preprocessing operators. They take a PPtable object as first argument and return a PPtable object, as shown by the following signature:

$$f_{<\mathrm{PP\_...}>} : \texttt{PPtable} \times \ldots \longrightarrow \texttt{PPtable}.$$

Settings (see figure B.7)

The main settings super-classes include PPAlgorithmLoadingSettings and PPSettings.

The first one extends the AlgorithmLoadingSettings class, containing the code to load an external algorithm (both preprocessing and mining). Operator attributes include the xml_dest and the algorithm_name containing the name of the algorithm to load. The last one is added to the list of attributes by the interpreter, during the parsing of the query. The methods *checkAttributes(attributes)* and *refreshAttributes(attributes)* are overloaded, in order to load the external algorithm (by using the AlgorithmFactory class) and to check the algorithm parameters (by using the AlgorithmSettingsTask interface methods).

The PPSettings class extends the OperatorSettings class and contains settings about all other preprocessing operators.

Resolver (see figure B.8)

The main resolver super-classes include `PPAlgorithmLoadingResolver` and `PPResolver`.

The first one extends the `AlgorithmLoadingResolver` class, containing the code to load an external algorithm (both preprocessing and mining). The method *execute(arguments, attributes)* is specialized in the `PP_SAMPLING_RESOLVER`, `PP_NUMERIC_DISCRETIZATION_RESOLVER`, `PP_NORMALIZATION_RESOLVER`, `PP_REWRITING_RESOLVER` classes. This method uses the `AlgorithmFactory` in order to create a new instance of the algorithm as `SamplingAlgorithmResolverTask`, `DiscretizationAlgorithmResolverTask`, `NormalizationAlgorithmResolverTask`, `RewritingAlgorithmResolverTask`, respectively. Then, it uses the methods containing in this interfaces to resolve the algorithm and generate the output preprocessing table.

The `PPResolver` class extends the `OperatorResolver` class and contains the resolver implementation about all other preprocessing operators. It distinguishes three implementation patterns for the *execute(arguments, attributes)* method, according to the type of transformation performed on input preprocessing table. The `PPResolver` class admits three sub-classes:

1. the `SchemaLevelTransformation` class includes preprocessing operators performing an input data scan that modify only the data schemata on the input preprocessing table, without change the physical content of each row (e.g. `PP_FILTER_ATTRIBUTES`, `PP_RENAME_ATTRIBUTES`);

2. the `InstanceLevelDependentTransformation` class includes preprocessing operators performing an input data scan that is dependent on the values assumed by all input instances (e.g. `PP_SAMPLING`, `PP_SORTING_ATTRIBUTES`). This class of operators is main-memory and they cannot work on a single instance at once, but require the entire dataset loaded in RAM. Data schemata do not change;

3. finally, the `InstanceLevelIndependentTransformation` class includes preprocessing operators performing an input data scan that is independent on the values assumed by all other instances (e.g. `PP_NORMALIZATION`, `PP_REMOVE_ROWS`). This class of operators is not main-memory and they can work on a single instance at once. The operator only reads a block of instances, computes the new tuples and forgets the block again; that is, at each point in time only a small sub-set of the database is in main memory. Data schemata do not change.

3.4 Package kddml.Operators.DataMining

The package `kddml.Operators.DataMining` contains operators to extract a mining model by using an external algorithm, such as `RDA_MINER`, `TREE_MINER`, `CLUSTER_MINER` and `SEQUENCE_MINER` operators. This operators admit a fixed signature taking a relational table as first argument, the algorithm specification as second argument and returning a mining model as output:

$$f_{<\mathtt{MODEL\_MINER}>} : \mathtt{table} \times \mathtt{alg} \longrightarrow \mathtt{model}.$$

Settings (see figure B.9)

The main settings super-class is `DMAlgorithmLoadingSettings` that extends the `AlgorithmLoadingSettings` class, containing the code to load an external algorithm (both pre-processing and mining). Operator attributes include the `xml_dest` and the `algorithm_name`, containing respectively the name of output destination and the name of the algorithm to load. The last one is added to the list of attributes by the interpreter, during the parsing of the query. In the `AlgorithmLoadingSettings` class, the methods *checkAttributes(attributes)* and *refreshAttributes(attributes)* are overloaded, in order to load the external algorithm (by using the `AlgorithmFactory` class) and then to check the algorithm parameters (by using the `AlgorithmSettingsTask` interface methods).

Resolver (see figure B.10)

The main resolver superclass is `DMAlgorithmLoadingResolver` that extends the `AlgorithmLoadingResolver` class, containing the code to load an external algorithm (both pre-processing and mining). The method *execute(arguments, attributes)* is specialized in the `RDA_MINER_RESOLVER`, `TREE_MINER_RESOLVER`, `CLUSTER_MINER_RESOLVER` and `SEQUENCE_MINER_RESOLVER` classes. This method uses the `AlgorithmFactory` in order to create a new instance of the algorithm as `AssociationAlgorithmResolverTask`, `ClassificationAlgorithmResolverTask`, `ClusteringAlgorithmResolverTask`, `SequenceAlgorithmResolverTask`, respectively. Then, it uses the methods containing in this interfaces to resolve the algorithm.

3.5 Package kddml.Operators.Postprocessing

This package includes operators as used in the post-processing step of the KDD process.

Settings (see figure B.11)

The class `PostProcessingSettings` manages the parameters settings of the post-processing operators.

Resolver (see figure B.12)

The class `PostProcessingResolver` is the resolver superclass that all post-processing operators must extend. Since a post-processing operator is generic, neither the *execute(settings, arguments)* method, nor other method of the superclass is specialized inside this class.

3.6 Package kddml.Operators.Unclassified

This package includes operators that do not cover a particular phase of the KDD process.

Settings (see figure B.13)

The class `UnclassifiedSettings` manages the parameters settings of the unclassified operators.

Resolver (see figure B.14)

The class `UnclassifiedResolver` is the resolver superclass that all unclassified operators must extend. Since an operator of this type is generic, neither the *execute(settings, arguments)* method, nor other method of the superclass is specialized inside this class.

3.7 Extending the operators layer

The extension related to the operator layer can be distinguished into:

- *preprocessing and mining algorithm extensibility*;
- *operator extensibility*;
- *model extensibility*.

3.7.1 Adding a new algorithm

`ALGORITHM_NAME` (e.g. `APRIORI`) is implemented as two Java classes for settings sub-layer and resolver sub-layer, named respectively `ALGORITHM_NAME_SETTINGS` and `ALGORITHM-_NAME_RESOLVER` (e.g. `APRIORI_SETTINGS`, `APRIORI_RESOLVER`). As reported in table 3.1, the two classes must be placed in the right package location, according to the type of knowledge extracted, and must implement the relative interfaces.

As final step, a new entry to the relative algorithm list must be added in the `kddml.Operators.AlgorithmFactory` class. Lists are implemented as final, static arrays of strings, containing the unique names of the algorithms[1].

3.7.2 Adding a new operator

`OPERATOR_NAME` is implemented as two Java classes for settings sub-layer and resolver sub-layer, named respectively `OPERATOR_NAME_SETTINGS` and `OPERATOR_NAME_RESOLVER`. As reported in table 3.2, the two classes must be placed in the right package location, according to the KDD step supported, and must extend the relative abstract class. All abstract methods of super-classes require to be implemented; other methods can be overloaded only if necessary.

[1]Names are not case sensitive.

Type of algorithm	**Package location**	**Interfaces to implement**
Discretization	kddml.Operators.Preprocessing. DiscretizationAlgorithms	DiscretizationAlgorithmSettingsTask DiscretizationAlgorithmResolverTask
Normalization	kddml.Operators.Preprocessing. NormalizationAlgorithms	NormalizationAlgorithmSettingsTask NormalizationAlgorithmResolverTask
Sampling	kddml.Operators.Preprocessing. SamplingAlgorithms	SamplingAlgorithmSettingsTask SamplingAlgorithmResolverTask
Rewriting	kddml.Operators.Preprocessing. RewritingAlgorithms	RewritingAlgorithmSettingsTask RewritingAlgorithmResolverTask
RdA extraction	kddml.Operators.DataMining. AssociationAlgorithms	AssociationAlgorithmSettingsTask AssociationAlgorithmResolverTask
Classification	kddml.Operators.DataMining. ClassificationAlgorithms	ClassificationAlgorithmSettingsTask ClassificationAlgorithmResolverTask
Clustering	kddml.Operators.DataMining. ClusteringAlgorithms	ClusteringAlgorithmSettingsTask ClusteringAlgorithmResolverTask
Sequence extraction	kddml.Operators.DataMining. SequenceAlgorithms	SequenceAlgorithmSettingsTask SequenceAlgorithmResolverTask

Table 3.1: Correspondence between algorithms and interfaces

3.7.3 Adding a new class of algorithms

KDDML can be extended with further knowledge, as instance with a regression model or a neural network[2]. This operation usually requires to add a new set of algorithm to the system, in order to extract the new mining model.

Step 1: create a new algorithms sub-package

A new package, named as instance `MyNewKnowledgeAlgorithms`, must be located in the `kddml.Operators.DataMining` sub-package. This package will contain all algorithms that can be used in order extract the new knowledge.

Step 2: define the setting and resolver task interfaces

Two interfaces must be defined in order to manage all algorithms belonging to the new knowledge class. The first one, named as instance `MyNewModelAlgorithmSettingsTask`, must extend the interface `DMAlgorithmSettingTask`. The second one, named as instance `MyNew-ModelAlgorithmResolverTask`, must extend the interface `DMAlgorithmSettings-Task`. Both interfaces must be placed in the package created at step 1.

[2]However, this considerations can be extended also to the algorithms used in the preprocessing step.

Type of operator	**Package location**	**Classes to extend**
I/O		
repository reader	kddml.Operators.IO. RepositoryResourceReader	RepositoryResourceReaderSettings RepositoryResourceReaderResolver
external reader	kddml.Operators.IO. ExternalResourceReader	ExternalResourceReaderSettings ExternalResourceReaderResolver
external writer	kddml.Operators.IO. ExternalResourceWriter	ExternalResourceWriterSettings ExternalResourceWriterResolver
Preprocessing		
schema level transformation	kddml.Operators. Preprocessing	PPSettings SchemaLevelTransformation
instance level dependent transformation	kddml.Operators. Preprocessing	PPSettings InstanceLevelDependentTransformation
instance level independent transformation	kddml.Operators. Preprocessing	PPSettings InstanceLevelIndependentTransformation
PP algorithm loading	kddml.Operators. Preprocessing	PPAlgorithmLoadingSettings PPAlgorithmLoadingResolver
Data Mining		
DM algorithm loading	kddml.Operators. DataMining	DMAlgorithmLoadingSettings DMAlgorithmLoadingResolver
Postprocessing		
generic postprocessing	kddml.Operators. Postprocessing	PostProcessingSettings PostProcessingResolver
Others		
unclassified	kddml.Operators. Unclassified	UnclassifiedSettings UnclassifiedResolver

Table 3.2: Correspondence between operators and abstract classes

Step 3: add algorithms to the system

The `kddml.Operators.AlgorithmFactory` produces KDDML algorithms. A new class of algorithms can be easily added to the system adding to this class the following Java code:

```
...
public final static AlgorithmFactory MY_NEW_MODEL_ALGORITHMS = new AlgorithmFactory(
      "MyNewClassAlg", {"my_new_alg_1", "my_new_alg_2"}, "MY_NEW_MODEL_MINER",
      "kddml.Operators.DataMining.MyNewKnowledgeAlgorithms");
...
```

The protected constructor of the class `AlgorithmFactory` that as input the following arguments:

1. a name identifying the class of algorithms;
2. the list of supported algorithms;
3. the XML tag name of the operator supporting the class of algorithms;
4. the package name that allows to locate the algorithms.

In order to implement `my_new_alg_1` and `my_new_alg_2`, their settings and resolver classes must be placed in the package created at step one and must extend the interfaces defined at step two.

Step 4: add a new mining operator

Finally, a new KDDML operator (e.g. `MY_NEW_MODEL_MINER`) that loads the new set of algorithms can be defined as specified in the previous section.

CHAPTER 4

Interpreter layer

This section describes the interpreter level using UML class diagrams. For details on the interfaces and classes depicted below, refer to the accompanying Java documentation produced using Javadoc. All UML figures are placed in appendix C.

4.1 Package kddml.Interpreter

The package `kddml.Interpreter` contains class and interfaces for the execution of a KDD query. The main class is `QueryExecutor` that accepts a validated KDDML query (either in XML format or as a DOM tree), evaluates it, save the final result in the repository and returns it as KDDMLObject. It implements three interfaces in order to provide his services to the upper layers:

1. `InterpreterIO`: it allows to set I/O proprieties to be applied before the query execution;
2. `InterpreterChecker`: it provides methods in order to validate a KDD query, before its execution;
3. `InterpreterRunner`: it contains the methods to manage the main execution of the interpreter cycle.

Exceptions (see figure C.1)

The main class `KDDMLInterpreterException` extends the `java.lang.Exception` to indicate conditions that a reasonable application might want to catch. It has sub-classes to provide more sophisticated exception handling. More in detail:

- `ExecutionException`: exception throws if a generic error occurs during the query execution;

- `TypeCheckingException`: exception throws when a type error occurs during the query execution: as instance, if an object of type `T` is returned by an operator, but a different type was expected. Type checking is mainly static, i.e. applied before the execution of the query during the parsing of the XML document, but some operators, such as `EXT_CALL`, may require a dynamic type checking performed at run-time.
- `InvalidKDDMLQueryException`: exception throws when the input query is not valid. This exception is static, i.e. it is throws before during the checking of the query, as instance when an operator (resp. algorithm) has an illegal attribute (resp. parameter).
- `DBMSConnectionException` exception throws if a connection to an external DBMS fails.
- `EmptyResultsException`: exception throws when an empty output object is returned by an operator, but it is not required by the interpreter.
- `UnsupportedAlgorithmException`: exceptions throws when a required algorithm is not supported by the system.

Interpreter I/O (see figure C.2)

The `InterpreterIO` interface contains the following abstract methods:

- *public void enableVerboseMode(boolean enable);*
  it enables (or disables) the output messages in the standard output during the execution of the query.
- *public void enableLogFile(boolean enable);*
  when the input argument is *true*, the method prints the output messages in a log file stored in the root directory during the execution of the query.

Interpreter checking (see figure C.3)

The `InterpreterChecker` interface contains the following abstract methods:

- *public void validateQuery() throws InvalidKDDMLQueryException, UnsupportedAlgorithmException, DBMSConnectionException;*
  tests if the query is valid before his execution. More precisely, it recursively tests that:
  1. the attributes are valid (i.e. they respect the DTD specification), for each operator belonging to the query;
  2. the parameters are valid (i.e. they respect the DTD specification), for each algorithm belonging to the query;
  3. all required algorithms inside the input query are supported by the system;
  4. the conditions signature are valid, for each condition specification belonging to the query.

If the query contains a `DATABASE_LOADER` or a `DATABASE_WRITER` operator, then the operator checks for user and password fields used during the JDBC connection. If user and password are not specified in the query syntax, then a pop-up frame will be open at run-time.

Interpreter execution (see figure C.4)

The `InterpreterRunner` interface contains the following abstract methods:

- *public KDDMLObject getResult();*
  returns the object obtained after the execution of the query. Can return `null` if the query do not provide a result (e.g. if an exception occurs) or before invoking the method *resolve()*.

- *public String getMessageResult();*
  returns the message describing the result of the execution: returns "*error: < exception description >*" if an exception occurs during the query execution; returns "*success*" if no errors occur; returns `null` if the method is invoked before the method *resolve()*.

- *public void resolve() throws ExecutionException, TypeCheckingException, EmptyResultsException, InvalidKDDMLQueryException, UnsupportedAlgorithmException;*
  starts the interpreter on the KDD query, storing the final object result inside a class variable. The method throws an exception if an error occurs.

The core algorithm of the interpreter is reported in Fig. 4.1. The interpreter recursively traverse the DOM tree representation of the query, yielding a `KDDMLObject` as a result. Also, the expected type of the result is passed together the query.

At each tag, the strict functional interpretation is applied. A `OperatorResolver` object and a `OperatorSettings` object are constructed from the XML tag using a factory of objects from the operators layer (*line 8-11*). At *line 13* the set of attributes for the current operator is stored inside an hash-table structure[1]. In the next two lines, the list of XML elements representing the children of the current tag operator are extracted. Then, each sub-element is evaluated (*line 17-21*), returning a vector of `KDDMLObject`. The operator provides the expected type for the sub-element. Finally, arguments types are checked at *line 24* and the current operator is executed on the vector and hash-table above (*line 25*). Exceptions are raised on critical situations (*lines 29-31*).

The general interpreter of KDDML is however a little bit more complex. Tags with meta-meaning, such as `<IF>` and `<CALL_QUERY>` must be taken into account. Figure 4.2 shows the overall interpreter, which distinguishes three cases:

- The `<IF>` tag (*line 13-23*) has a non-strict semantics. The method *getConditionInputStm()* returns the `<COND>` sub-element that is evaluated first, returning a `KDDMLObject`. The XQuery expression is then evaluated (*line 18*), returning a truth value. Based on that, the `<THEN>` or the `<ELSE>` branch are evaluated and their result returned as the overall result. A run-time checking in performed on output result.

[1] The interpreter assumes that the query is valid before executing it; i.e. the set of attributes and algorithm parameters has been checked by using the appropriate method.

```
protected KDDMLObject resolveCore(Element query, ResultType type)
      throws ExecutionException, TypeCheckingException,
             EmptyResultsException, InvalidKDDMLQueryException,
             UnsupportedAlgorithmException
{
  try {
    String tag_name = query.getTagName();
    OperatorResolver opr =
          OperatorFactory.newResolverInstance(tag_name);
    OperatorSettings ops =
          OperatorFactory.newSettingsInstance(tag_name);

    Hashtable attributes = prepareAttributes(query);
    Vector params = new Vector();
    Vector children = XMLDocument.getChildren(query);

    for (int i=0; i<children.size(); i++) {
        Element elem = (Element) children.get(i);
        KDDMLObject obj = resolve(elem, opr.getArgumentType(i));
        params.add(obj);
    }

    attributes = ops.refreshAttributes(attributes);
    opr.checkArgumentTypes(params);
    return opr.execute(attributes, params);
  catch(OperatorException e1) {
    throw new ExecutionException(...);
  }
  catch(KDDMLCoreException e2) {
    throw new ExecutionException(...);
  }
}
```

Figure 4.1: The core KDDML interpreter

- The `<CALL_QUERY>` tag (*line 27-31*) has a meta-interpretation. We recall that attributes specify actual and formal parameters for the called queries. The method *loadQuery()* loads the called query from the system repository. Them, it performs a parameters substitution (i.e. each formal parameter is replaced with the actual parameter) returning the query that must be evaluated by the interpreter. Also in this case, a run-time checking is performed on output result.

- The third case is the KDDML core interpreter of Fig. 4.1 (*lines 34-53*). When the method *resolveCore(query)* terminates, the integrity of the final result is checked, as shown at *line 35*. If the operator claims for dynamic type checking, the returned result is checked against the expected type (*lines 42-44*). Further control concerns empty output, as shown at *lines 46-48*. Finally, if the result needed to be stored in the data/model repository, the method *saveToRepository()* is invoked (*lines 50-52*).

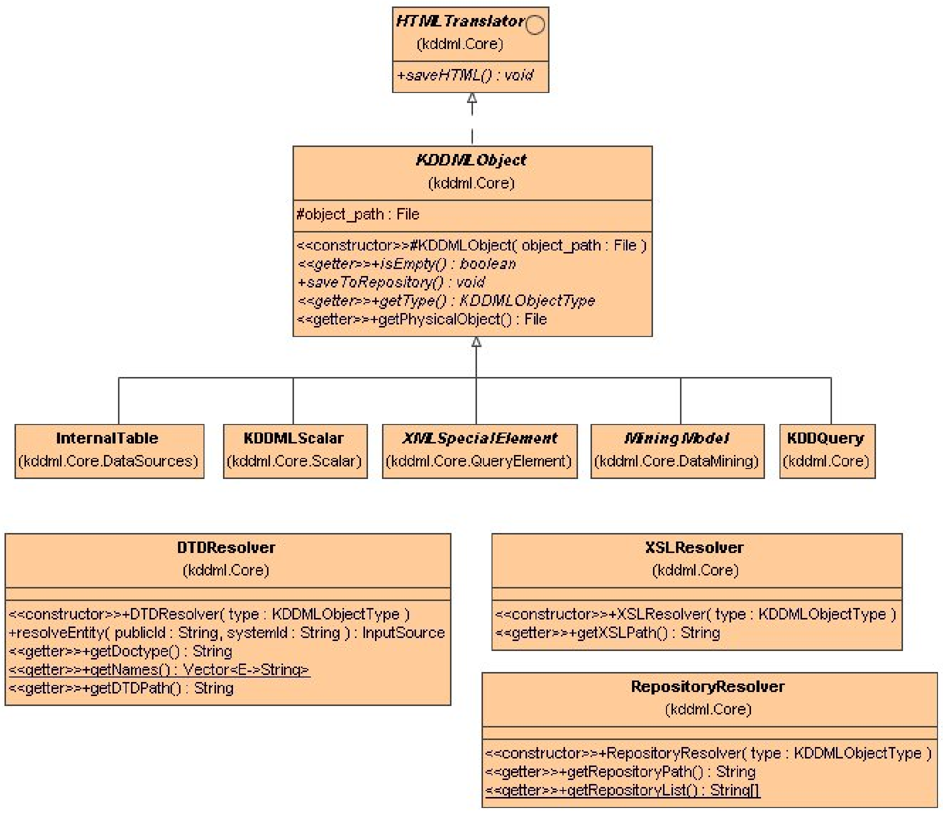
Finally, observe that as shown by the overall query execution code, the KDDML interpreter layer is not affected by any system extension, both concerning *algorithms extensibility*, *operators extensibility* and *model extensibility*.

```
protected KDDMLObject resolve(Element query, KDDMLObjectType type)
      throws ExecutionException, TypeCheckingException,
             EmptyResultsException, InvalidKDDMLQueryException,
             UnsupportedAlgorithmException
{
  KDDMLObject result = null;
  Element children = query.getChildren();
  String tag_name = query.getTagName();

   switch( tag_name ) {
     case "IF":

           IfResolver ifr = new IfResolver(query);
           // evaluates the condition
           KDDMLObject cond_obj = resolve(ifr.getConditionInputStm(),
                                          KDDMLObjectType.ANY);

           if(ifr.evaluateXQueryExpression(cond_obj.getPhysicalObject())
               result = resolve( ifr.getThenStm(), type );
           else
               result = resolve( ifr.getElseStm(), type );
           runTimeChecking(result, type);
           break;

     case "CALL_QUERY":

           CallQueryResolver resolver = new CallQueryResolver(query);
           Element calledQuery = resolver.loadQuery();
           result = resolve( calledQuery, type );
           runTimeChecking( result, type );
           break;

     default:
           result = resolveCore(query);
           if (result == null) {
               throw new EmptyResultsException(tag_name);
           }

           String xml_dest = query.getAttribute("xml_dest");
           OperatorResolver opr =
                     OperatorFactory.newResolverInstance(tag_name);
           if ( (opr.runTimeCheckNeeded())) {
               runTimeChecking(result, type);
           }

           if ( (result.isEmpty()) && (opr.abortIfIsEmpty())) {
               throw new EmptyResultsException(tag_name);
           }

           if ( (xml_dest != null) && (opr.saveXMLOutput())) {
               result.saveToRepository();
           }
           break;
   }
   return result;
 }
```

Figure 4.2: The KDDML interpreter

APPENDIX A

UML core layer diagrams



HTMLTranslator
(kddml.Core)
+saveHTML() : void
KDDMLObject
(kddml.Core)
#object_path : File
<<constructor>>#KDDMLObject(object_path : File)
<<getter>>+isEmpty() : boolean
+saveToRepository() : void
<<getter>>+getType() : KDDMLObjectType
<<getter>>+getPhysicalObject() : File
InternalTable
(kddml.Core.DataSources)
KDDMLScalar
(kddml.Core.Scalar)
XMLSpecialElement
(kddml.Core.QueryElement)
MiningModel
(kddml.Core.DataMining)
KDDQuery
(kddml.Core)
DTDResolver
(kddml.Core)
<<constructor>>+DTDResolver(type : KDDMLObjectType)
+resolveEntity(publicId : String, systemId : String) : InputSource
<<getter>>+getDoctype() : String
<<getter>>+getNames() : Vector<E->String>
<<getter>>+getDTDPath() : String
XSLResolver
(kddml.Core)
<<constructor>>+XSLResolver(type : KDDMLObjectType)
<<getter>>+getXSLPath() : String
RepositoryResolver
(kddml.Core)
<<constructor>>+RepositoryResolver(type : KDDMLObjectType)
<<getter>>+getRepositoryPath() : String
<<getter>>+getRepositoryList() : String[]

Figure A.1: Package kddml.Core - Manager

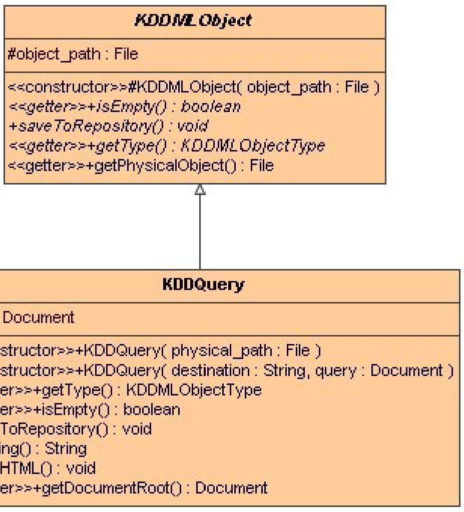

KDDMLObject
#object_path : File
<<constructor>>#KDDMLObject(object_path : File)
<<getter>>+isEmpty() : boolean
+saveToRepository() : void
<<getter>>+getType() : KDDMLObjectType
<<getter>>+getPhysicalObject() : File
KDDQuery
-root : Document
<<constructor>>+KDDQuery(physical_path : File)
<<constructor>>+KDDQuery(destination : String, query : Document)
<<getter>>+getType() : KDDMLObjectType
<<getter>>+isEmpty() : boolean
+saveToRepository() : void
+toString() : String
+saveHTML() : void
<<getter>>+getDocumentRoot() : Document

Figure A.2: Package kddml.Core - KDD query

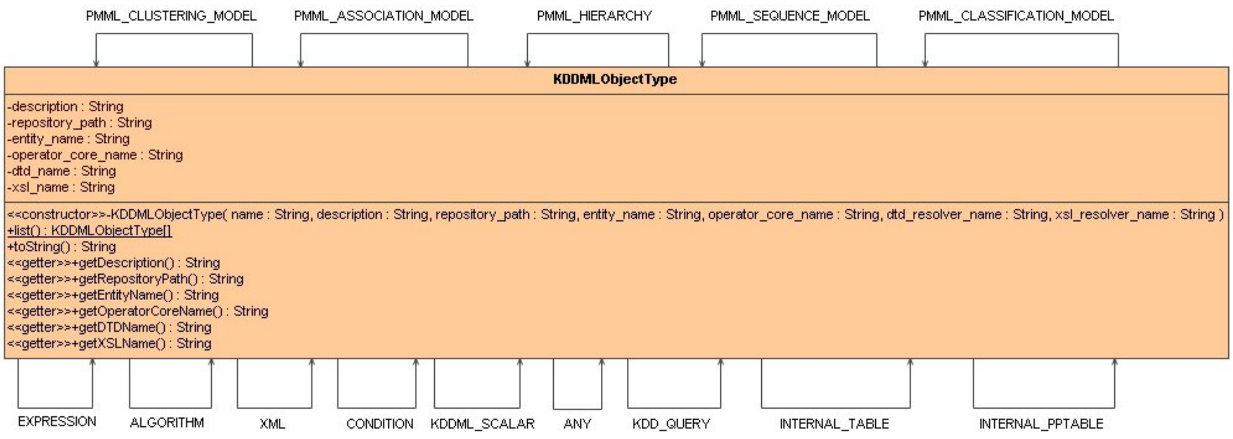

PMML_CLUSTERING_MODEL
PMML_ASSOCIATION_MODEL
PMML_HIERARCHY
PMML_SEQUENCE_MODEL
PMML_CLASSIFICATION_MODEL
KDDMLObjectType
-description : String
-repository_path : String
-entity_name : String
-operator_core_name : String
-dtd_name : String
-xsl_name : String
<<constructor>>-KDDMLObjectType(name : String, description : String, repository_path : String, entity_name : String, operator_core_name : String, dtd_resolver_name : String, xsl_resolver_name : String)
+list() : KDDMLObjectType[]
+toString() : String
<<getter>>+getDescription() : String
<<getter>>+getRepositoryPath() : String
<<getter>>+getEntityName() : String
<<getter>>+getOperatorCoreName() : String
<<getter>>+getDTDName() : String
<<getter>>+getXSLName() : String
EXPRESSION
ALGORITHM
XML
CONDITION
KDDML_SCALAR
ANY
KDD_QUERY
INTERNAL_TABLE
INTERNAL_PPTABLE

Figure A.3: Package kddml.Core - Types

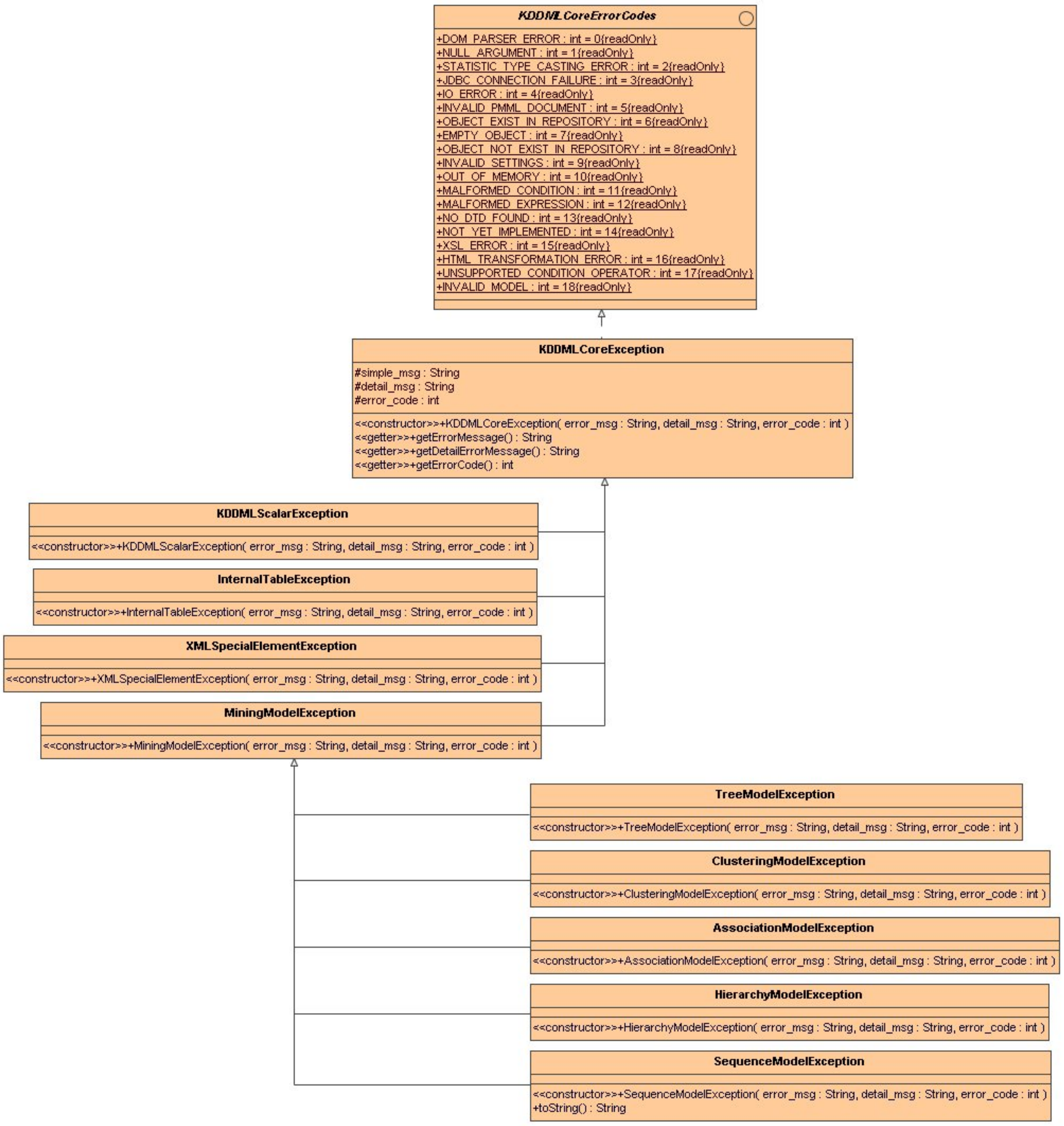

KDDMLCoreErrorCodes
+DOM_PARSER_ERROR : int = 0{readOnly}
+NULL_ARGUMENT : int = 1{readOnly}
+STATISTIC_TYPE_CASTING_ERROR : int = 2{readOnly}
+JDBC_CONNECTION_FAILURE : int = 3{readOnly}
+IO_ERROR : int = 4{readOnly}
+INVALID_PMML_DOCUMENT : int = 5{readOnly}
+OBJECT_EXIST_IN_REPOSITORY : int = 6{readOnly}
+EMPTY_OBJECT : int = 7{readOnly}
+OBJECT_NOT_EXIST_IN_REPOSITORY : int = 8{readOnly}
+INVALID_SETTINGS : int = 9{readOnly}
+OUT_OF_MEMORY : int = 10{readOnly}
+MALFORMED_CONDITION : int = 11{readOnly}
+MALFORMED_EXPRESSION : int = 12{readOnly}
+NO_DTD_FOUND : int = 13{readOnly}
+NOT_YET_IMPLEMENTED : int = 14{readOnly}
+XSL_ERROR : int = 15{readOnly}
+HTML_TRANSFORMATION_ERROR : int = 16{readOnly}
+UNSUPPORTED_CONDITION_OPERATOR : int = 17{readOnly}
+INVALID_MODEL : int = 18{readOnly}
KDDMLCoreException
#simple_msg : String
#detail_msg : String
#error_code : int
<<constructor>>+KDDMLCoreException(error_msg : String, detail_msg : String, error_code : int)
<<getter>>+getErrorMessage() : String
<<getter>>+getDetailErrorMessage() : String
<<getter>>+getErrorCode() : int
KDDMLScalarException
<<constructor>>+KDDMLScalarException(error_msg : String, detail_msg : String, error_code : int)
InternalTableException
<<constructor>>+InternalTableException(error_msg : String, detail_msg : String, error_code : int)
XMLSpecialElementException
<<constructor>>+XMLSpecialElementException(error_msg : String, detail_msg : String, error_code : int)
MiningModelException
<<constructor>>+MiningModelException(error_msg : String, detail_msg : String, error_code : int)
TreeModelException
<<constructor>>+TreeModelException(error_msg : String, detail_msg : String, error_code : int)
ClusteringModelException
<<constructor>>+ClusteringModelException(error_msg : String, detail_msg : String, error_code : int)
AssociationModelException
<<constructor>>+AssociationModelException(error_msg : String, detail_msg : String, error_code : int)
HierarchyModelException
<<constructor>>+HierarchyModelException(error_msg : String, detail_msg : String, error_code : int)
SequenceModelException
<<constructor>>+SequenceModelException(error_msg : String, detail_msg : String, error_code : int)
+toString() : String

Figure A.4: Package kddml.Core - Exceptions

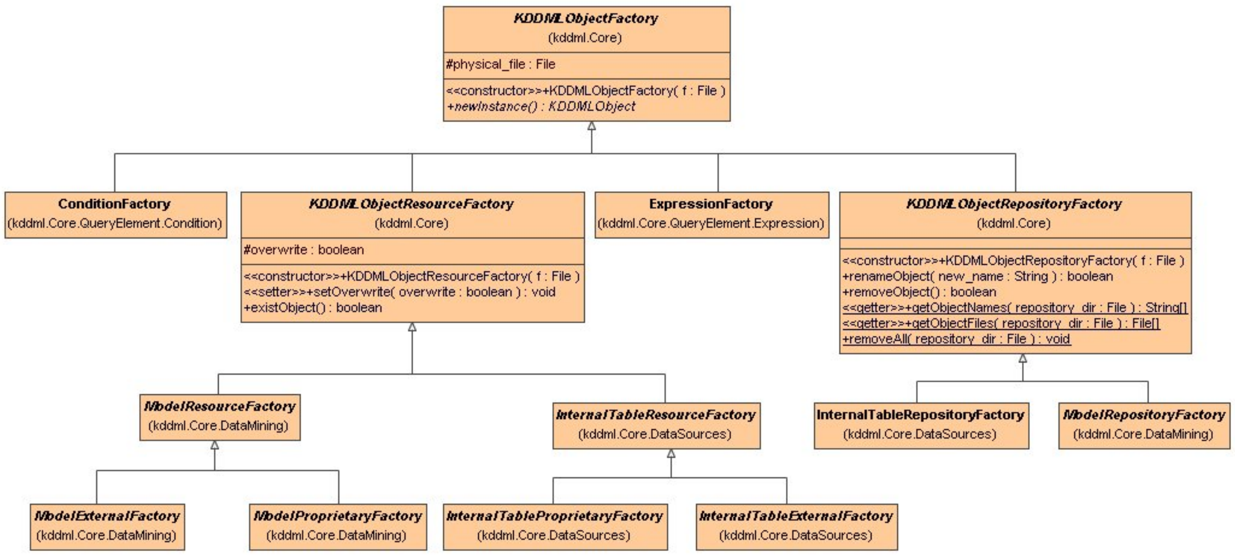

KDDMLObjectFactory
(kddml.Core)
#physical_file : File
<<constructor>>+KDDMLObjectFactory(f : File)
+newInstance() : KDDMLObject
ConditionFactory
(kddml.Core.QueryElement.Condition)
KDDMLObjectResourceFactory
(kddml.Core)
#overwrite : boolean
<<constructor>>+KDDMLObjectResourceFactory(f : File)
<<setter>>+setOverwrite(overwrite : boolean) : void
+existObject() : boolean
ExpressionFactory
(kddml.Core.QueryElement.Expression)
KDDMLObjectRepositoryFactory
(kddml.Core)
<<constructor>>+KDDMLObjectRepositoryFactory(f : File)
+renameObject(new_name : String) : boolean
+removeObject() : boolean
<<getter>>+getObjectNames(repository_dir : File) : String[]
<<getter>>+getObjectFiles(repository_dir : File) : File[]
+removeAll(repository_dir : File) : void
ModelResourceFactory
(kddml.Core.DataMining)
InternalTableResourceFactory
(kddml.Core.DataSources)
InternalTableRepositoryFactory
(kddml.Core.DataSources)
ModelRepositoryFactory
(kddml.Core.DataMining)
ModelExternalFactory
(kddml.Core.DataMining)
ModelProprietaryFactory
(kddml.Core.DataMining)
InternalTableProprietaryFactory
(kddml.Core.DataSources)
InternalTableExternalFactory
(kddml.Core.DataSources)

Figure A.5: Package kddml.Core - Factory

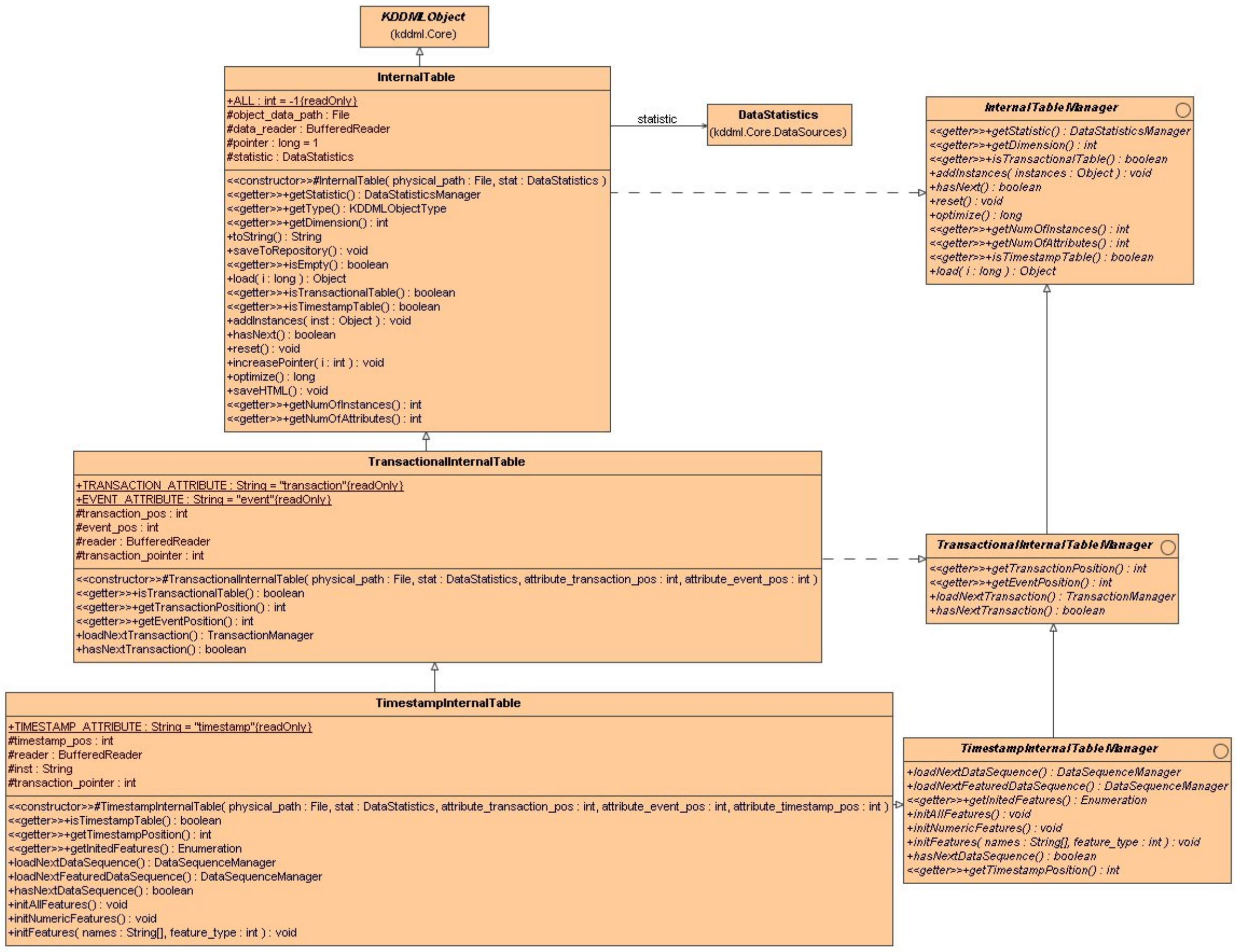

KDDMLObject
(kddml.Core)
InternalTable
+ALL : int = -1{readOnly}
#object_data_path : File
#data_reader : BufferedReader
#pointer : long = 1
#statistic : DataStatistics
<<constructor>>#InternalTable(physical_path : File, stat : DataStatistics)
<<getter>>+getStatistic() : DataStatisticsManager
<<getter>>+getType() : KDDMLObjectType
<<getter>>+getDimension() : int
+toString() : String
+saveToRepository() : void
<<getter>>+isEmpty() : boolean
+load(i : long) : Object
<<getter>>+isTransactionalTable() : boolean
<<getter>>+isTimestampTable() : boolean
+addInstances(inst : Object) : void
+hasNext() : boolean
+reset() : void
+increasePointer(i : int) : void
+optimize() : long
+saveHTML() : void
<<getter>>+getNumOfInstances() : int
<<getter>>+getNumOfAttributes() : int
statistic
DataStatistics
(kddml.Core.DataSources)
InternalTableManager
<<getter>>+getStatistic() : DataStatisticsManager
<<getter>>+getDimension() : int
<<getter>>+isTransactionalTable() : boolean
+addInstances(instances : Object) : void
+hasNext() : boolean
+reset() : void
+optimize() : long
<<getter>>+getNumOfInstances() : int
<<getter>>+getNumOfAttributes() : int
<<getter>>+isTimestampTable() : boolean
+load(i : long) : Object
TransactionalInternalTable
+TRANSACTION_ATTRIBUTE : String = "transaction"{readOnly}
+EVENT_ATTRIBUTE : String = "event"{readOnly}
#transaction_pos : int
#event_pos : int
#reader : BufferedReader
#transaction_pointer : int
<<constructor>>#TransactionalInternalTable(physical_path : File, stat : DataStatistics, attribute_transaction_pos : int, attribute_event_pos : int)
<<getter>>+isTransactionalTable() : boolean
<<getter>>+getTransactionPosition() : int
<<getter>>+getEventPosition() : int
+loadNextTransaction() : TransactionManager
+hasNextTransaction() : boolean
TransactionalInternalTableManager
<<getter>>+getTransactionPosition() : int
<<getter>>+getEventPosition() : int
+loadNextTransaction() : TransactionManager
+hasNextTransaction() : boolean
TimestampInternalTable
+TIMESTAMP_ATTRIBUTE : String = "timestamp"{readOnly}
#timestamp_pos : int
#reader : BufferedReader
#inst : String
#transaction_pointer : int
<<constructor>>#TimestampInternalTable(physical_path : File, stat : DataStatistics, attribute_transaction_pos : int, attribute_event_pos : int, attribute_timestamp_pos : int)
<<getter>>+isTimestampTable() : boolean
<<getter>>+getTimestampPosition() : int
<<getter>>+getInitedFeatures() : Enumeration
+loadNextDataSequence() : DataSequenceManager
+loadNextFeaturedDataSequence() : DataSequenceManager
+hasNextDataSequence() : boolean
+initAllFeatures() : void
+initNumericFeatures() : void
+initFeatures(names : String[], feature_type : int) : void
TimestampInternalTableManager
+loadNextDataSequence() : DataSequenceManager
+loadNextFeaturedDataSequence() : DataSequenceManager
<<getter>>+getInitedFeatures() : Enumeration
+initAllFeatures() : void
+initNumericFeatures() : void
+initFeatures(names : String[], feature_type : int) : void
+hasNextDataSequence() : boolean
<<getter>>+getTimestampPosition() : int

