Enabling End-User Datawarehouse Mining
Contract No. IST-1999-11993
Deliverable No. D12.4

The Concept Editor

A Description of Part of the Mining Mart HCI Deliverable D12.4

Erik Darwinkel and Olaf Rem

Perot Systems Nederland B.V.
P.O. box 2729, NL-3800 GG Amersfoort, The Netherlands
{Erik.Darwinkel, Olaf.Rem}@ps.net

December 20, 2002

Abstract

The main objective for workpackage 12 is to deliver a Human Computer Interface for the Mining Mart system and to integrate all components. The Concept Editor is part of the HCI. It allows to create and manipulate concepts and connect them to the business data. These concepts are inputs for preprocessing operators that can be specified using the Chain Editor.

In this report the functional requirements for the Concept Editor are described; the design and implementation is discussed and information is provided about using the Concept Editor.

We conclude that the current version of the Concept Editor fulfills the functional requirements. Some functionality can be made more user-friendly or more scalable. On the whole, however, the Concept Editor does what it is supposed to do and allows to conveniently work with concepts.

Contents

Chapter 1

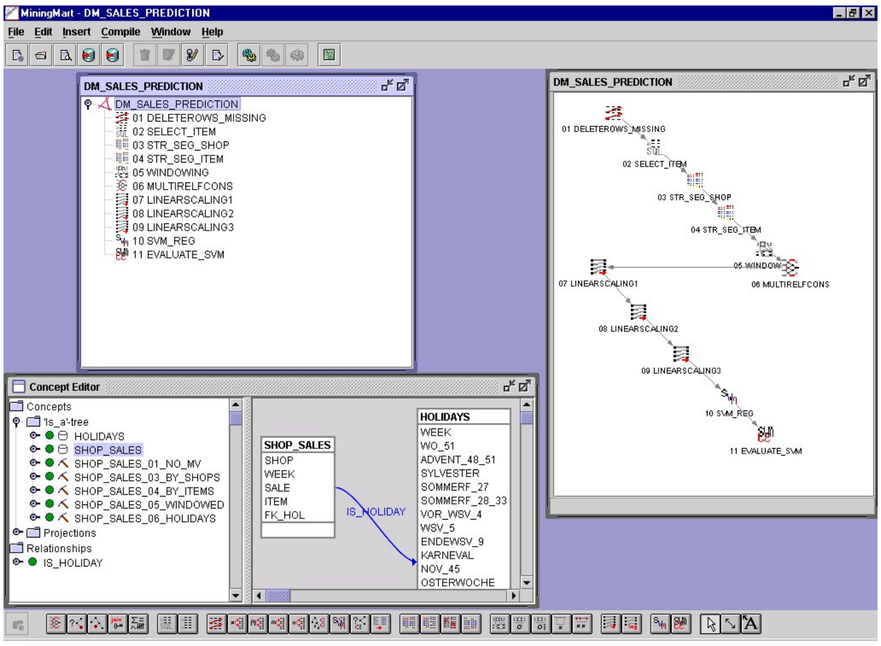
Introduction

The main objective for the Mining Mart system (see Figure 1.1) is to provide a user-friendly interface for enhanced preprocessing of data for a knowledge discovery task. The system architecture (see Figure 1.2) consists of several components of which the Concept Editor is one. The other major components are: Chain Editor, Compiler, Mining Mart Meta Model (M4) Schema, M4 Interface, and Business Data Schema. The heart of the Mining Mart system is the M4. It stores meta information about preprocessing steps and data. The M4 Interface provides a Java object interface to access the M4. The Concept Editor and Chain Editor act closely together and are both part of the Mining Mart system HCI (human computer interface). They both use the M4 Interface to manipulate the M4. For the end user they provide a user-friendly way to work with the meta data. The Concept Editor allows users to work with meta data about business data. The user needs this information when working with the Chain Editor for defining preprocessing steps. The Compiler manages the execution of preprocessing steps. It triggers operators and writes the results back in the M4.

The objective for workpackage 12 is to design and implement the Concept Editor, Chain Editor and Compiler and integrate all components into a working system. This report focuses specifically on the design, implementation and use of the Concept Editor.