Figure A.6: Package kddml.Core.DataSources - Manager

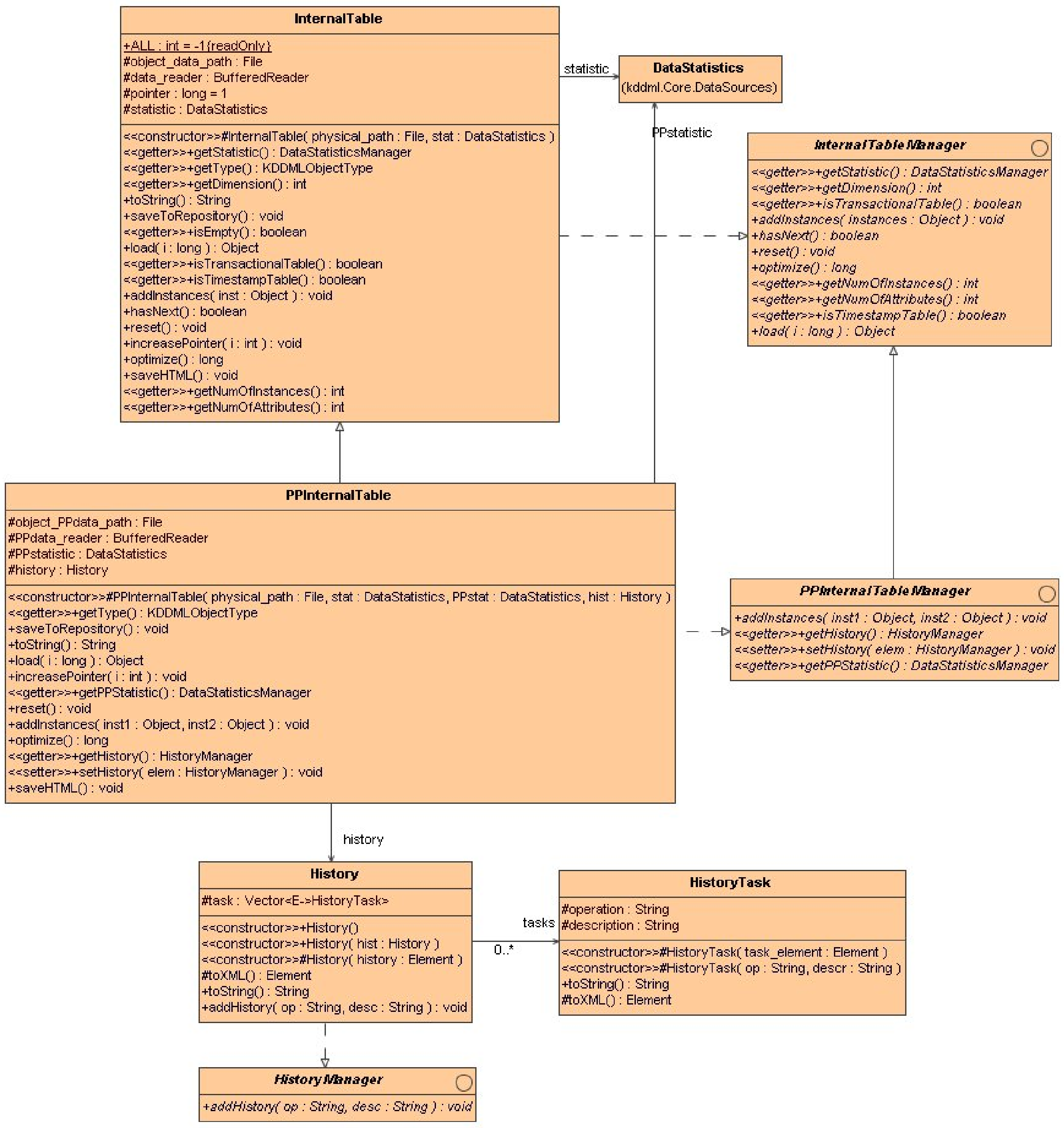

InternalTable
+ALL : int = -1{readOnly}
#object_data_path : File
#data_reader : BufferedReader
#pointer : long = 1
#statistic : DataStatistics
<<constructor>>#InternalTable(physical_path : File, stat : DataStatistics)
<<getter>>+getStatistic() : DataStatisticsManager
<<getter>>+getType() : KDDMLObjectType
<<getter>>+getDimension() : int
+toString() : String
+saveToRepository() : void
<<getter>>+isEmpty() : boolean
+load(i : long) : Object
<<getter>>+isTransactionalTable() : boolean
<<getter>>+isTimestampTable() : boolean
+addInstances(inst : Object) : void
+hasNext() : boolean
+reset() : void
+increasePointer(i : int) : void
+optimize() : long
+saveHTML() : void
<<getter>>+getNumOfInstances() : int
<<getter>>+getNumOfAttributes() : int
statistic
DataStatistics
(kddml.Core.DataSources)
PPstatistic
InternalTableManager
<<getter>>+getStatistic() : DataStatisticsManager
<<getter>>+getDimension() : int
<<getter>>+isTransactionalTable() : boolean
+addInstances(instances : Object) : void
+hasNext() : boolean
+reset() : void
+optimize() : long
<<getter>>+getNumOfInstances() : int
<<getter>>+getNumOfAttributes() : int
<<getter>>+isTimestampTable() : boolean
+load(i : long) : Object
PPInternalTable
#object_PPdata_path : File
#PPdata_reader : BufferedReader
#PPstatistic : DataStatistics
#history : History
<<constructor>>#PPInternalTable(physical_path : File, stat : DataStatistics, PPstat : DataStatistics, hist : History)
<<getter>>+getType() : KDDMLObjectType
+saveToRepository() : void
+toString() : String
+load(i : long) : Object
+increasePointer(i : int) : void
<<getter>>+getPPStatistic() : DataStatisticsManager
+reset() : void
+addInstances(inst1 : Object, inst2 : Object) : void
+optimize() : long
<<getter>>+getHistory() : HistoryManager
<<setter>>+setHistory(elem : HistoryManager) : void
+saveHTML() : void
PPInternalTableManager
+addInstances(inst1 : Object, inst2 : Object) : void
<<getter>>+getHistory() : HistoryManager
<<setter>>+setHistory(elem : HistoryManager) : void
<<getter>>+getPPStatistic() : DataStatisticsManager
history
History
#task : Vector<E->HistoryTask>
<<constructor>>+History()
<<constructor>>+History(hist : History)
<<constructor>>#History(history : Element)
#toXML() : Element
+toString() : String
+addHistory(op : String, desc : String) : void
tasks
0..*
HistoryTask
#operation : String
#description : String
<<constructor>>#HistoryTask(task_element : Element)
<<constructor>>#HistoryTask(op : String, descr : String)
+toString() : String
#toXML() : Element
HistoryManager
+addHistory(op : String, desc : String) : void

Figure A.7: Package kddml.Core.DataSources - Preprocessing

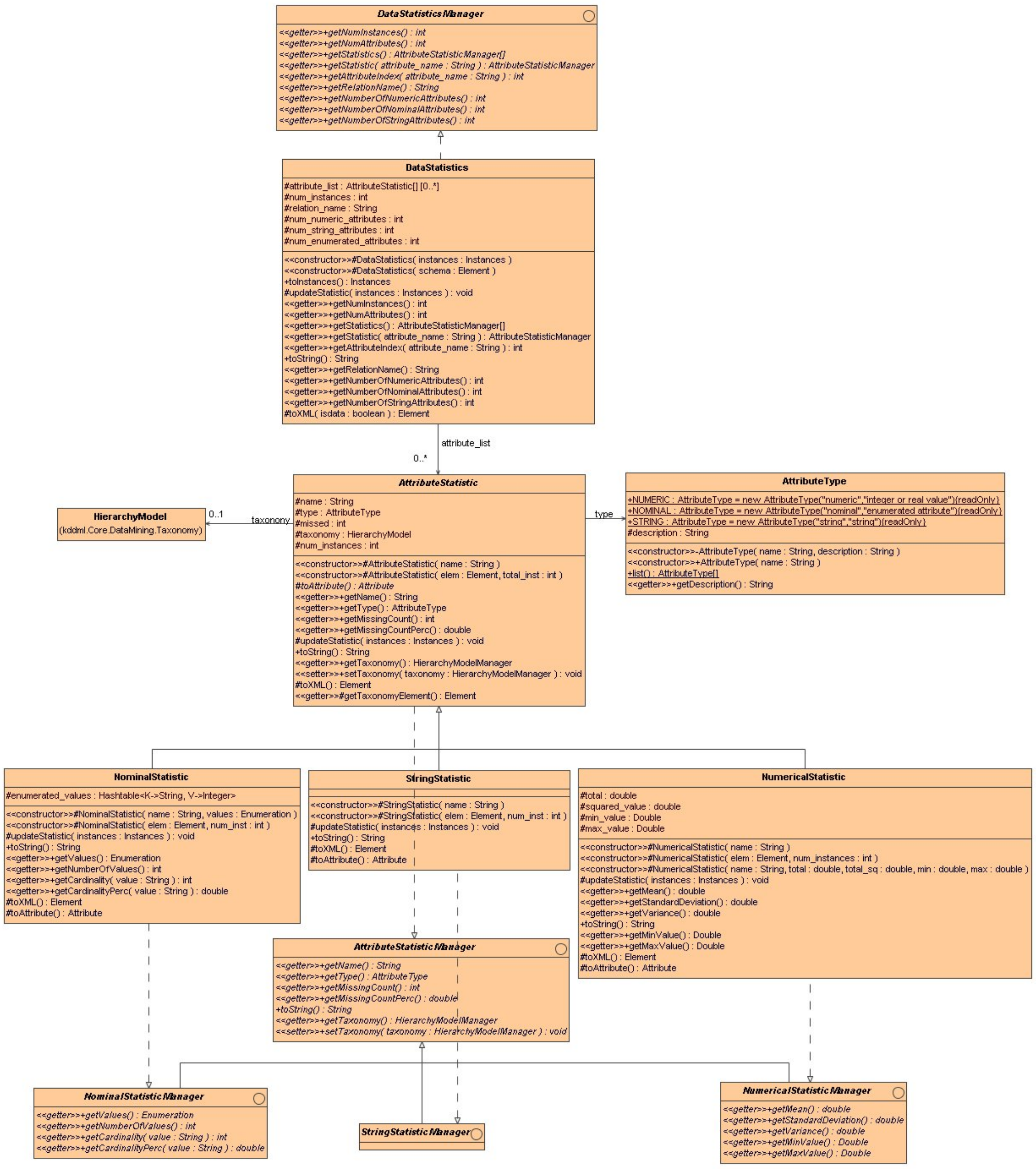

DataStatisticsManager
<<getter>>+getNumInstances() : int
<<getter>>+getNumAttributes() : int
<<getter>>+getStatistics() : AttributeStatisticManager[]
<<getter>>+getStatistic(attribute_name : String) : AttributeStatisticManager
<<getter>>+getAttributeIndex(attribute_name : String) : int
<<getter>>+getRelationName() : String
<<getter>>+getNumberOfNumericAttributes() : int
<<getter>>+getNumberOfNominalAttributes() : int
<<getter>>+getNumberOfStringAttributes() : int
DataStatistics
#attribute_list : AttributeStatistic[] [0..*]
#num_instances : int
#relation_name : String
#num_numeric_attributes : int
#num_string_attributes : int
#num_enumerated_attributes : int
<<constructor>>#DataStatistics(instances : Instances)
<<constructor>>#DataStatistics(schema : Element)
+toInstances() : Instances
#updateStatistic(instances : Instances) : void
<<getter>>+getNumInstances() : int
<<getter>>+getNumAttributes() : int
<<getter>>+getStatistics() : AttributeStatisticManager[]
<<getter>>+getStatistic(attribute_name : String) : AttributeStatisticManager
<<getter>>+getAttributeIndex(attribute_name : String) : int
+toString() : String
<<getter>>+getRelationName() : String
<<getter>>+getNumberOfNumericAttributes() : int
<<getter>>+getNumberOfNominalAttributes() : int
<<getter>>+getNumberOfStringAttributes() : int
#toXML(isdata : boolean) : Element
attribute_list
0..*
AttributeStatistic
#name : String
#type : AttributeType
#missed : int
#taxonomy : HierarchyModel
#num_instances : int
<<constructor>>#AttributeStatistic(name : String)
<<constructor>>#AttributeStatistic(elem : Element, total_inst : int)
#toAttribute() : Attribute
<<getter>>+getName() : String
<<getter>>+getType() : AttributeType
<<getter>>+getMissingCount() : int
<<getter>>+getMissingCountPerc() : double
#updateStatistic(instances : Instances) : void
+toString() : String
<<getter>>+getTaxonomy() : HierarchyModelManager
<<setter>>+setTaxonomy(taxonomy : HierarchyModelManager) : void
#toXML() : Element
<<getter>>#getTaxonomyElement() : Element
HierarchyModel
(kddml.Core.DataMining.Taxonomy)
0..1
taxonomy
type
AttributeType
+NUMERIC : AttributeType = new AttributeType("numeric","integer or real value"){readOnly}
+NOMINAL : AttributeType = new AttributeType("nominal","enumerated attribute"){readOnly}
+STRING : AttributeType = new AttributeType("string","string"){readOnly}
#description : String
<<constructor>>-AttributeType(name : String, description : String)
<<constructor>>+AttributeType(name : String)
+list() : AttributeType[]
<<getter>>+getDescription() : String
NominalStatistic
#enumerated_values : Hashtable<K->String, V->Integer>
<<constructor>>#NominalStatistic(name : String, values : Enumeration)
<<constructor>>#NominalStatistic(elem : Element, num_inst : int)
#updateStatistic(instances : Instances) : void
+toString() : String
<<getter>>+getValues() : Enumeration
<<getter>>+getNumberOfValues() : int
<<getter>>+getCardinality(value : String) : int
<<getter>>+getCardinalityPerc(value : String) : double
#toXML() : Element
#toAttribute() : Attribute
StringStatistic
<<constructor>>#StringStatistic(name : String)
<<constructor>>#StringStatistic(elem : Element, num_inst : int)
#updateStatistic(instances : Instances) : void
+toString() : String
#toXML() : Element
#toAttribute() : Attribute
NumericalStatistic
#total : double
#squared_value : double
#min_value : Double
#max_value : Double
<<constructor>>#NumericalStatistic(name : String)
<<constructor>>#NumericalStatistic(elem : Element, num_instances : int)
<<constructor>>#NumericalStatistic(name : String, total : double, total_sq : double, min : double, max : double)
#updateStatistic(instances : Instances) : void
<<getter>>+getMean() : double
<<getter>>+getStandardDeviation() : double
<<getter>>+getVariance() : double
+toString() : String
<<getter>>+getMinValue() : Double
<<getter>>+getMaxValue() : Double
#toXML() : Element
#toAttribute() : Attribute
AttributeStatisticManager
<<getter>>+getName() : String
<<getter>>+getType() : AttributeType
<<getter>>+getMissingCount() : int
<<getter>>+getMissingCountPerc() : double
+toString() : String
<<getter>>+getTaxonomy() : HierarchyModelManager
<<setter>>+setTaxonomy(taxonomy : HierarchyModelManager) : void
NominalStatisticManager
<<getter>>+getValues() : Enumeration
<<getter>>+getNumberOfValues() : int
<<getter>>+getCardinality(value : String) : int
<<getter>>+getCardinalityPerc(value : String) : double
StringStatisticManager
NumericalStatisticManager
<<getter>>+getMean() : double
<<getter>>+getStandardDeviation() : double
<<getter>>+getVariance() : double
<<getter>>+getMinValue() : Double
<<getter>>+getMaxValue() : Double

Figure A.8: Package kddml.Core.DataSources - Statistics

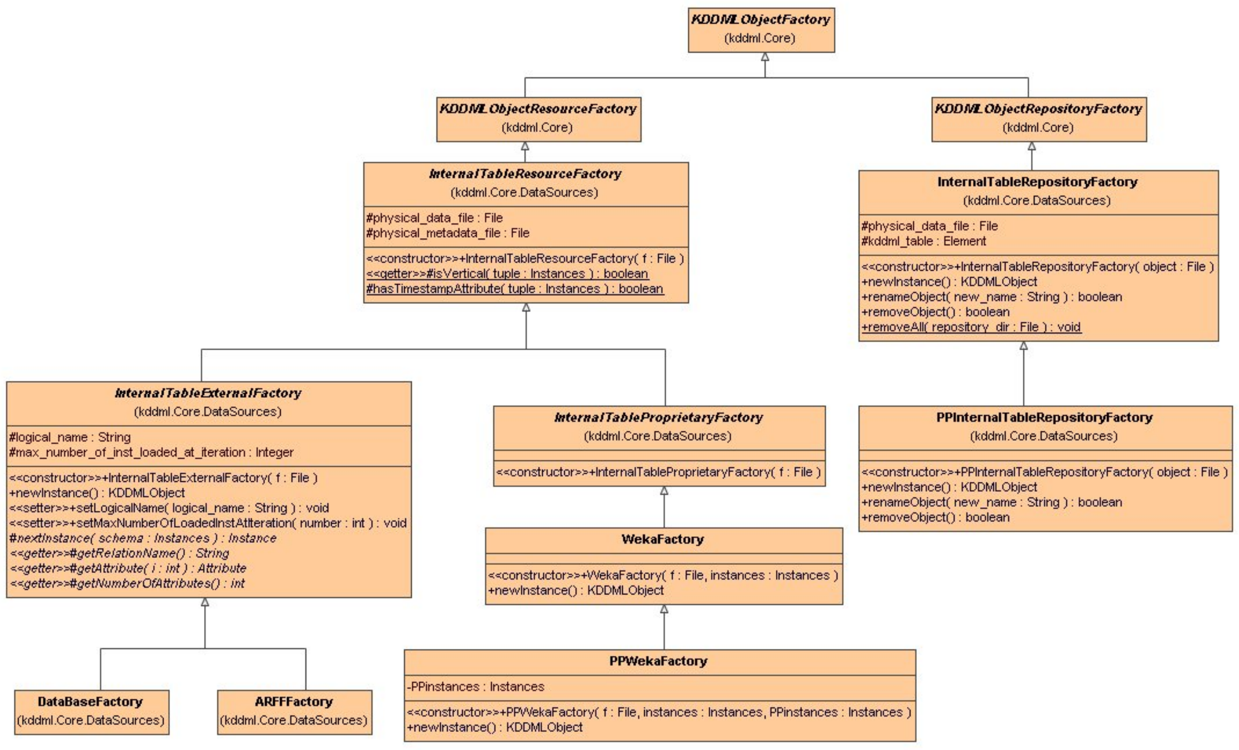

KDDMLObjectFactory
(kddml.Core)
KDDMLObjectResourceFactory
(kddml.Core)
KDDMLObjectRepositoryFactory
(kddml.Core)
InternalTableResourceFactory
(kddml.Core.DataSources)
#physical_data_file : File
#physical_metadata_file : File
<<constructor>>+InternalTableResourceFactory(f : File)
<<getter>>#isVertical(tuple : Instances) : boolean
#hasTimestampAttribute(tuple : Instances) : boolean
InternalTableRepositoryFactory
(kddml.Core.DataSources)
#physical_data_file : File
#kddml_table : Element
<<constructor>>+InternalTableRepositoryFactory(object : File)
+newInstance() : KDDMLObject
+renameObject(new_name : String) : boolean
+removeObject() : boolean
+removeAll(repository_dir : File) : void
InternalTableExternalFactory
(kddml.Core.DataSources)
#logical_name : String
#max_number_of_inst_loaded_at_iteration : Integer
<<constructor>>+InternalTableExternalFactory(f : File)
+newInstance() : KDDMLObject
<<setter>>+setLogicalName(logical_name : String) : void
<<setter>>+setMaxNumberOfLoadedInstAtIteration(number : int) : void
#nextInstance(schema : Instances) : Instance
<<getter>>#getRelationName() : String
<<getter>>#getAttribute(i : int) : Attribute
<<getter>>#getNumberOfAttributes() : int
InternalTableProprietaryFactory
(kddml.Core.DataSources)
<<constructor>>+InternalTableProprietaryFactory(f : File)
PPInternalTableRepositoryFactory
(kddml.Core.DataSources)
<<constructor>>+PPInternalTableRepositoryFactory(object : File)
+newInstance() : KDDMLObject
+renameObject(new_name : String) : boolean
+removeObject() : boolean
WekaFactory
<<constructor>>+WekaFactory(f : File, instances : Instances)
+newInstance() : KDDMLObject
PPWekaFactory
-PPinstances : Instances
<<constructor>>+PPWekaFactory(f : File, instances : Instances, PPinstances : Instances)
+newInstance() : KDDMLObject
DataBaseFactory
(kddml.Core.DataSources)
ARFFFactory
(kddml.Core.DataSources)

Figure A.9: Package kddml.Core.DataSources - Factory

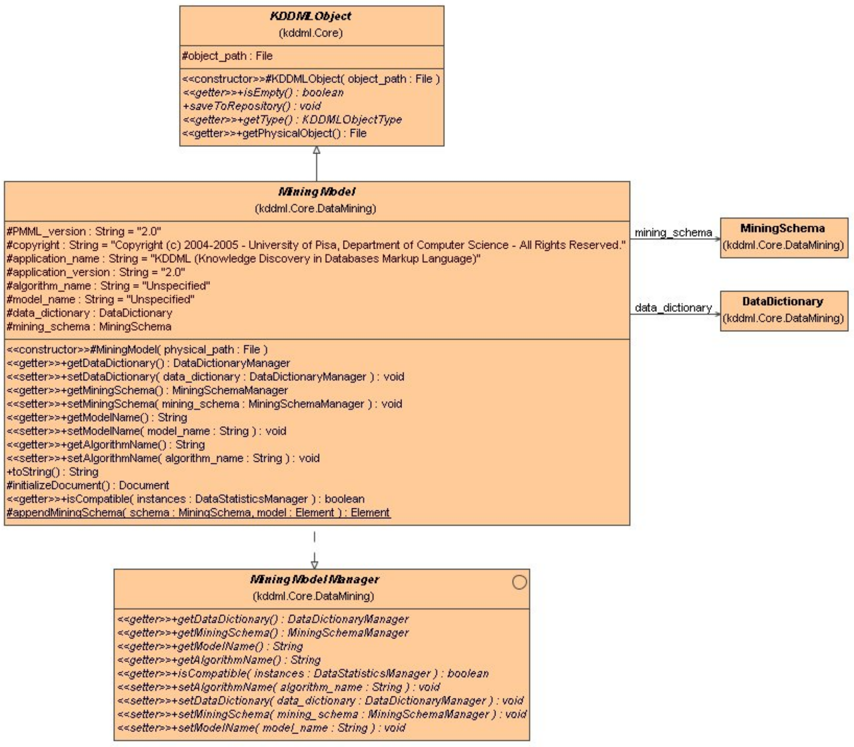

KDDMLObject
(kddml.Core)
#object_path : File
<<constructor>>#KDDMLObject(object_path : File)
<<getter>>+isEmpty() : boolean
+saveToRepository() : void
<<getter>>+getType() : KDDMLObjectType
<<getter>>+getPhysicalObject() : File
MiningModel
(kddml.Core.DataMining)
#PMML_version : String = "2.0"
#copyright : String = "Copyright (c) 2004-2005 - University of Pisa, Department of Computer Science - All Rights Reserved."
#application_name : String = "KDDML (Knowledge Discovery in Databases Markup Language)"
#application_version : String = "2.0"
#algorithm_name : String = "Unspecified"
#model_name : String = "Unspecified"
#data_dictionary : DataDictionary
#mining_schema : MiningSchema
<<constructor>>#MiningModel(physical_path : File)
<<getter>>+getDataDictionary() : DataDictionaryManager
<<setter>>+setDataDictionary(data_dictionary : DataDictionaryManager) : void
<<getter>>+getMiningSchema() : MiningSchemaManager
<<setter>>+setMiningSchema(mining_schema : MiningSchemaManager) : void
<<getter>>+getModelName() : String
<<setter>>+setModelName(model_name : String) : void
<<getter>>+getAlgorithmName() : String
<<setter>>+setAlgorithmName(algorithm_name : String) : void
+toString() : String
#initializeDocument() : Document
<<getter>>+isCompatible(instances : DataStatisticsManager) : boolean
#appendMiningSchema(schema : MiningSchema, model : Element) : Element
mining_schema
MiningSchema
(kddml.Core.DataMining)
data_dictionary
DataDictionary
(kddml.Core.DataMining)
MiningModelManager
(kddml.Core.DataMining)
<<getter>>+getDataDictionary() : DataDictionaryManager
<<getter>>+getMiningSchema() : MiningSchemaManager
<<getter>>+getModelName() : String
<<getter>>+getAlgorithmName() : String
<<getter>>+isCompatible(instances : DataStatisticsManager) : boolean
<<setter>>+setAlgorithmName(algorithm_name : String) : void
<<setter>>+setDataDictionary(data_dictionary : DataDictionaryManager) : void
<<setter>>+setMiningSchema(mining_schema : MiningSchemaManager) : void
<<setter>>+setModelName(model_name : String) : void

Figure A.10: Package kddml.Core.DataMining - Model manager

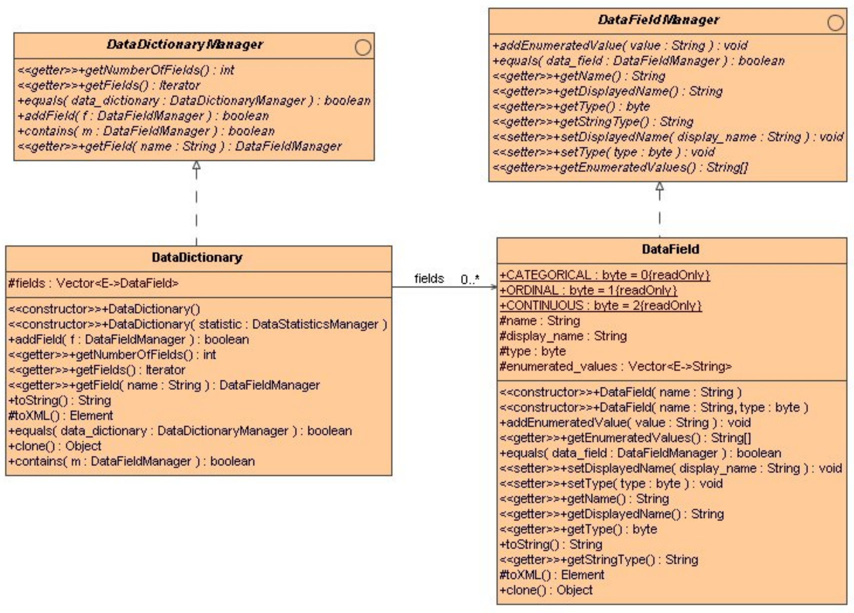

DataDictionaryManager
<<getter>>+getNumberOfFields() : int
<<getter>>+getFields() : Iterator
+equals(data_dictionary : DataDictionaryManager) : boolean
+addField(f : DataFieldManager) : boolean
+contains(m : DataFieldManager) : boolean
<<getter>>+getField(name : String) : DataFieldManager
DataFieldManager
+addEnumeratedValue(value : String) : void
+equals(data_field : DataFieldManager) : boolean
<<getter>>+getName() : String
<<getter>>+getDisplayedName() : String
<<getter>>+getType() : byte
<<getter>>+getStringType() : String
<<setter>>+setDisplayedName(display_name : String) : void
<<setter>>+setType(type : byte) : void
<<getter>>+getEnumeratedValues() : String[]
DataDictionary
#fields : Vector<E->DataField>
<<constructor>>+DataDictionary()
<<constructor>>+DataDictionary(statistic : DataStatisticsManager)
+addField(f : DataFieldManager) : boolean
<<getter>>+getNumberOfFields() : int
<<getter>>+getFields() : Iterator
<<getter>>+getField(name : String) : DataFieldManager
+toString() : String
#toXML() : Element
+equals(data_dictionary : DataDictionaryManager) : boolean
+clone() : Object
+contains(m : DataFieldManager) : boolean
fields
0..*
DataField
+CATEGORICAL : byte = 0{readOnly}
+ORDINAL : byte = 1{readOnly}
+CONTINUOUS : byte = 2{readOnly}
#name : String
#display_name : String
#type : byte
#enumerated_values : Vector<E->String>
<<constructor>>+DataField(name : String)
<<constructor>>+DataField(name : String, type : byte)
+addEnumeratedValue(value : String) : void
<<getter>>+getEnumeratedValues() : String[]
+equals(data_field : DataFieldManager) : boolean
<<setter>>+setDisplayedName(display_name : String) : void
<<setter>>+setType(type : byte) : void
<<getter>>+getName() : String
<<getter>>+getDisplayedName() : String
<<getter>>+getType() : byte
+toString() : String
<<getter>>+getStringType() : String
#toXML() : Element
+clone() : Object

Figure A.11: Package kddml.Core.DataMining - Data dictionary

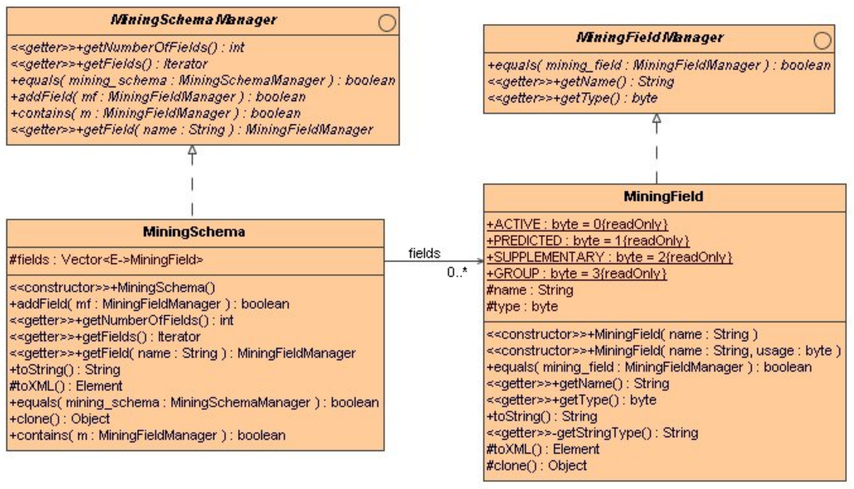

MiningSchemaManager
<<getter>>+getNumberOfFields() : int
<<getter>>+getFields() : Iterator
+equals(mining_schema : MiningSchemaManager) : boolean
+addField(mf : MiningFieldManager) : boolean
+contains(m : MiningFieldManager) : boolean
<<getter>>+getField(name : String) : MiningFieldManager
MiningFieldManager
+equals(mining_field : MiningFieldManager) : boolean
<<getter>>+getName() : String
<<getter>>+getType() : byte
MiningSchema
#fields : Vector<E->MiningField>
<<constructor>>+MiningSchema()
+addField(mf : MiningFieldManager) : boolean
<<getter>>+getNumberOfFields() : int
<<getter>>+getFields() : Iterator
<<getter>>+getField(name : String) : MiningFieldManager
+toString() : String
#toXML() : Element
+equals(mining_schema : MiningSchemaManager) : boolean
+clone() : Object
+contains(m : MiningFieldManager) : boolean
fields
0..*
MiningField
+ACTIVE : byte = 0{readOnly}
+PREDICTED : byte = 1{readOnly}
+SUPPLEMENTARY : byte = 2{readOnly}
+GROUP : byte = 3{readOnly}
#name : String
#type : byte
<<constructor>>+MiningField(name : String)
<<constructor>>+MiningField(name : String, usage : byte)
+equals(mining_field : MiningFieldManager) : boolean
<<getter>>+getName() : String
<<getter>>+getType() : byte
+toString() : String
<<getter>>-getStringType() : String
#toXML() : Element
#clone() : Object

Figure A.12: Package kddml.Core.DataMining - Mining schema

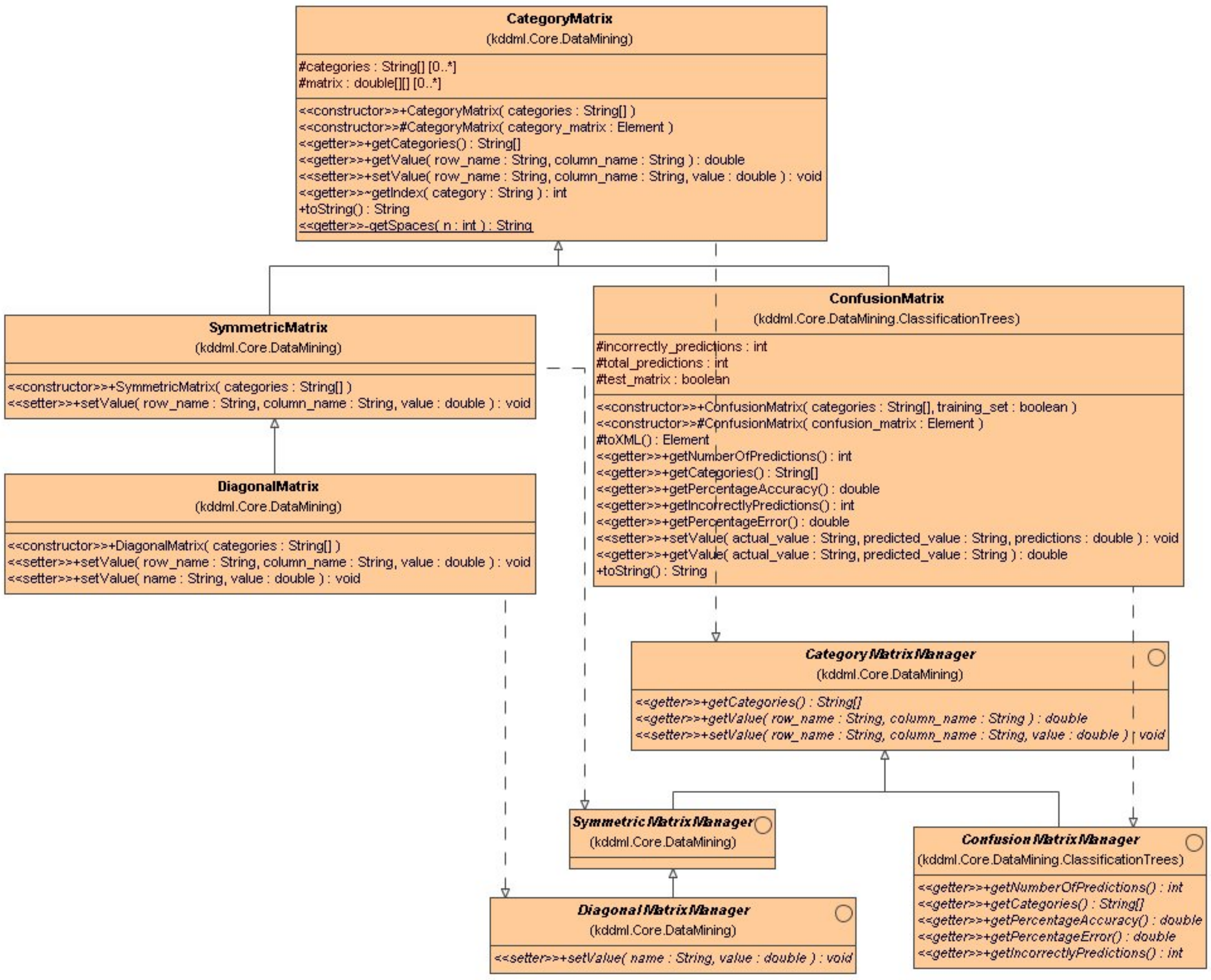

CategoryMatrix
(kddml.Core.DataMining)
#categories : String[] [0..*]
#matrix : double[][] [0..*]
<<constructor>>+CategoryMatrix(categories : String[])
<<constructor>>#CategoryMatrix(category_matrix : Element)
<<getter>>+getCategories() : String[]
<<getter>>+getValue(row_name : String, column_name : String) : double
<<setter>>+setValue(row_name : String, column_name : String, value : double) : void
<<getter>>~getIndex(category : String) : int
+toString() : String
<<getter>>-getSpaces(n : int) : String
SymmetricMatrix
(kddml.Core.DataMining)
<<constructor>>+SymmetricMatrix(categories : String[])
<<setter>>+setValue(row_name : String, column_name : String, value : double) : void
DiagonalMatrix
(kddml.Core.DataMining)
<<constructor>>+DiagonalMatrix(categories : String[])
<<setter>>+setValue(row_name : String, column_name : String, value : double) : void
<<setter>>+setValue(name : String, value : double) : void
ConfusionMatrix
(kddml.Core.DataMining.ClassificationTrees)
#incorrectly_predictions : int
#total_predictions : int
#test_matrix : boolean
<<constructor>>+ConfusionMatrix(categories : String[], training_set : boolean)
<<constructor>>#ConfusionMatrix(confusion_matrix : Element)
#toXML() : Element
<<getter>>+getNumberOfPredictions() : int
<<getter>>+getCategories() : String[]
<<getter>>+getPercentageAccuracy() : double
<<getter>>+getIncorrectlyPredictions() : int
<<getter>>+getPercentageError() : double
<<setter>>+setValue(actual_value : String, predicted_value : String, predictions : double) : void
<<getter>>+getValue(actual_value : String, predicted_value : String) : double
+toString() : String
CategoryMatrixManager
(kddml.Core.DataMining)
<<getter>>+getCategories() : String[]
<<getter>>+getValue(row_name : String, column_name : String) : double
<<setter>>+setValue(row_name : String, column_name : String, value : double) : void
SymmetricMatrixManager
(kddml.Core.DataMining)
ConfusionMatrixManager
(kddml.Core.DataMining.ClassificationTrees)
<<getter>>+getNumberOfPredictions() : int
<<getter>>+getCategories() : String[]
<<getter>>+getPercentageAccuracy() : double
<<getter>>+getPercentageError() : double
<<getter>>+getIncorrectlyPredictions() : int
DiagonalMatrixManager
(kddml.Core.DataMining)
<<setter>>+setValue(name : String, value : double) : void

Figure A.13: Package kddml.Core.DataMining - Matrixes

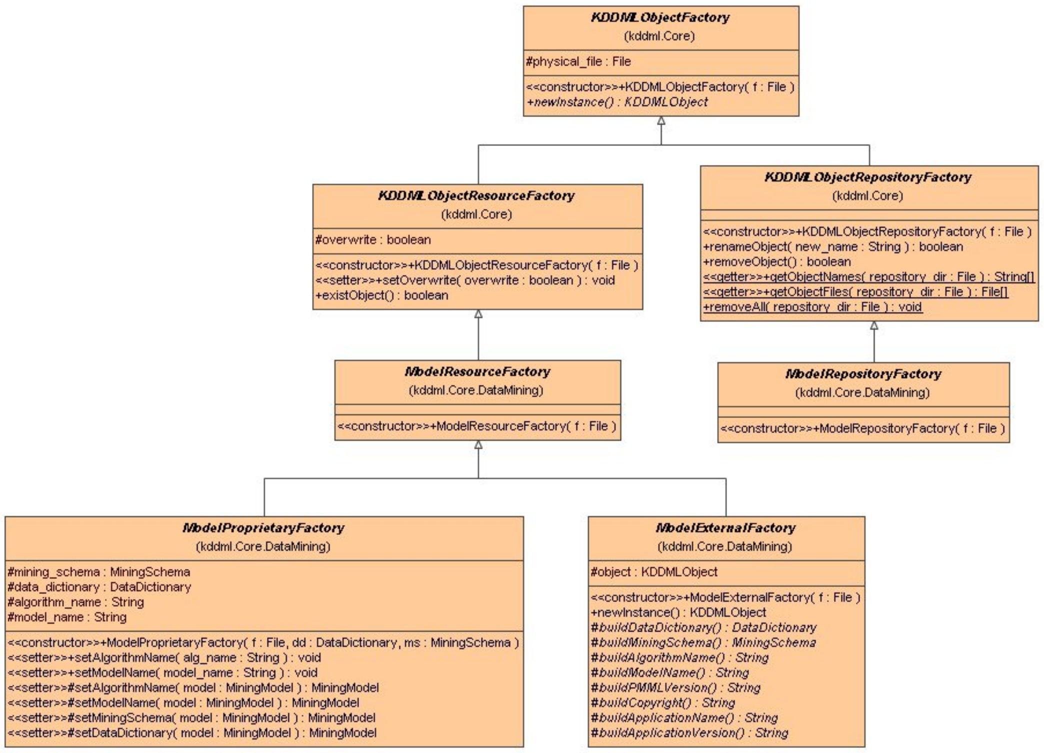

KDDMLObjectFactory
(kddml.Core)
#physical_file : File
<<constructor>>+KDDMLObjectFactory(f : File)
+newInstance() : KDDMLObject
KDDMLObjectResourceFactory
(kddml.Core)
#overwrite : boolean
<<constructor>>+KDDMLObjectResourceFactory(f : File)
<<setter>>+setOverwrite(overwrite : boolean) : void
+existObject() : boolean
KDDMLObjectRepositoryFactory
(kddml.Core)
<<constructor>>+KDDMLObjectRepositoryFactory(f : File)
+renameObject(new_name : String) : boolean
+removeObject() : boolean
<<getter>>+getObjectNames(repository_dir : File) : String[]
<<getter>>+getObjectFiles(repository_dir : File) : File[]
+removeAll(repository_dir : File) : void
ModelResourceFactory
(kddml.Core.DataMining)
<<constructor>>+ModelResourceFactory(f : File)
ModelRepositoryFactory
(kddml.Core.DataMining)
<<constructor>>+ModelRepositoryFactory(f : File)
ModelProprietaryFactory
(kddml.Core.DataMining)
#mining_schema : MiningSchema
#data_dictionary : DataDictionary
#algorithm_name : String
#model_name : String
<<constructor>>+ModelProprietaryFactory(f : File, dd : DataDictionary, ms : MiningSchema)
<<setter>>+setAlgorithmName(alg_name : String) : void
<<setter>>+setModelName(model_name : String) : void
<<setter>>#setAlgorithmName(model : MiningModel) : MiningModel
<<setter>>#setModelName(model : MiningModel) : MiningModel
<<setter>>#setMiningSchema(model : MiningModel) : MiningModel
<<setter>>#setDataDictionary(model : MiningModel) : MiningModel
ModelExternalFactory
(kddml.Core.DataMining)
#object : KDDMLObject
<<constructor>>+ModelExternalFactory(f : File)
+newInstance() : KDDMLObject
#buildDataDictionary() : DataDictionary
#buildMiningSchema() : MiningSchema
#buildAlgorithmName() : String
#buildModelName() : String
#buildPMMLVersion() : String
#buildCopyright() : String
#buildApplicationName() : String
#buildApplicationVersion() : String

Figure A.14: Package kddml.Core.DataMining - Model factory

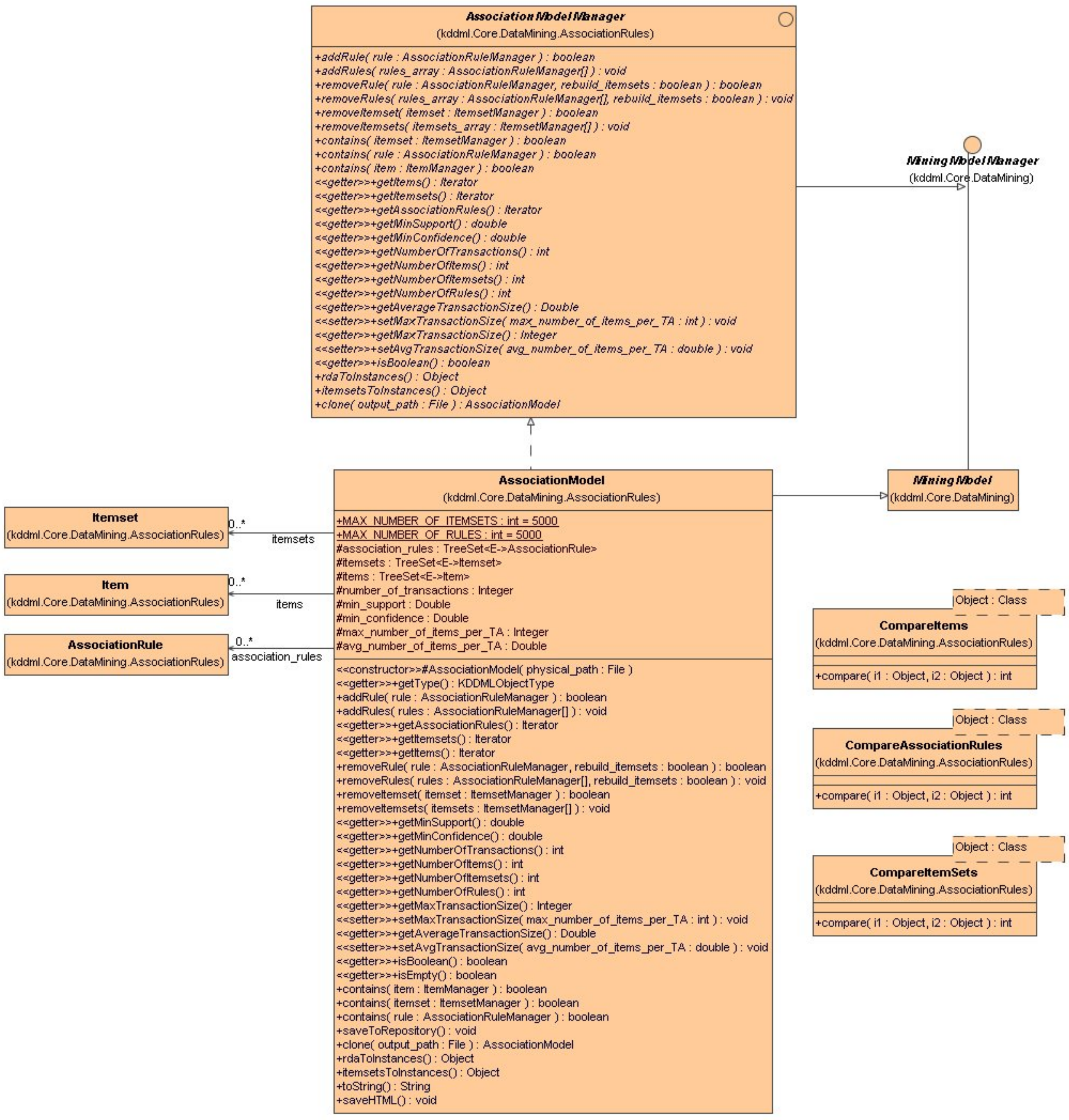

AssociationModelManager
(kddml.Core.DataMining.AssociationRules)
+addRule(rule : AssociationRuleManager) : boolean
+addRules(rules_array : AssociationRuleManager[]) : void
+removeRule(rule : AssociationRuleManager, rebuild_itemsets : boolean) : boolean
+removeRules(rules_array : AssociationRuleManager[], rebuild_itemsets : boolean) : void
+removeItemset(itemset : ItemsetManager) : boolean
+removeItemsets(itemsets_array : ItemsetManager[]) : void
+contains(itemset : ItemsetManager) : boolean
+contains(rule : AssociationRuleManager) : boolean
+contains(item : ItemManager) : boolean
<<getter>>+getItems() : Iterator
<<getter>>+getItemsets() : Iterator
<<getter>>+getAssociationRules() : Iterator
<<getter>>+getMinSupport() : double
<<getter>>+getMinConfidence() : double
<<getter>>+getNumberOfTransactions() : int
<<getter>>+getNumberOfItems() : int
<<getter>>+getNumberOfItemsets() : int
<<getter>>+getNumberOfRules() : int
<<getter>>+getAverageTransactionSize() : Double
<<setter>>+setMaxTransactionSize(max_number_of_items_per_TA : int) : void
<<getter>>+getMaxTransactionSize() : Integer
<<setter>>+setAvgTransactionSize(avg_number_of_items_per_TA : double) : void
<<getter>>+isBoolean() : boolean
+rdaToInstances() : Object
+itemsetsToInstances() : Object
+clone(output_path : File) : AssociationModel
MiningModelManager
(kddml.Core.DataMining)
AssociationModel
(kddml.Core.DataMining.AssociationRules)
+MAX_NUMBER_OF_ITEMSETS : int = 5000
+MAX_NUMBER_OF_RULES : int = 5000
#association_rules : TreeSet<E->AssociationRule>
#itemsets : TreeSet<E->Itemset>
#items : TreeSet<E->Item>
#number_of_transactions : Integer
#min_support : Double
#min_confidence : Double
#max_number_of_items_per_TA : Integer
#avg_number_of_items_per_TA : Double
<<constructor>>#AssociationModel(physical_path : File)
<<getter>>+getType() : KDDMLObjectType
+addRule(rule : AssociationRuleManager) : boolean
+addRules(rules : AssociationRuleManager[]) : void
<<getter>>+getAssociationRules() : Iterator
<<getter>>+getItemsets() : Iterator
<<getter>>+getItems() : Iterator
+removeRule(rule : AssociationRuleManager, rebuild_itemsets : boolean) : boolean
+removeRules(rules : AssociationRuleManager[], rebuild_itemsets : boolean) : void
+removeItemset(itemset : ItemsetManager) : boolean
+removeItemsets(itemsets : ItemsetManager[]) : void
<<getter>>+getMinSupport() : double
<<getter>>+getMinConfidence() : double
<<getter>>+getNumberOfTransactions() : int
<<getter>>+getNumberOfItems() : int
<<getter>>+getNumberOfItemsets() : int
<<getter>>+getNumberOfRules() : int
<<getter>>+getMaxTransactionSize() : Integer
<<setter>>+setMaxTransactionSize(max_number_of_items_per_TA : int) : void
<<getter>>+getAverageTransactionSize() : Double
<<setter>>+setAvgTransactionSize(avg_number_of_items_per_TA : double) : void
<<getter>>+isBoolean() : boolean
<<getter>>+isEmpty() : boolean
+contains(item : ItemManager) : boolean
+contains(itemset : ItemsetManager) : boolean
+contains(rule : AssociationRuleManager) : boolean
+saveToRepository() : void
+clone(output_path : File) : AssociationModel
+rdaToInstances() : Object
+itemsetsToInstances() : Object
+toString() : String
+saveHTML() : void
MiningModel
(kddml.Core.DataMining)
Itemset
(kddml.Core.DataMining.AssociationRules)
0..*
itemsets
Item
(kddml.Core.DataMining.AssociationRules)
0..*
items
AssociationRule
(kddml.Core.DataMining.AssociationRules)
0..*
association_rules
Object : Class
CompareItems
(kddml.Core.DataMining.AssociationRules)
+compare(i1 : Object, i2 : Object) : int
Object : Class
CompareAssociationRules
(kddml.Core.DataMining.AssociationRules)
+compare(i1 : Object, i2 : Object) : int
Object : Class
CompareItemSets
(kddml.Core.DataMining.AssociationRules)
+compare(i1 : Object, i2 : Object) : int

Figure A.15: Package kddml.Core.DataMining.AssociationRules - Manager

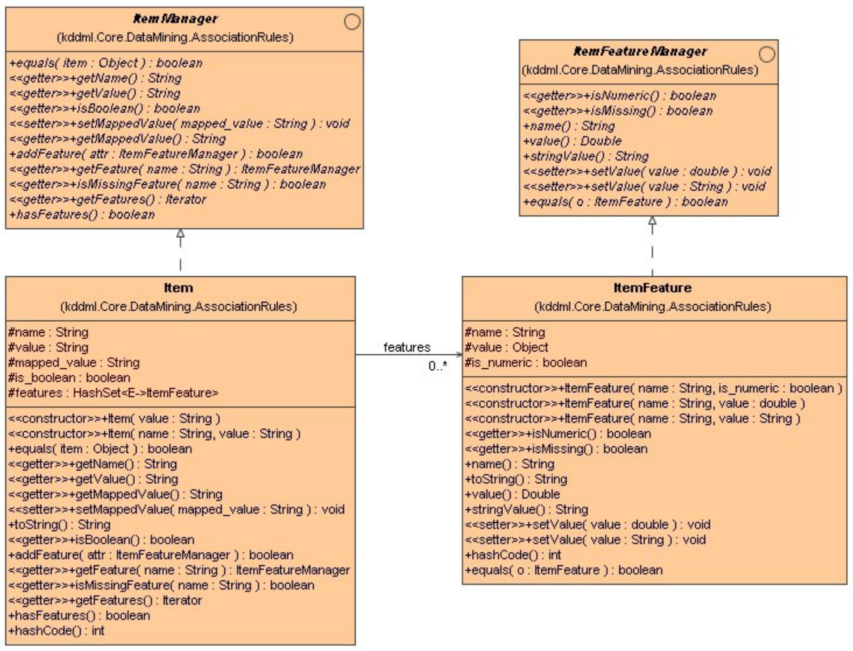

ItemManager
(kddml.Core.DataMining.AssociationRules)
+equals(item : Object) : boolean
<<getter>>+getName() : String
<<getter>>+getValue() : String
<<getter>>+isBoolean() : boolean
<<setter>>+setMappedValue(mapped_value : String) : void
<<getter>>+getMappedValue() : String
+addFeature(attr : ItemFeatureManager) : boolean
<<getter>>+getFeature(name : String) : ItemFeatureManager
<<getter>>+isMissingFeature(name : String) : boolean
<<getter>>+getFeatures() : Iterator
+hasFeatures() : boolean
ItemFeatureManager
(kddml.Core.DataMining.AssociationRules)
<<getter>>+isNumeric() : boolean
<<getter>>+isMissing() : boolean
+name() : String
+value() : Double
+stringValue() : String
<<setter>>+setValue(value : double) : void
<<setter>>+setValue(value : String) : void
+equals(o : ItemFeature) : boolean
Item
(kddml.Core.DataMining.AssociationRules)
#name : String
#value : String
#mapped_value : String
#is_boolean : boolean
#features : HashSet<E->ItemFeature>
<<constructor>>+Item(value : String)
<<constructor>>+Item(name : String, value : String)
+equals(item : Object) : boolean
<<getter>>+getName() : String
<<getter>>+getValue() : String
<<getter>>+getMappedValue() : String
<<setter>>+setMappedValue(mapped_value : String) : void
+toString() : String
<<getter>>+isBoolean() : boolean
+addFeature(attr : ItemFeatureManager) : boolean
<<getter>>+getFeature(name : String) : ItemFeatureManager
<<getter>>+isMissingFeature(name : String) : boolean
<<getter>>+getFeatures() : Iterator
+hasFeatures() : boolean
+hashCode() : int
features
0..*
ItemFeature
(kddml.Core.DataMining.AssociationRules)
#name : String
#value : Object
#is_numeric : boolean
<<constructor>>+ItemFeature(name : String, is_numeric : boolean)
<<constructor>>+ItemFeature(name : String, value : double)
<<constructor>>+ItemFeature(name : String, value : String)
<<getter>>+isNumeric() : boolean
<<getter>>+isMissing() : boolean
+name() : String
+toString() : String
+value() : Double
+stringValue() : String
<<setter>>+setValue(value : double) : void
<<setter>>+setValue(value : String) : void
+hashCode() : int
+equals(o : ItemFeature) : boolean

Figure A.16: Package kddml.Core.DataMining.AssociationRules - Items

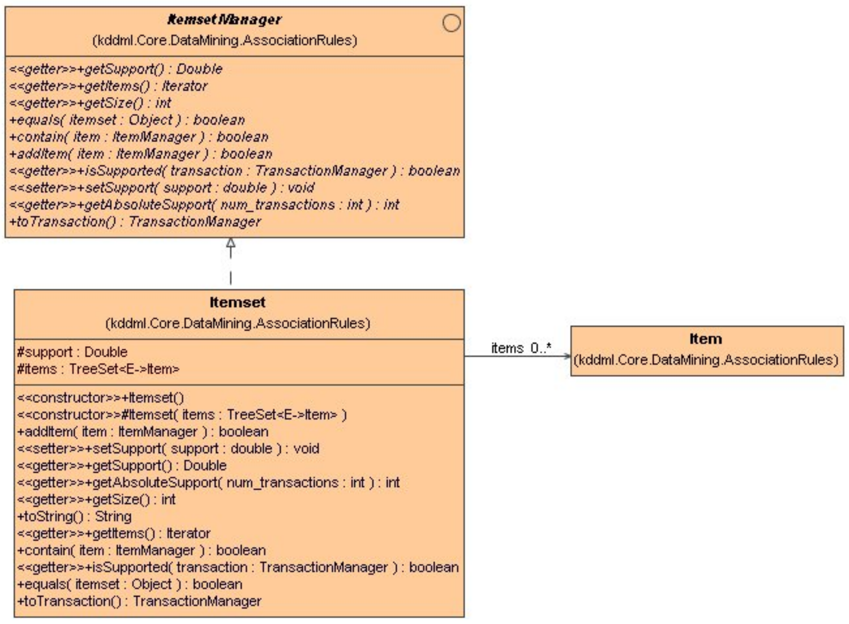

ItemsetManager
(kddml.Core.DataMining.AssociationRules)
<<getter>>+getSupport() : Double
<<getter>>+getItems() : Iterator
<<getter>>+getSize() : int
+equals(itemset : Object) : boolean
+contain(item : ItemManager) : boolean
+addItem(item : ItemManager) : boolean
<<getter>>+isSupported(transaction : TransactionManager) : boolean
<<setter>>+setSupport(support : double) : void
<<getter>>+getAbsoluteSupport(num_transactions : int) : int
+toTransaction() : TransactionManager
Itemset
(kddml.Core.DataMining.AssociationRules)
#support : Double
#items : TreeSet<E->Item>
<<constructor>>+Itemset()
<<constructor>>#Itemset(items : TreeSet<E->Item>)
+addItem(item : ItemManager) : boolean
<<setter>>+setSupport(support : double) : void
<<getter>>+getSupport() : Double
<<getter>>+getAbsoluteSupport(num_transactions : int) : int
<<getter>>+getSize() : int
+toString() : String
<<getter>>+getItems() : Iterator
+contain(item : ItemManager) : boolean
<<getter>>+isSupported(transaction : TransactionManager) : boolean
+equals(itemset : Object) : boolean
+toTransaction() : TransactionManager
items 0..*
Item
(kddml.Core.DataMining.AssociationRules)

Figure A.17: Package kddml.Core.DataMining.AssociationRules - Itemsets

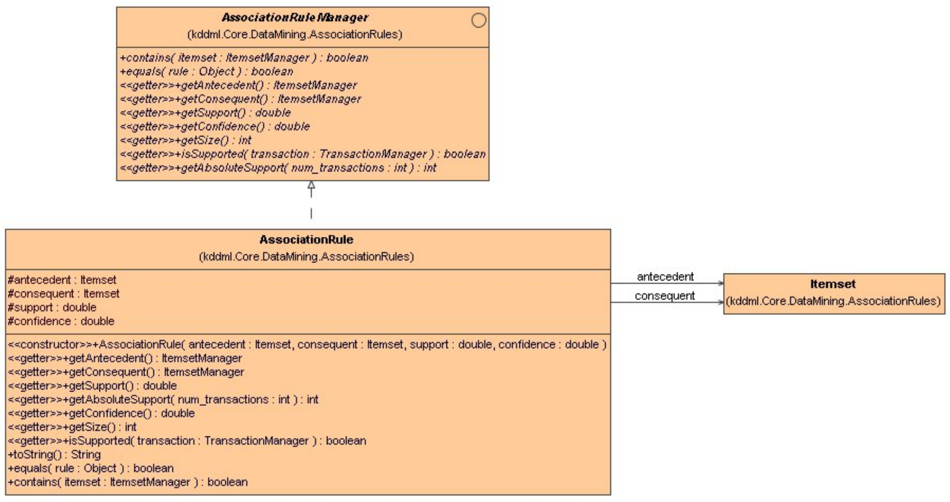

AssociationRuleManager
(kddml.Core.DataMining.AssociationRules)
+contains(itemset : ItemsetManager) : boolean
+equals(rule : Object) : boolean
<<getter>>+getAntecedent() : ItemsetManager
<<getter>>+getConsequent() : ItemsetManager
<<getter>>+getSupport() : double
<<getter>>+getConfidence() : double
<<getter>>+getSize() : int
<<getter>>+isSupported(transaction : TransactionManager) : boolean
<<getter>>+getAbsoluteSupport(num_transactions : int) : int
AssociationRule
(kddml.Core.DataMining.AssociationRules)
#antecedent : Itemset
#consequent : Itemset
#support : double
#confidence : double
<<constructor>>+AssociationRule(antecedent : Itemset, consequent : Itemset, support : double, confidence : double)
<<getter>>+getAntecedent() : ItemsetManager
<<getter>>+getConsequent() : ItemsetManager
<<getter>>+getSupport() : double
<<getter>>+getAbsoluteSupport(num_transactions : int) : int
<<getter>>+getConfidence() : double
<<getter>>+getSize() : int
<<getter>>+isSupported(transaction : TransactionManager) : boolean
+toString() : String
+equals(rule : Object) : boolean
+contains(itemset : ItemsetManager) : boolean
antecedent
consequent
Itemset
(kddml.Core.DataMining.AssociationRules)

Figure A.18: Package kddml.Core.DataMining.AssociationRules - Association Rules

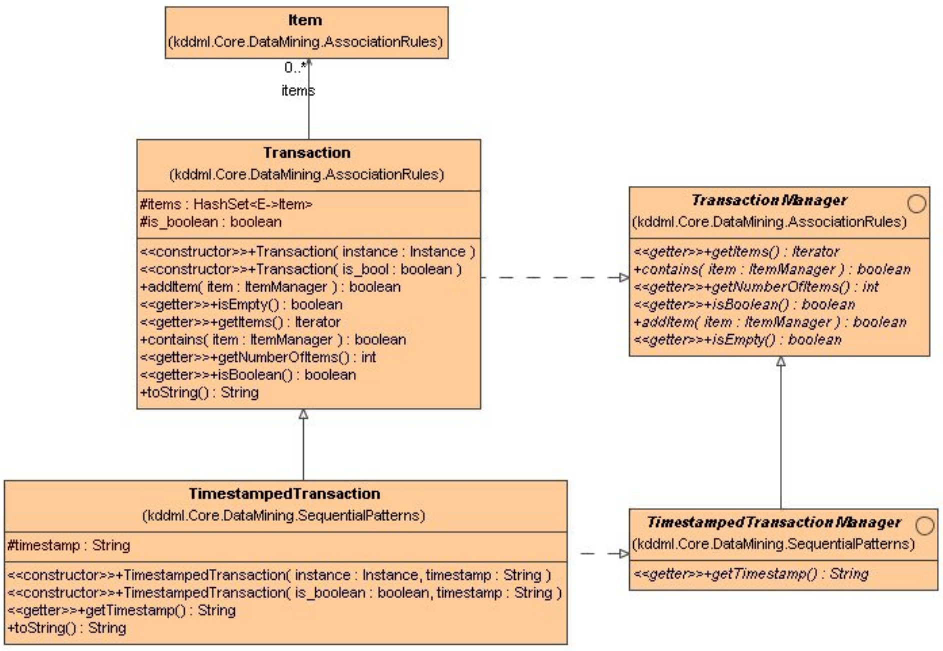

Item
(kddml.Core.DataMining.AssociationRules)
0..*
items
Transaction
(kddml.Core.DataMining.AssociationRules)
#items : HashSet<E->Item>
#is_boolean : boolean
<<constructor>>+Transaction(instance : Instance)
<<constructor>>+Transaction(is_bool : boolean)
+addItem(item : ItemManager) : boolean
<<getter>>+isEmpty() : boolean
<<getter>>+getItems() : Iterator
+contains(item : ItemManager) : boolean
<<getter>>+getNumberOfItems() : int
<<getter>>+isBoolean() : boolean
+toString() : String
TransactionManager
(kddml.Core.DataMining.AssociationRules)
<<getter>>+getItems() : Iterator
+contains(item : ItemManager) : boolean
<<getter>>+getNumberOfItems() : int
<<getter>>+isBoolean() : boolean
+addItem(item : ItemManager) : boolean
<<getter>>+isEmpty() : boolean
TimestampedTransaction
(kddml.Core.DataMining.SequentialPatterns)
#timestamp : String
<<constructor>>+TimestampedTransaction(instance : Instance, timestamp : String)
<<constructor>>+TimestampedTransaction(is_boolean : boolean, timestamp : String)
<<getter>>+getTimestamp() : String
+toString() : String
TimestampedTransactionManager
(kddml.Core.DataMining.SequentialPatterns)
<<getter>>+getTimestamp() : String

Figure A.19: Package kddml.Core.DataMining.AssociationRules - Transactions

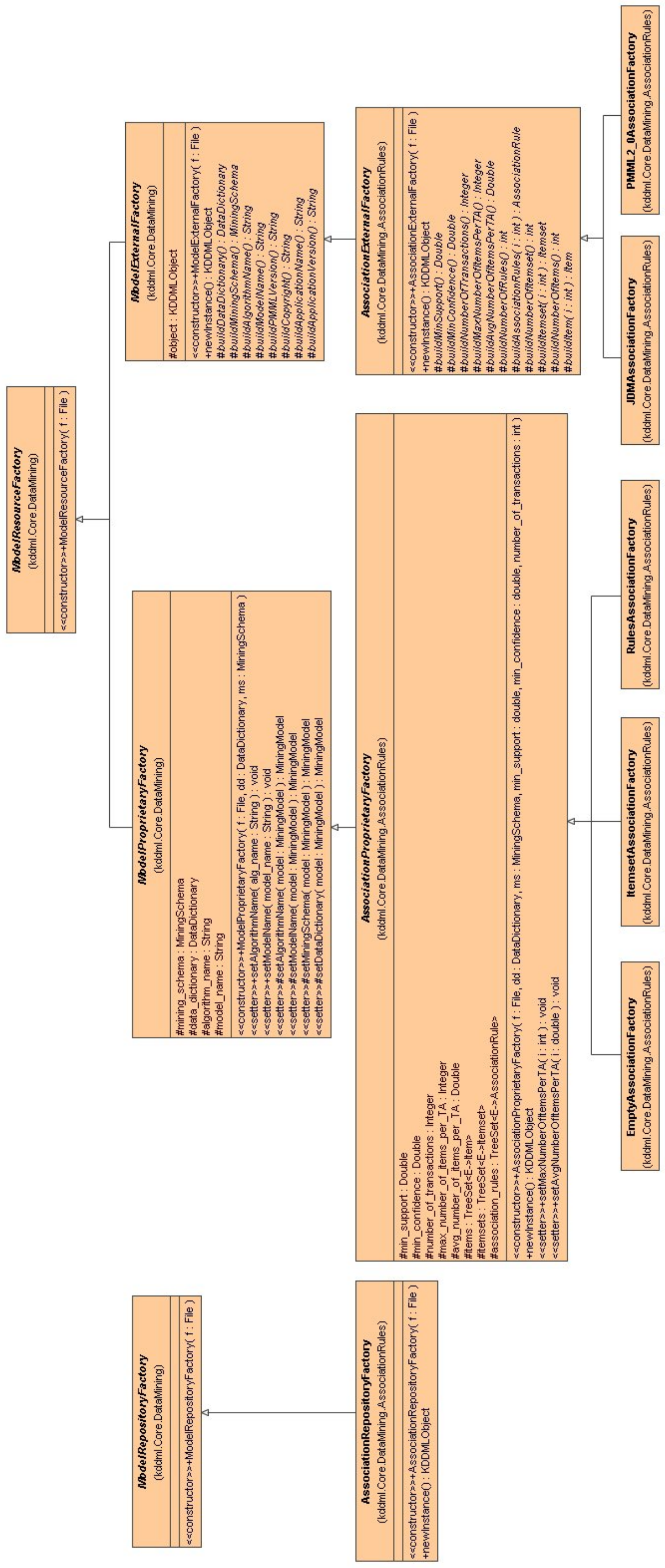

ModelResourceFactory
(kddml.Core.DataMining)
<<constructor>>+ModelResourceFactory(f : File)
ModelExternalFactory
(kddml.Core.DataMining)
#object : KDDMLObject
<<constructor>>+ModelExternalFactory(f : File)
+newInstance() : KDDMLObject
#buildDataDictionary() : DataDictionary
#buildMiningSchema() : MiningSchema
#buildAlgorithmName() : String
#buildModelName() : String
#buildPMMLVersion() : String
#buildCopyright() : String
#buildApplicationName() : String
#buildApplicationVersion() : String
ModelProprietaryFactory
(kddml.Core.DataMining)
#mining_schema : MiningSchema
#data_dictionary : DataDictionary
#algorithm_name : String
#model_name : String
<<constructor>>+ModelProprietaryFactory(f : File, dd : DataDictionary, ms : MiningSchema)
<<setter>>+setAlgorithmName(alg_name : String) : void
<<setter>>+setModelName(model_name : String) : void
<<setter>>#setAlgorithmName(model : MiningModel) : MiningModel
<<setter>>#setModelName(model : MiningModel) : MiningModel
<<setter>>#setMiningSchema(model : MiningModel) : MiningModel
<<setter>>#setDataDictionary(model : MiningModel) : MiningModel
ModelRepositoryFactory
(kddml.Core.DataMining)
<<constructor>>+ModelRepositoryFactory(f : File)
AssociationExternalFactory
(kddml.Core.DataMining.AssociationRules)
<<constructor>>+AssociationExternalFactory(f : File)
+newInstance() : KDDMLObject
#buildMinSupport() : Double
#buildMinConfidence() : Double
#buildNumberOfTransactions() : Integer
#buildMaxNumberOfItemsPerTA() : Integer
#buildAvgNumberOfItemsPerTA() : Double
#buildNumberOfRules() : int
#buildAssociationRules(i : int) : AssociationRule
#buildNumberOfItemset() : int
#buildItemset(i : int) : Itemset
#buildNumberOfItems() : int
#buildItem(i : int) : Item
AssociationProprietaryFactory
(kddml.Core.DataMining.AssociationRules)
#min_support : Double
#min_confidence : Double
#number_of_transactions : Integer
#max_number_of_items_per_TA : Integer
#avg_number_of_items_per_TA : Double
#items : TreeSet<E->Item>
#itemsets : TreeSet<E->Itemset>
#association_rules : TreeSet<E->AssociationRule>
<<constructor>>+AssociationProprietaryFactory(f : File, dd : DataDictionary, ms : MiningSchema, min_support : double, min_confidence : double, number_of_transactions : int)
+newInstance() : KDDMLObject
<<setter>>+setMaxNumberOfItemsPerTA(i : int) : void
<<setter>>+setAvgNumberOfItemsPerTA(i : double) : void
AssociationRepositoryFactory
(kddml.Core.DataMining.AssociationRules)
<<constructor>>+AssociationRepositoryFactory(f : File)
+newInstance() : KDDMLObject
PMML2_0AssociationFactory
(kddml.Core.DataMining.AssociationRules)
JDMAssociationFactory
(kddml.Core.DataMining.AssociationRules)
RulesAssociationFactory
(kddml.Core.DataMining.AssociationRules)
ItemsetAssociationFactory
(kddml.Core.DataMining.AssociationRules)
EmptyAssociationFactory
(kddml.Core.DataMining.AssociationRules)

Figure A.20: Package kddml.Core.DataMining.AssociationRules - Factory

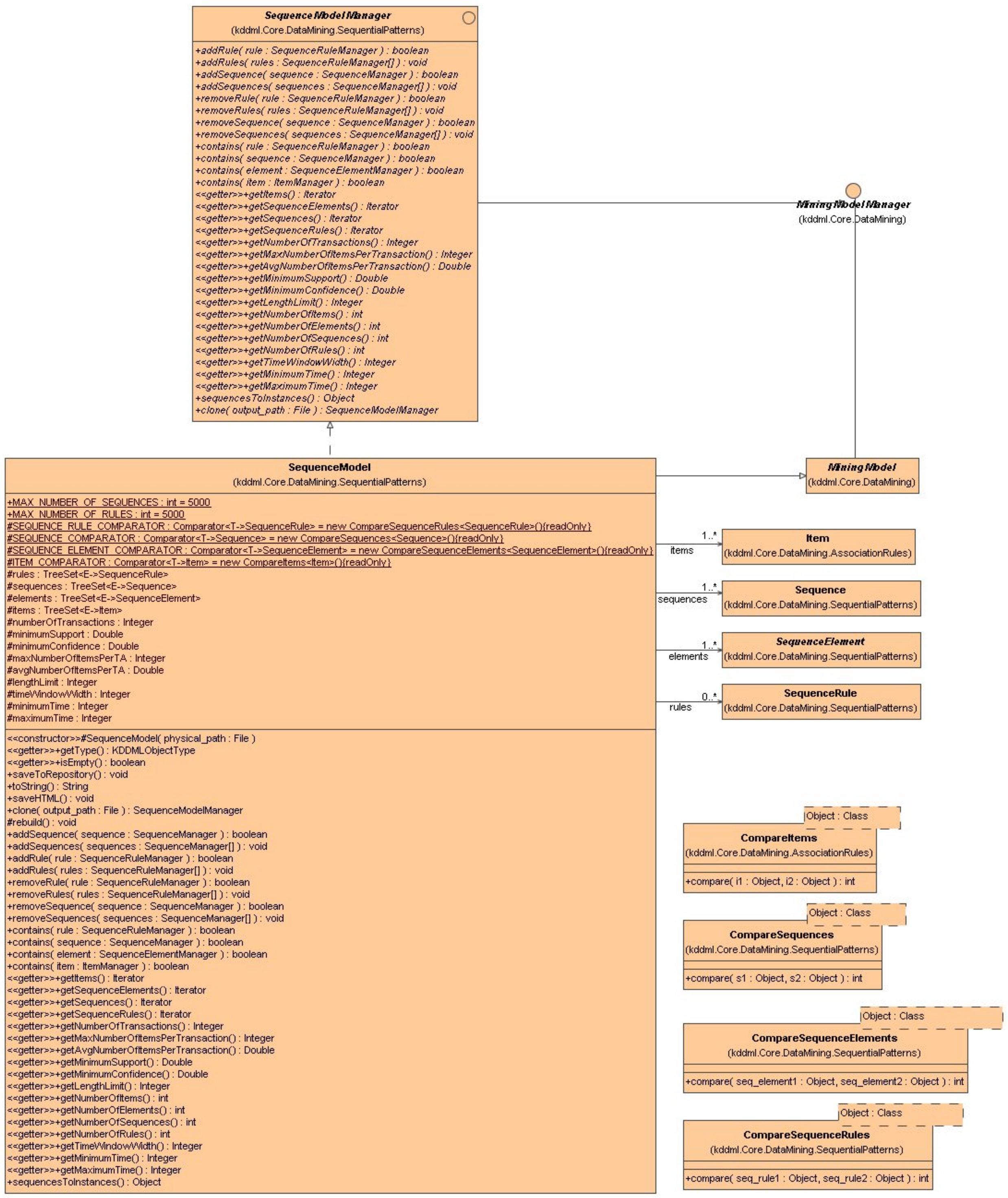

SequenceModelManager
(kddml.Core.DataMining.SequentialPatterns)
+addRule(rule : SequenceRuleManager) : boolean
+addRules(rules : SequenceRuleManager[]) : void
+addSequence(sequence : SequenceManager) : boolean
+addSequences(sequences : SequenceManager[]) : void
+removeRule(rule : SequenceRuleManager) : boolean
+removeRules(rules : SequenceRuleManager[]) : void
+removeSequence(sequence : SequenceManager) : boolean
+removeSequences(sequences : SequenceManager[]) : void
+contains(rule : SequenceRuleManager) : boolean
+contains(sequence : SequenceManager) : boolean
+contains(element : SequenceElementManager) : boolean
+contains(item : ItemManager) : boolean
<<getter>>+getItems() : Iterator
<<getter>>+getSequenceElements() : Iterator
<<getter>>+getSequences() : Iterator
<<getter>>+getSequenceRules() : Iterator
<<getter>>+getNumberOfTransactions() : Integer
<<getter>>+getMaxNumberOfItemsPerTransaction() : Integer
<<getter>>+getAvgNumberOfItemsPerTransaction() : Double
<<getter>>+getMinimumSupport() : Double
<<getter>>+getMinimumConfidence() : Double
<<getter>>+getLengthLimit() : Integer
<<getter>>+getNumberOfItems() : int
<<getter>>+getNumberOfElements() : int
<<getter>>+getNumberOfSequences() : int
<<getter>>+getNumberOfRules() : int
<<getter>>+getTimeWindowWidth() : Integer
<<getter>>+getMinimumTime() : Integer
<<getter>>+getMaximumTime() : Integer
+sequencesToInstances() : Object
+clone(output_path : File) : SequenceModelManager
MiningModelManager
(kddml.Core.DataMining)
SequenceModel
(kddml.Core.DataMining.SequentialPatterns)
+MAX_NUMBER_OF_SEQUENCES : int = 5000
+MAX_NUMBER_OF_RULES : int = 5000
#SEQUENCE_RULE_COMPARATOR : Comparator<T->SequenceRule> = new CompareSequenceRules<SequenceRule>(){readOnly}
#SEQUENCE_COMPARATOR : Comparator<T->Sequence> = new CompareSequences<Sequence>(){readOnly}
#SEQUENCE_ELEMENT_COMPARATOR : Comparator<T->SequenceElement> = new CompareSequenceElements<SequenceElement>(){readOnly}
#ITEM_COMPARATOR : Comparator<T->Item> = new CompareItems<Item>(){readOnly}
#rules : TreeSet<E->SequenceRule>
#sequences : TreeSet<E->Sequence>
#elements : TreeSet<E->SequenceElement>
#items : TreeSet<E->Item>
#numberOfTransactions : Integer
#minimumSupport : Double
#minimumConfidence : Double
#maxNumberOfItemsPerTA : Integer
#avgNumberOfItemsPerTA : Double
#lengthLimit : Integer
#timeWindowWidth : Integer
#minimumTime : Integer
#maximumTime : Integer
<<constructor>>#SequenceModel(physical_path : File)
<<getter>>+getType() : KDDMLObjectType
<<getter>>+isEmpty() : boolean
+saveToRepository() : void
+toString() : String
+saveHTML() : void
+clone(output_path : File) : SequenceModelManager
#rebuild() : void
+addSequence(sequence : SequenceManager) : boolean
+addSequences(sequences : SequenceManager[]) : void
+addRule(rule : SequenceRuleManager) : boolean
+addRules(rules : SequenceRuleManager[]) : void
+removeRule(rule : SequenceRuleManager) : boolean
+removeRules(rules : SequenceRuleManager[]) : void
+removeSequence(sequence : SequenceManager) : boolean
+removeSequences(sequences : SequenceManager[]) : void
+contains(rule : SequenceRuleManager) : boolean
+contains(sequence : SequenceManager) : boolean
+contains(element : SequenceElementManager) : boolean
+contains(item : ItemManager) : boolean
<<getter>>+getItems() : Iterator
<<getter>>+getSequenceElements() : Iterator
<<getter>>+getSequences() : Iterator
<<getter>>+getSequenceRules() : Iterator
<<getter>>+getNumberOfTransactions() : Integer
<<getter>>+getMaxNumberOfItemsPerTransaction() : Integer
<<getter>>+getAvgNumberOfItemsPerTransaction() : Double
<<getter>>+getMinimumSupport() : Double
<<getter>>+getMinimumConfidence() : Double
<<getter>>+getLengthLimit() : Integer
<<getter>>+getNumberOfItems() : int
<<getter>>+getNumberOfElements() : int
<<getter>>+getNumberOfSequences() : int
<<getter>>+getNumberOfRules() : int
<<getter>>+getTimeWindowWidth() : Integer
<<getter>>+getMinimumTime() : Integer
<<getter>>+getMaximumTime() : Integer
+sequencesToInstances() : Object
MiningModel
(kddml.Core.DataMining)
1..*
items
Item
(kddml.Core.DataMining.AssociationRules)
1..*
sequences
Sequence
(kddml.Core.DataMining.SequentialPatterns)
1..*
elements
SequenceElement
(kddml.Core.DataMining.SequentialPatterns)
0..*
rules
SequenceRule
(kddml.Core.DataMining.SequentialPatterns)
Object : Class
CompareItems
(kddml.Core.DataMining.AssociationRules)
+compare(i1 : Object, i2 : Object) : int
Object : Class
CompareSequences
(kddml.Core.DataMining.SequentialPatterns)
+compare(s1 : Object, s2 : Object) : int
Object : Class
CompareSequenceElements
(kddml.Core.DataMining.SequentialPatterns)
+compare(seq_element1 : Object, seq_element2 : Object) : int
Object : Class
CompareSequenceRules
(kddml.Core.DataMining.SequentialPatterns)
+compare(seq_rule1 : Object, seq_rule2 : Object) : int