There are various other sources available that provide more information about the Mining Mart project and the Mining Mart system. A good place to start is the Mining Mart website[1] which offers a good overview of the available documentation. Here also many documents can be downloaded directly. The Mining Mart approach is described in: [MoS02], [ZKV01] and [KZV00]. Further information about the Mining Mart system can be found in: [PeF02] (requirements, system overview), [Eul02] (Compiler and operators), [LaR02] (M4Interface), [VKZ01] (the Mining Mart Meta Model), [Zu01b] (Compiler) and [Zu01a] (M4 schema).

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


MiningMart - DM_SALES_PREDICTION
File Edit Insert Compile Window Help
Concept Editor
IS_HOLIDAY

DM_SALES_PREDICTION
- DM_SALES_PREDICTION
  - 01 DELETEROWS_MISSING
  - 02 SELECT_ITEM
  - 03 STR_SEG_SHOP
  - 04 STR_SEG_ITEM
  - 05 WINDOWING
  - 06 MULTIRELFCONS
  - 07 LINEARSCALING1
  - 08 LINEARSCALING2
  - 09 LINEARSCALING3
  - 10 SVM_REG
  - 11 EVALUATE_SVM

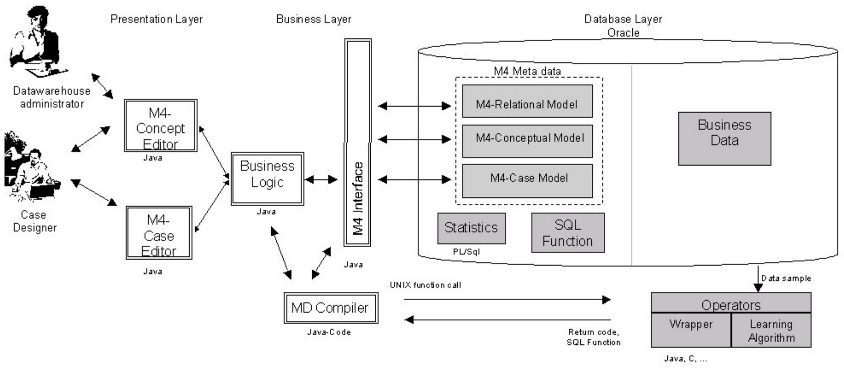
Concepts
- 'Is_a'-tree
  - HOLIDAYS
  - SHOP_SALES
  - SHOP_SALES_01_NO_MV
  - SHOP_SALES_03_BY_SHOPS
  - SHOP_SALES_04_BY_ITEMS
  - SHOP_SALES_05_WINDOWED
  - SHOP_SALES_06_HOLIDAYS
- Projections
- Relationships
  - IS_HOLIDAY

SHOP_SALES
SHOP
WEEK
SALE
ITEM
FK_HOL

HOLIDAYS
WEEK
WO_51
ADVENT_48_51
SYLVESTER
SOMMERF_27
SOMMERF_28_33
VOR_WSV_4
WSV_5
ENDEWSV_9
KARNEVAL
NOV_45
OSTERWOCHE

This screen shot of the Mining Mart HCI (human computer interface) depicts three internal windows. The upper and right windows form the Chain Editor; the lower left window shows the Concept Editor.

Figure 1.1: Screen shot of the Mining Mart HCI.



Schematic view of the Mining Mart components. The Concept Editor and Case Editor are part of the HCI. The M4 Interface provides a Java object interface to the M4 and is divided over the client (Java Swing) and the application server (JBoss). The Compiler (Java RMI server) executes operators and creates resulting tables and views. The database (Oracle) contains the M4 and the business data.

Figure 1.2: Mining Mart components.

Chapter 2

Concept Editor Architecture, Design, and Implementation

In this chapter an overview will be given of the architecture of the concept editor and we will focus on some aspects of the design and implementation.