Figure A.21: Package kddml.Core.DataMining.SequentialPatterns - Manager

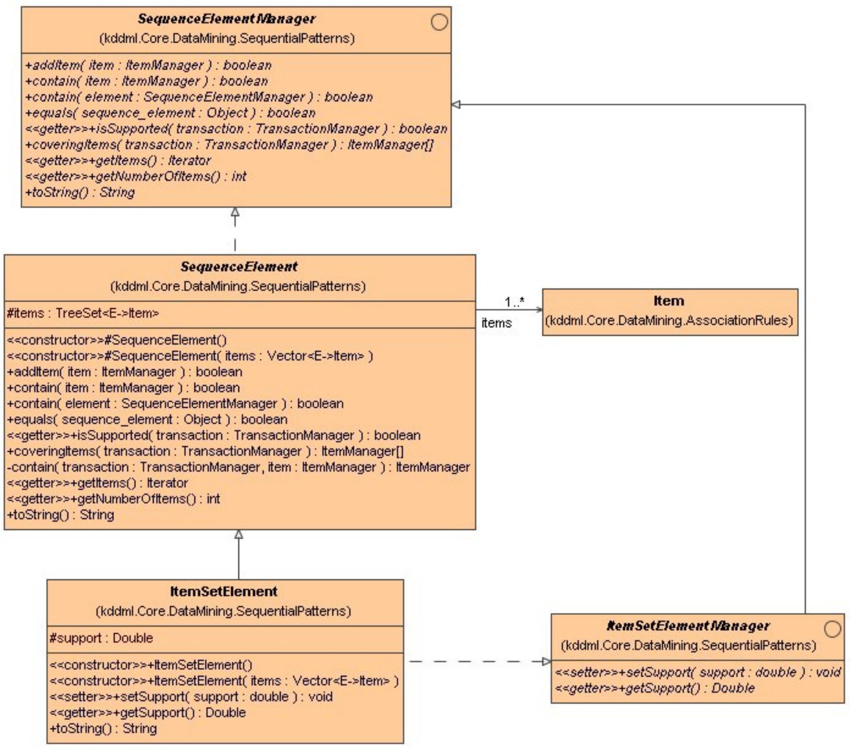

SequenceElementManager
(kddml.Core.DataMining.SequentialPatterns)
+addItem(item : ItemManager) : boolean
+contain(item : ItemManager) : boolean
+contain(element : SequenceElementManager) : boolean
+equals(sequence_element : Object) : boolean
<<getter>>+isSupported(transaction : TransactionManager) : boolean
+coveringItems(transaction : TransactionManager) : ItemManager[]
<<getter>>+getItems() : Iterator
<<getter>>+getNumberOfItems() : int
+toString() : String
SequenceElement
(kddml.Core.DataMining.SequentialPatterns)
#items : TreeSet<E->Item>
<<constructor>>#SequenceElement()
<<constructor>>#SequenceElement(items : Vector<E->Item>)
+addItem(item : ItemManager) : boolean
+contain(item : ItemManager) : boolean
+contain(element : SequenceElementManager) : boolean
+equals(sequence_element : Object) : boolean
<<getter>>+isSupported(transaction : TransactionManager) : boolean
+coveringItems(transaction : TransactionManager) : ItemManager[]
-contain(transaction : TransactionManager, item : ItemManager) : ItemManager
<<getter>>+getItems() : Iterator
<<getter>>+getNumberOfItems() : int
+toString() : String
1..*
items
Item
(kddml.Core.DataMining.AssociationRules)
ItemSetElement
(kddml.Core.DataMining.SequentialPatterns)
#support : Double
<<constructor>>+ItemSetElement()
<<constructor>>+ItemSetElement(items : Vector<E->Item>)
<<setter>>+setSupport(support : double) : void
<<getter>>+getSupport() : Double
+toString() : String
ItemSetElementManager
(kddml.Core.DataMining.SequentialPatterns)
<<setter>>+setSupport(support : double) : void
<<getter>>+getSupport() : Double

Figure A.22: Package kddml.Core.DataMining.SequentialPatterns - Sequence element

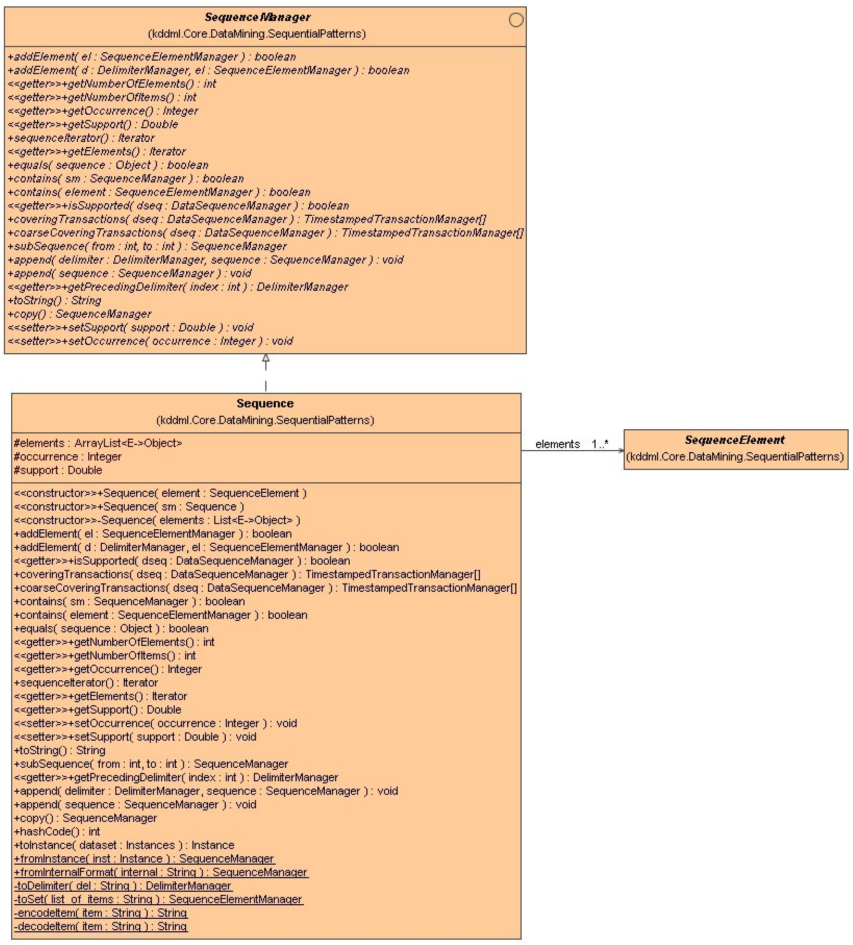

SequenceManager
(kddml.Core.DataMining.SequentialPatterns)
+addElement(el : SequenceElementManager) : boolean
+addElement(d : DelimiterManager, el : SequenceElementManager) : boolean
<<getter>>+getNumberOfElements() : int
<<getter>>+getNumberOfItems() : int
<<getter>>+getOccurrence() : Integer
<<getter>>+getSupport() : Double
+sequenceIterator() : Iterator
<<getter>>+getElements() : Iterator
+equals(sequence : Object) : boolean
+contains(sm : SequenceManager) : boolean
+contains(element : SequenceElementManager) : boolean
<<getter>>+isSupported(dseq : DataSequenceManager) : boolean
+coveringTransactions(dseq : DataSequenceManager) : TimestampedTransactionManager[]
+coarseCoveringTransactions(dseq : DataSequenceManager) : TimestampedTransactionManager[]
+subSequence(from : int, to : int) : SequenceManager
+append(delimiter : DelimiterManager, sequence : SequenceManager) : void
+append(sequence : SequenceManager) : void
<<getter>>+getPrecedingDelimiter(index : int) : DelimiterManager
+toString() : String
+copy() : SequenceManager
<<setter>>+setSupport(support : Double) : void
<<setter>>+setOccurrence(occurrence : Integer) : void
Sequence
(kddml.Core.DataMining.SequentialPatterns)
#elements : ArrayList<E->Object>
#occurrence : Integer
#support : Double
<<constructor>>+Sequence(element : SequenceElement)
<<constructor>>+Sequence(sm : Sequence)
<<constructor>>-Sequence(elements : List<E->Object>)
+addElement(el : SequenceElementManager) : boolean
+addElement(d : DelimiterManager, el : SequenceElementManager) : boolean
<<getter>>+isSupported(dseq : DataSequenceManager) : boolean
+coveringTransactions(dseq : DataSequenceManager) : TimestampedTransactionManager[]
+coarseCoveringTransactions(dseq : DataSequenceManager) : TimestampedTransactionManager[]
+contains(sm : SequenceManager) : boolean
+contains(element : SequenceElementManager) : boolean
+equals(sequence : Object) : boolean
<<getter>>+getNumberOfElements() : int
<<getter>>+getNumberOfItems() : int
<<getter>>+getOccurrence() : Integer
+sequenceIterator() : Iterator
<<getter>>+getElements() : Iterator
<<getter>>+getSupport() : Double
<<setter>>+setOccurrence(occurrence : Integer) : void
<<setter>>+setSupport(support : Double) : void
+toString() : String
+subSequence(from : int, to : int) : SequenceManager
<<getter>>+getPrecedingDelimiter(index : int) : DelimiterManager
+append(delimiter : DelimiterManager, sequence : SequenceManager) : void
+append(sequence : SequenceManager) : void
+copy() : SequenceManager
+hashCode() : int
+toInstance(dataset : Instances) : Instance
+fromInstance(inst : Instance) : SequenceManager
+fromInternalFormat(internal : String) : SequenceManager
-toDelimiter(del : String) : DelimiterManager
-toSet(list_of_items : String) : SequenceElementManager
-encodeItem(item : String) : String
-decodeItem(item : String) : String
elements 1..*
SequenceElement
(kddml.Core.DataMining.SequentialPatterns)

Figure A.23: Package kddml.Core.DataMining.SequentialPatterns - Sequence

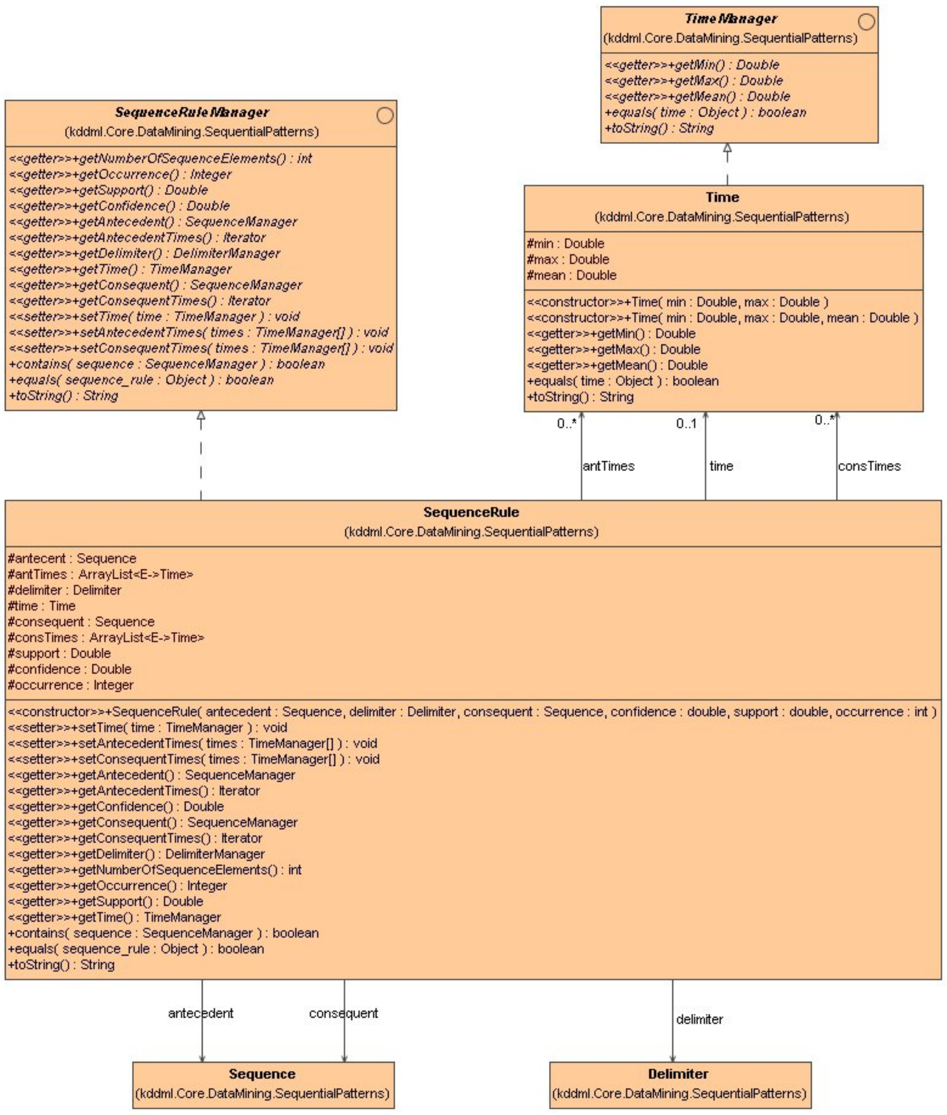

TimeManager
(kddml.Core.DataMining.SequentialPatterns)
<<getter>>+getMin() : Double
<<getter>>+getMax() : Double
<<getter>>+getMean() : Double
+equals(time : Object) : boolean
+toString() : String
SequenceRuleManager
(kddml.Core.DataMining.SequentialPatterns)
<<getter>>+getNumberOfSequenceElements() : int
<<getter>>+getOccurrence() : Integer
<<getter>>+getSupport() : Double
<<getter>>+getConfidence() : Double
<<getter>>+getAntecedent() : SequenceManager
<<getter>>+getAntecedentTimes() : Iterator
<<getter>>+getDelimiter() : DelimiterManager
<<getter>>+getTime() : TimeManager
<<getter>>+getConsequent() : SequenceManager
<<getter>>+getConsequentTimes() : Iterator
<<setter>>+setTime(time : TimeManager) : void
<<setter>>+setAntecedentTimes(times : TimeManager[]) : void
<<setter>>+setConsequentTimes(times : TimeManager[]) : void
+contains(sequence : SequenceManager) : boolean
+equals(sequence_rule : Object) : boolean
+toString() : String
Time
(kddml.Core.DataMining.SequentialPatterns)
#min : Double
#max : Double
#mean : Double
<<constructor>>+Time(min : Double, max : Double)
<<constructor>>+Time(min : Double, max : Double, mean : Double)
<<getter>>+getMin() : Double
<<getter>>+getMax() : Double
<<getter>>+getMean() : Double
+equals(time : Object) : boolean
+toString() : String
0..*
antTimes
0..1
time
0..*
consTimes
SequenceRule
(kddml.Core.DataMining.SequentialPatterns)
#antecent : Sequence
#antTimes : ArrayList<E->Time>
#delimiter : Delimiter
#time : Time
#consequent : Sequence
#consTimes : ArrayList<E->Time>
#support : Double
#confidence : Double
#occurrence : Integer
<<constructor>>+SequenceRule(antecedent : Sequence, delimiter : Delimiter, consequent : Sequence, confidence : double, support : double, occurrence : int)
<<setter>>+setTime(time : TimeManager) : void
<<setter>>+setAntecedentTimes(times : TimeManager[]) : void
<<setter>>+setConsequentTimes(times : TimeManager[]) : void
<<getter>>+getAntecedent() : SequenceManager
<<getter>>+getAntecedentTimes() : Iterator
<<getter>>+getConfidence() : Double
<<getter>>+getConsequent() : SequenceManager
<<getter>>+getConsequentTimes() : Iterator
<<getter>>+getDelimiter() : DelimiterManager
<<getter>>+getNumberOfSequenceElements() : int
<<getter>>+getOccurrence() : Integer
<<getter>>+getSupport() : Double
<<getter>>+getTime() : TimeManager
+contains(sequence : SequenceManager) : boolean
+equals(sequence_rule : Object) : boolean
+toString() : String
antecedent
consequent
delimiter
Sequence
(kddml.Core.DataMining.SequentialPatterns)
Delimiter
(kddml.Core.DataMining.SequentialPatterns)

Figure A.24: Package kddml.Core.DataMining.SequentialPatterns - Sequence rule

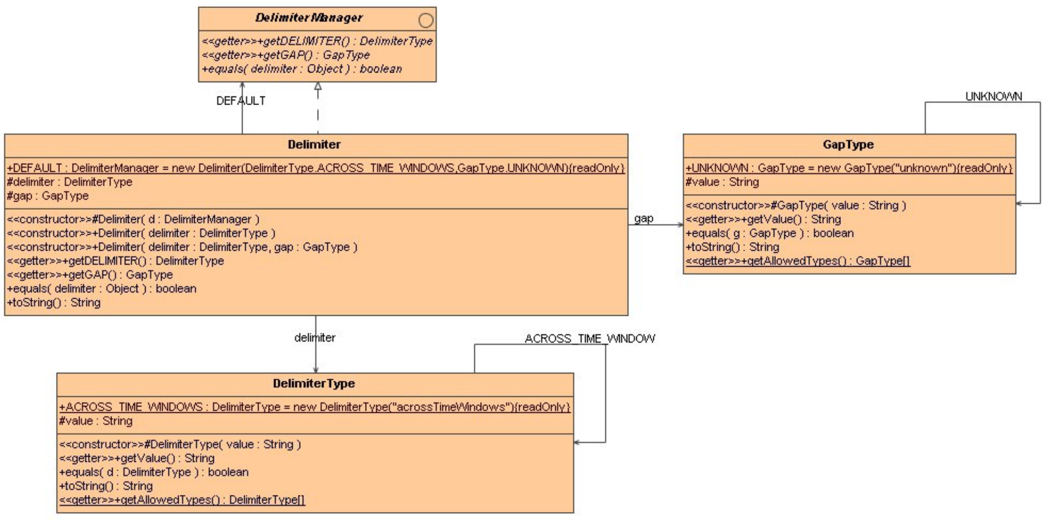

DelimiterManager
<<getter>>+getDELIMITER() : DelimiterType
<<getter>>+getGAP() : GapType
+equals(delimiter : Object) : boolean
DEFAULT
UNKNOWN
Delimiter
+DEFAULT : DelimiterManager = new Delimiter(DelimiterType.ACROSS_TIME_WINDOWS,GapType.UNKNOWN){readOnly}
#delimiter : DelimiterType
#gap : GapType
<<constructor>>#Delimiter(d : DelimiterManager)
<<constructor>>+Delimiter(delimiter : DelimiterType)
<<constructor>>+Delimiter(delimiter : DelimiterType, gap : GapType)
<<getter>>+getDELIMITER() : DelimiterType
<<getter>>+getGAP() : GapType
+equals(delimiter : Object) : boolean
+toString() : String
gap
GapType
+UNKNOWN : GapType = new GapType("unknown"){readOnly}
#value : String
<<constructor>>#GapType(value : String)
<<getter>>+getValue() : String
+equals(g : GapType) : boolean
+toString() : String
<<getter>>+getAllowedTypes() : GapType[]
delimiter
ACROSS_TIME_WINDOW
DelimiterType
+ACROSS_TIME_WINDOWS : DelimiterType = new DelimiterType("acrossTimeWindows"){readOnly}
#value : String
<<constructor>>#DelimiterType(value : String)
<<getter>>+getValue() : String
+equals(d : DelimiterType) : boolean
+toString() : String
<<getter>>+getAllowedTypes() : DelimiterType[]

Figure A.25: Package kddml.Core.DataMining.SequentialPatterns - Delimiter

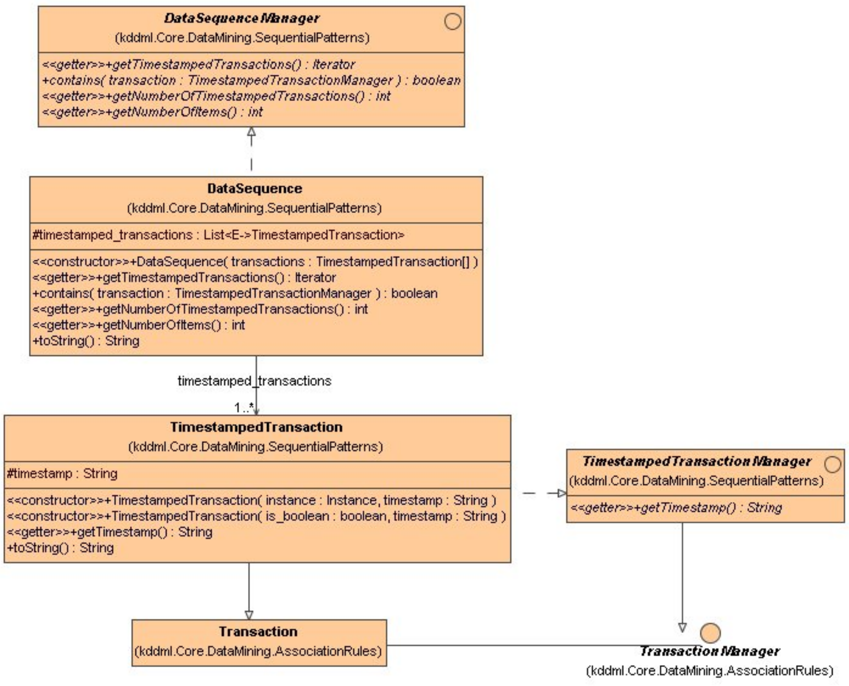

DataSequenceManager
(kddml.Core.DataMining.SequentialPatterns)
<<getter>>+getTimestampedTransactions() : Iterator
+contains(transaction : TimestampedTransactionManager) : boolean
<<getter>>+getNumberOfTimestampedTransactions() : int
<<getter>>+getNumberOfItems() : int
DataSequence
(kddml.Core.DataMining.SequentialPatterns)
#timestamped_transactions : List<E->TimestampedTransaction>
<<constructor>>+DataSequence(transactions : TimestampedTransaction[])
<<getter>>+getTimestampedTransactions() : Iterator
+contains(transaction : TimestampedTransactionManager) : boolean
<<getter>>+getNumberOfTimestampedTransactions() : int
<<getter>>+getNumberOfItems() : int
+toString() : String
timestamped_transactions
1..*
TimestampedTransaction
(kddml.Core.DataMining.SequentialPatterns)
#timestamp : String
<<constructor>>+TimestampedTransaction(instance : Instance, timestamp : String)
<<constructor>>+TimestampedTransaction(is_boolean : boolean, timestamp : String)
<<getter>>+getTimestamp() : String
+toString() : String
TimestampedTransactionManager
(kddml.Core.DataMining.SequentialPatterns)
<<getter>>+getTimestamp() : String
Transaction
(kddml.Core.DataMining.AssociationRules)
TransactionManager
(kddml.Core.DataMining.AssociationRules)

Figure A.26: Package kddml.Core.DataMining.SequentialPatterns - Data sequence

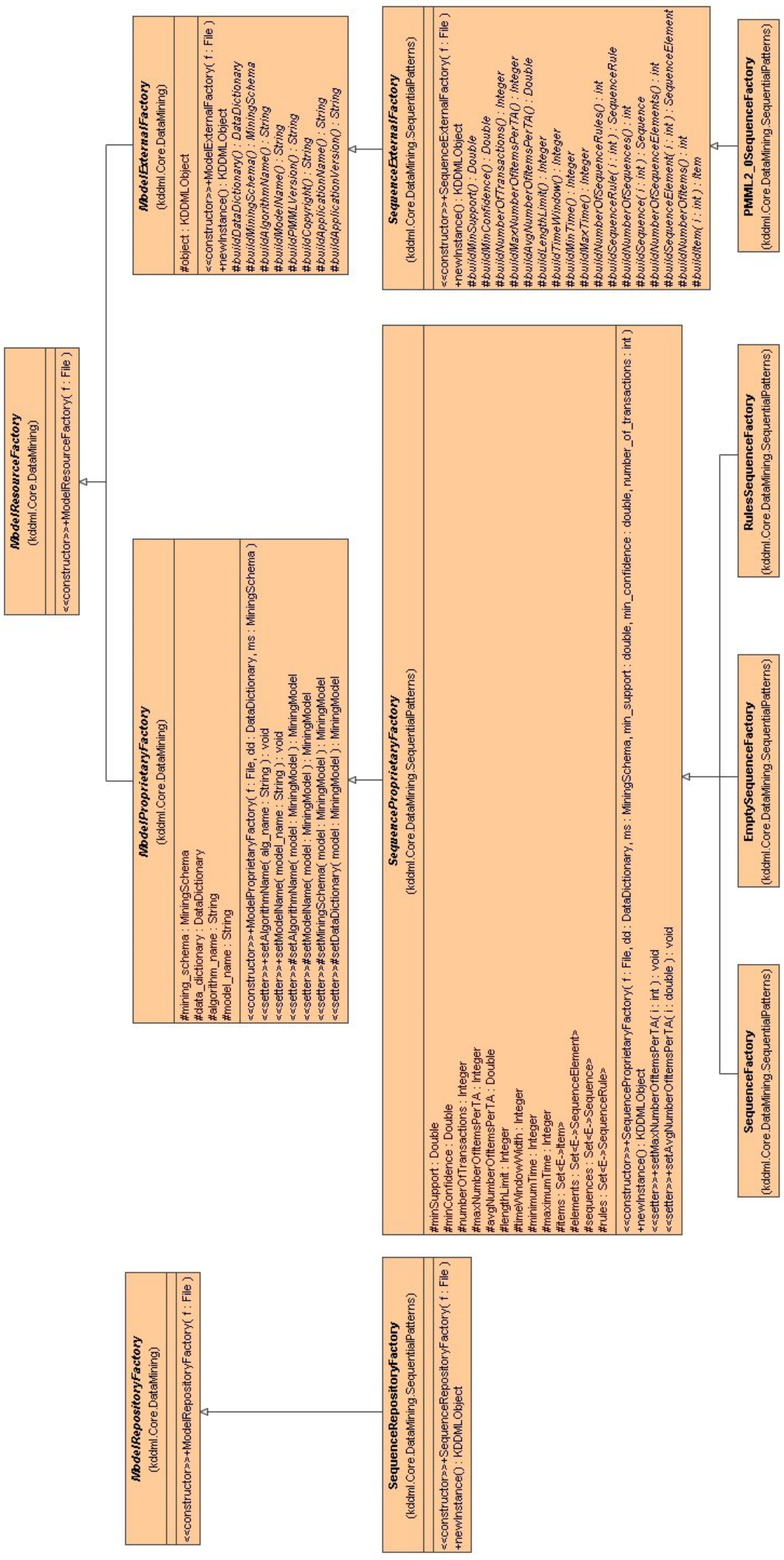

ModelResourceFactory
(kddml.Core.DataMining)
<<constructor>>+ModelResourceFactory(f : File)
ModelExternalFactory
(kddml.Core.DataMining)
#object : KDDMLObject
<<constructor>>+ModelExternalFactory(f : File)
+newInstance() : KDDMLObject
#buildDataDictionary() : DataDictionary
#buildMiningSchema() : MiningSchema
#buildAlgorithmName() : String
#buildModelName() : String
#buildPMMLVersion() : String
#buildCopyright() : String
#buildApplicationName() : String
#buildApplicationVersion() : String
SequenceExternalFactory
(kddml.Core.DataMining.SequentialPatterns)
<<constructor>>+SequenceExternalFactory(f : File)
+newInstance() : KDDMLObject
#buildMinSupport() : Double
#buildMinConfidence() : Double
#buildNumberOfTransactions() : Integer
#buildMaxNumberOfItemsPerTA() : Integer
#buildAvgNumberOfItemsPerTA() : Double
#buildLengthLimit() : Integer
#buildTimeWindow() : Integer
#buildMinTime() : Integer
#buildMaxTime() : Integer
#buildNumberOfSequenceRules() : int
#buildSequenceRule(i : int) : SequenceRule
#buildNumberOfSequences() : int
#buildSequence(i : int) : Sequence
#buildNumberOfSequenceElements() : int
#buildSequenceElement(i : int) : SequenceElement
#buildNumberOfItems() : int
#buildItem(i : int) : Item
PMML2_0SequenceFactory
(kddml.Core.DataMining.SequentialPatterns)
ModelProprietaryFactory
(kddml.Core.DataMining)
#mining_schema : MiningSchema
#data_dictionary : DataDictionary
#algorithm_name : String
#model_name : String
<<constructor>>+ModelProprietaryFactory(f : File, dd : DataDictionary, ms : MiningSchema)
<<setter>>+setAlgorithmName(alg_name : String) : void
<<setter>>+setModelName(model_name : String) : void
<<setter>>#setAlgorithmName(model : MiningModel) : MiningModel
<<setter>>#setModelName(model : MiningModel) : MiningModel
<<setter>>#setMiningSchema(model : MiningModel) : MiningModel
<<setter>>#setDataDictionary(model : MiningModel) : MiningModel
SequenceProprietaryFactory
(kddml.Core.DataMining.SequentialPatterns)
#minSupport : Double
#minConfidence : Double
#numberOfTransactions : Integer
#maxNumberOfItemsPerTA : Integer
#avgNumberOfItemsPerTA : Double
#lengthLimit : Integer
#timeWindowWidth : Integer
#minimumTime : Integer
#maximumTime : Integer
#items : Set<E->Item>
#elements : Set<E->SequenceElement>
#sequences : Set<E->Sequence>
#rules : Set<E->SequenceRule>
<<constructor>>+SequenceProprietaryFactory(f : File, dd : DataDictionary, ms : MiningSchema, min_support : double, min_confidence : double, number_of_transactions : int)
+newInstance() : KDDMLObject
<<setter>>+setMaxNumberOfItemsPerTA(i : int) : void
<<setter>>+setAvgNumberOfItemsPerTA(i : double) : void
RulesSequenceFactory
(kddml.Core.DataMining.SequentialPatterns)
EmptySequenceFactory
(kddml.Core.DataMining.SequentialPatterns)
SequenceFactory
(kddml.Core.DataMining.SequentialPatterns)
ModelRepositoryFactory
(kddml.Core.DataMining)
<<constructor>>+ModelRepositoryFactory(f : File)
SequenceRepositoryFactory
(kddml.Core.DataMining.SequentialPatterns)
<<constructor>>+SequenceRepositoryFactory(f : File)
+newInstance() : KDDMLObject

Figure A.27: Package kddml.Core.DataMining.SequentialPatterns - Factory

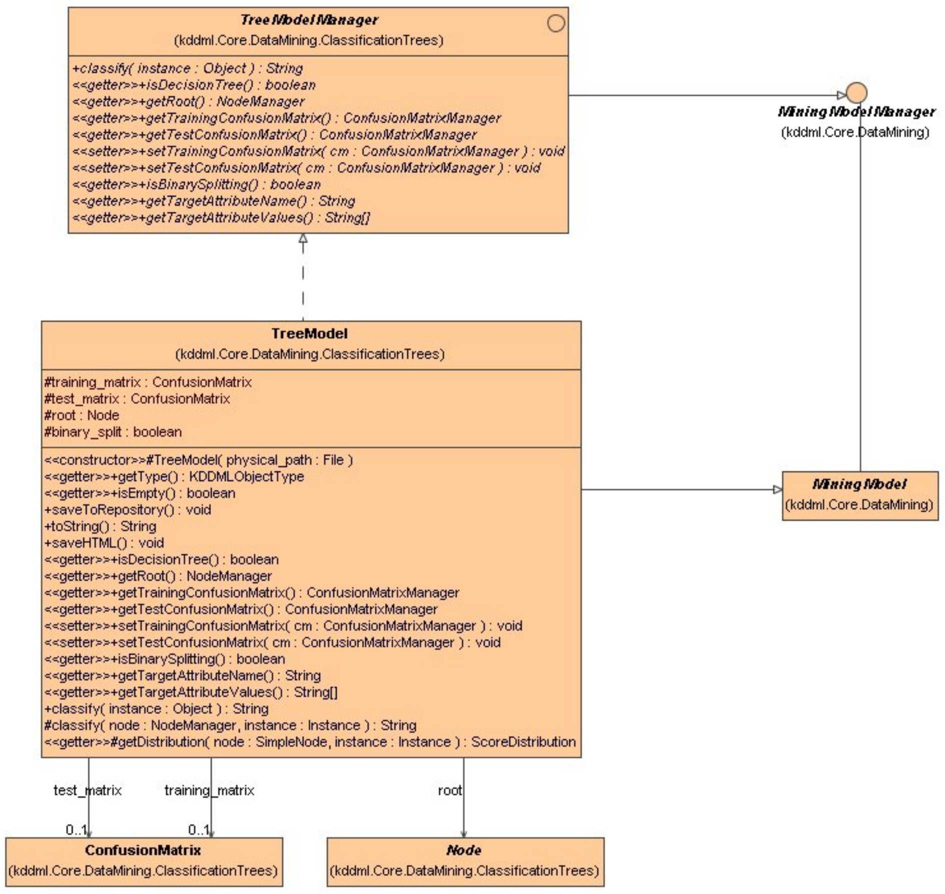

TreeModelManager
(kddml.Core.DataMining.ClassificationTrees)
+classify(instance : Object) : String
<<getter>>+isDecisionTree() : boolean
<<getter>>+getRoot() : NodeManager
<<getter>>+getTrainingConfusionMatrix() : ConfusionMatrixManager
<<getter>>+getTestConfusionMatrix() : ConfusionMatrixManager
<<setter>>+setTrainingConfusionMatrix(cm : ConfusionMatrixManager) : void
<<setter>>+setTestConfusionMatrix(cm : ConfusionMatrixManager) : void
<<getter>>+isBinarySplitting() : boolean
<<getter>>+getTargetAttributeName() : String
<<getter>>+getTargetAttributeValues() : String[]
MiningModelManager
(kddml.Core.DataMining)
TreeModel
(kddml.Core.DataMining.ClassificationTrees)
#training_matrix : ConfusionMatrix
#test_matrix : ConfusionMatrix
#root : Node
#binary_split : boolean
<<constructor>>#TreeModel(physical_path : File)
<<getter>>+getType() : KDDMLObjectType
<<getter>>+isEmpty() : boolean
+saveToRepository() : void
+toString() : String
+saveHTML() : void
<<getter>>+isDecisionTree() : boolean
<<getter>>+getRoot() : NodeManager
<<getter>>+getTrainingConfusionMatrix() : ConfusionMatrixManager
<<getter>>+getTestConfusionMatrix() : ConfusionMatrixManager
<<setter>>+setTrainingConfusionMatrix(cm : ConfusionMatrixManager) : void
<<setter>>+setTestConfusionMatrix(cm : ConfusionMatrixManager) : void
<<getter>>+isBinarySplitting() : boolean
<<getter>>+getTargetAttributeName() : String
<<getter>>+getTargetAttributeValues() : String[]
+classify(instance : Object) : String
#classify(node : NodeManager, instance : Instance) : String
<<getter>>#getDistribution(node : SimpleNode, instance : Instance) : ScoreDistribution
MiningModel
(kddml.Core.DataMining)
test_matrix
training_matrix
root
0..1
0..1
ConfusionMatrix
(kddml.Core.DataMining.ClassificationTrees)
Node
(kddml.Core.DataMining.ClassificationTrees)

Figure A.28: Package kddml.Core.DataMining.ClassificationTrees - Manager

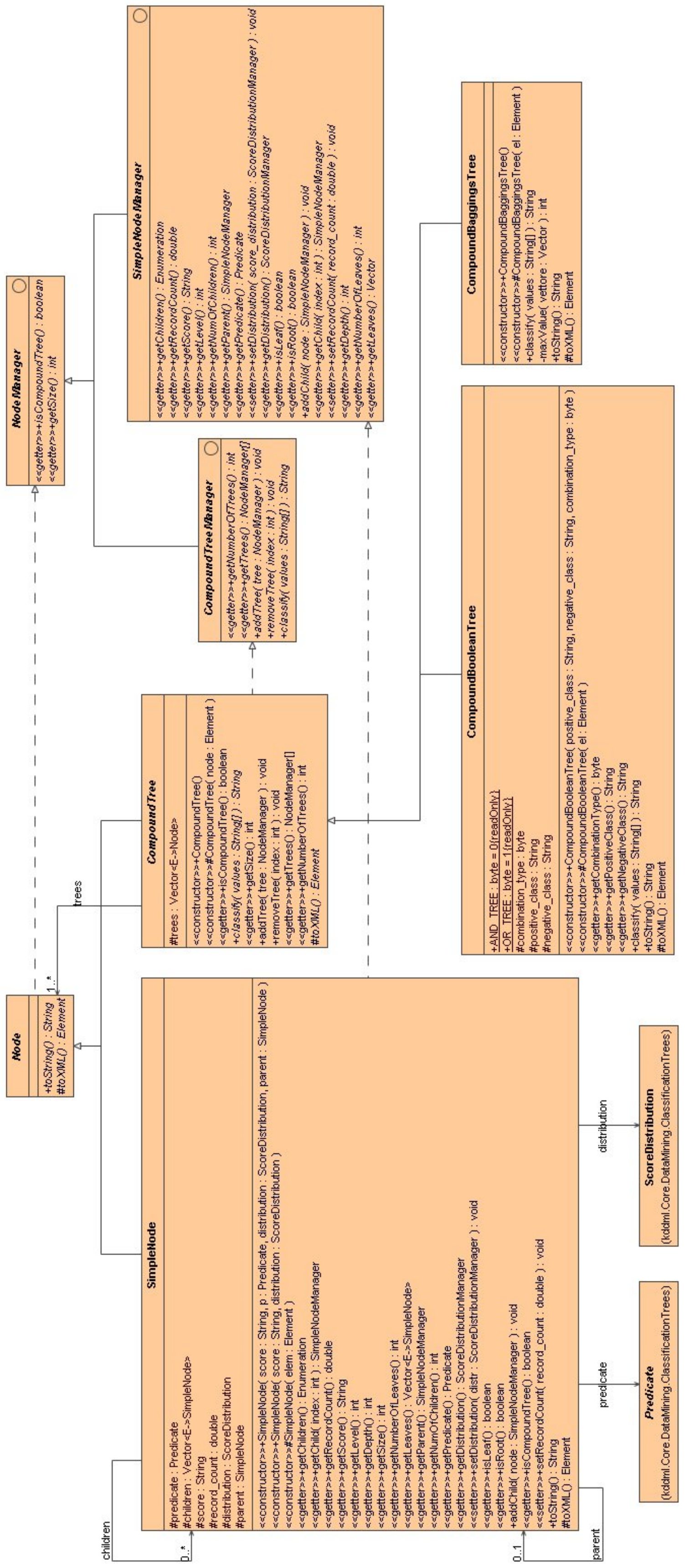

Node
+toString() : String
#toXML() : Element
NodeManager
<<getter>>+isCompoundTree() : boolean
<<getter>>+getSize() : int
1..*
trees
children
0..*
SimpleNode
#predicate : Predicate
#children : Vector<E->SimpleNode>
#score : String
#record_count : double
#distribution : ScoreDistribution
#parent : SimpleNode
<<constructor>>+SimpleNode(score : String, p : Predicate, distribution : ScoreDistribution, parent : SimpleNode)
<<constructor>>+SimpleNode(score : String, distribution : ScoreDistribution)
<<constructor>>#SimpleNode(elem : Element)
<<getter>>+getChildren() : Enumeration
<<getter>>+getChild(index : int) : SimpleNodeManager
<<getter>>+getRecordCount() : double
<<getter>>+getScore() : String
<<getter>>+getLevel() : int
<<getter>>+getDepth() : int
<<getter>>+getSize() : int
<<getter>>+getNumberOfLeaves() : int
<<getter>>+getLeaves() : Vector<E->SimpleNode>
<<getter>>+getParent() : SimpleNodeManager
<<getter>>+getNumOfChildren() : int
<<getter>>+getPredicate() : Predicate
<<getter>>+getDistribution() : ScoreDistributionManager
<<setter>>+setDistribution(distr : ScoreDistributionManager) : void
<<getter>>+isLeaf() : boolean
<<getter>>+isRoot() : boolean
+addChild(node : SimpleNodeManager) : void
<<getter>>+isCompoundTree() : boolean
<<setter>>+setRecordCount(record_count : double) : void
+toString() : String
#toXML() : Element
0..1
parent
predicate
distribution
Predicate
(kddml.Core.DataMining.ClassificationTrees)
ScoreDistribution
(kddml.Core.DataMining.ClassificationTrees)
CompoundTree
#trees : Vector<E->Node>
<<constructor>>+CompoundTree()
<<constructor>>#CompoundTree(node : Element)
<<getter>>+isCompoundTree() : boolean
+classify(values : String[]) : String
<<getter>>+getSize() : int
+addTree(tree : NodeManager) : void
+removeTree(index : int) : void
<<getter>>+getTrees() : NodeManager[]
<<getter>>+getNumberOfTrees() : int
#toXML() : Element
CompoundTreeManager
<<getter>>+getNumberOfTrees() : int
<<getter>>+getTrees() : NodeManager[]
+addTree(tree : NodeManager) : void
+removeTree(index : int) : void
+classify(values : String[]) : String
SimpleNodeManager
<<getter>>+getChildren() : Enumeration
<<getter>>+getRecordCount() : double
<<getter>>+getScore() : String
<<getter>>+getLevel() : int
<<getter>>+getNumOfChildren() : int
<<getter>>+getParent() : SimpleNodeManager
<<getter>>+getPredicate() : Predicate
<<setter>>+setDistribution(score_distribution : ScoreDistributionManager) : void
<<getter>>+getDistribution() : ScoreDistributionManager
<<getter>>+isLeaf() : boolean
<<getter>>+isRoot() : boolean
+addChild(node : SimpleNodeManager) : void
<<getter>>+getChild(index : int) : SimpleNodeManager
<<setter>>+setRecordCount(record_count : double) : void
<<getter>>+getDepth() : int
<<getter>>+getNumberOfLeaves() : int
<<getter>>+getLeaves() : Vector
CompoundBooleanTree
+AND_TREE : byte = 0{readOnly}
+OR_TREE : byte = 1{readOnly}
#combination_type : byte
#positive_class : String
#negative_class : String
<<constructor>>+CompoundBooleanTree(positive_class : String, negative_class : String, combination_type : byte)
<<constructor>>#CompoundBooleanTree(el : Element)
<<getter>>+getCombinationType() : byte
<<getter>>+getPositiveClass() : String
<<getter>>+getNegativeClass() : String
+classify(values : String[]) : String
+toString() : String
#toXML() : Element
CompoundBaggingsTree
<<constructor>>+CompoundBaggingsTree()
<<constructor>>#CompoundBaggingsTree(el : Element)
+classify(values : String[]) : String
-maxValue(vettore : Vector) : int
+toString() : String
#toXML() : Element