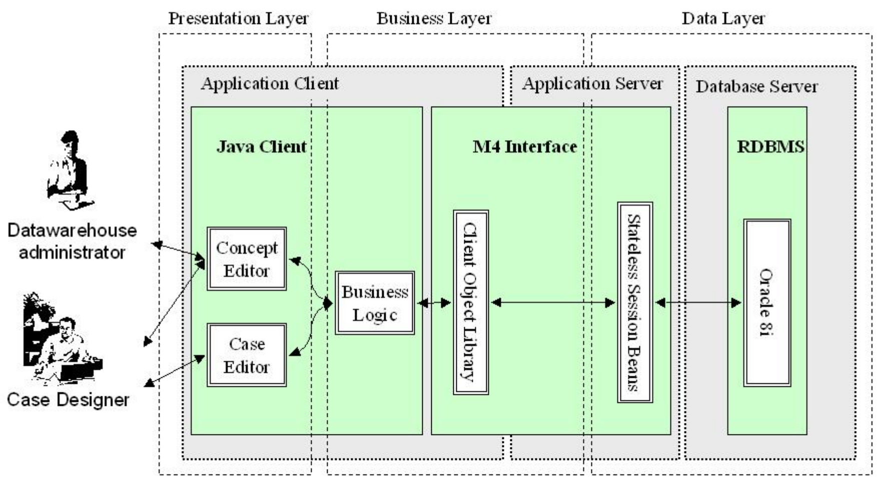
2.1 Architecture

The Concept Editor and the Case Editor are part of the presentation layer (see Figure 2.1). The business logic is part of the business layer and handles the communication of the presentation layer with the database layer and the Compiler. The M4 interface forms a buffer between the business logic and the database. It provides methods for creating, updating, deleting, and finding information in an M4 instance.

Figure 2.1 presents the architectural view, showing the three tier model superimposed on the major Mining Mart components. It shows how the different components are distributed over the client, the application server and the database server. The M4 Interface consists of two parts: the Client Object Library (COL) and several session beans. The COL abstracts the data centric view used in the data layer for the application client and hides the communication with the application server. Further "down" into the data layer Session Beans are used to provide access to the data stored in the database.

2.2 Design and Implementation

The Concept Editor has been implemented using Java. For the graphical components the Swing components were used. The Concept Editor consists of many Java classes that work together to provide the required functionality. All these classes reside within the



The figure shows a conceptual view of the Mining Mart architecture. The case designer and datawarehouse administrator use the Mining Mart HCI. The HCI consists of the Chain Editor and the Concept Editor, which are both part of the presentation layer. The Concept Editor also contains some business logic and access the database through the Client Object Interface. The COL provides an object interface and shields the data centric view from the client.

Figure 2.1: Mining Mart architecture.

`com.syllogic.miningmart.concepteditor`

package and its subpackages. The main component is the ConceptEditor class. For most of the functionality it is the starting point. Other classes are grouped logically together in subpackages. In Table 2.1 an overview is presented of the packages used for the Concept Editor.

Package Name	**Description**
db	Utilities for database access.
diagram	Utilities for master detail view.
dialogs	Contains all dialogs.
event	Contains Concept Editor events.
images	Contains images for icons.
tree	Utilities for the concept and relation tree.
util	General utilities.
wizard	Wizard for creating many to many relation between Concept and business data table.

Table 2.1: Overview of packages used for Concept Editor

The classes have been documented using the Java documentation features. Using the standard javadoc tool an API has been produces in HTML form. For further detailed information about the classes we refer to this API. Figure 2.4 shows a screen shot of the API.

For two use cases we have created sequence diagrams to illustrate how some of the classes work together in providing functionality. The first use case is to create a new concept. It is illustrated in Figure 2.2.

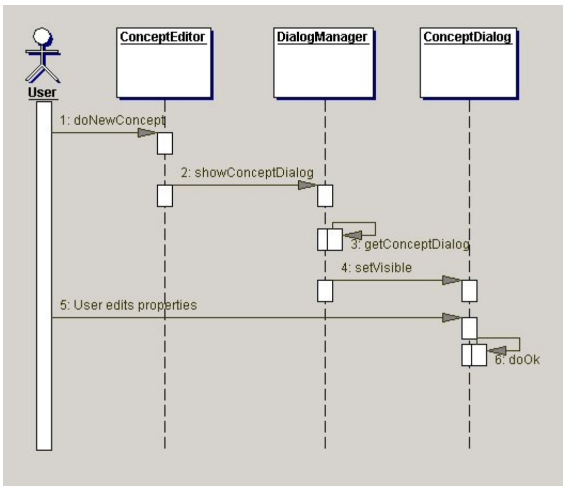
The second use case is connecting a concept to the business data. This involves quite a few classes as can be seen from the sequence diagram in Figure 2.3.