Figure A.29: Package kddml.Core.DataMining.ClassificationTrees - Nodes

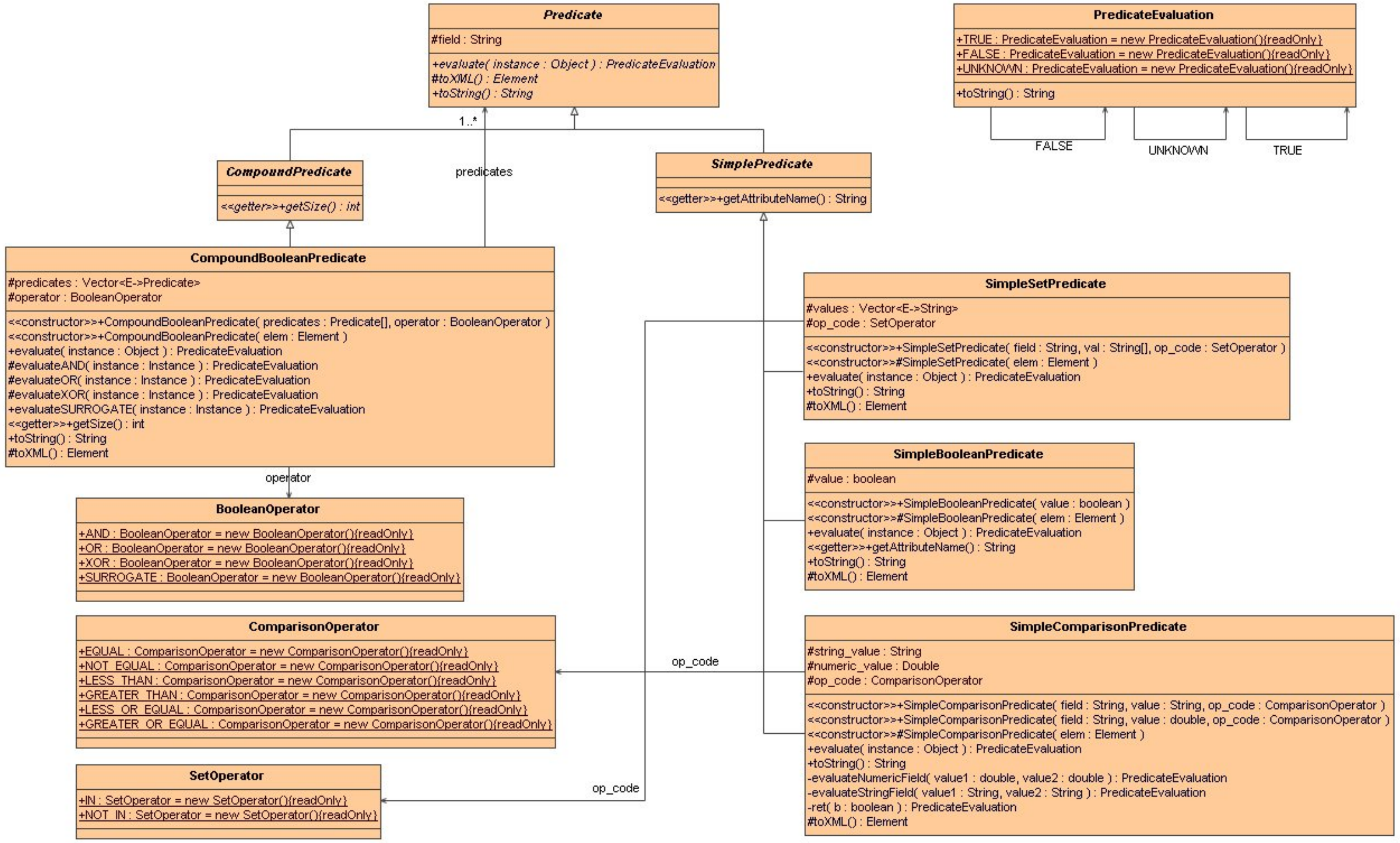

PredicateEvaluation
+TRUE : PredicateEvaluation = new PredicateEvaluation(){readOnly}
+FALSE : PredicateEvaluation = new PredicateEvaluation(){readOnly}
+UNKNOWN : PredicateEvaluation = new PredicateEvaluation(){readOnly}
+toString() : String
FALSE
UNKNOWN
TRUE
Predicate
#field : String
+evaluate(instance : Object) : PredicateEvaluation
#toXML() : Element
+toString() : String
1..*
predicates
CompoundPredicate
<<getter>>+getSize() : int
SimplePredicate
<<getter>>+getAttributeName() : String
CompoundBooleanPredicate
#predicates : Vector<E->Predicate>
#operator : BooleanOperator
<<constructor>>+CompoundBooleanPredicate(predicates : Predicate[], operator : BooleanOperator)
<<constructor>>+CompoundBooleanPredicate(elem : Element)
+evaluate(instance : Object) : PredicateEvaluation
#evaluateAND(instance : Instance) : PredicateEvaluation
#evaluateOR(instance : Instance) : PredicateEvaluation
#evaluateXOR(instance : Instance) : PredicateEvaluation
+evaluateSURROGATE(instance : Instance) : PredicateEvaluation
<<getter>>+getSize() : int
+toString() : String
#toXML() : Element
operator
BooleanOperator
+AND : BooleanOperator = new BooleanOperator(){readOnly}
+OR : BooleanOperator = new BooleanOperator(){readOnly}
+XOR : BooleanOperator = new BooleanOperator(){readOnly}
+SURROGATE : BooleanOperator = new BooleanOperator(){readOnly}
ComparisonOperator
+EQUAL : ComparisonOperator = new ComparisonOperator(){readOnly}
+NOT_EQUAL : ComparisonOperator = new ComparisonOperator(){readOnly}
+LESS_THAN : ComparisonOperator = new ComparisonOperator(){readOnly}
+GREATER_THAN : ComparisonOperator = new ComparisonOperator(){readOnly}
+LESS_OR_EQUAL : ComparisonOperator = new ComparisonOperator(){readOnly}
+GREATER_OR_EQUAL : ComparisonOperator = new ComparisonOperator(){readOnly}
SetOperator
+IN : SetOperator = new SetOperator(){readOnly}
+NOT_IN : SetOperator = new SetOperator(){readOnly}
op_code
op_code
SimpleSetPredicate
#values : Vector<E->String>
#op_code : SetOperator
<<constructor>>+SimpleSetPredicate(field : String, val : String[], op_code : SetOperator)
<<constructor>>#SimpleSetPredicate(elem : Element)
+evaluate(instance : Object) : PredicateEvaluation
+toString() : String
#toXML() : Element
SimpleBooleanPredicate
#value : boolean
<<constructor>>+SimpleBooleanPredicate(value : boolean)
<<constructor>>#SimpleBooleanPredicate(elem : Element)
+evaluate(instance : Object) : PredicateEvaluation
<<getter>>+getAttributeName() : String
+toString() : String
#toXML() : Element
SimpleComparisonPredicate
#string_value : String
#numeric_value : Double
#op_code : ComparisonOperator
<<constructor>>+SimpleComparisonPredicate(field : String, value : String, op_code : ComparisonOperator)
<<constructor>>+SimpleComparisonPredicate(field : String, value : double, op_code : ComparisonOperator)
<<constructor>>#SimpleComparisonPredicate(elem : Element)
+evaluate(instance : Object) : PredicateEvaluation
+toString() : String
-evaluateNumericField(value1 : double, value2 : double) : PredicateEvaluation
-evaluateStringField(value1 : String, value2 : String) : PredicateEvaluation
-ret(b : boolean) : PredicateEvaluation
#toXML() : Element

Figure A.30: Package kddml.Core.DataMining.ClassificationTrees - Predicates

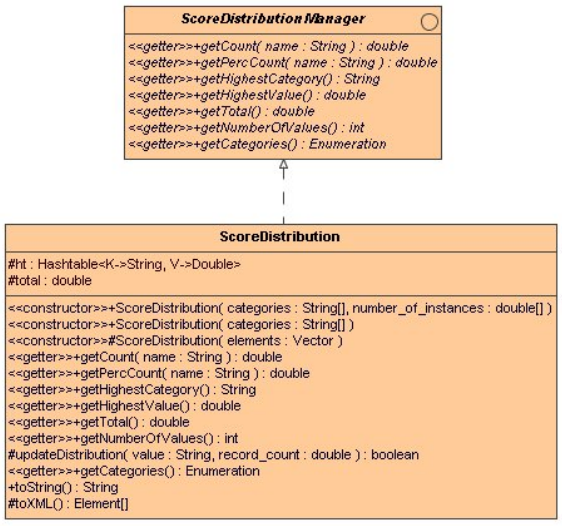

ScoreDistributionManager
<<getter>>+getCount(name : String) : double
<<getter>>+getPercCount(name : String) : double
<<getter>>+getHighestCategory() : String
<<getter>>+getHighestValue() : double
<<getter>>+getTotal() : double
<<getter>>+getNumberOfValues() : int
<<getter>>+getCategories() : Enumeration
ScoreDistribution
#ht : Hashtable<K->String, V->Double>
#total : double
<<constructor>>+ScoreDistribution(categories : String[], number_of_instances : double[])
<<constructor>>+ScoreDistribution(categories : String[])
<<constructor>>#ScoreDistribution(elements : Vector)
<<getter>>+getCount(name : String) : double
<<getter>>+getPercCount(name : String) : double
<<getter>>+getHighestCategory() : String
<<getter>>+getHighestValue() : double
<<getter>>+getTotal() : double
<<getter>>+getNumberOfValues() : int
#updateDistribution(value : String, record_count : double) : boolean
<<getter>>+getCategories() : Enumeration
+toString() : String
#toXML() : Element[]

Figure A.31: Package kddml.Core.DataMining.ClassificationTrees - ScoreDistribution

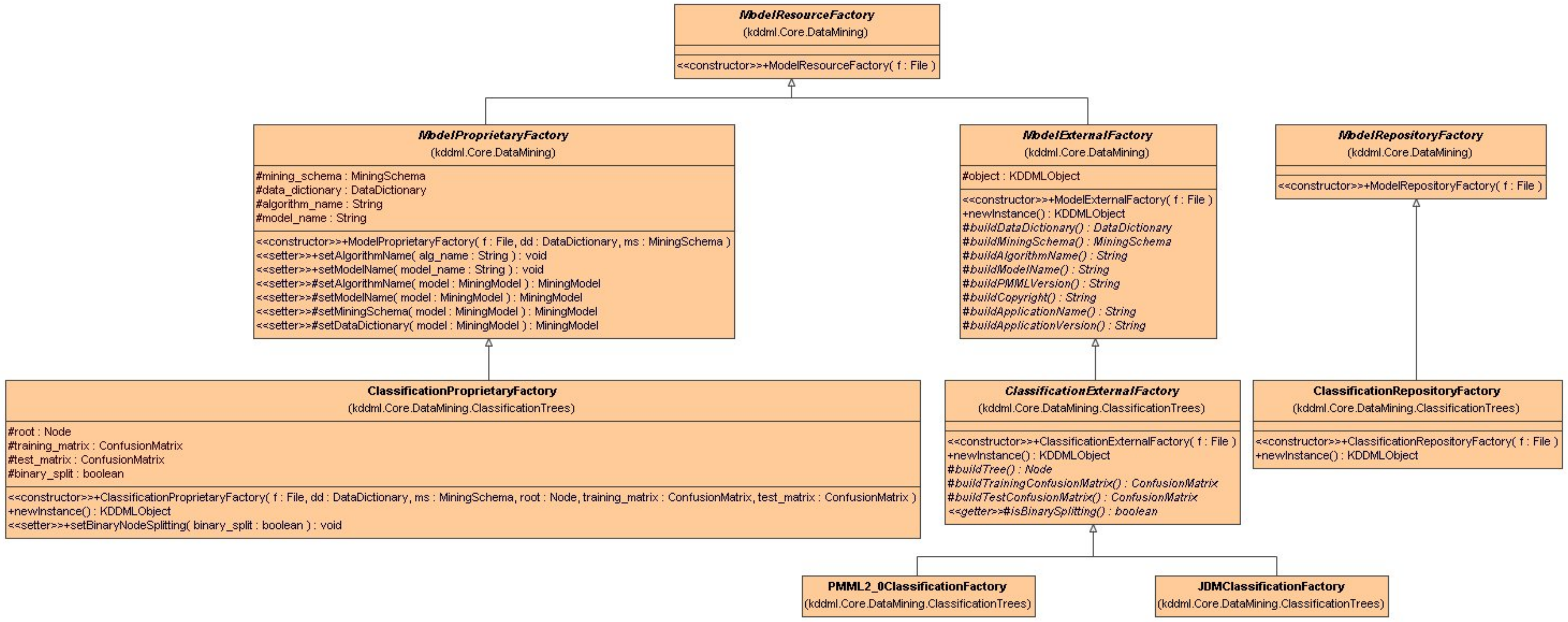

ModelResourceFactory
(kddml.Core.DataMining)
<<constructor>>+ModelResourceFactory(f : File)
ModelProprietaryFactory
(kddml.Core.DataMining)
#mining_schema : MiningSchema
#data_dictionary : DataDictionary
#algorithm_name : String
#model_name : String
<<constructor>>+ModelProprietaryFactory(f : File, dd : DataDictionary, ms : MiningSchema)
<<setter>>+setAlgorithmName(alg_name : String) : void
<<setter>>+setModelName(model_name : String) : void
<<setter>>#setAlgorithmName(model : MiningModel) : MiningModel
<<setter>>#setModelName(model : MiningModel) : MiningModel
<<setter>>#setMiningSchema(model : MiningModel) : MiningModel
<<setter>>#setDataDictionary(model : MiningModel) : MiningModel
ModelExternalFactory
(kddml.Core.DataMining)
#object : KDDMLObject
<<constructor>>+ModelExternalFactory(f : File)
+newInstance() : KDDMLObject
#buildDataDictionary() : DataDictionary
#buildMiningSchema() : MiningSchema
#buildAlgorithmName() : String
#buildModelName() : String
#buildPMMLVersion() : String
#buildCopyright() : String
#buildApplicationName() : String
#buildApplicationVersion() : String
ModelRepositoryFactory
(kddml.Core.DataMining)
<<constructor>>+ModelRepositoryFactory(f : File)
ClassificationProprietaryFactory
(kddml.Core.DataMining.ClassificationTrees)
#root : Node
#training_matrix : ConfusionMatrix
#test_matrix : ConfusionMatrix
#binary_split : boolean
<<constructor>>+ClassificationProprietaryFactory(f : File, dd : DataDictionary, ms : MiningSchema, root : Node, training_matrix : ConfusionMatrix, test_matrix : ConfusionMatrix)
+newInstance() : KDDMLObject
<<setter>>+setBinaryNodeSplitting(binary_split : boolean) : void
ClassificationExternalFactory
(kddml.Core.DataMining.ClassificationTrees)
<<constructor>>+ClassificationExternalFactory(f : File)
+newInstance() : KDDMLObject
#buildTree() : Node
#buildTrainingConfusionMatrix() : ConfusionMatrix
#buildTestConfusionMatrix() : ConfusionMatrix
<<getter>>#isBinarySplitting() : boolean
ClassificationRepositoryFactory
(kddml.Core.DataMining.ClassificationTrees)
<<constructor>>+ClassificationRepositoryFactory(f : File)
+newInstance() : KDDMLObject
PMML2_0ClassificationFactory
(kddml.Core.DataMining.ClassificationTrees)
JDMClassificationFactory
(kddml.Core.DataMining.ClassificationTrees)

Figure A.32: Package kddml.Core.DataMining.ClassificationTrees - Factory

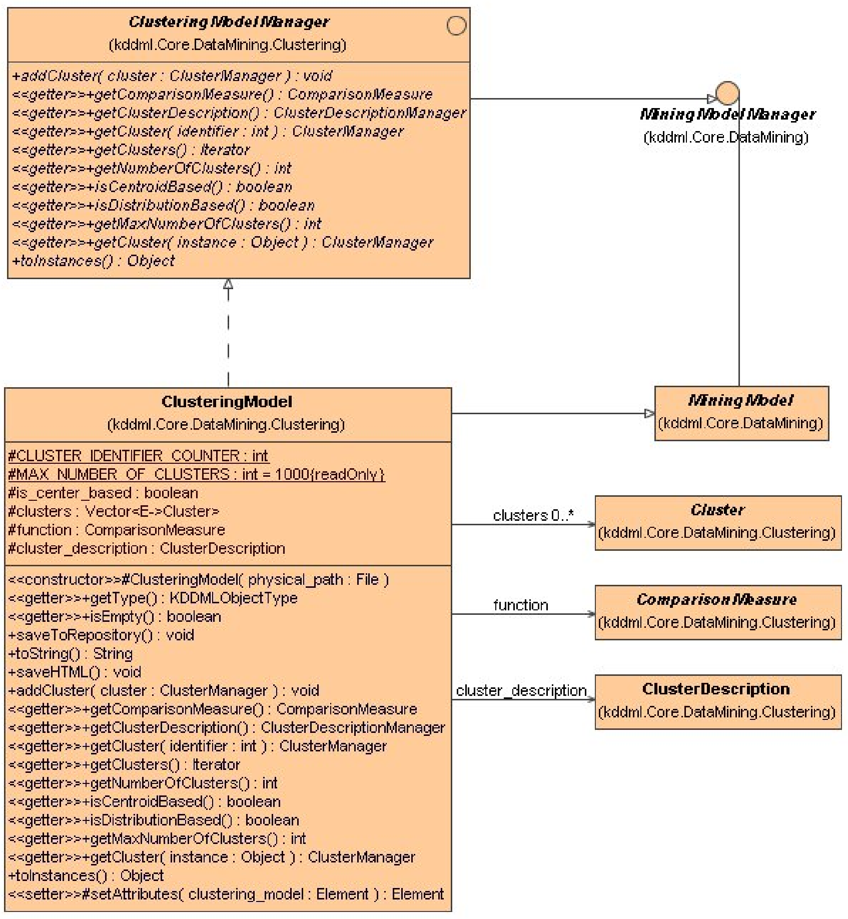

ClusteringModelManager
(kddml.Core.DataMining.Clustering)
+addCluster(cluster : ClusterManager) : void
<<getter>>+getComparisonMeasure() : ComparisonMeasure
<<getter>>+getClusterDescription() : ClusterDescriptionManager
<<getter>>+getCluster(identifier : int) : ClusterManager
<<getter>>+getClusters() : Iterator
<<getter>>+getNumberOfClusters() : int
<<getter>>+isCentroidBased() : boolean
<<getter>>+isDistributionBased() : boolean
<<getter>>+getMaxNumberOfClusters() : int
<<getter>>+getCluster(instance : Object) : ClusterManager
+toInstances() : Object
MiningModelManager
(kddml.Core.DataMining)
ClusteringModel
(kddml.Core.DataMining.Clustering)
#CLUSTER_IDENTIFIER_COUNTER : int
#MAX_NUMBER_OF_CLUSTERS : int = 1000{readOnly}
#is_center_based : boolean
#clusters : Vector<E->Cluster>
#function : ComparisonMeasure
#cluster_description : ClusterDescription
<<constructor>>#ClusteringModel(physical_path : File)
<<getter>>+getType() : KDDMLObjectType
<<getter>>+isEmpty() : boolean
+saveToRepository() : void
+toString() : String
+saveHTML() : void
+addCluster(cluster : ClusterManager) : void
<<getter>>+getComparisonMeasure() : ComparisonMeasure
<<getter>>+getClusterDescription() : ClusterDescriptionManager
<<getter>>+getCluster(identifier : int) : ClusterManager
<<getter>>+getClusters() : Iterator
<<getter>>+getNumberOfClusters() : int
<<getter>>+isCentroidBased() : boolean
<<getter>>+isDistributionBased() : boolean
<<getter>>+getMaxNumberOfClusters() : int
<<getter>>+getCluster(instance : Object) : ClusterManager
+toInstances() : Object
<<setter>>#setAttributes(clustering_model : Element) : Element
MiningModel
(kddml.Core.DataMining)
clusters 0..*
Cluster
(kddml.Core.DataMining.Clustering)
function
ComparisonMeasure
(kddml.Core.DataMining.Clustering)
cluster_description
ClusterDescription
(kddml.Core.DataMining.Clustering)

Figure A.33: Package kddml.Core.DataMining.Clustering - Manager

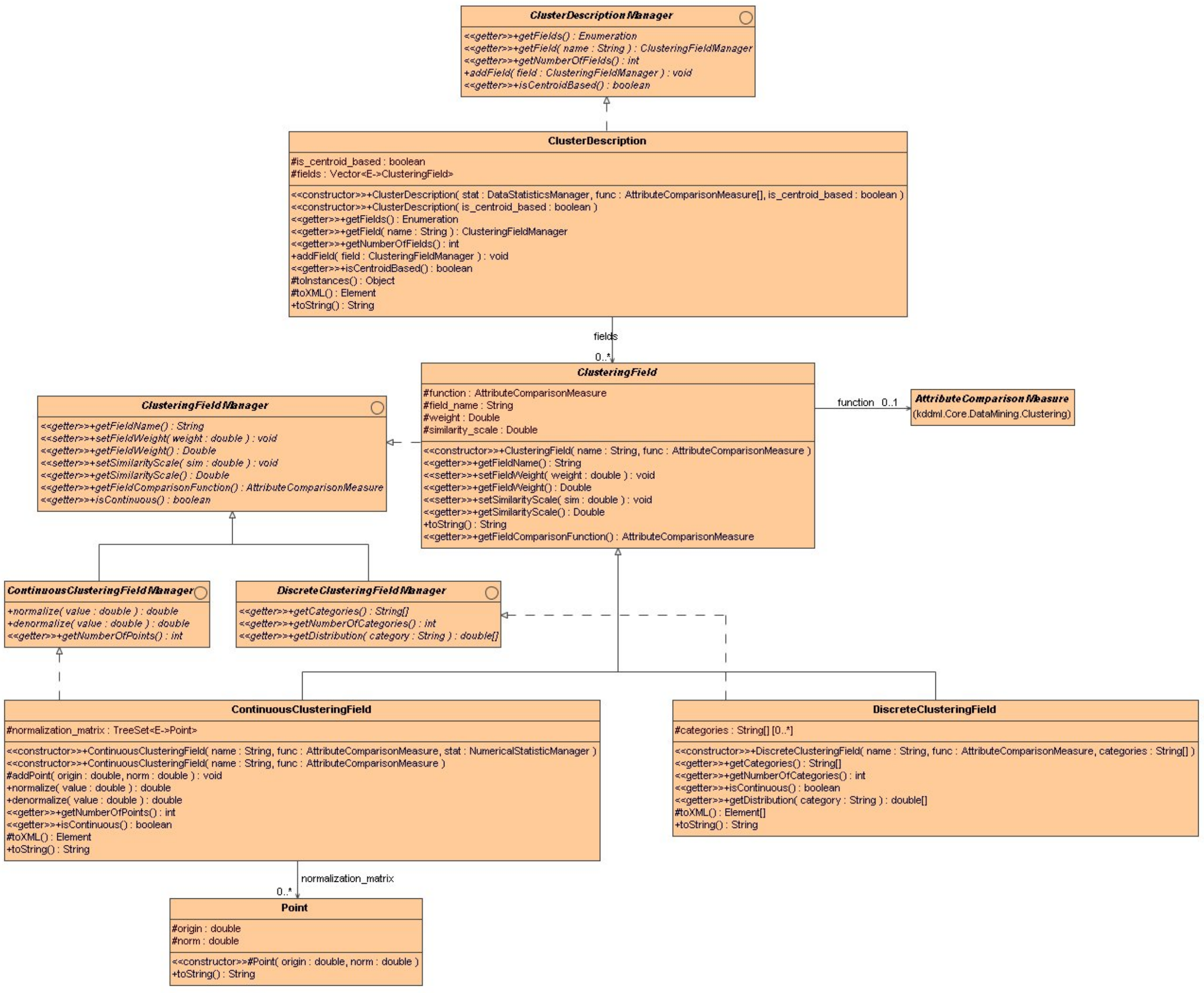

ClusterDescriptionManager
<<getter>>+getFields() : Enumeration
<<getter>>+getField(name : String) : ClusteringFieldManager
<<getter>>+getNumberOfFields() : int
+addField(field : ClusteringFieldManager) : void
<<getter>>+isCentroidBased() : boolean
ClusterDescription
#is_centroid_based : boolean
#fields : Vector<E->ClusteringField>
<<constructor>>+ClusterDescription(stat : DataStatisticsManager, func : AttributeComparisonMeasure[], is_centroid_based : boolean)
<<constructor>>+ClusterDescription(is_centroid_based : boolean)
<<getter>>+getFields() : Enumeration
<<getter>>+getField(name : String) : ClusteringFieldManager
<<getter>>+getNumberOfFields() : int
+addField(field : ClusteringFieldManager) : void
<<getter>>+isCentroidBased() : boolean
#toInstances() : Object
#toXML() : Element
+toString() : String
fields
0..*
ClusteringField
#function : AttributeComparisonMeasure
#field_name : String
#weight : Double
#similarity_scale : Double
<<constructor>>+ClusteringField(name : String, func : AttributeComparisonMeasure)
<<getter>>+getFieldName() : String
<<setter>>+setFieldWeight(weight : double) : void
<<getter>>+getFieldWeight() : Double
<<setter>>+setSimilarityScale(sim : double) : void
<<getter>>+getSimilarityScale() : Double
+toString() : String
<<getter>>+getFieldComparisonFunction() : AttributeComparisonMeasure
function 0..1
AttributeComparisonMeasure
(kddml.Core.DataMining.Clustering)
ClusteringFieldManager
<<getter>>+getFieldName() : String
<<setter>>+setFieldWeight(weight : double) : void
<<getter>>+getFieldWeight() : Double
<<setter>>+setSimilarityScale(sim : double) : void
<<getter>>+getSimilarityScale() : Double
<<getter>>+getFieldComparisonFunction() : AttributeComparisonMeasure
<<getter>>+isContinuous() : boolean
ContinuousClusteringFieldManager
+normalize(value : double) : double
+denormalize(value : double) : double
<<getter>>+getNumberOfPoints() : int
DiscreteClusteringFieldManager
<<getter>>+getCategories() : String[]
<<getter>>+getNumberOfCategories() : int
<<getter>>+getDistribution(category : String) : double[]
ContinuousClusteringField
#normalization_matrix : TreeSet<E->Point>
<<constructor>>+ContinuousClusteringField(name : String, func : AttributeComparisonMeasure, stat : NumericalStatisticManager)
<<constructor>>+ContinuousClusteringField(name : String, func : AttributeComparisonMeasure)
#addPoint(origin : double, norm : double) : void
+normalize(value : double) : double
+denormalize(value : double) : double
<<getter>>+getNumberOfPoints() : int
<<getter>>+isContinuous() : boolean
#toXML() : Element
+toString() : String
DiscreteClusteringField
#categories : String[] [0..*]
<<constructor>>+DiscreteClusteringField(name : String, func : AttributeComparisonMeasure, categories : String[])
<<getter>>+getCategories() : String[]
<<getter>>+getNumberOfCategories() : int
<<getter>>+isContinuous() : boolean
<<getter>>+getDistribution(category : String) : double[]
#toXML() : Element[]
+toString() : String
normalization_matrix
0..*
Point
#origin : double
#norm : double
<<constructor>>#Point(origin : double, norm : double)
+toString() : String

Figure A.34: Package kddml.Core.DataMining.Clustering - Cluster description

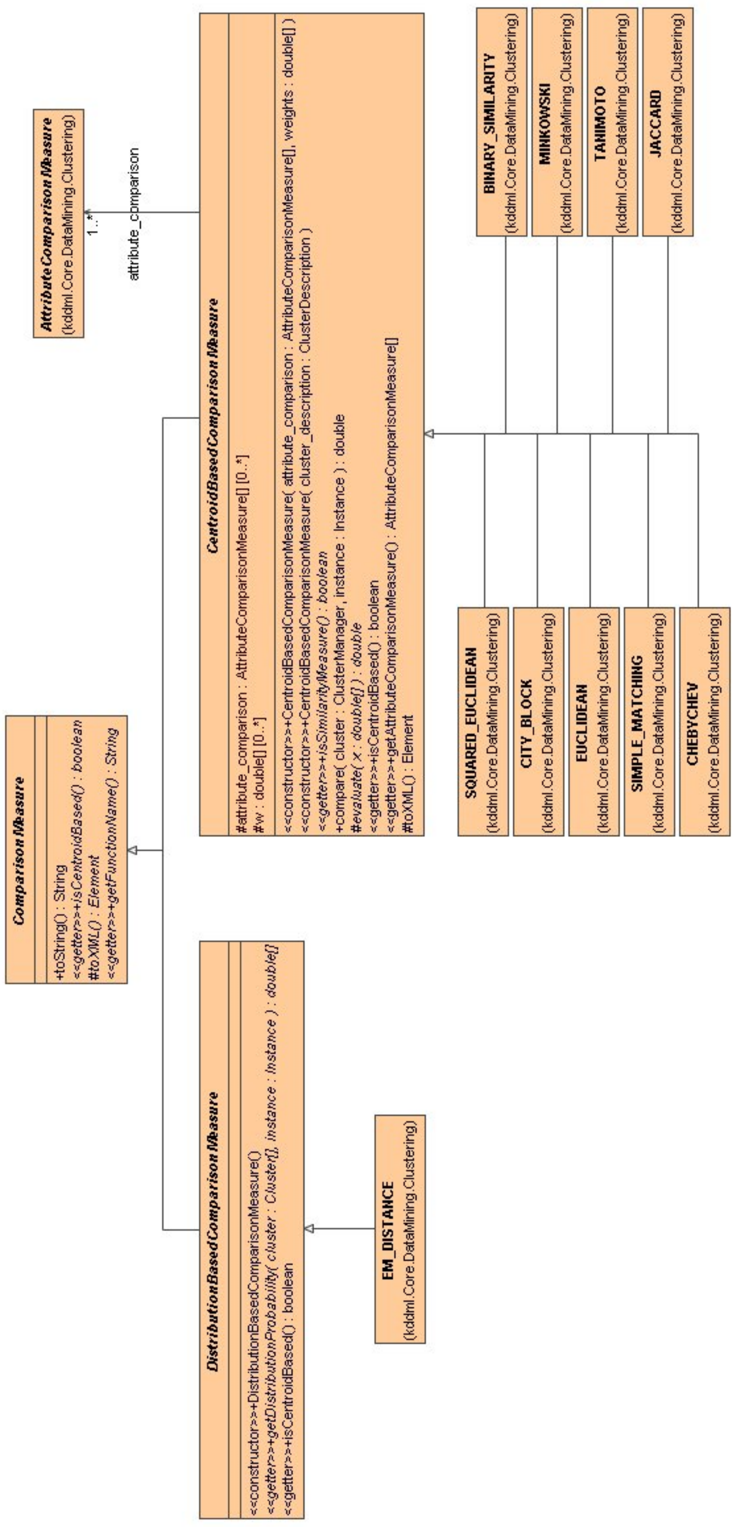

AttributeComparisonMeasure
(kddml.Core.DataMining.Clustering)
attribute_comparison
1..*
ComparisonMeasure
+toString() : String
<<getter>>+isCentroidBased() : boolean
#toXML() : Element
<<getter>>+getFunctionName() : String
CentroidBasedComparisonMeasure
#attribute_comparison : AttributeComparisonMeasure[] [0..*]
#w : double[] [0..*]
<<constructor>>+CentroidBasedComparisonMeasure(attribute_comparison : AttributeComparisonMeasure[], weights : double[])
<<constructor>>+CentroidBasedComparisonMeasure(cluster_description : ClusterDescription)
<<getter>>+isSimilarityMeasure() : boolean
+compare(cluster : ClusterManager, instance : Instance) : double
#evaluate(x : double[]) : double
<<getter>>+isCentroidBased() : boolean
<<getter>>+getAttributeComparisonMeasure() : AttributeComparisonMeasure[]
#toXML() : Element
BINARY_SIMILARITY
(kddml.Core.DataMining.Clustering)
MINKOWSKI
(kddml.Core.DataMining.Clustering)
TANIMOTO
(kddml.Core.DataMining.Clustering)
JACCARD
(kddml.Core.DataMining.Clustering)
SQUARED_EUCLIDEAN
(kddml.Core.DataMining.Clustering)
CITY_BLOCK
(kddml.Core.DataMining.Clustering)
EUCLIDEAN
(kddml.Core.DataMining.Clustering)
SIMPLE_MATCHING
(kddml.Core.DataMining.Clustering)
CHEBYCHEV
(kddml.Core.DataMining.Clustering)
DistributionBasedComparisonMeasure
<<constructor>>+DistributionBasedComparisonMeasure()
<<getter>>+getDistributionProbability(cluster : Cluster[], instance : Instance) : double[]
<<getter>>+isCentroidBased() : boolean
EM_DISTANCE
(kddml.Core.DataMining.Clustering)

Figure A.35: Package kddml.Core.DataMining.Clustering - Comparison measure

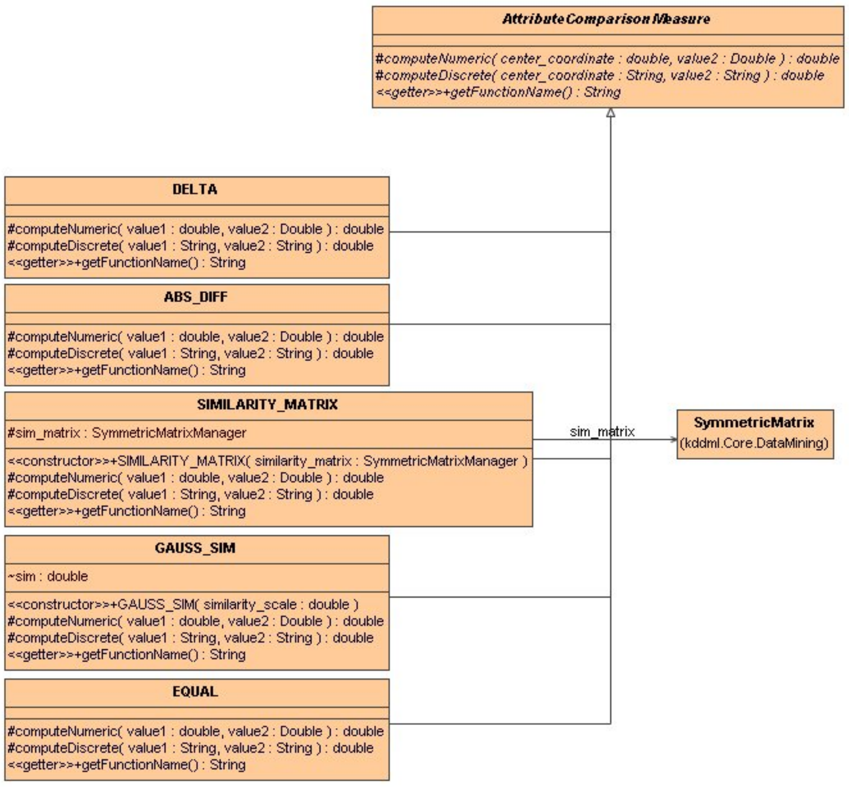

AttributeComparisonMeasure
#computeNumeric(center_coordinate : double, value2 : Double) : double
#computeDiscrete(center_coordinate : String, value2 : String) : double
<<getter>>+getFunctionName() : String
DELTA
#computeNumeric(value1 : double, value2 : Double) : double
#computeDiscrete(value1 : String, value2 : String) : double
<<getter>>+getFunctionName() : String
ABS_DIFF
#computeNumeric(value1 : double, value2 : Double) : double
#computeDiscrete(value1 : String, value2 : String) : double
<<getter>>+getFunctionName() : String
SIMILARITY_MATRIX
#sim_matrix : SymmetricMatrixManager
<<constructor>>+SIMILARITY_MATRIX(similarity_matrix : SymmetricMatrixManager)
#computeNumeric(value1 : double, value2 : Double) : double
#computeDiscrete(value1 : String, value2 : String) : double
<<getter>>+getFunctionName() : String
sim_matrix
SymmetricMatrix
(kddml.Core.DataMining)
GAUSS_SIM
~sim : double
<<constructor>>+GAUSS_SIM(similarity_scale : double)
#computeNumeric(value1 : double, value2 : Double) : double
#computeDiscrete(value1 : String, value2 : String) : double
<<getter>>+getFunctionName() : String
EQUAL
#computeNumeric(value1 : double, value2 : Double) : double
#computeDiscrete(value1 : String, value2 : String) : double
<<getter>>+getFunctionName() : String

Figure A.36: Package kddml.Core.DataMining.Clustering - Attribute comparison measure

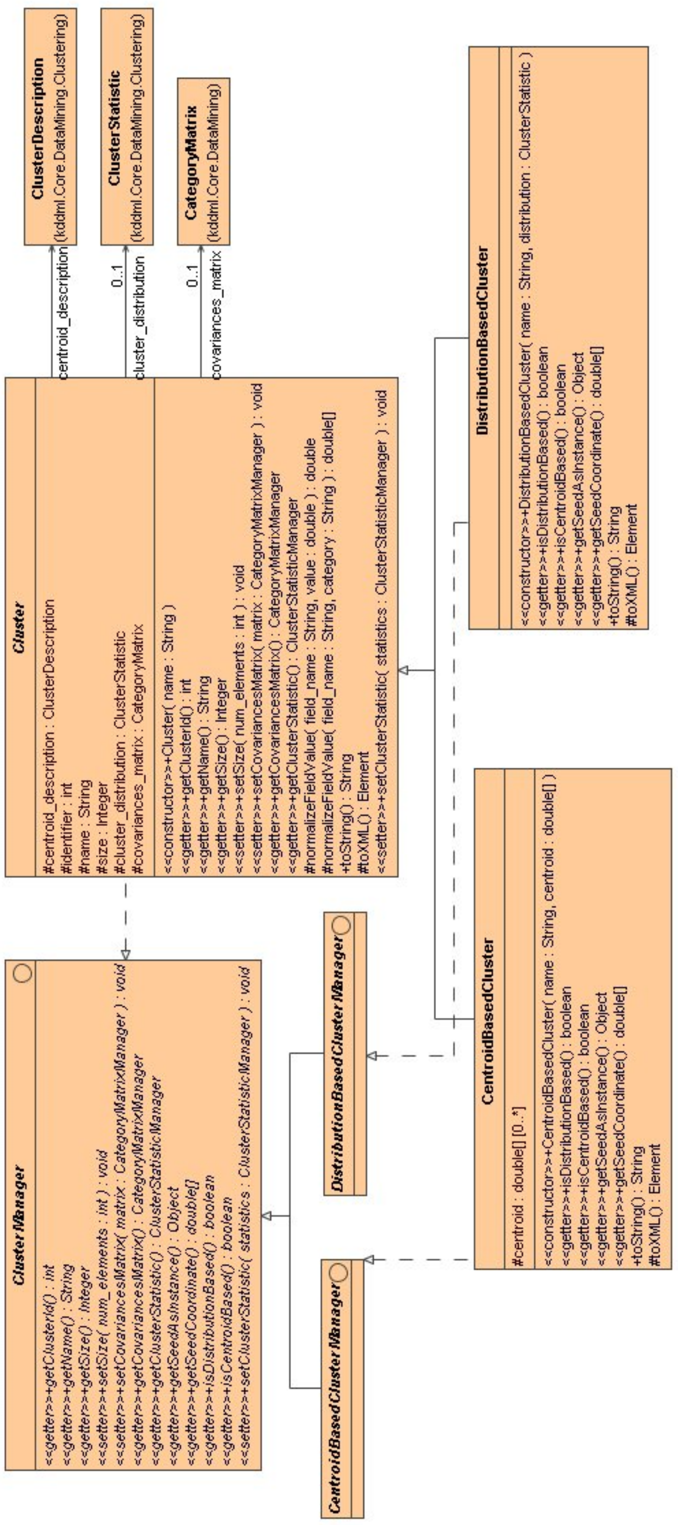

ClusterDescription
(kddml.Core.DataMining.Clustering)
ClusterStatistic
(kddml.Core.DataMining.Clustering)
CategoryMatrix
(kddml.Core.DataMining)
centroid_description
0..1
cluster_distribution
0..1
covariances_matrix
Cluster
#centroid_description : ClusterDescription
#identifier : int
#name : String
#size : Integer
#cluster_distribution : ClusterStatistic
#covariances_matrix : CategoryMatrix
<<constructor>>+Cluster(name : String)
<<getter>>+getClusterId() : int
<<getter>>+getName() : String
<<getter>>+getSize() : Integer
<<setter>>+setSize(num_elements : int) : void
<<setter>>+setCovariancesMatrix(matrix : CategoryMatrixManager) : void
<<getter>>+getCovariancesMatrix() : CategoryMatrixManager
<<getter>>+getClusterStatistic() : ClusterStatisticManager
#normalizeFieldValue(field_name : String, value : double) : double
#normalizeFieldValue(field_name : String, category : String) : double[]
+toString() : String
#toXML() : Element
<<setter>>+setClusterStatistic(statistics : ClusterStatisticManager) : void
DistributionBasedCluster
<<constructor>>+DistributionBasedCluster(name : String, distribution : ClusterStatistic)
<<getter>>+isDistributionBased() : boolean
<<getter>>+isCentroidBased() : boolean
<<getter>>+getSeedAsInstance() : Object
<<getter>>+getSeedCoordinate() : double[]
+toString() : String
#toXML() : Element
ClusterManager
<<getter>>+getClusterId() : int
<<getter>>+getName() : String
<<getter>>+getSize() : Integer
<<setter>>+setSize(num_elements : int) : void
<<setter>>+setCovariancesMatrix(matrix : CategoryMatrixManager) : void
<<getter>>+getCovariancesMatrix() : CategoryMatrixManager
<<getter>>+getClusterStatistic() : ClusterStatisticManager
<<getter>>+getSeedAsInstance() : Object
<<getter>>+getSeedCoordinate() : double[]
<<getter>>+isDistributionBased() : boolean
<<getter>>+isCentroidBased() : boolean
<<setter>>+setClusterStatistic(statistics : ClusterStatisticManager) : void
DistributionBasedClusterManager
CentroidBasedClusterManager
CentroidBasedCluster
#centroid : double[] [0..*]
<<constructor>>+CentroidBasedCluster(name : String, centroid : double[])
<<getter>>+isDistributionBased() : boolean
<<getter>>+isCentroidBased() : boolean
<<getter>>+getSeedAsInstance() : Object
<<getter>>+getSeedCoordinate() : double[]
+toString() : String
#toXML() : Element

Figure A.37: Package kddml.Core.DataMining.Clustering - Clusters

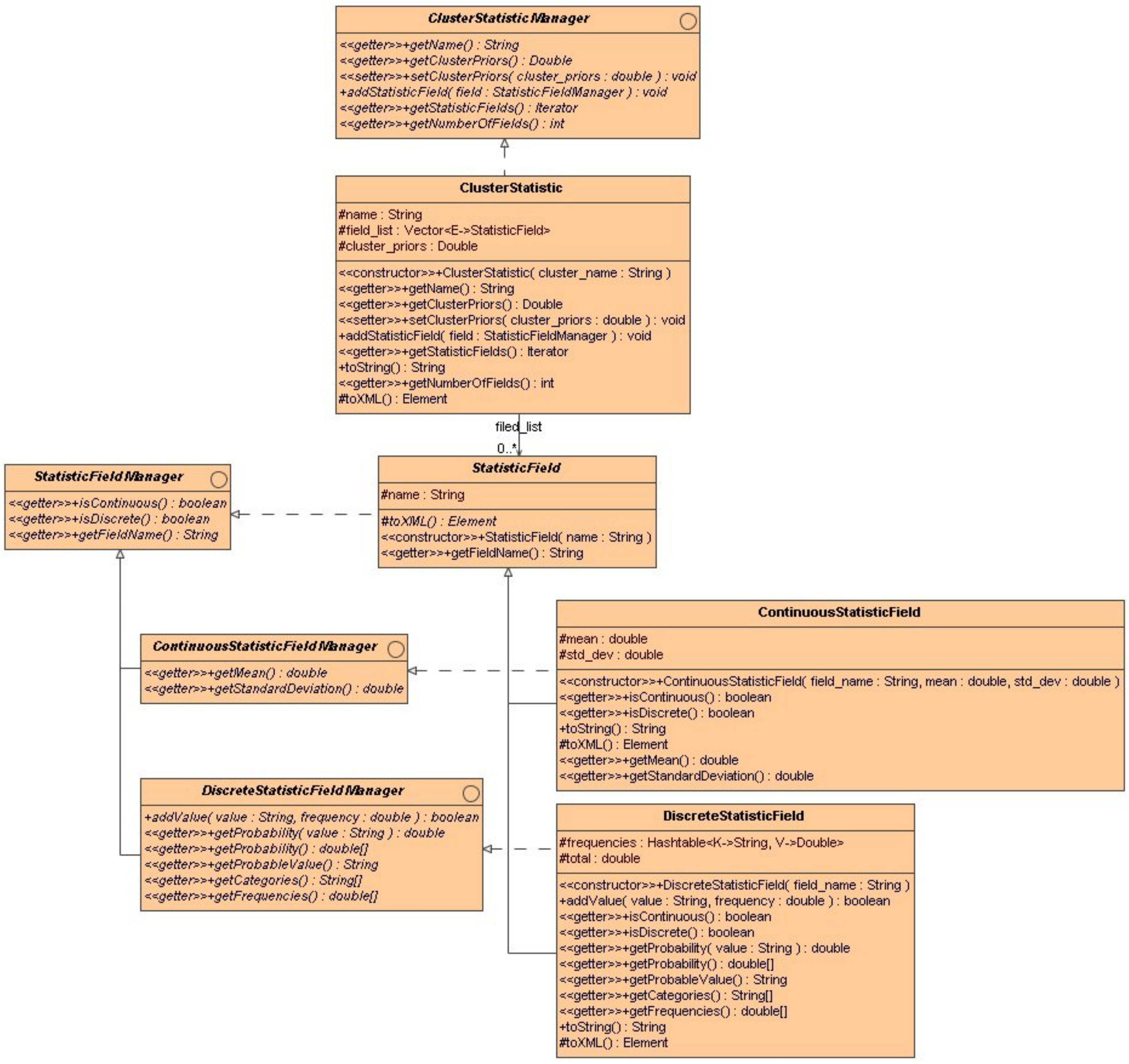

ClusterStatisticManager
<<getter>>+getName() : String
<<getter>>+getClusterPriors() : Double
<<setter>>+setClusterPriors(cluster_priors : double) : void
+addStatisticField(field : StatisticFieldManager) : void
<<getter>>+getStatisticFields() : Iterator
<<getter>>+getNumberOfFields() : int
ClusterStatistic
#name : String
#field_list : Vector<E->StatisticField>
#cluster_priors : Double
<<constructor>>+ClusterStatistic(cluster_name : String)
<<getter>>+getName() : String
<<getter>>+getClusterPriors() : Double
<<setter>>+setClusterPriors(cluster_priors : double) : void
+addStatisticField(field : StatisticFieldManager) : void
<<getter>>+getStatisticFields() : Iterator
+toString() : String
<<getter>>+getNumberOfFields() : int
#toXML() : Element
filed_list
0..*
StatisticFieldManager
<<getter>>+isContinuous() : boolean
<<getter>>+isDiscrete() : boolean
<<getter>>+getFieldName() : String
StatisticField
#name : String
#toXML() : Element
<<constructor>>+StatisticField(name : String)
<<getter>>+getFieldName() : String
ContinuousStatisticFieldManager
<<getter>>+getMean() : double
<<getter>>+getStandardDeviation() : double
ContinuousStatisticField
#mean : double
#std_dev : double
<<constructor>>+ContinuousStatisticField(field_name : String, mean : double, std_dev : double)
<<getter>>+isContinuous() : boolean
<<getter>>+isDiscrete() : boolean
+toString() : String
#toXML() : Element
<<getter>>+getMean() : double
<<getter>>+getStandardDeviation() : double
DiscreteStatisticFieldManager
+addValue(value : String, frequency : double) : boolean
<<getter>>+getProbability(value : String) : double
<<getter>>+getProbability() : double[]
<<getter>>+getProbableValue() : String
<<getter>>+getCategories() : String[]
<<getter>>+getFrequencies() : double[]
DiscreteStatisticField
#frequencies : Hashtable<K->String, V->Double>
#total : double
<<constructor>>+DiscreteStatisticField(field_name : String)
+addValue(value : String, frequency : double) : boolean
<<getter>>+isContinuous() : boolean
<<getter>>+isDiscrete() : boolean
<<getter>>+getProbability(value : String) : double
<<getter>>+getProbability() : double[]
<<getter>>+getProbableValue() : String
<<getter>>+getCategories() : String[]
<<getter>>+getFrequencies() : double[]
+toString() : String
#toXML() : Element

Figure A.38: Package kddml.Core.DataMining.Clustering - Cluster statistics

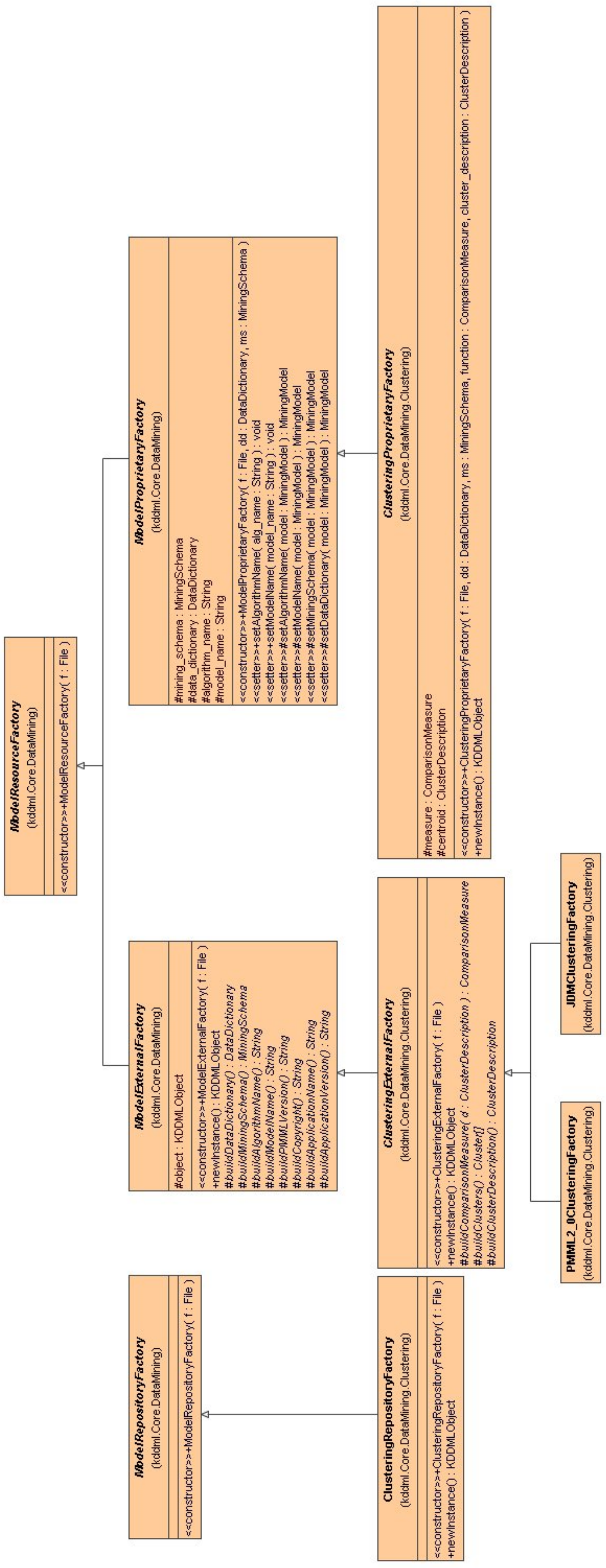

ModelResourceFactory
(kddml.Core.DataMining)
<<constructor>>+ModelResourceFactory(f : File)
ModelRepositoryFactory
(kddml.Core.DataMining)
<<constructor>>+ModelRepositoryFactory(f : File)
ModelExternalFactory
(kddml.Core.DataMining)
#object : KDDMLObject
<<constructor>>+ModelExternalFactory(f : File)
+newInstance() : KDDMLObject
#buildDataDictionary() : DataDictionary
#buildMiningSchema() : MiningSchema
#buildAlgorithmName() : String
#buildModelName() : String
#buildPMMLVersion() : String
#buildCopyright() : String
#buildApplicationName() : String
#buildApplicationVersion() : String
ModelProprietaryFactory
(kddml.Core.DataMining)
#mining_schema : MiningSchema
#data_dictionary : DataDictionary
#algorithm_name : String
#model_name : String
<<constructor>>+ModelProprietaryFactory(f : File, dd : DataDictionary, ms : MiningSchema)
<<setter>>+setAlgorithmName(alg_name : String) : void
<<setter>>+setModelName(model_name : String) : void
<<setter>>#setAlgorithmName(model : MiningModel) : MiningModel
<<setter>>#setModelName(model : MiningModel) : MiningModel
<<setter>>#setMiningSchema(model : MiningModel) : MiningModel
<<setter>>#setDataDictionary(model : MiningModel) : MiningModel
ClusteringRepositoryFactory
(kddml.Core.DataMining.Clustering)
<<constructor>>+ClusteringRepositoryFactory(f : File)
+newInstance() : KDDMLObject
ClusteringExternalFactory
(kddml.Core.DataMining.Clustering)
<<constructor>>+ClusteringExternalFactory(f : File)
+newInstance() : KDDMLObject
#buildComparisonMeasure(d : ClusterDescription) : ComparisonMeasure
#buildClusters() : Cluster[]
#buildClusterDescription() : ClusterDescription
ClusteringProprietaryFactory
(kddml.Core.DataMining.Clustering)
#measure : ComparisonMeasure
#centroid : ClusterDescription
<<constructor>>+ClusteringProprietaryFactory(f : File, dd : DataDictionary, ms : MiningSchema, function : ComparisonMeasure, cluster_description : ClusterDescription)
+newInstance() : KDDMLObject
PMML2_0ClusteringFactory
(kddml.Core.DataMining.Clustering)
JDMClusteringFactory
(kddml.Core.DataMining.Clustering)

Figure A.39: Package kddml.Core.DataMining.Clustering - Factory

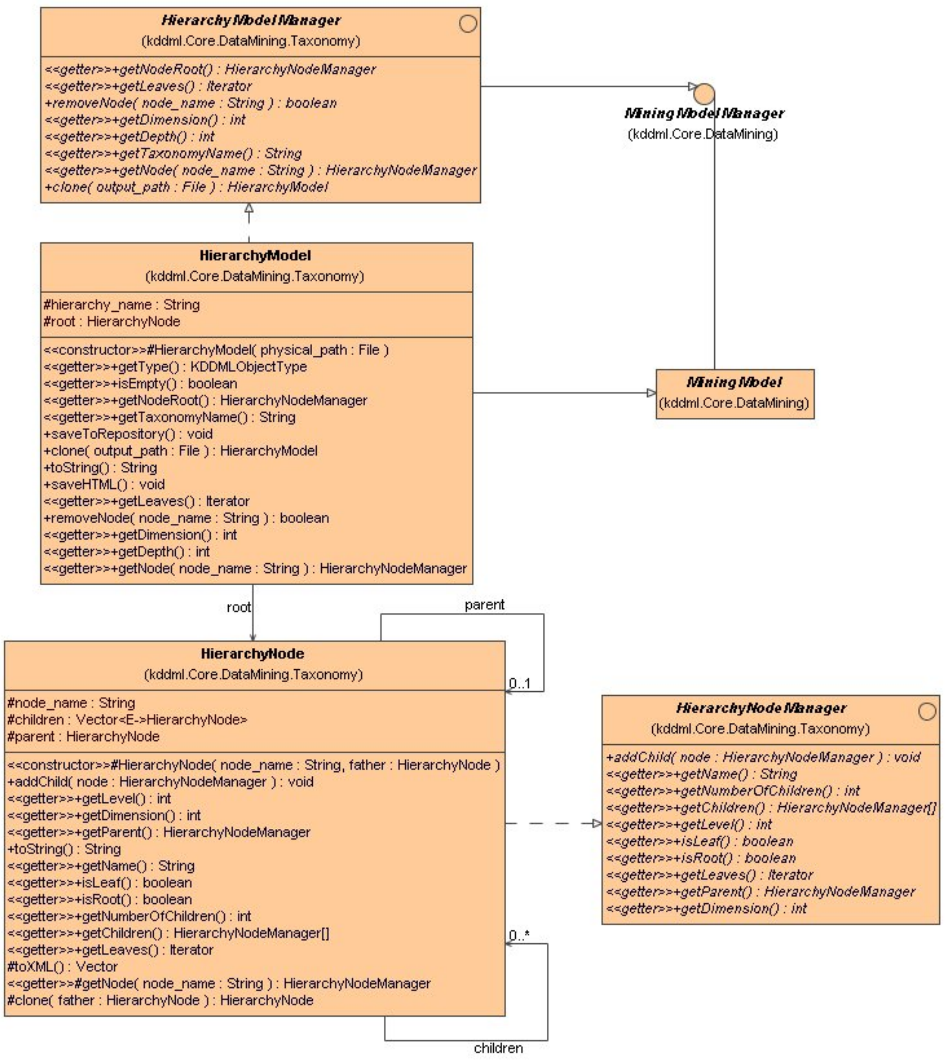

HierarchyModelManager
(kddml.Core.DataMining.Taxonomy)
<<getter>>+getNodeRoot() : HierarchyNodeManager
<<getter>>+getLeaves() : Iterator
+removeNode(node_name : String) : boolean
<<getter>>+getDimension() : int
<<getter>>+getDepth() : int
<<getter>>+getTaxonomyName() : String
<<getter>>+getNode(node_name : String) : HierarchyNodeManager
+clone(output_path : File) : HierarchyModel
MiningModelManager
(kddml.Core.DataMining)
HierarchyModel
(kddml.Core.DataMining.Taxonomy)
#hierarchy_name : String
#root : HierarchyNode
<<constructor>>#HierarchyModel(physical_path : File)
<<getter>>+getType() : KDDMLObjectType
<<getter>>+isEmpty() : boolean
<<getter>>+getNodeRoot() : HierarchyNodeManager
<<getter>>+getTaxonomyName() : String
+saveToRepository() : void
+clone(output_path : File) : HierarchyModel
+toString() : String
+saveHTML() : void
<<getter>>+getLeaves() : Iterator
+removeNode(node_name : String) : boolean
<<getter>>+getDimension() : int
<<getter>>+getDepth() : int
<<getter>>+getNode(node_name : String) : HierarchyNodeManager
MiningModel
(kddml.Core.DataMining)
root
parent
0..1
HierarchyNode
(kddml.Core.DataMining.Taxonomy)
#node_name : String
#children : Vector<E->HierarchyNode>
#parent : HierarchyNode
<<constructor>>#HierarchyNode(node_name : String, father : HierarchyNode)
+addChild(node : HierarchyNodeManager) : void
<<getter>>+getLevel() : int
<<getter>>+getDimension() : int
<<getter>>+getParent() : HierarchyNodeManager
+toString() : String
<<getter>>+getName() : String
<<getter>>+isLeaf() : boolean
<<getter>>+isRoot() : boolean
<<getter>>+getNumberOfChildren() : int
<<getter>>+getChildren() : HierarchyNodeManager[]
<<getter>>+getLeaves() : Iterator
#toXML() : Vector
<<getter>>#getNode(node_name : String) : HierarchyNodeManager
#clone(father : HierarchyNode) : HierarchyNode
0..*
children
HierarchyNodeManager
(kddml.Core.DataMining.Taxonomy)
+addChild(node : HierarchyNodeManager) : void
<<getter>>+getName() : String
<<getter>>+getNumberOfChildren() : int
<<getter>>+getChildren() : HierarchyNodeManager[]
<<getter>>+getLevel() : int
<<getter>>+isLeaf() : boolean
<<getter>>+isRoot() : boolean
<<getter>>+getLeaves() : Iterator
<<getter>>+getParent() : HierarchyNodeManager
<<getter>>+getDimension() : int

Figure A.40: Package kddml.Core.DataMining.Taxonomy - Manager

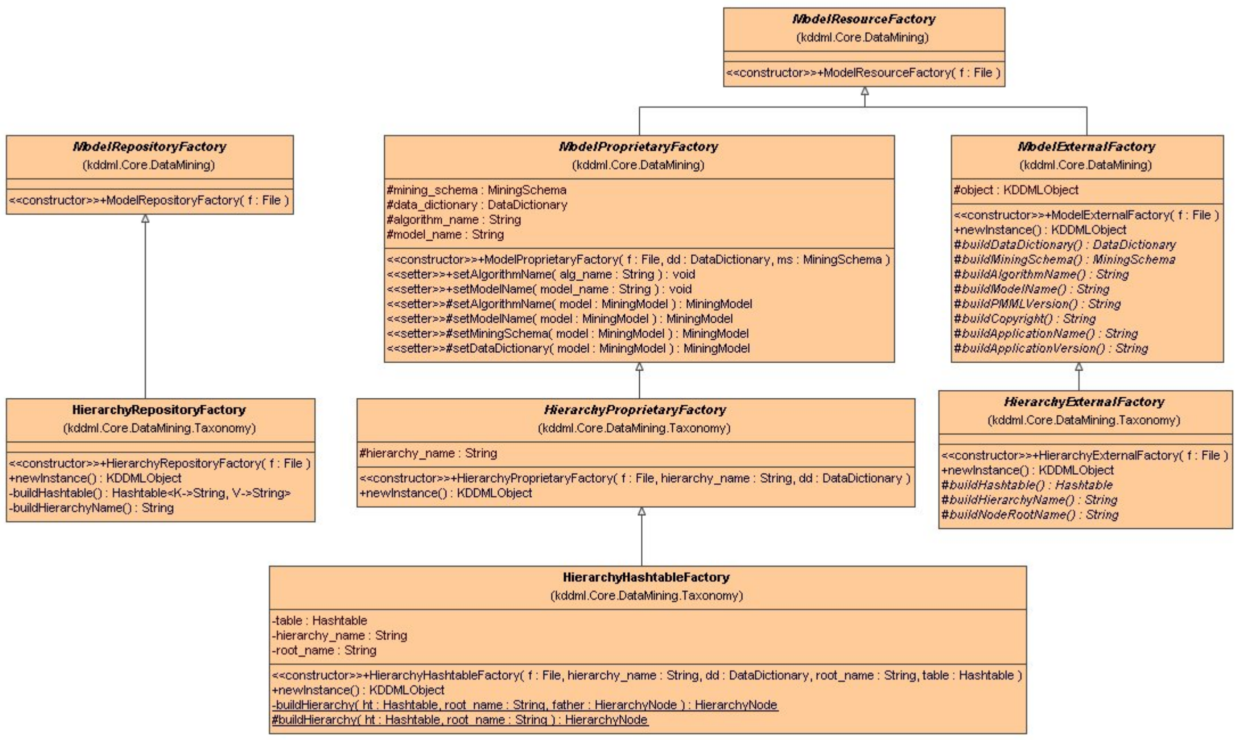

ModelResourceFactory
(kddml.Core.DataMining)
<<constructor>>+ModelResourceFactory(f : File)
ModelRepositoryFactory
(kddml.Core.DataMining)
<<constructor>>+ModelRepositoryFactory(f : File)
ModelProprietaryFactory
(kddml.Core.DataMining)
#mining_schema : MiningSchema
#data_dictionary : DataDictionary
#algorithm_name : String
#model_name : String
<<constructor>>+ModelProprietaryFactory(f : File, dd : DataDictionary, ms : MiningSchema)
<<setter>>+setAlgorithmName(alg_name : String) : void
<<setter>>+setModelName(model_name : String) : void
<<setter>>#setAlgorithmName(model : MiningModel) : MiningModel
<<setter>>#setModelName(model : MiningModel) : MiningModel
<<setter>>#setMiningSchema(model : MiningModel) : MiningModel
<<setter>>#setDataDictionary(model : MiningModel) : MiningModel
ModelExternalFactory
(kddml.Core.DataMining)
#object : KDDMLObject
<<constructor>>+ModelExternalFactory(f : File)
+newInstance() : KDDMLObject
#buildDataDictionary() : DataDictionary
#buildMiningSchema() : MiningSchema
#buildAlgorithmName() : String
#buildModelName() : String
#buildPMMLVersion() : String
#buildCopyright() : String
#buildApplicationName() : String
#buildApplicationVersion() : String
HierarchyRepositoryFactory
(kddml.Core.DataMining.Taxonomy)
<<constructor>>+HierarchyRepositoryFactory(f : File)
+newInstance() : KDDMLObject
-buildHashtable() : Hashtable<K->String, V->String>
-buildHierarchyName() : String
HierarchyProprietaryFactory
(kddml.Core.DataMining.Taxonomy)
#hierarchy_name : String
<<constructor>>+HierarchyProprietaryFactory(f : File, hierarchy_name : String, dd : DataDictionary)
+newInstance() : KDDMLObject
HierarchyExternalFactory
(kddml.Core.DataMining.Taxonomy)
<<constructor>>+HierarchyExternalFactory(f : File)
+newInstance() : KDDMLObject
#buildHashtable() : Hashtable
#buildHierarchyName() : String
#buildNodeRootName() : String
HierarchyHashtableFactory
(kddml.Core.DataMining.Taxonomy)
-table : Hashtable
-hierarchy_name : String
-root_name : String
<<constructor>>+HierarchyHashtableFactory(f : File, hierarchy_name : String, dd : DataDictionary, root_name : String, table : Hashtable)
+newInstance() : KDDMLObject
-buildHierarchy(ht : Hashtable, root_name : String, father : HierarchyNode) : HierarchyNode
#buildHierarchy(ht : Hashtable, root_name : String) : HierarchyNode

Figure A.41: Package kddml.Core.DataMining.Taxonomy - Factory

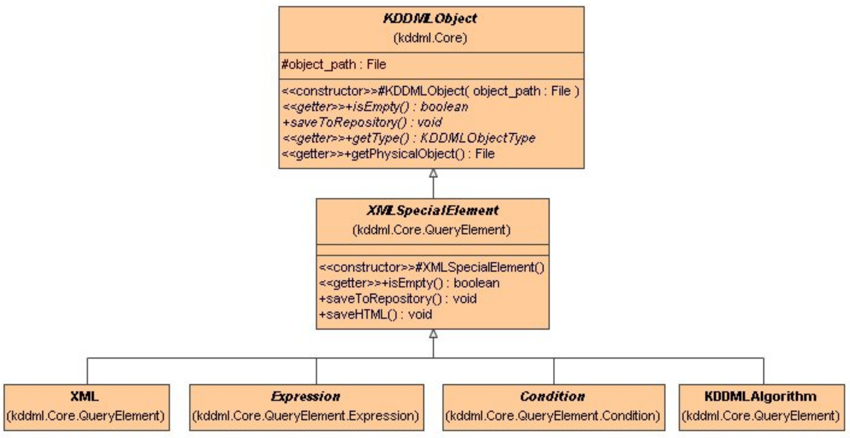

KDDMLObject
(kddml.Core)
#object_path : File
<<constructor>>#KDDMLObject(object_path : File)
<<getter>>+isEmpty() : boolean
+saveToRepository() : void
<<getter>>+getType() : KDDMLObjectType
<<getter>>+getPhysicalObject() : File
XMLSpecialElement
(kddml.Core.QueryElement)
<<constructor>>#XMLSpecialElement()
<<getter>>+isEmpty() : boolean
+saveToRepository() : void
+saveHTML() : void
XML
(kddml.Core.QueryElement)
Expression
(kddml.Core.QueryElement.Expression)
Condition
(kddml.Core.QueryElement.Condition)
KDDMLAlgorithm
(kddml.Core.QueryElement)

Figure A.42: Package kddml.Core.QueryElement - Query element manager

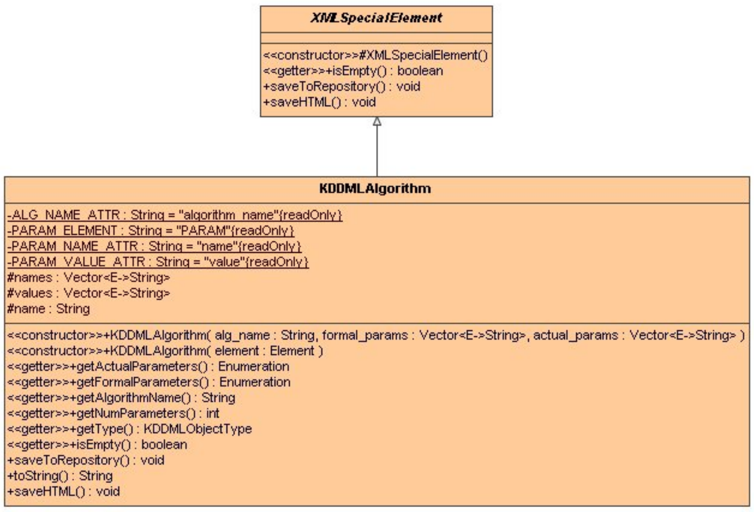

XMLSpecialElement
<<constructor>>#XMLSpecialElement()
<<getter>>+isEmpty() : boolean
+saveToRepository() : void
+saveHTML() : void
KDDMLAlgorithm
-ALG_NAME_ATTR : String = "algorithm_name"{readOnly}
-PARAM_ELEMENT : String = "PARAM"{readOnly}
-PARAM_NAME_ATTR : String = "name"{readOnly}
-PARAM_VALUE_ATTR : String = "value"{readOnly}
#names : Vector<E->String>
#values : Vector<E->String>
#name : String
<<constructor>>+KDDMLAlgorithm(alg_name : String, formal_params : Vector<E->String>, actual_params : Vector<E->String>)
<<constructor>>+KDDMLAlgorithm(element : Element)
<<getter>>+getActualParameters() : Enumeration
<<getter>>+getFormalParameters() : Enumeration
<<getter>>+getAlgorithmName() : String
<<getter>>+getNumParameters() : int
<<getter>>+getType() : KDDMLObjectType
<<getter>>+isEmpty() : boolean
+saveToRepository() : void
+toString() : String
+saveHTML() : void

Figure A.43: Package kddml.Core.QueryElement - Algorithm specification

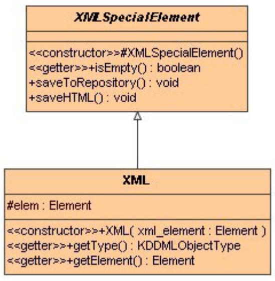

XMLSpecialElement
<<constructor>>#XMLSpecialElement()
<<getter>>+isEmpty() : boolean
+saveToRepository() : void
+saveHTML() : void
XML
#elem : Element
<<constructor>>+XML(xml_element : Element)
<<getter>>+getType() : KDDMLObjectType
<<getter>>+getElement() : Element

Figure A.44: Package kddml.Core.QueryElement - XML element

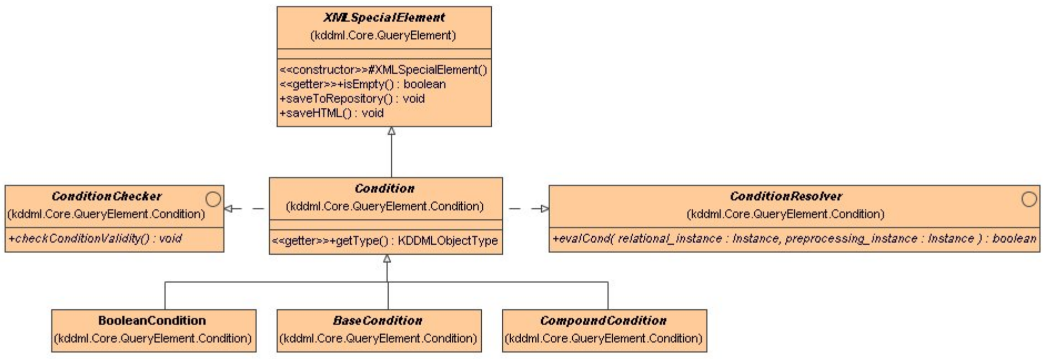

XMLSpecialElement
(kddml.Core.QueryElement)
<<constructor>>#XMLSpecialElement()
<<getter>>+isEmpty() : boolean
+saveToRepository() : void
+saveHTML() : void
ConditionChecker
(kddml.Core.QueryElement.Condition)
+checkConditionValidity() : void
Condition
(kddml.Core.QueryElement.Condition)
<<getter>>+getType() : KDDMLObjectType
ConditionResolver
(kddml.Core.QueryElement.Condition)
+evalCond(relational_instance : Instance, preprocessing_instance : Instance) : boolean
BooleanCondition
(kddml.Core.QueryElement.Condition)
BaseCondition
(kddml.Core.QueryElement.Condition)
CompoundCondition
(kddml.Core.QueryElement.Condition)

Figure A.45: Package kddml.Core.QueryElement.Condition - Manager

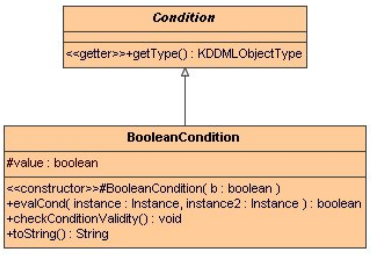

Condition
<<getter>>+getType() : KDDMLObjectType
BooleanCondition
#value : boolean
<<constructor>>#BooleanCondition(b : boolean)
+evalCond(instance : Instance, instance2 : Instance) : boolean
+checkConditionValidity() : void
+toString() : String

Figure A.46: Package kddml.Core.QueryElement.Condition - Boolean condition

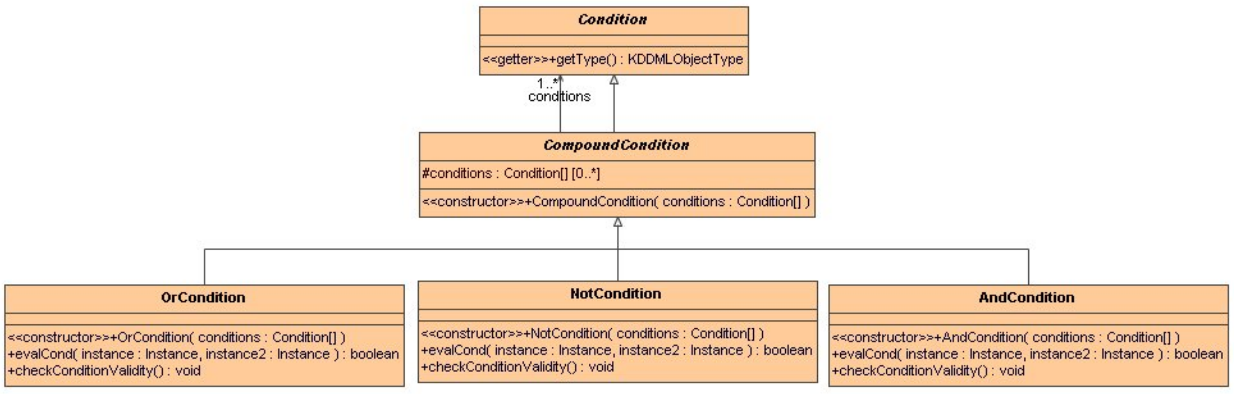

Condition
<<getter>>+getType() : KDDMLObjectType
1..*
conditions
CompoundCondition
#conditions : Condition[] [0..*]
<<constructor>>+CompoundCondition(conditions : Condition[])
OrCondition
<<constructor>>+OrCondition(conditions : Condition[])
+evalCond(instance : Instance, instance2 : Instance) : boolean
+checkConditionValidity() : void
NotCondition
<<constructor>>+NotCondition(conditions : Condition[])
+evalCond(instance : Instance, instance2 : Instance) : boolean
+checkConditionValidity() : void
AndCondition
<<constructor>>+AndCondition(conditions : Condition[])
+evalCond(instance : Instance, instance2 : Instance) : boolean
+checkConditionValidity() : void

Figure A.47: Package kddml.Core.QueryElement.Condition - Compound condition

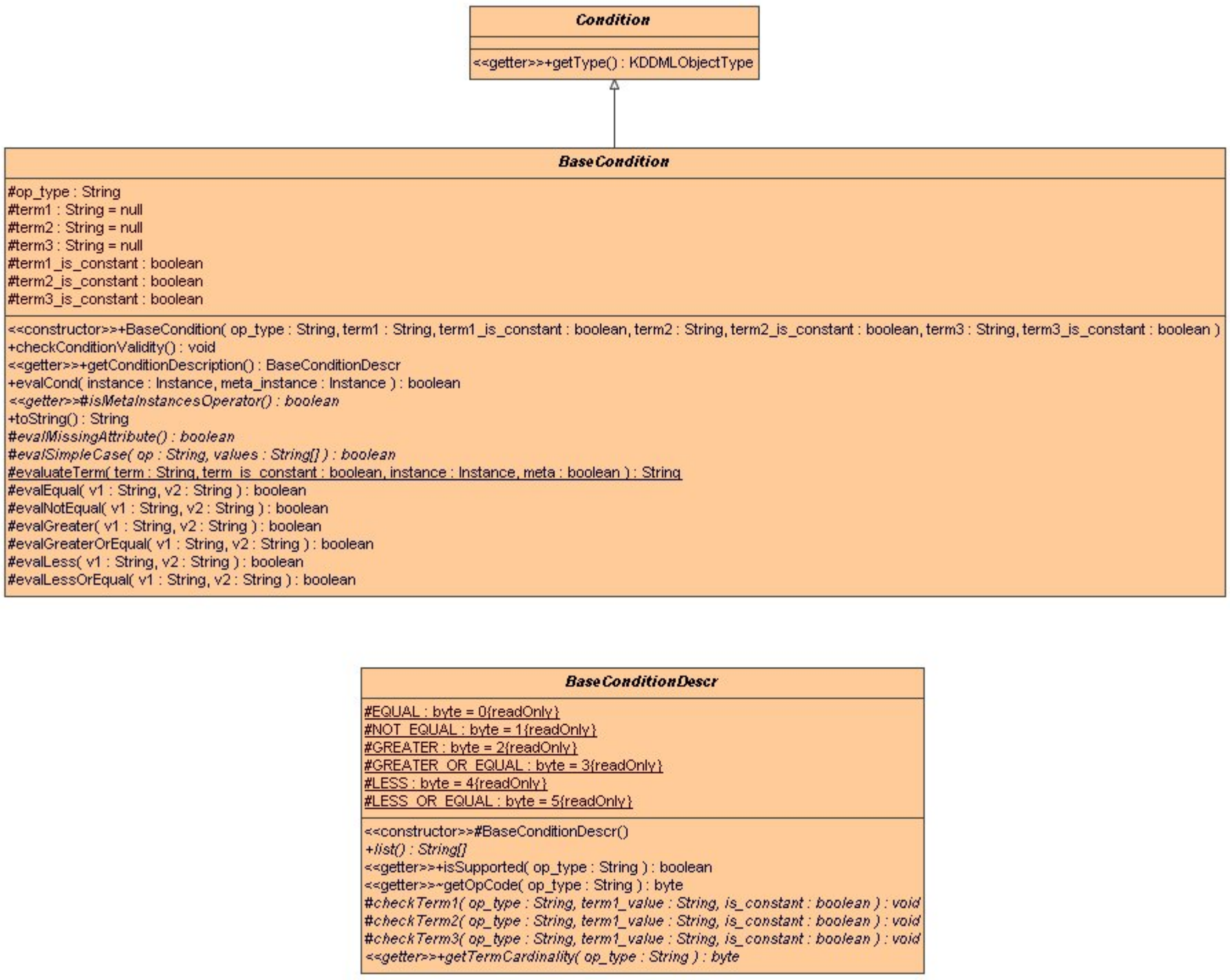

Condition
<<getter>>+getType() : KDDMLObjectType
BaseCondition
#op_type : String
#term1 : String = null
#term2 : String = null
#term3 : String = null
#term1_is_constant : boolean
#term2_is_constant : boolean
#term3_is_constant : boolean
<<constructor>>+BaseCondition(op_type : String, term1 : String, term1_is_constant : boolean, term2 : String, term2_is_constant : boolean, term3 : String, term3_is_constant : boolean)
+checkConditionValidity() : void
<<getter>>+getConditionDescription() : BaseConditionDescr
+evalCond(instance : Instance, meta_instance : Instance) : boolean
<<getter>>#isMetaInstancesOperator() : boolean
+toString() : String
#evalMissingAttribute() : boolean
#evalSimpleCase(op : String, values : String[]) : boolean
#evaluateTerm(term : String, term_is_constant : boolean, instance : Instance, meta : boolean) : String
#evalEqual(v1 : String, v2 : String) : boolean
#evalNotEqual(v1 : String, v2 : String) : boolean
#evalGreater(v1 : String, v2 : String) : boolean
#evalGreaterOrEqual(v1 : String, v2 : String) : boolean
#evalLess(v1 : String, v2 : String) : boolean
#evalLessOrEqual(v1 : String, v2 : String) : boolean
BaseConditionDescr
#EQUAL : byte = 0{readOnly}
#NOT_EQUAL : byte = 1{readOnly}
#GREATER : byte = 2{readOnly}
#GREATER_OR_EQUAL : byte = 3{readOnly}
#LESS : byte = 4{readOnly}
#LESS_OR_EQUAL : byte = 5{readOnly}
<<constructor>>#BaseConditionDescr()
+list() : String[]
<<getter>>+isSupported(op_type : String) : boolean
<<getter>>~getOpCode(op_type : String) : byte
#checkTerm1(op_type : String, term1_value : String, is_constant : boolean) : void
#checkTerm2(op_type : String, term1_value : String, is_constant : boolean) : void
#checkTerm3(op_type : String, term1_value : String, is_constant : boolean) : void
<<getter>>+getTermCardinality(op_type : String) : byte

Figure A.48: Package kddml.Core.QueryElement.Condition - Base condition

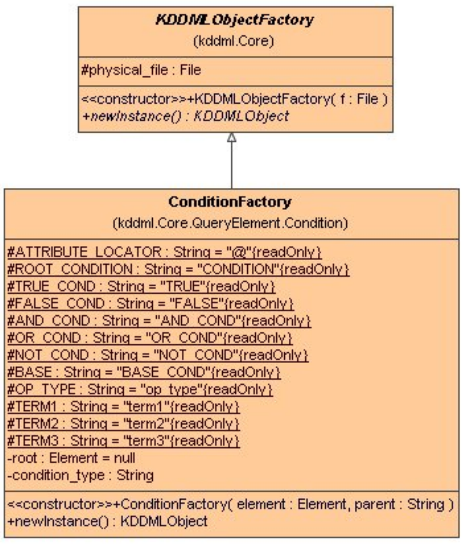

KDDMLObjectFactory
(kddml.Core)
#physical_file : File
<<constructor>>+KDDMLObjectFactory(f : File)
+newInstance() : KDDMLObject
ConditionFactory
(kddml.Core.QueryElement.Condition)
#ATTRIBUTE_LOCATOR : String = "@"{readOnly}
#ROOT_CONDITION : String = "CONDITION"{readOnly}
#TRUE_COND : String = "TRUE"{readOnly}
#FALSE_COND : String = "FALSE"{readOnly}
#AND_COND : String = "AND_COND"{readOnly}
#OR_COND : String = "OR_COND"{readOnly}
#NOT_COND : String = "NOT_COND"{readOnly}
#BASE : String = "BASE_COND"{readOnly}
#OP_TYPE : String = "op_type"{readOnly}
#TERM1 : String = "term1"{readOnly}
#TERM2 : String = "term2"{readOnly}
#TERM3 : String = "term3"{readOnly}
-root : Element = null
-condition_type : String
<<constructor>>+ConditionFactory(element : Element, parent : String)
+newInstance() : KDDMLObject

Figure A.49: Package kddml.Core.QueryElement.Condition - Factory

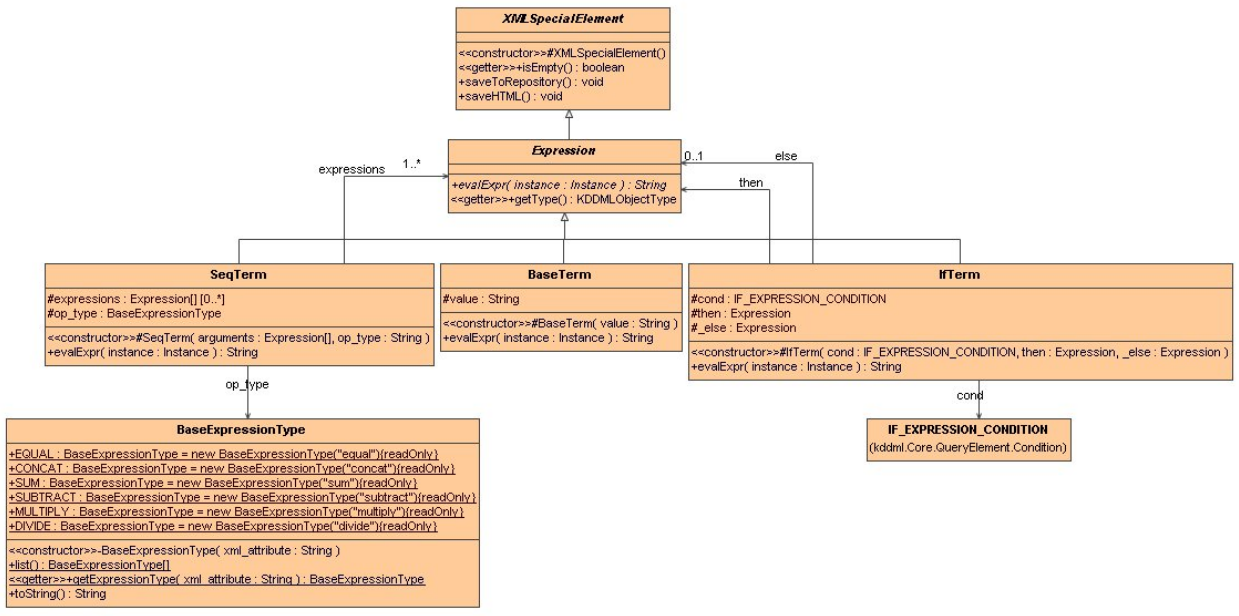

XMLSpecialElement
<<constructor>>#XMLSpecialElement()
<<getter>>+isEmpty() : boolean
+saveToRepository() : void
+saveHTML() : void
Expression
+evalExpr(instance : Instance) : String
<<getter>>+getType() : KDDMLObjectType
expressions
1..*
0..1
else
then
SeqTerm
#expressions : Expression[] [0..*]
#op_type : BaseExpressionType
<<constructor>>#SeqTerm(arguments : Expression[], op_type : String)
+evalExpr(instance : Instance) : String
BaseTerm
#value : String
<<constructor>>#BaseTerm(value : String)
+evalExpr(instance : Instance) : String
IfTerm
#cond : IF_EXPRESSION_CONDITION
#then : Expression
#_else : Expression
<<constructor>>#IfTerm(cond : IF_EXPRESSION_CONDITION, then : Expression, _else : Expression)
+evalExpr(instance : Instance) : String
op_type
cond
BaseExpressionType
+EQUAL : BaseExpressionType = new BaseExpressionType("equal"){readOnly}
+CONCAT : BaseExpressionType = new BaseExpressionType("concat"){readOnly}
+SUM : BaseExpressionType = new BaseExpressionType("sum"){readOnly}
+SUBTRACT : BaseExpressionType = new BaseExpressionType("subtract"){readOnly}
+MULTIPLY : BaseExpressionType = new BaseExpressionType("multiply"){readOnly}
+DIVIDE : BaseExpressionType = new BaseExpressionType("divide"){readOnly}
<<constructor>>-BaseExpressionType(xml_attribute : String)
+list() : BaseExpressionType[]
<<getter>>+getExpressionType(xml_attribute : String) : BaseExpressionType
+toString() : String
IF_EXPRESSION_CONDITION
(kddml.Core.QueryElement.Condition)