2.3 Recommendations for Future Development

Although all the functional requirements for the Concept Editor have been implemented (at least to a certain level), there is room for improvement. These are mainly improvements in the areas of increased user-friendliness and better scalability. Points for improvement are:

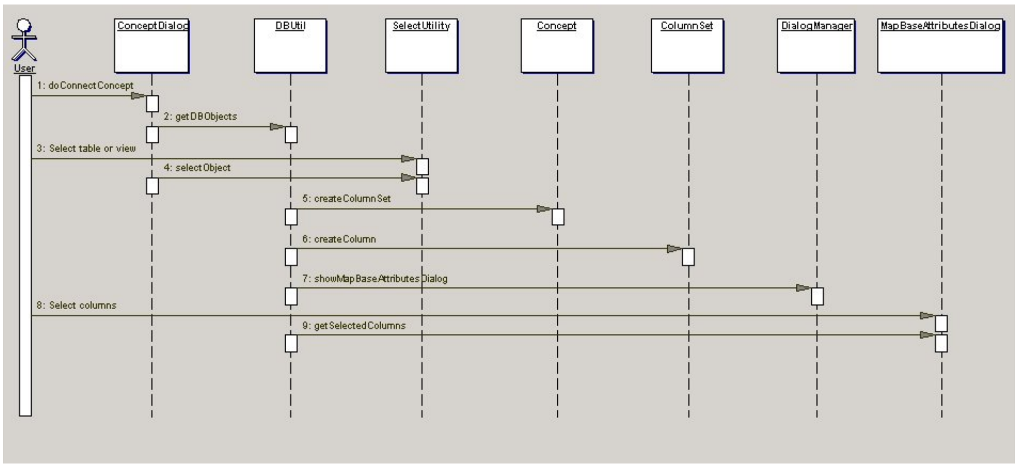
- Requests have been made to add an option to the Concept Editor to automatically create new concepts and base attributes given one or more tables. This would be especially practical for concepts with many base attributes.

- Further the mapping of base attributes to columns could be made more user-friendly. Currently this mapping cannot be edited or viewed after it has been defined. The only way to change it, is by defining it once more.



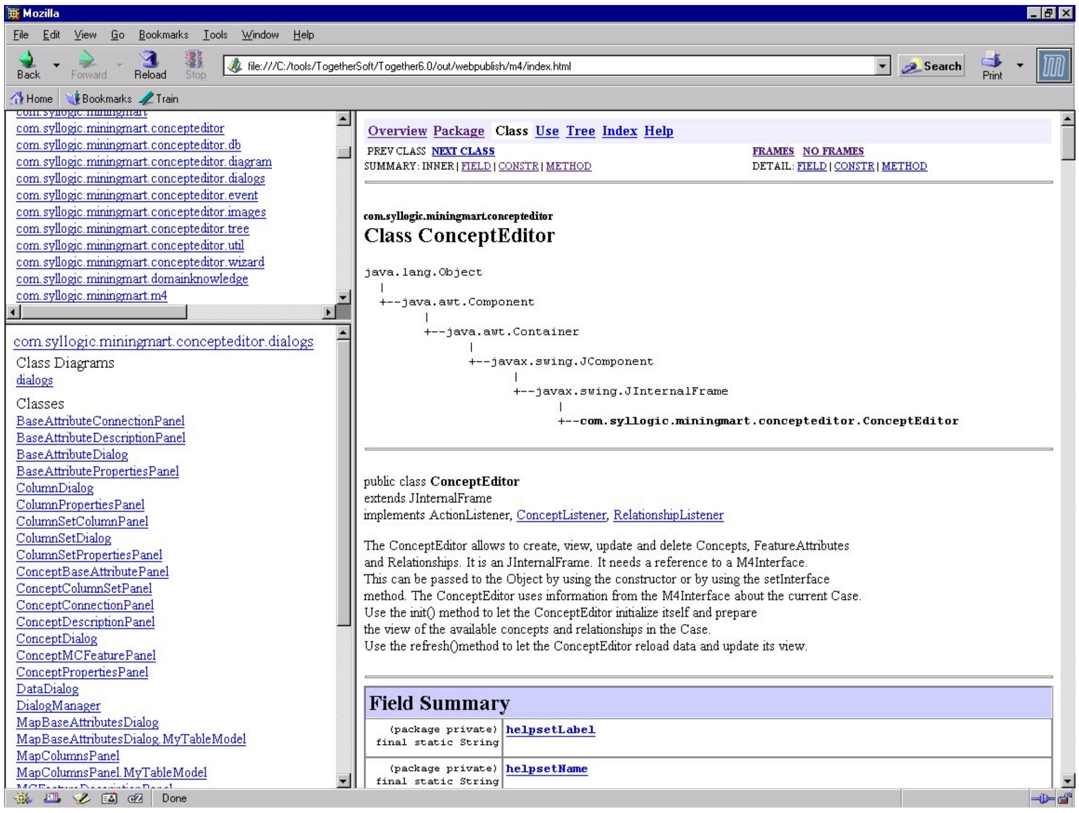
The figure shows which main classes are involved in creating a Concept. The method names are shown that are called sequentially on the different objects.