Figure A.50: Package kddml.Core.QueryElement.Expression - Manager

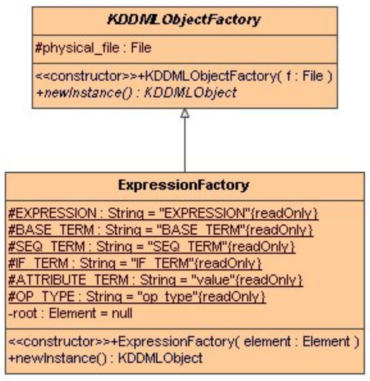

KDDMLObjectFactory
#physical_file : File
<<constructor>>+KDDMLObjectFactory(f : File)
+newInstance() : KDDMLObject
ExpressionFactory
#EXPRESSION : String = "EXPRESSION"{readOnly}
#BASE_TERM : String = "BASE_TERM"{readOnly}
#SEQ_TERM : String = "SEQ_TERM"{readOnly}
#IF_TERM : String = "IF_TERM"{readOnly}
#ATTRIBUTE_TERM : String = "value"{readOnly}
#OP_TYPE : String = "op_type"{readOnly}
-root : Element = null
<<constructor>>+ExpressionFactory(element : Element)
+newInstance() : KDDMLObject

Figure A.51: Package kddml.Core.QueryElement.Expression - Factory

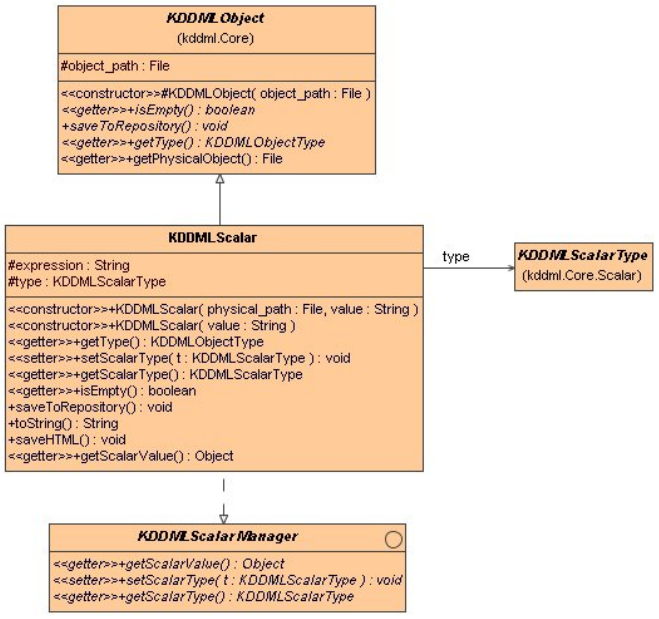

KDDMLObject
(kddml.Core)
#object_path : File
<<constructor>>#KDDMLObject(object_path : File)
<<getter>>+isEmpty() : boolean
+saveToRepository() : void
<<getter>>+getType() : KDDMLObjectType
<<getter>>+getPhysicalObject() : File
KDDMLScalar
#expression : String
#type : KDDMLScalarType
<<constructor>>+KDDMLScalar(physical_path : File, value : String)
<<constructor>>+KDDMLScalar(value : String)
<<getter>>+getType() : KDDMLObjectType
<<setter>>+setScalarType(t : KDDMLScalarType) : void
<<getter>>+getScalarType() : KDDMLScalarType
<<getter>>+isEmpty() : boolean
+saveToRepository() : void
+toString() : String
+saveHTML() : void
<<getter>>+getScalarValue() : Object
type
KDDMLScalarType
(kddml.Core.Scalar)
KDDMLScalarManager
<<getter>>+getScalarValue() : Object
<<setter>>+setScalarType(t : KDDMLScalarType) : void
<<getter>>+getScalarType() : KDDMLScalarType

Figure A.52: Package kddml.Core.Scalar - Scalar manager

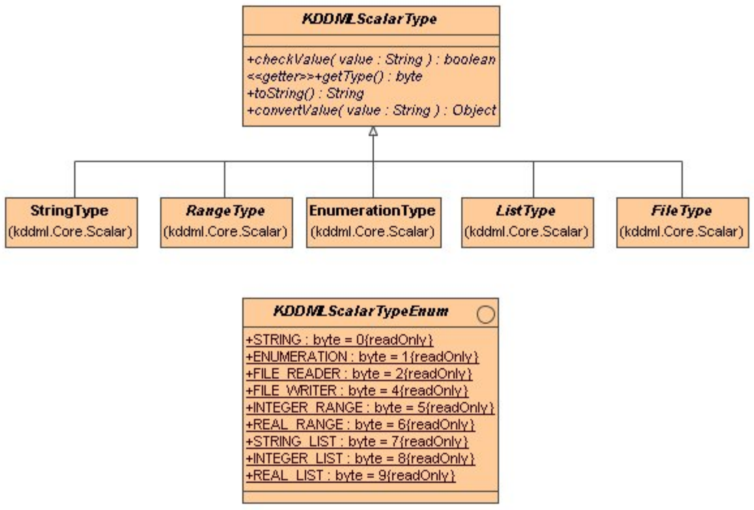

KDDMLScalarType
+checkValue(value : String) : boolean
<<getter>>+getType() : byte
+toString() : String
+convertValue(value : String) : Object
StringType
(kddml.Core.Scalar)
RangeType
(kddml.Core.Scalar)
EnumerationType
(kddml.Core.Scalar)
ListType
(kddml.Core.Scalar)
FileType
(kddml.Core.Scalar)
KDDMLScalarTypeEnum
+STRING : byte = 0{readOnly}
+ENUMERATION : byte = 1{readOnly}
+FILE_READER : byte = 2{readOnly}
+FILE_WRITER : byte = 4{readOnly}
+INTEGER_RANGE : byte = 5{readOnly}
+REAL_RANGE : byte = 6{readOnly}
+STRING_LIST : byte = 7{readOnly}
+INTEGER_LIST : byte = 8{readOnly}
+REAL_LIST : byte = 9{readOnly}

Figure A.53: Package kddml.Core.Scalar - Scalar types

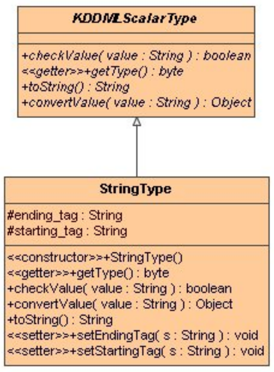

KDDMLScalarType
+checkValue(value : String) : boolean
<<getter>>+getType() : byte
+toString() : String
+convertValue(value : String) : Object
StringType
#ending_tag : String
#starting_tag : String
<<constructor>>+StringType()
<<getter>>+getType() : byte
+checkValue(value : String) : boolean
+convertValue(value : String) : Object
+toString() : String
<<setter>>+setEndingTag(s : String) : void
<<setter>>+setStartingTag(s : String) : void

Figure A.54: Package kddml.Core.Scalar - String type

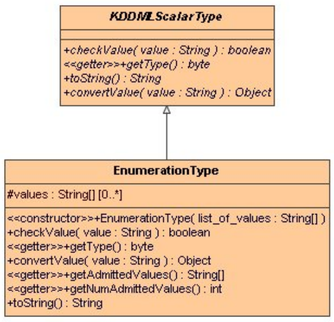

KDDMLScalarType
+checkValue(value : String) : boolean
<<getter>>+getType() : byte
+toString() : String
+convertValue(value : String) : Object
EnumerationType
#values : String[] [0..*]
<<constructor>>+EnumerationType(list_of_values : String[])
+checkValue(value : String) : boolean
<<getter>>+getType() : byte
+convertValue(value : String) : Object
<<getter>>+getAdmittedValues() : String[]
<<getter>>+getNumAdmittedValues() : int
+toString() : String

Figure A.55: Package kddml.Core.Scalar - Enumeration type

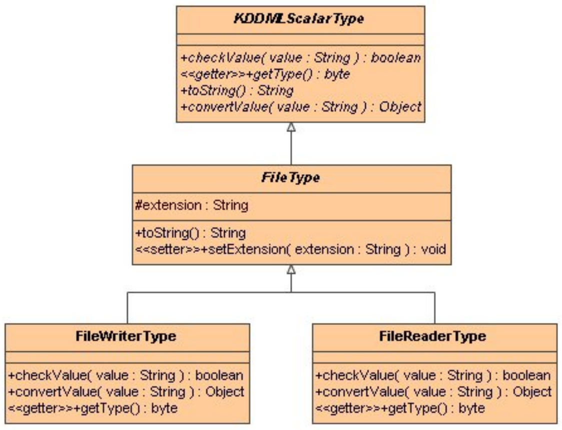

KDDMLScalarType
+checkValue(value : String) : boolean
<<getter>>+getType() : byte
+toString() : String
+convertValue(value : String) : Object
FileType
#extension : String
+toString() : String
<<setter>>+setExtension(extension : String) : void
FileWriterType
+checkValue(value : String) : boolean
+convertValue(value : String) : Object
<<getter>>+getType() : byte
FileReaderType
+checkValue(value : String) : boolean
+convertValue(value : String) : Object
<<getter>>+getType() : byte

Figure A.56: Package kddml.Core.Scalar - File type

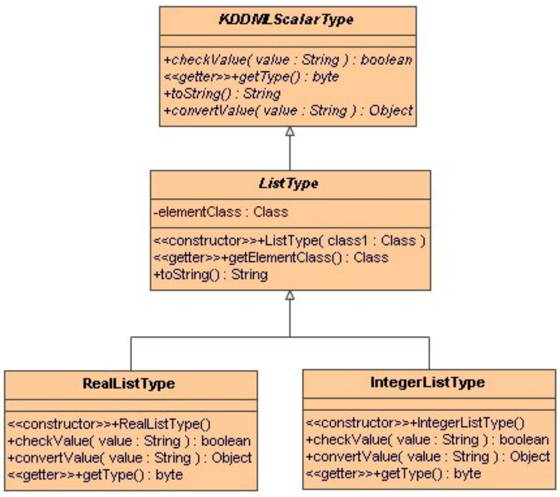

KDDMLScalarType
+checkValue(value : String) : boolean
<<getter>>+getType() : byte
+toString() : String
+convertValue(value : String) : Object
ListType
-elementClass : Class
<<constructor>>+ListType(class1 : Class)
<<getter>>+getElementClass() : Class
+toString() : String
RealListType
<<constructor>>+RealListType()
+checkValue(value : String) : boolean
+convertValue(value : String) : Object
<<getter>>+getType() : byte
IntegerListType
<<constructor>>+IntegerListType()
+checkValue(value : String) : boolean
+convertValue(value : String) : Object
<<getter>>+getType() : byte

Figure A.57: Package kddml.Core.Scalar - List type

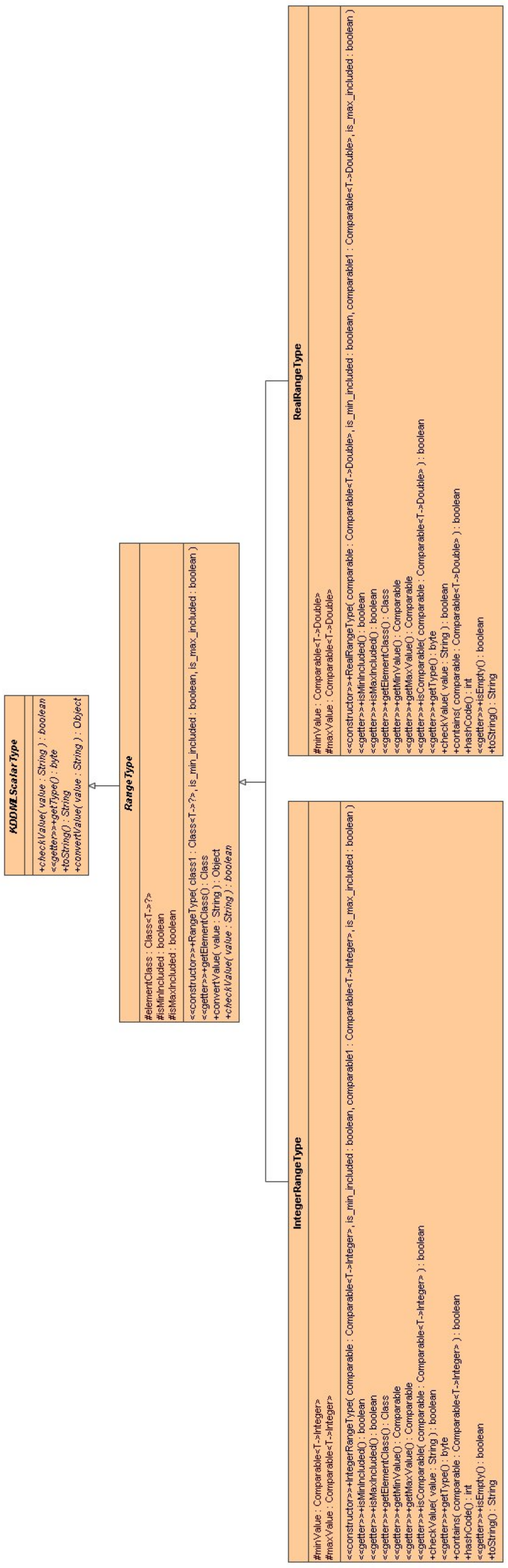

KDDMLScalarType
+checkValue(value : String) : boolean
<<getter>>+getType() : byte
+toString() : String
+convertValue(value : String) : Object
RangeType
#elementClass : Class<T->?>
#isMinIncluded : boolean
#isMaxIncluded : boolean
<<constructor>>+RangeType(class1 : Class<T->?>, is_min_included : boolean, is_max_included : boolean)
<<getter>>+getElementClass() : Class
+convertValue(value : String) : Object
+checkValue(value : String) : boolean
IntegerRangeType
#minValue : Comparable<T->Integer>
#maxValue : Comparable<T->Integer>
<<constructor>>+IntegerRangeType(comparable : Comparable<T->Integer>, is_min_included : boolean, comparable1 : Comparable<T->Integer>, is_max_included : boolean)
<<getter>>+isMinIncluded() : boolean
<<getter>>+isMaxIncluded() : boolean
<<getter>>+getElementClass() : Class
<<getter>>+getMinValue() : Comparable
<<getter>>+getMaxValue() : Comparable
<<getter>>+isComparable(comparable : Comparable<T->Integer>) : boolean
+checkValue(value : String) : boolean
<<getter>>+getType() : byte
+contains(comparable : Comparable<T->Integer>) : boolean
+hashCode() : int
<<getter>>+isEmpty() : boolean
+toString() : String
RealRangeType
#minValue : Comparable<T->Double>
#maxValue : Comparable<T->Double>
<<constructor>>+RealRangeType(comparable : Comparable<T->Double>, is_min_included : boolean, comparable1 : Comparable<T->Double>, is_max_included : boolean)
<<getter>>+isMinIncluded() : boolean
<<getter>>+isMaxIncluded() : boolean
<<getter>>+getElementClass() : Class
<<getter>>+getMinValue() : Comparable
<<getter>>+getMaxValue() : Comparable
<<getter>>+isComparable(comparable : Comparable<T->Double>) : boolean
<<getter>>+getType() : byte
+checkValue(value : String) : boolean
+contains(comparable : Comparable<T->Double>) : boolean
+hashCode() : int
<<getter>>+isEmpty() : boolean
+toString() : String

Figure A.58: Package kddml.Core.Scalar - Range type

APPENDIX B

UML operator layer diagrams

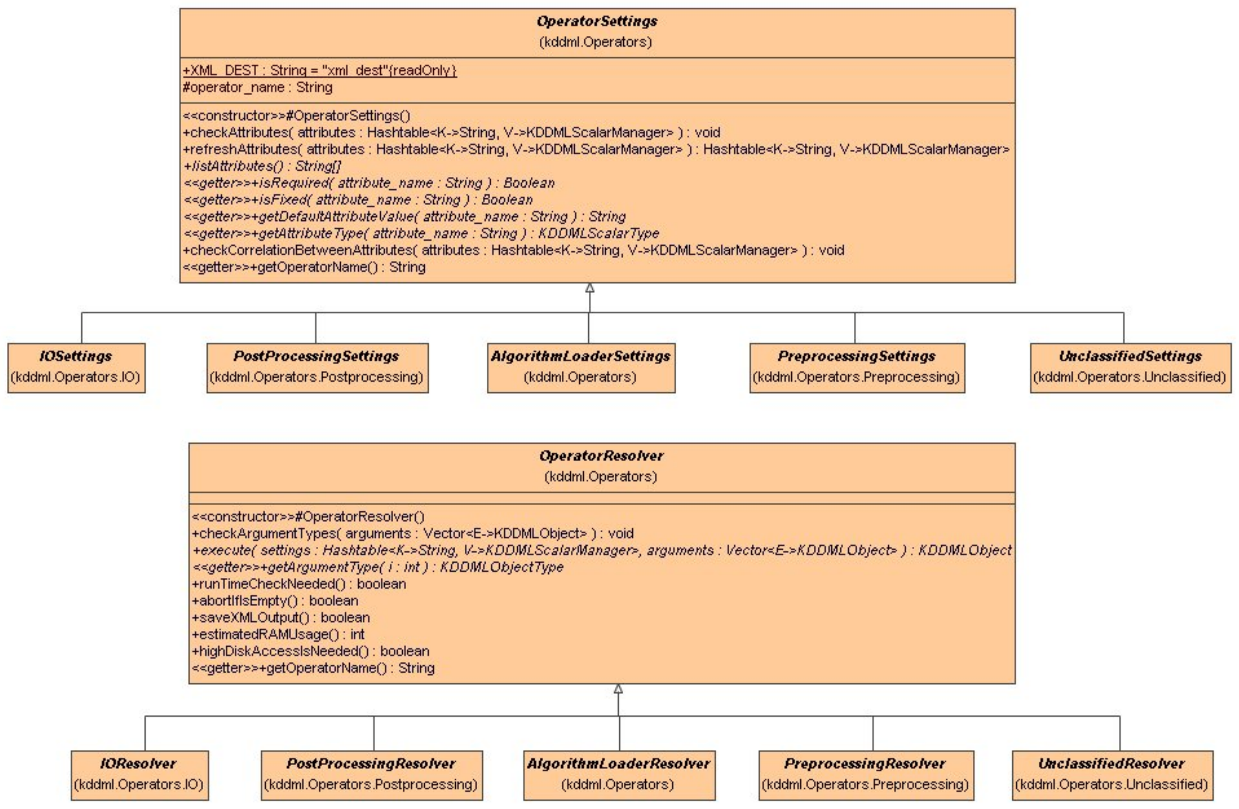

OperatorSettings
(kddml.Operators)
+XML_DEST : String = "xml_dest"{readOnly}
#operator_name : String
<<constructor>>#OperatorSettings()
+checkAttributes(attributes : Hashtable<K->String, V->KDDMLScalarManager>) : void
+refreshAttributes(attributes : Hashtable<K->String, V->KDDMLScalarManager>) : Hashtable<K->String, V->KDDMLScalarManager>
+listAttributes() : String[]
<<getter>>+isRequired(attribute_name : String) : Boolean
<<getter>>+isFixed(attribute_name : String) : Boolean
<<getter>>+getDefaultAttributeValue(attribute_name : String) : String
<<getter>>+getAttributeType(attribute_name : String) : KDDMLScalarType
+checkCorrelationBetweenAttributes(attributes : Hashtable<K->String, V->KDDMLScalarManager>) : void
<<getter>>+getOperatorName() : String
IOSettings
(kddml.Operators.IO)
PostProcessingSettings
(kddml.Operators.Postprocessing)
AlgorithmLoaderSettings
(kddml.Operators)
PreprocessingSettings
(kddml.Operators.Preprocessing)
UnclassifiedSettings
(kddml.Operators.Unclassified)
OperatorResolver
(kddml.Operators)
<<constructor>>#OperatorResolver()
+checkArgumentTypes(arguments : Vector<E->KDDMLObject>) : void
+execute(settings : Hashtable<K->String, V->KDDMLScalarManager>, arguments : Vector<E->KDDMLObject>) : KDDMLObject
<<getter>>+getArgumentType(i : int) : KDDMLObjectType
+runTimeCheckNeeded() : boolean
+abortIfIsEmpty() : boolean
+saveXMLOutput() : boolean
+estimatedRAMUsage() : int
+highDiskAccessIsNeeded() : boolean
<<getter>>+getOperatorName() : String
IOResolver
(kddml.Operators.IO)
PostProcessingResolver
(kddml.Operators.Postprocessing)
AlgorithmLoaderResolver
(kddml.Operators)
PreprocessingResolver
(kddml.Operators.Preprocessing)
UnclassifiedResolver
(kddml.Operators.Unclassified)

Figure B.1: Package kddml.Operators - Operator settings and operator resolver

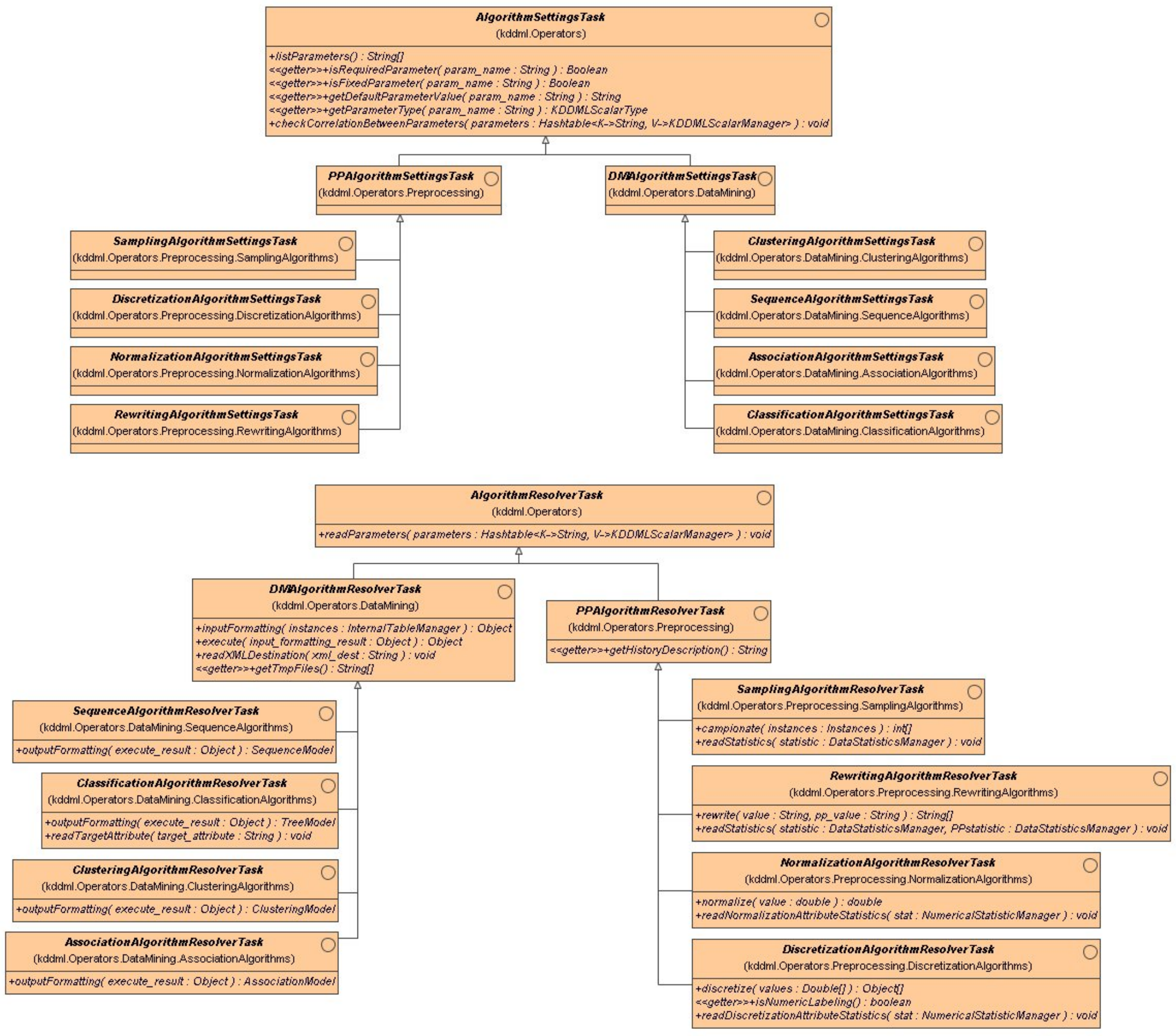

AlgorithmSettingsTask
(kddml.Operators)
+listParameters() : String[]
<<getter>>+isRequiredParameter(param_name : String) : Boolean
<<getter>>+isFixedParameter(param_name : String) : Boolean
<<getter>>+getDefaultParameterValue(param_name : String) : String
<<getter>>+getParameterType(param_name : String) : KDDMLScalarType
+checkCorrelationBetweenParameters(parameters : Hashtable<K->String, V->KDDMLScalarManager>) : void
PPAlgorithmSettingsTask
(kddml.Operators.Preprocessing)
DMAlgorithmSettingsTask
(kddml.Operators.DataMining)
SamplingAlgorithmSettingsTask
(kddml.Operators.Preprocessing.SamplingAlgorithms)
DiscretizationAlgorithmSettingsTask
(kddml.Operators.Preprocessing.DiscretizationAlgorithms)
NormalizationAlgorithmSettingsTask
(kddml.Operators.Preprocessing.NormalizationAlgorithms)
RewritingAlgorithmSettingsTask
(kddml.Operators.Preprocessing.RewritingAlgorithms)
ClusteringAlgorithmSettingsTask
(kddml.Operators.DataMining.ClusteringAlgorithms)
SequenceAlgorithmSettingsTask
(kddml.Operators.DataMining.SequenceAlgorithms)
AssociationAlgorithmSettingsTask
(kddml.Operators.DataMining.AssociationAlgorithms)
ClassificationAlgorithmSettingsTask
(kddml.Operators.DataMining.ClassificationAlgorithms)
AlgorithmResolverTask
(kddml.Operators)
+readParameters(parameters : Hashtable<K->String, V->KDDMLScalarManager>) : void
DMAlgorithmResolverTask
(kddml.Operators.DataMining)
+inputFormatting(instances : InternalTableManager) : Object
+execute(input_formatting_result : Object) : Object
+readXMLDestination(xml_dest : String) : void
<<getter>>+getTmpFiles() : String[]
PPAlgorithmResolverTask
(kddml.Operators.Preprocessing)
<<getter>>+getHistoryDescription() : String
SequenceAlgorithmResolverTask
(kddml.Operators.DataMining.SequenceAlgorithms)
+outputFormatting(execute_result : Object) : SequenceModel
ClassificationAlgorithmResolverTask
(kddml.Operators.DataMining.ClassificationAlgorithms)
+outputFormatting(execute_result : Object) : TreeModel
+readTargetAttribute(target_attribute : String) : void
ClusteringAlgorithmResolverTask
(kddml.Operators.DataMining.ClusteringAlgorithms)
+outputFormatting(execute_result : Object) : ClusteringModel
AssociationAlgorithmResolverTask
(kddml.Operators.DataMining.AssociationAlgorithms)
+outputFormatting(execute_result : Object) : AssociationModel
SamplingAlgorithmResolverTask
(kddml.Operators.Preprocessing.SamplingAlgorithms)
+campionate(instances : Instances) : int[]
+readStatistics(statistic : DataStatisticsManager) : void
RewritingAlgorithmResolverTask
(kddml.Operators.Preprocessing.RewritingAlgorithms)
+rewrite(value : String, pp_value : String) : String[]
+readStatistics(statistic : DataStatisticsManager, PPstatistic : DataStatisticsManager) : void
NormalizationAlgorithmResolverTask
(kddml.Operators.Preprocessing.NormalizationAlgorithms)
+normalize(value : double) : double
+readNormalizationAttributeStatistics(stat : NumericalStatisticManager) : void
DiscretizationAlgorithmResolverTask
(kddml.Operators.Preprocessing.DiscretizationAlgorithms)
+discretize(values : Double[]) : Object[]
<<getter>>+isNumericLabeling() : boolean
+readDiscretizationAttributeStatistics(stat : NumericalStatisticManager) : void

Figure B.2: Package kddml.Operators - Algorithm settings task and algorithm resolver task

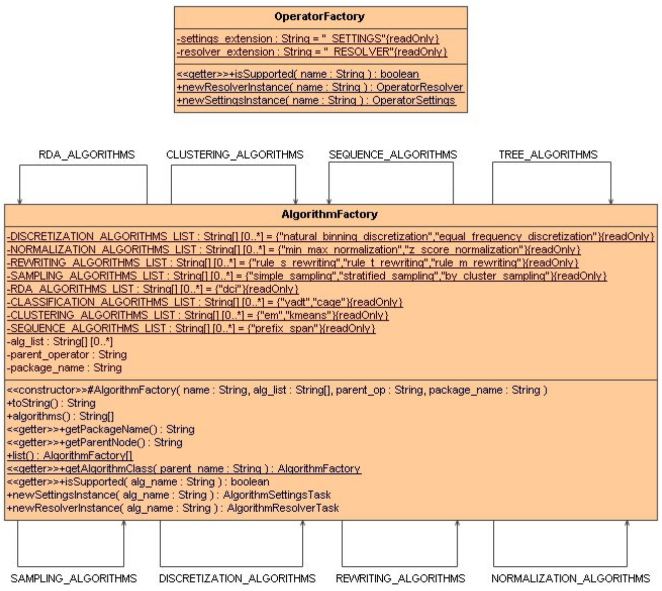

OperatorFactory
-settings_extension : String = "_SETTINGS"{readOnly}
-resolver_extension : String = "_RESOLVER"{readOnly}
<<getter>>+isSupported(name : String) : boolean
+newResolverInstance(name : String) : OperatorResolver
+newSettingsInstance(name : String) : OperatorSettings
RDA_ALGORITHMS
CLUSTERING_ALGORITHMS
SEQUENCE_ALGORITHMS
TREE_ALGORITHMS
AlgorithmFactory
-DISCRETIZATION_ALGORITHMS_LIST : String[] [0..*] = {"natural_binning_discretization","equal_frequency_discretization"}{readOnly}
-NORMALIZATION_ALGORITHMS_LIST : String[] [0..*] = {"min_max_normalization","z_score_normalization"}{readOnly}
-REWRITING_ALGORITHMS_LIST : String[] [0..*] = {"rule_s_rewriting","rule_t_rewriting","rule_m_rewriting"}{readOnly}
-SAMPLING_ALGORITHMS_LIST : String[] [0..*] = {"simple_sampling","stratified_sampling","by_cluster_sampling"}{readOnly}
-RDA_ALGORITHMS_LIST : String[] [0..*] = {"dci"}{readOnly}
-CLASSIFICATION_ALGORITHMS_LIST : String[] [0..*] = {"yadt","cage"}{readOnly}
-CLUSTERING_ALGORITHMS_LIST : String[] [0..*] = {"em","kmeans"}{readOnly}
-SEQUENCE_ALGORITHMS_LIST : String[] [0..*] = {"prefix_span"}{readOnly}
-alg_list : String[] [0..*]
-parent_operator : String
-package_name : String
<<constructor>>#AlgorithmFactory(name : String, alg_list : String[], parent_op : String, package_name : String)
+toString() : String
+algorithms() : String[]
<<getter>>+getPackageName() : String
<<getter>>+getParentNode() : String
+list() : AlgorithmFactory[]
<<getter>>+getAlgorithmClass(parent_name : String) : AlgorithmFactory
<<getter>>+isSupported(alg_name : String) : boolean
+newSettingsInstance(alg_name : String) : AlgorithmSettingsTask
+newResolverInstance(alg_name : String) : AlgorithmResolverTask
SAMPLING_ALGORITHMS
DISCRETIZATION_ALGORITHMS
REWRITING_ALGORITHMS
NORMALIZATION_ALGORITHMS

Figure B.3: Package kddml.Operators - Operator and algorithm factory

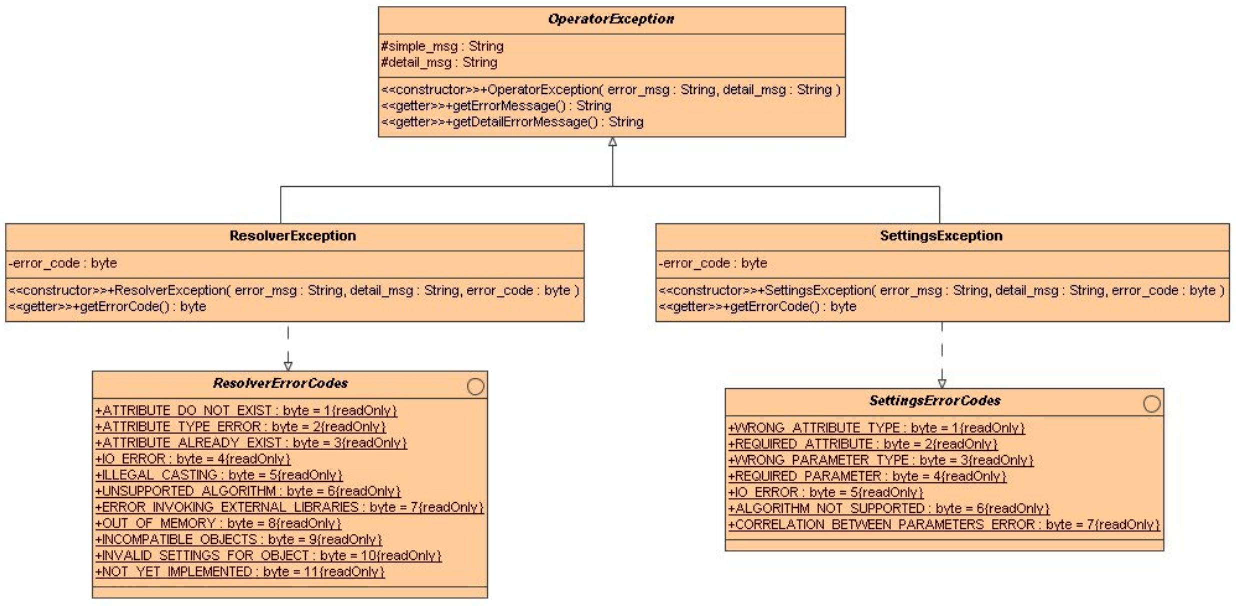

OperatorException
#simple_msg : String
#detail_msg : String
<<constructor>>+OperatorException(error_msg : String, detail_msg : String)
<<getter>>+getErrorMessage() : String
<<getter>>+getDetailErrorMessage() : String
ResolverException
-error_code : byte
<<constructor>>+ResolverException(error_msg : String, detail_msg : String, error_code : byte)
<<getter>>+getErrorCode() : byte
SettingsException
-error_code : byte
<<constructor>>+SettingsException(error_msg : String, detail_msg : String, error_code : byte)
<<getter>>+getErrorCode() : byte
ResolverErrorCodes
+ATTRIBUTE_DO_NOT_EXIST : byte = 1{readOnly}
+ATTRIBUTE_TYPE_ERROR : byte = 2{readOnly}
+ATTRIBUTE_ALREADY_EXIST : byte = 3{readOnly}
+IO_ERROR : byte = 4{readOnly}
+ILLEGAL_CASTING : byte = 5{readOnly}
+UNSUPPORTED_ALGORITHM : byte = 6{readOnly}
+ERROR_INVOKING_EXTERNAL_LIBRARIES : byte = 7{readOnly}
+OUT_OF_MEMORY : byte = 8{readOnly}
+INCOMPATIBLE_OBJECTS : byte = 9{readOnly}
+INVALID_SETTINGS_FOR_OBJECT : byte = 10{readOnly}
+NOT_YET_IMPLEMENTED : byte = 11{readOnly}
SettingsErrorCodes
+WRONG_ATTRIBUTE_TYPE : byte = 1{readOnly}
+REQUIRED_ATTRIBUTE : byte = 2{readOnly}
+WRONG_PARAMETER_TYPE : byte = 3{readOnly}
+REQUIRED_PARAMETER : byte = 4{readOnly}
+IO_ERROR : byte = 5{readOnly}
+ALGORITHM_NOT_SUPPORTED : byte = 6{readOnly}
+CORRELATION_BETWEEN_PARAMETERS_ERROR : byte = 7{readOnly}

Figure B.4: Package kddml.Operators - Exceptions

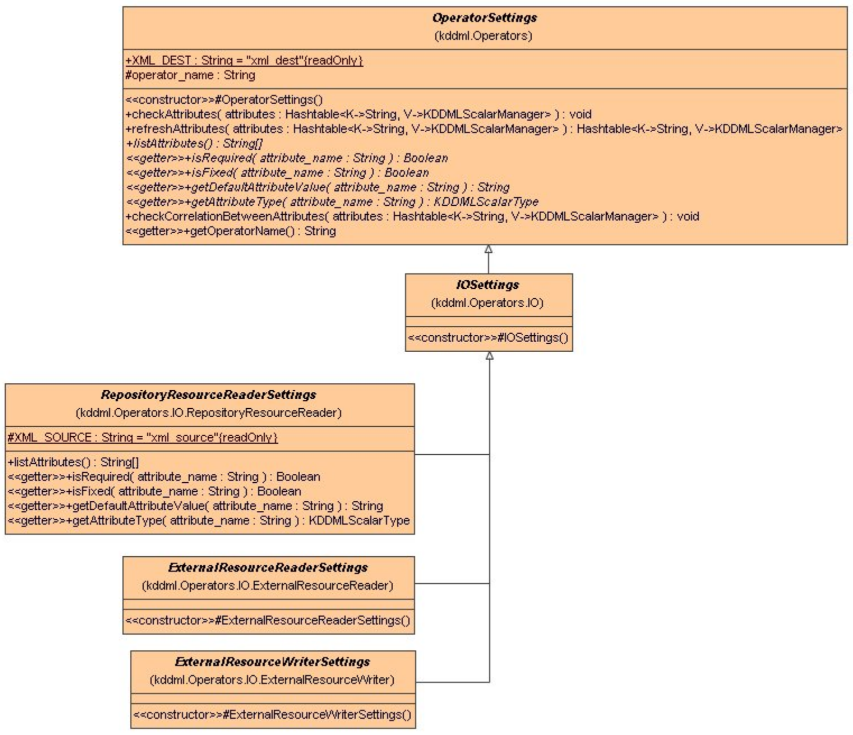

OperatorSettings
(kddml.Operators)
+XML_DEST : String = "xml_dest"{readOnly}
#operator_name : String
<<constructor>>#OperatorSettings()
+checkAttributes(attributes : Hashtable<K->String, V->KDDMLScalarManager>) : void
+refreshAttributes(attributes : Hashtable<K->String, V->KDDMLScalarManager>) : Hashtable<K->String, V->KDDMLScalarManager>
+listAttributes() : String[]
<<getter>>+isRequired(attribute_name : String) : Boolean
<<getter>>+isFixed(attribute_name : String) : Boolean
<<getter>>+getDefaultAttributeValue(attribute_name : String) : String
<<getter>>+getAttributeType(attribute_name : String) : KDDMLScalarType
+checkCorrelationBetweenAttributes(attributes : Hashtable<K->String, V->KDDMLScalarManager>) : void
<<getter>>+getOperatorName() : String
IOSettings
(kddml.Operators.IO)
<<constructor>>#IOSettings()
RepositoryResourceReaderSettings
(kddml.Operators.IO.RepositoryResourceReader)
#XML_SOURCE : String = "xml_source"{readOnly}
+listAttributes() : String[]
<<getter>>+isRequired(attribute_name : String) : Boolean
<<getter>>+isFixed(attribute_name : String) : Boolean
<<getter>>+getDefaultAttributeValue(attribute_name : String) : String
<<getter>>+getAttributeType(attribute_name : String) : KDDMLScalarType
ExternalResourceReaderSettings
(kddml.Operators.IO.ExternalResourceReader)
<<constructor>>#ExternalResourceReaderSettings()
ExternalResourceWriterSettings
(kddml.Operators.IO.ExternalResourceWriter)
<<constructor>>#ExternalResourceWriterSettings()

Figure B.5: Package kddml.Operators.IO - Settings

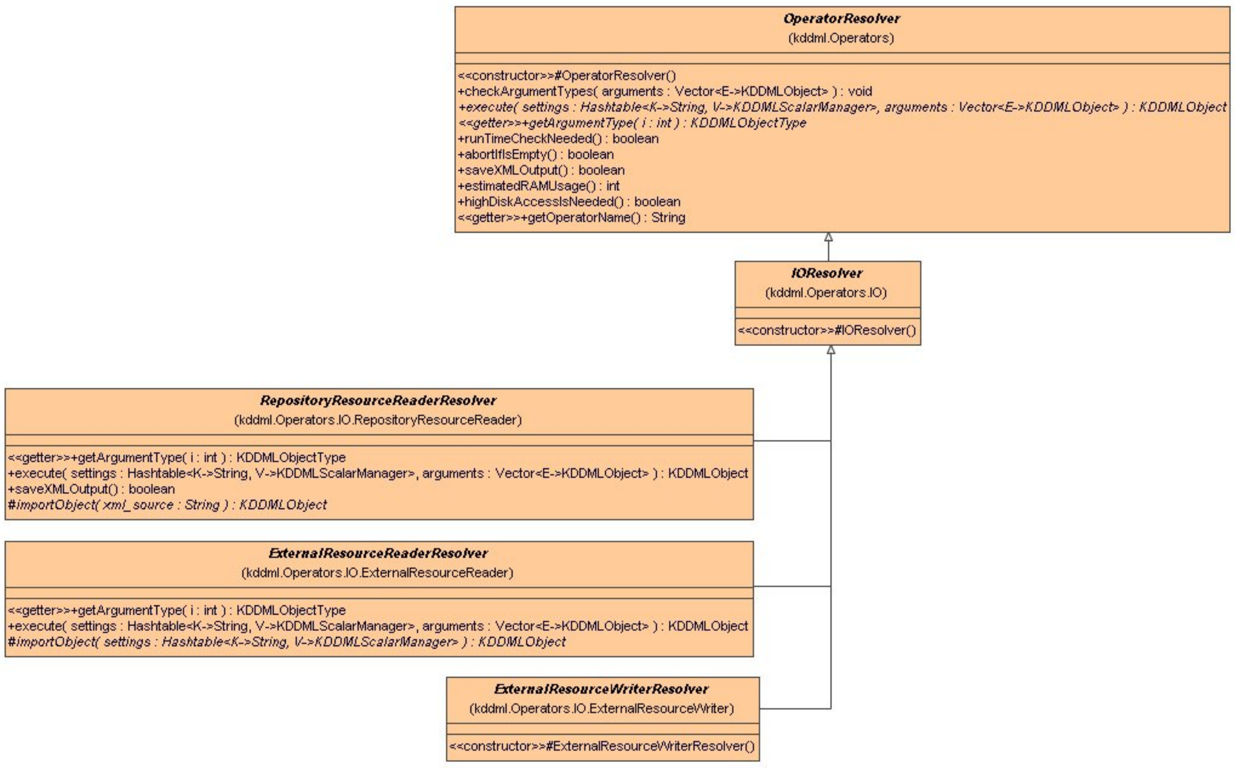

OperatorResolver
(kddml.Operators)
<<constructor>>#OperatorResolver()
+checkArgumentTypes(arguments : Vector<E->KDDMLObject>) : void
+execute(settings : Hashtable<K->String, V->KDDMLScalarManager>, arguments : Vector<E->KDDMLObject>) : KDDMLObject
<<getter>>+getArgumentType(i : int) : KDDMLObjectType
+runTimeCheckNeeded() : boolean
+abortIfIsEmpty() : boolean
+saveXMLOutput() : boolean
+estimatedRAMUsage() : int
+highDiskAccessIsNeeded() : boolean
<<getter>>+getOperatorName() : String
IOResolver
(kddml.Operators.IO)
<<constructor>>#IOResolver()
RepositoryResourceReaderResolver
(kddml.Operators.IO.RepositoryResourceReader)
<<getter>>+getArgumentType(i : int) : KDDMLObjectType
+execute(settings : Hashtable<K->String, V->KDDMLScalarManager>, arguments : Vector<E->KDDMLObject>) : KDDMLObject
+saveXMLOutput() : boolean
#importObject(xml_source : String) : KDDMLObject
ExternalResourceReaderResolver
(kddml.Operators.IO.ExternalResourceReader)
<<getter>>+getArgumentType(i : int) : KDDMLObjectType
+execute(settings : Hashtable<K->String, V->KDDMLScalarManager>, arguments : Vector<E->KDDMLObject>) : KDDMLObject
#importObject(settings : Hashtable<K->String, V->KDDMLScalarManager>) : KDDMLObject
ExternalResourceWriterResolver
(kddml.Operators.IO.ExternalResourceWriter)
<<constructor>>#ExternalResourceWriterResolver()

Figure B.6: Package kddml.Operators.IO - Resolver

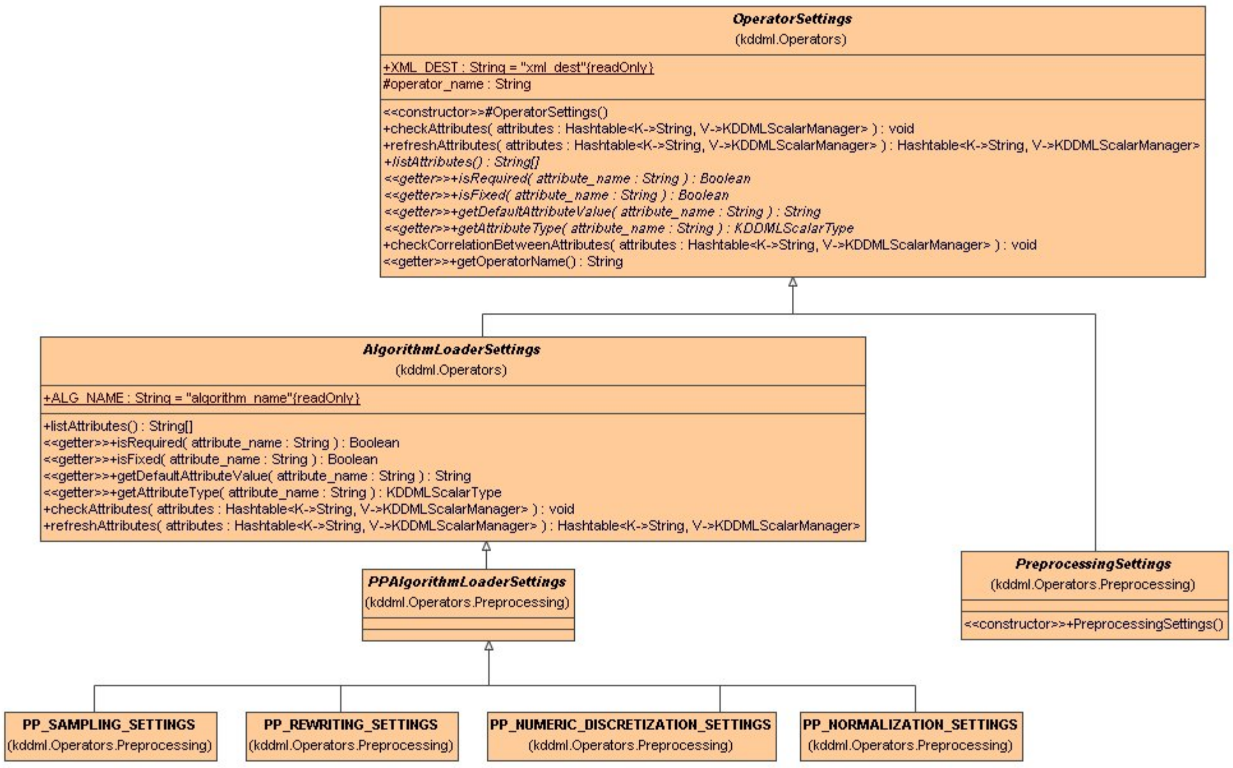

OperatorSettings
(kddml.Operators)
+XML_DEST : String = "xml_dest"{readOnly}
#operator_name : String
<<constructor>>#OperatorSettings()
+checkAttributes(attributes : Hashtable<K->String, V->KDDMLScalarManager>) : void
+refreshAttributes(attributes : Hashtable<K->String, V->KDDMLScalarManager>) : Hashtable<K->String, V->KDDMLScalarManager>
+listAttributes() : String[]
<<getter>>+isRequired(attribute_name : String) : Boolean
<<getter>>+isFixed(attribute_name : String) : Boolean
<<getter>>+getDefaultAttributeValue(attribute_name : String) : String
<<getter>>+getAttributeType(attribute_name : String) : KDDMLScalarType
+checkCorrelationBetweenAttributes(attributes : Hashtable<K->String, V->KDDMLScalarManager>) : void
<<getter>>+getOperatorName() : String
AlgorithmLoaderSettings
(kddml.Operators)
+ALG_NAME : String = "algorithm_name"{readOnly}
+listAttributes() : String[]
<<getter>>+isRequired(attribute_name : String) : Boolean
<<getter>>+isFixed(attribute_name : String) : Boolean
<<getter>>+getDefaultAttributeValue(attribute_name : String) : String
<<getter>>+getAttributeType(attribute_name : String) : KDDMLScalarType
+checkAttributes(attributes : Hashtable<K->String, V->KDDMLScalarManager>) : void
+refreshAttributes(attributes : Hashtable<K->String, V->KDDMLScalarManager>) : Hashtable<K->String, V->KDDMLScalarManager>
PreprocessingSettings
(kddml.Operators.Preprocessing)
<<constructor>>+PreprocessingSettings()
PPAlgorithmLoaderSettings
(kddml.Operators.Preprocessing)
PP_SAMPLING_SETTINGS
(kddml.Operators.Preprocessing)
PP_REWRITING_SETTINGS
(kddml.Operators.Preprocessing)
PP_NUMERIC_DISCRETIZATION_SETTINGS
(kddml.Operators.Preprocessing)
PP_NORMALIZATION_SETTINGS
(kddml.Operators.Preprocessing)

Figure B.7: Package kddml.Operators.Preprocessing - Settings

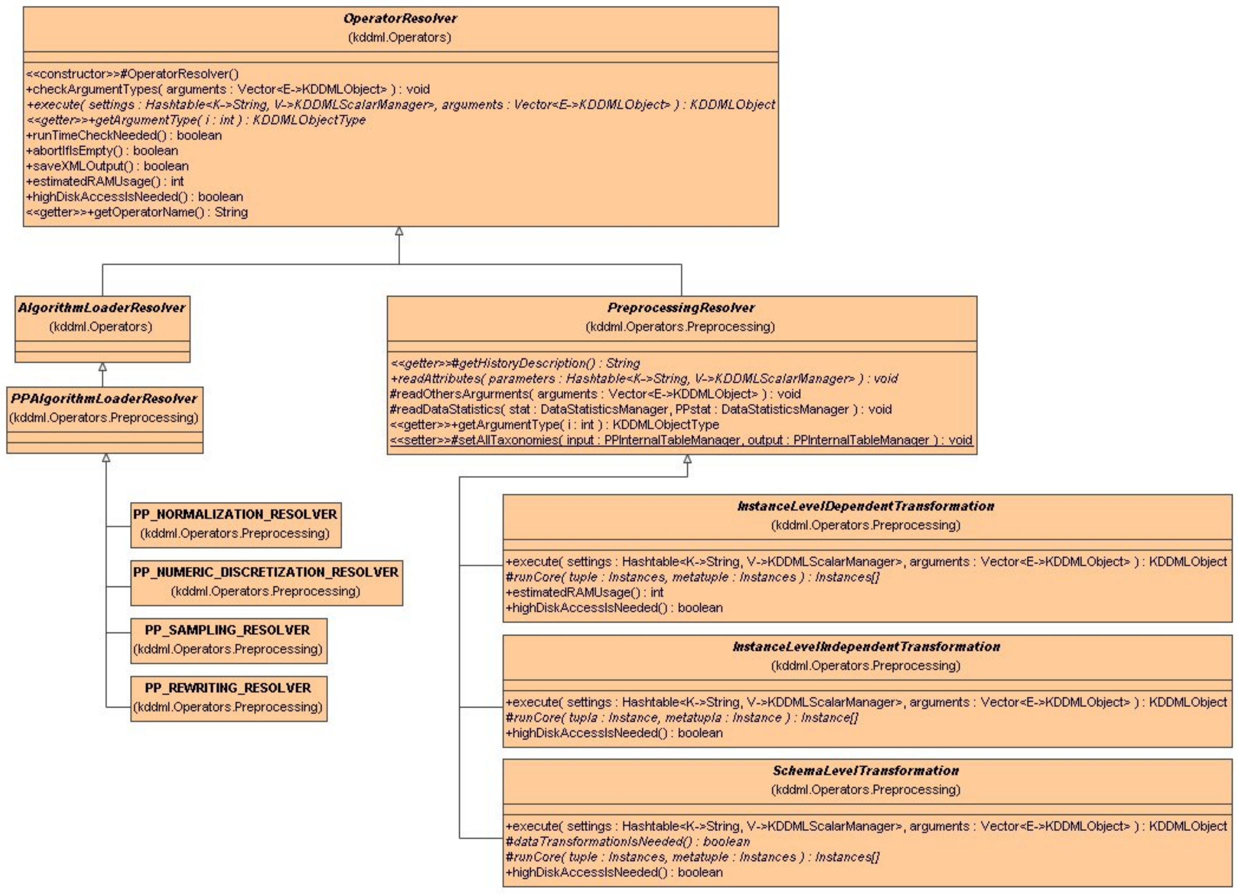

OperatorResolver
(kddml.Operators)
<<constructor>>#OperatorResolver()
+checkArgumentTypes(arguments : Vector<E->KDDMLObject>) : void
+execute(settings : Hashtable<K->String, V->KDDMLScalarManager>, arguments : Vector<E->KDDMLObject>) : KDDMLObject
<<getter>>+getArgumentType(i : int) : KDDMLObjectType
+runTimeCheckNeeded() : boolean
+abortIfIsEmpty() : boolean
+saveXMLOutput() : boolean
+estimatedRAMUsage() : int
+highDiskAccessIsNeeded() : boolean
<<getter>>+getOperatorName() : String
AlgorithmLoaderResolver
(kddml.Operators)
PPAlgorithmLoaderResolver
(kddml.Operators.Preprocessing)
PreprocessingResolver
(kddml.Operators.Preprocessing)
<<getter>>#getHistoryDescription() : String
+readAttributes(parameters : Hashtable<K->String, V->KDDMLScalarManager>) : void
#readOthersArgurments(arguments : Vector<E->KDDMLObject>) : void
#readDataStatistics(stat : DataStatisticsManager, PPstat : DataStatisticsManager) : void
<<getter>>+getArgumentType(i : int) : KDDMLObjectType
<<setter>>#setAllTaxonomies(input : PPInternalTableManager, output : PPInternalTableManager) : void
PP_NORMALIZATION_RESOLVER
(kddml.Operators.Preprocessing)
PP_NUMERIC_DISCRETIZATION_RESOLVER
(kddml.Operators.Preprocessing)
PP_SAMPLING_RESOLVER
(kddml.Operators.Preprocessing)
PP_REWRITING_RESOLVER
(kddml.Operators.Preprocessing)
InstanceLevelDependentTransformation
(kddml.Operators.Preprocessing)
+execute(settings : Hashtable<K->String, V->KDDMLScalarManager>, arguments : Vector<E->KDDMLObject>) : KDDMLObject
#runCore(tuple : Instances, metatuple : Instances) : Instances[]
+estimatedRAMUsage() : int
+highDiskAccessIsNeeded() : boolean
InstanceLevelIndependentTransformation
(kddml.Operators.Preprocessing)
+execute(settings : Hashtable<K->String, V->KDDMLScalarManager>, arguments : Vector<E->KDDMLObject>) : KDDMLObject
#runCore(tupla : Instance, metatupla : Instance) : Instance[]
+highDiskAccessIsNeeded() : boolean
SchemaLevelTransformation
(kddml.Operators.Preprocessing)
+execute(settings : Hashtable<K->String, V->KDDMLScalarManager>, arguments : Vector<E->KDDMLObject>) : KDDMLObject
#dataTransformationIsNeeded() : boolean
#runCore(tuple : Instances, metatuple : Instances) : Instances[]
+highDiskAccessIsNeeded() : boolean

Figure B.8: Package kddml.Operators.Preprocessing - Resolver

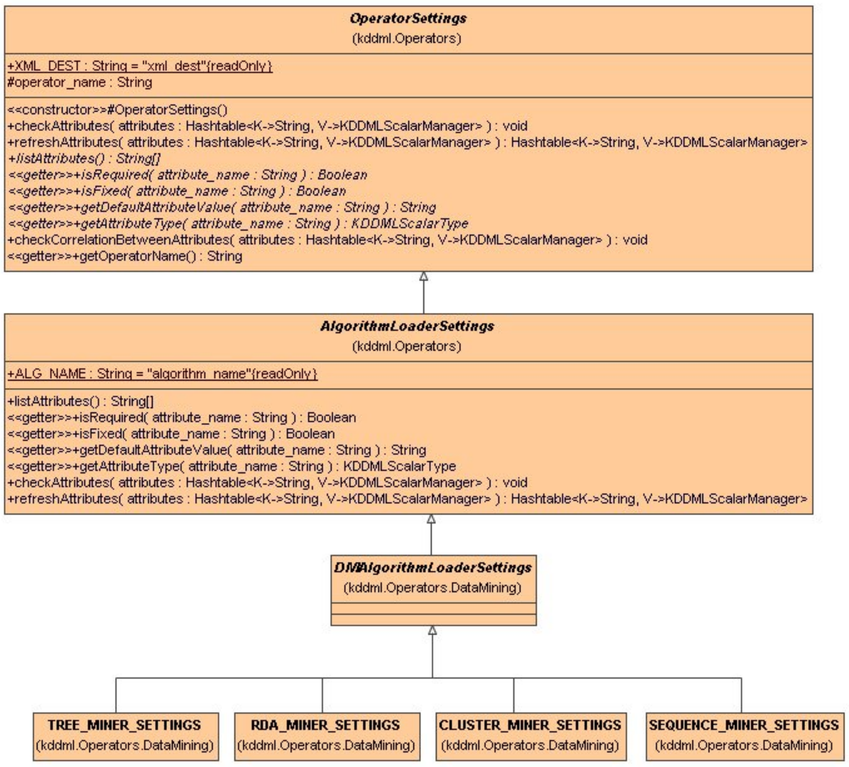

OperatorSettings
(kddml.Operators)
+XML_DEST : String = "xml_dest"{readOnly}
#operator_name : String
<<constructor>>#OperatorSettings()
+checkAttributes(attributes : Hashtable<K->String, V->KDDMLScalarManager>) : void
+refreshAttributes(attributes : Hashtable<K->String, V->KDDMLScalarManager>) : Hashtable<K->String, V->KDDMLScalarManager>
+listAttributes() : String[]
<<getter>>+isRequired(attribute_name : String) : Boolean
<<getter>>+isFixed(attribute_name : String) : Boolean
<<getter>>+getDefaultAttributeValue(attribute_name : String) : String
<<getter>>+getAttributeType(attribute_name : String) : KDDMLScalarType
+checkCorrelationBetweenAttributes(attributes : Hashtable<K->String, V->KDDMLScalarManager>) : void
<<getter>>+getOperatorName() : String
AlgorithmLoaderSettings
(kddml.Operators)
+ALG_NAME : String = "algorithm_name"{readOnly}
+listAttributes() : String[]
<<getter>>+isRequired(attribute_name : String) : Boolean
<<getter>>+isFixed(attribute_name : String) : Boolean
<<getter>>+getDefaultAttributeValue(attribute_name : String) : String
<<getter>>+getAttributeType(attribute_name : String) : KDDMLScalarType
+checkAttributes(attributes : Hashtable<K->String, V->KDDMLScalarManager>) : void
+refreshAttributes(attributes : Hashtable<K->String, V->KDDMLScalarManager>) : Hashtable<K->String, V->KDDMLScalarManager>
DMAlgorithmLoaderSettings
(kddml.Operators.DataMining)
TREE_MINER_SETTINGS
(kddml.Operators.DataMining)
RDA_MINER_SETTINGS
(kddml.Operators.DataMining)
CLUSTER_MINER_SETTINGS
(kddml.Operators.DataMining)
SEQUENCE_MINER_SETTINGS
(kddml.Operators.DataMining)

Figure B.9: Package kddml.Operators.DataMining - Settings

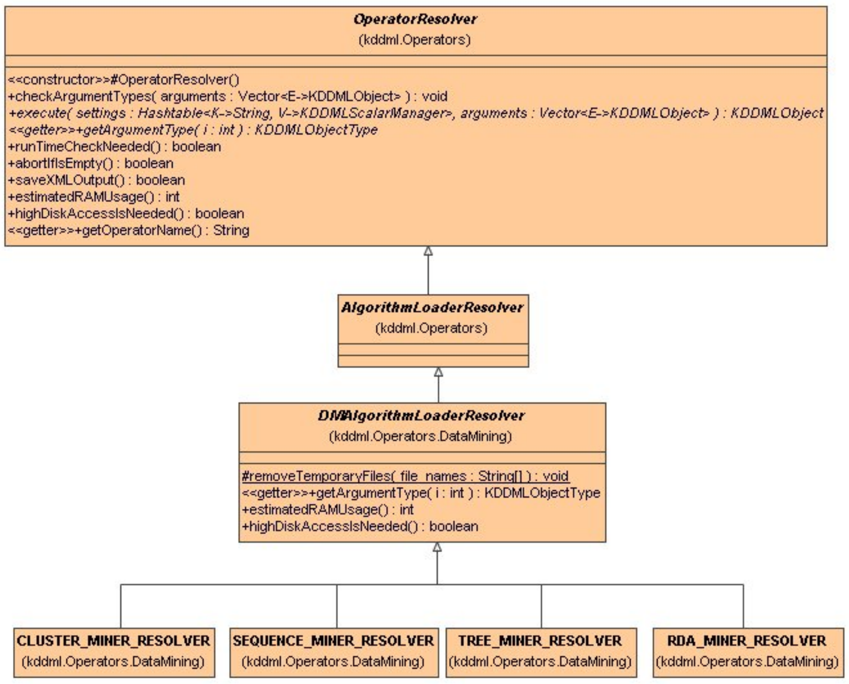

OperatorResolver
(kddml.Operators)
<<constructor>>#OperatorResolver()
+checkArgumentTypes(arguments : Vector<E->KDDMLObject>) : void
+execute(settings : Hashtable<K->String, V->KDDMLScalarManager>, arguments : Vector<E->KDDMLObject>) : KDDMLObject
<<getter>>+getArgumentType(i : int) : KDDMLObjectType
+runTimeCheckNeeded() : boolean
+abortIfIsEmpty() : boolean
+saveXMLOutput() : boolean
+estimatedRAMUsage() : int
+highDiskAccessIsNeeded() : boolean
<<getter>>+getOperatorName() : String
AlgorithmLoaderResolver
(kddml.Operators)
DMAlgorithmLoaderResolver
(kddml.Operators.DataMining)
#removeTemporaryFiles(file_names : String[]) : void
<<getter>>+getArgumentType(i : int) : KDDMLObjectType
+estimatedRAMUsage() : int
+highDiskAccessIsNeeded() : boolean
CLUSTER_MINER_RESOLVER
(kddml.Operators.DataMining)
SEQUENCE_MINER_RESOLVER
(kddml.Operators.DataMining)
TREE_MINER_RESOLVER
(kddml.Operators.DataMining)
RDA_MINER_RESOLVER
(kddml.Operators.DataMining)

Figure B.10: Package kddml.Operators.DataMining - Resolver

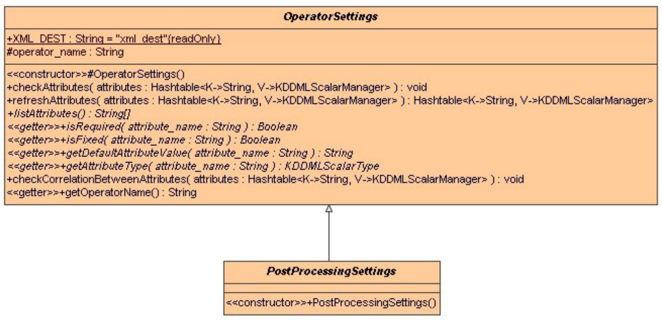

OperatorSettings
+XML_DEST : String = "xml_dest"{readOnly}
#operator_name : String
<<constructor>>#OperatorSettings()
+checkAttributes(attributes : Hashtable<K->String, V->KDDMLScalarManager>) : void
+refreshAttributes(attributes : Hashtable<K->String, V->KDDMLScalarManager>) : Hashtable<K->String, V->KDDMLScalarManager>
+listAttributes() : String[]
<<getter>>+isRequired(attribute_name : String) : Boolean
<<getter>>+isFixed(attribute_name : String) : Boolean
<<getter>>+getDefaultAttributeValue(attribute_name : String) : String
<<getter>>+getAttributeType(attribute_name : String) : KDDMLScalarType
+checkCorrelationBetweenAttributes(attributes : Hashtable<K->String, V->KDDMLScalarManager>) : void
<<getter>>+getOperatorName() : String
PostProcessingSettings
<<constructor>>+PostProcessingSettings()

Figure B.11: Package kddml.Operators.Postprocessing - Settings

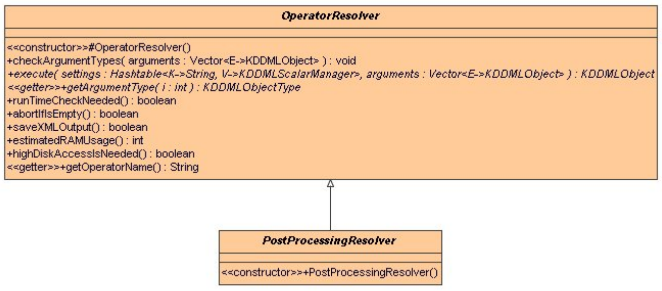

OperatorResolver
<<constructor>>#OperatorResolver()
+checkArgumentTypes(arguments : Vector<E->KDDMLObject>) : void
+execute(settings : Hashtable<K->String, V->KDDMLScalarManager>, arguments : Vector<E->KDDMLObject>) : KDDMLObject
<<getter>>+getArgumentType(i : int) : KDDMLObjectType
+runTimeCheckNeeded() : boolean
+abortIfIsEmpty() : boolean
+saveXMLOutput() : boolean
+estimatedRAMUsage() : int
+highDiskAccessIsNeeded() : boolean
<<getter>>+getOperatorName() : String
PostProcessingResolver
<<constructor>>+PostProcessingResolver()

Figure B.12: Package kddml.Operators.Postprocessing - Resolver

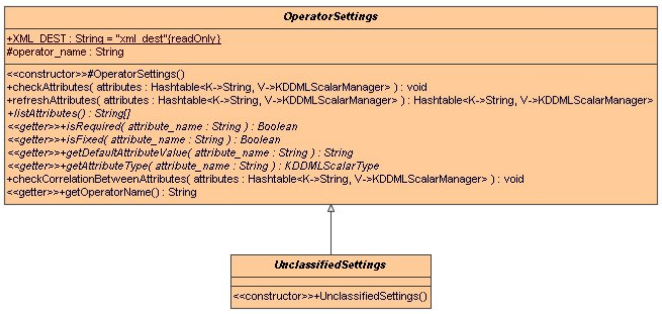

OperatorSettings
+XML_DEST : String = "xml_dest"{readOnly}
#operator_name : String
<<constructor>>#OperatorSettings()
+checkAttributes(attributes : Hashtable<K->String, V->KDDMLScalarManager>) : void
+refreshAttributes(attributes : Hashtable<K->String, V->KDDMLScalarManager>) : Hashtable<K->String, V->KDDMLScalarManager>
+listAttributes() : String[]
<<getter>>+isRequired(attribute_name : String) : Boolean
<<getter>>+isFixed(attribute_name : String) : Boolean
<<getter>>+getDefaultAttributeValue(attribute_name : String) : String
<<getter>>+getAttributeType(attribute_name : String) : KDDMLScalarType
+checkCorrelationBetweenAttributes(attributes : Hashtable<K->String, V->KDDMLScalarManager>) : void
<<getter>>+getOperatorName() : String
UnclassifiedSettings
<<constructor>>+UnclassifiedSettings()

Figure B.13: Package kddml.Operators.Unclassified - Settings

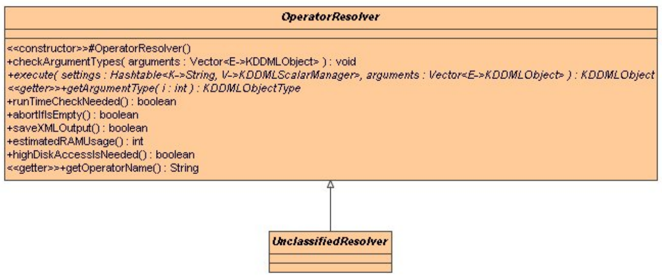

OperatorResolver
<<constructor>>#OperatorResolver()
+checkArgumentTypes(arguments : Vector<E->KDDMLObject>) : void
+execute(settings : Hashtable<K->String, V->KDDMLScalarManager>, arguments : Vector<E->KDDMLObject>) : KDDMLObject
<<getter>>+getArgumentType(i : int) : KDDMLObjectType
+runTimeCheckNeeded() : boolean
+abortIfIsEmpty() : boolean
+saveXMLOutput() : boolean
+estimatedRAMUsage() : int
+highDiskAccessIsNeeded() : boolean
<<getter>>+getOperatorName() : String
UnclassifiedResolver

Figure B.14: Package kddml.Operators.Unclassified - Resolver

APPENDIX C

UML interpreter layer diagrams

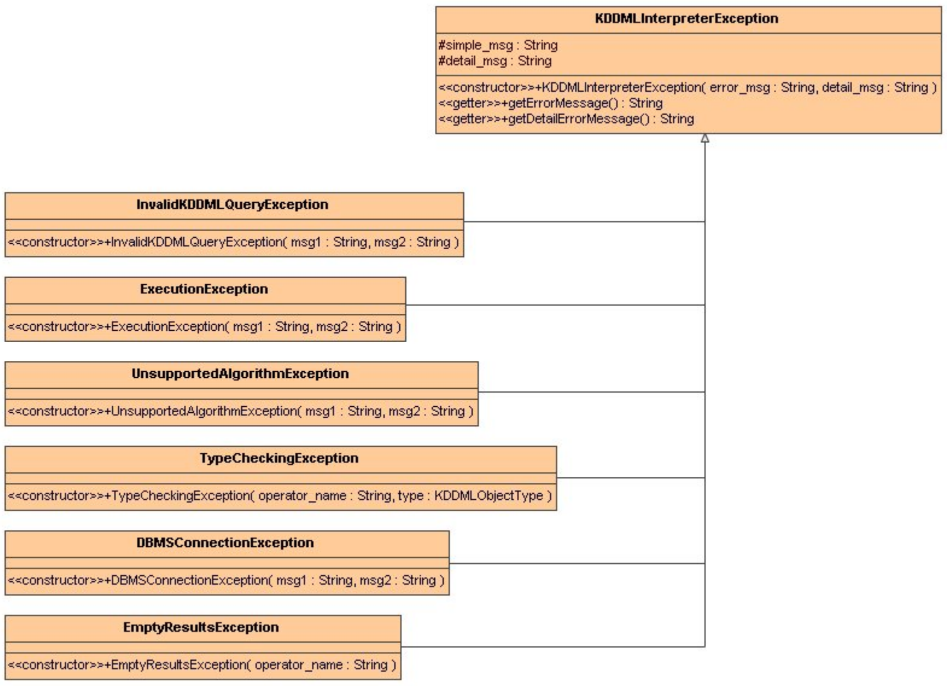

KDDMLInterpreterException
#simple_msg : String
#detail_msg : String
<<constructor>>+KDDMLInterpreterException(error_msg : String, detail_msg : String)
<<getter>>+getErrorMessage() : String
<<getter>>+getDetailErrorMessage() : String
InvalidKDDMLQueryException
<<constructor>>+InvalidKDDMLQueryException(msg1 : String, msg2 : String)
ExecutionException
<<constructor>>+ExecutionException(msg1 : String, msg2 : String)
UnsupportedAlgorithmException
<<constructor>>+UnsupportedAlgorithmException(msg1 : String, msg2 : String)
TypeCheckingException
<<constructor>>+TypeCheckingException(operator_name : String, type : KDDMLObjectType)
DBMSConnectionException
<<constructor>>+DBMSConnectionException(msg1 : String, msg2 : String)
EmptyResultsException
<<constructor>>+EmptyResultsException(operator_name : String)

Figure C.1: Package kddml.Interpreter - Exceptions

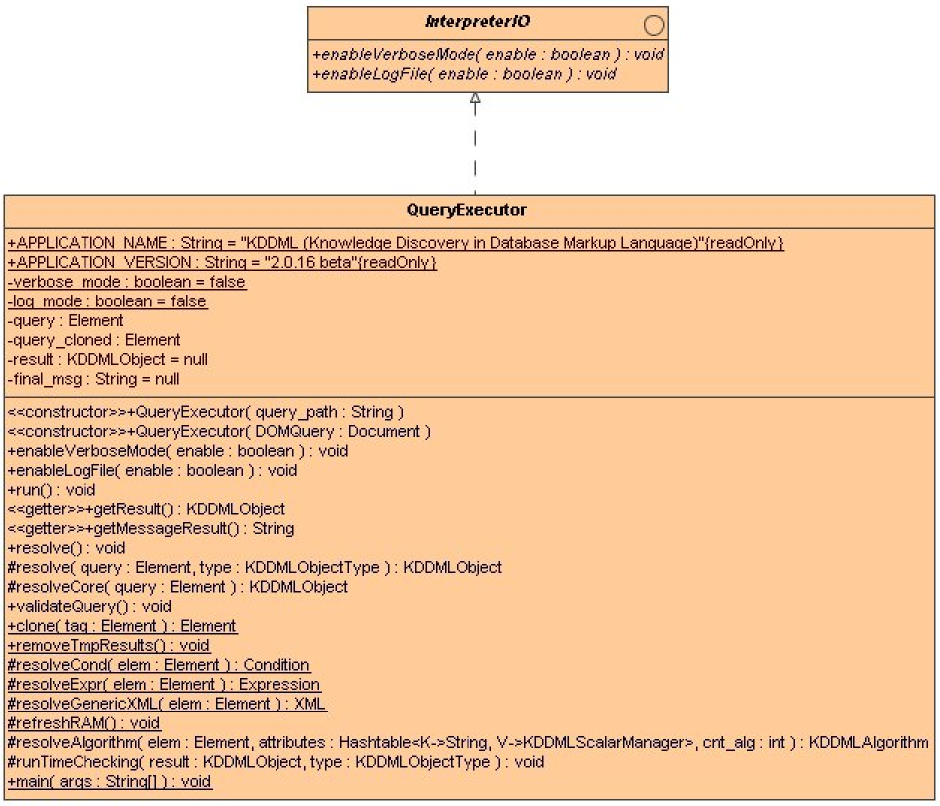

InterpreterIO
+enableVerboseMode(enable : boolean) : void
+enableLogFile(enable : boolean) : void
QueryExecutor
+APPLICATION_NAME : String = "KDDML (Knowledge Discovery in Database Markup Language)"{readOnly}
+APPLICATION_VERSION : String = "2.0.16 beta"{readOnly}
-verbose_mode : boolean = false
-log_mode : boolean = false
-query : Element
-query_cloned : Element
-result : KDDMLObject = null
-final_msg : String = null
<<constructor>>+QueryExecutor(query_path : String)
<<constructor>>+QueryExecutor(DOMQuery : Document)
+enableVerboseMode(enable : boolean) : void
+enableLogFile(enable : boolean) : void
+run() : void
<<getter>>+getResult() : KDDMLObject
<<getter>>+getMessageResult() : String
+resolve() : void
#resolve(query : Element, type : KDDMLObjectType) : KDDMLObject
#resolveCore(query : Element) : KDDMLObject
+validateQuery() : void
+clone(tag : Element) : Element
+removeTmpResults() : void
#resolveCond(elem : Element) : Condition
#resolveExpr(elem : Element) : Expression
#resolveGenericXML(elem : Element) : XML
#refreshRAM() : void
#resolveAlgorithm(elem : Element, attributes : Hashtable<K->String, V->KDDMLScalarManager>, cnt_alg : int) : KDDMLAlgorithm
#runTimeChecking(result : KDDMLObject, type : KDDMLObjectType) : void
+main(args : String[]) : void

Figure C.2: Package kddml.Interpreter - I/O

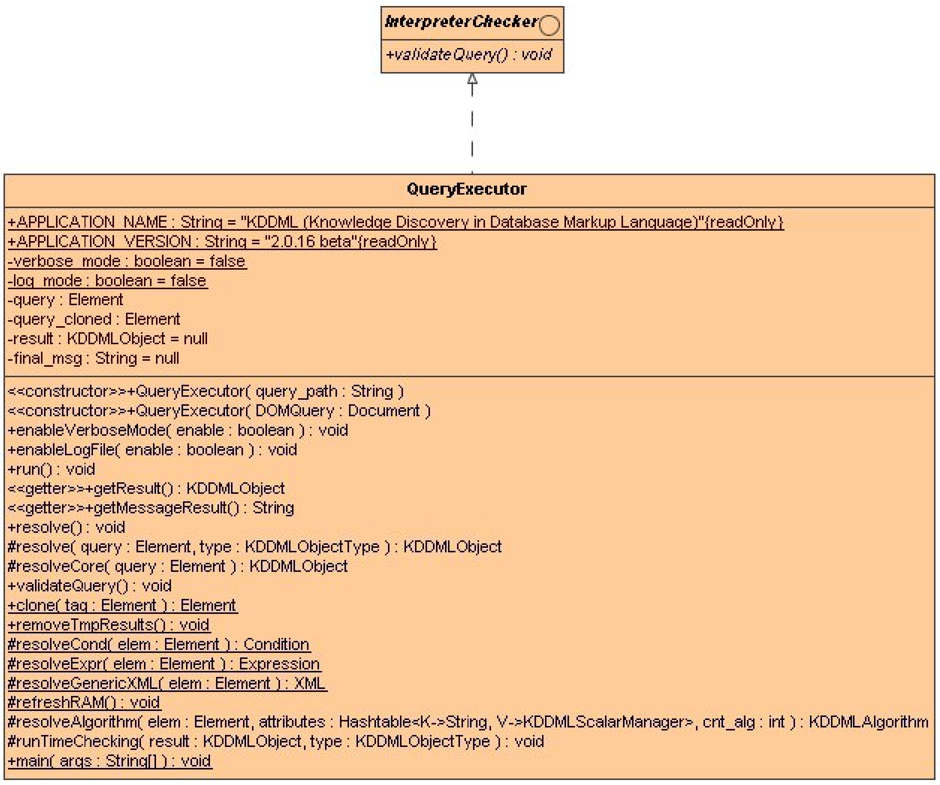

InterpreterChecker
+validateQuery() : void
QueryExecutor
+APPLICATION_NAME : String = "KDDML (Knowledge Discovery in Database Markup Language)"{readOnly}
+APPLICATION_VERSION : String = "2.0.16 beta"{readOnly}
-verbose_mode : boolean = false
-log_mode : boolean = false
-query : Element
-query_cloned : Element
-result : KDDMLObject = null
-final_msg : String = null
<<constructor>>+QueryExecutor(query_path : String)
<<constructor>>+QueryExecutor(DOMQuery : Document)
+enableVerboseMode(enable : boolean) : void
+enableLogFile(enable : boolean) : void
+run() : void
<<getter>>+getResult() : KDDMLObject
<<getter>>+getMessageResult() : String
+resolve() : void
#resolve(query : Element, type : KDDMLObjectType) : KDDMLObject
#resolveCore(query : Element) : KDDMLObject
+validateQuery() : void
+clone(tag : Element) : Element
+removeTmpResults() : void
#resolveCond(elem : Element) : Condition
#resolveExpr(elem : Element) : Expression
#resolveGenericXML(elem : Element) : XML
#refreshRAM() : void
#resolveAlgorithm(elem : Element, attributes : Hashtable<K->String, V->KDDMLScalarManager>, cnt_alg : int) : KDDMLAlgorithm
#runTimeChecking(result : KDDMLObject, type : KDDMLObjectType) : void
+main(args : String[]) : void

Figure C.3: Package kddml.Interpreter - Checker

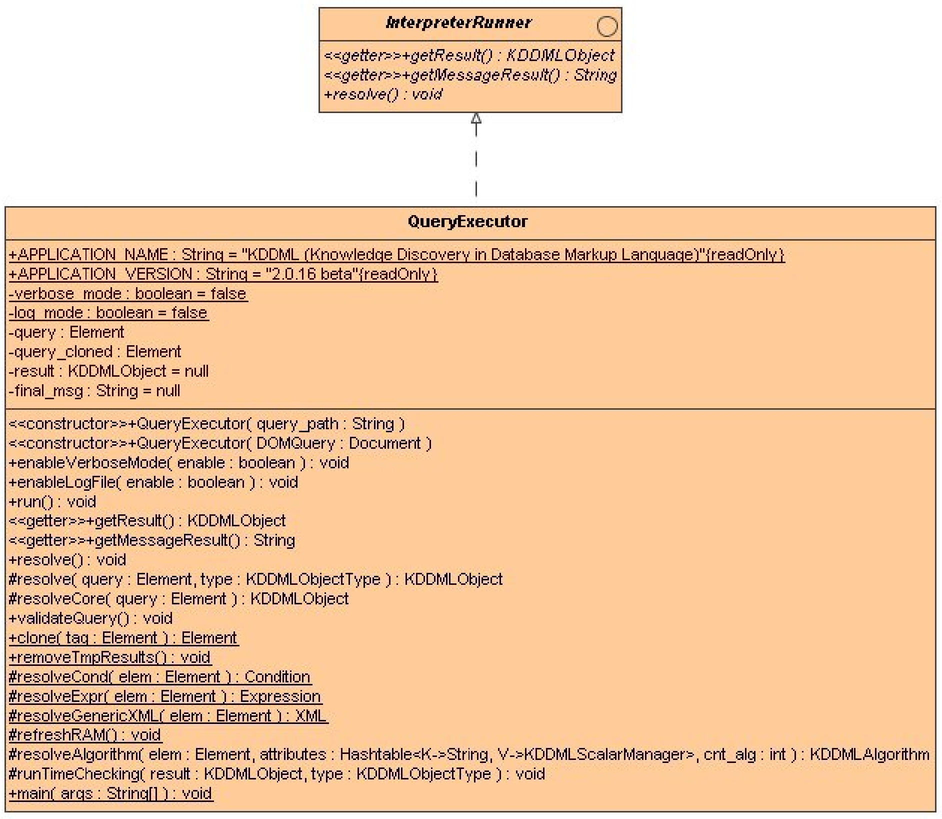

InterpreterRunner
<<getter>>+getResult() : KDDMLObject
<<getter>>+getMessageResult() : String
+resolve() : void
QueryExecutor
+APPLICATION_NAME : String = "KDDML (Knowledge Discovery in Database Markup Language)"{readOnly}
+APPLICATION_VERSION : String = "2.0.16 beta"{readOnly}
-verbose_mode : boolean = false
-log_mode : boolean = false
-query : Element
-query_cloned : Element
-result : KDDMLObject = null
-final_msg : String = null
<<constructor>>+QueryExecutor(query_path : String)
<<constructor>>+QueryExecutor(DOMQuery : Document)
+enableVerboseMode(enable : boolean) : void
+enableLogFile(enable : boolean) : void
+run() : void
<<getter>>+getResult() : KDDMLObject
<<getter>>+getMessageResult() : String
+resolve() : void
#resolve(query : Element, type : KDDMLObjectType) : KDDMLObject
#resolveCore(query : Element) : KDDMLObject
+validateQuery() : void
+clone(tag : Element) : Element
+removeTmpResults() : void
#resolveCond(elem : Element) : Condition
#resolveExpr(elem : Element) : Expression
#resolveGenericXML(elem : Element) : XML
#refreshRAM() : void
#resolveAlgorithm(elem : Element, attributes : Hashtable<K->String, V->KDDMLScalarManager>, cnt_alg : int) : KDDMLAlgorithm
#runTimeChecking(result : KDDMLObject, type : KDDMLObjectType) : void
+main(args : String[]) : void

Figure C.4: Package kddml.Interpreter - Executor