Figure 2.2: Create Concept sequence diagram.



The diagram shows which classes are involved in connecting a Concept to the business data. The method names that are called on the objects are also shown in the diagram.

Figure 2.3: Connect Concept sequence diagram.


Mozilla
File Edit View Go Bookmarks Tools Window Help
Back Forward Reload Stop
file:///C:/tools/TogetherSoft/Together6.0/out/webpublish/m4/index.html
Search
Print
Home Bookmarks Train
Done

com.syllogic.miningmart.concepteditor
com.syllogic.miningmart.concepteditor.db
com.syllogic.miningmart.concepteditor.diagram
com.syllogic.miningmart.concepteditor.dialogs
com.syllogic.miningmart.concepteditor.event
com.syllogic.miningmart.concepteditor.images
com.syllogic.miningmart.concepteditor.tree
com.syllogic.miningmart.concepteditor.util
com.syllogic.miningmart.concepteditor.wizard
com.syllogic.miningmart.domainknowledge
com.syllogic.miningmart.m4

com.syllogic.miningmart.concepteditor.dialogs

Class Diagrams
dialogs

Classes
BaseAttributeConnectionPanel
BaseAttributeDescriptionPanel
BaseAttributeDialog
BaseAttributePropertiesPanel
ColumnDialog
ColumnPropertiesPanel
ColumnSetColumnPanel
ColumnSetDialog
ColumnSetPropertiesPanel
ConceptBaseAttributePanel
ConceptColumnSetPanel
ConceptConnectionPanel
ConceptDescriptionPanel
ConceptDialog
ConceptMCFeaturePanel
ConceptPropertiesPanel
DataDialog
DialogManager
MapBaseAttributesDialog
MapBaseAttributesDialog.MyTableModel
MapColumnsPanel
MapColumnsPanel.MyTableModel

Overview Package **Class** Use Tree Index Help
PREV CLASS **NEXT CLASS** FRAMES NO FRAMES
SUMMARY: INNER | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

com.syllogic.miningmart.concepteditor
Class ConceptEditor

```
java.lang.Object
  |
  +--java.awt.Component
        |
        +--java.awt.Container
              |
              +--javax.swing.JComponent
                    |
                    +--javax.swing.JInternalFrame
                          |
                          +--com.syllogic.miningmart.concepteditor.ConceptEditor
```

public class **ConceptEditor**
extends JInternalFrame
implements ActionListener, ConceptListener, RelationshipListener

The ConceptEditor allows to create, view, update and delete Concepts, FeatureAttributes and Relationships. It is an JInternalFrame. It needs a reference to a M4Interface. This can be passed to the Object by using the constructor or by using the setInterface method. The ConceptEditor uses information from the M4Interface about the current Case. Use the init() method to let the ConceptEditor initialize itself and prepare the view of the available concepts and relationships in the Case. Use the refresh()method to let the ConceptEditor reload data and update its view.

Field Summary	
(package private) final static String	**helpsetLabel**
(package private) final static String	**helpsetName**

Overview of HTML documentation for the Concept Editor using a standard webbrowser. It has been generated using the standard javadoc tool. For more extensive information about the Java classes that build up the Concept Editor we refer the reader to this API.

Figure 2.4: Screenshot of HTML documentation for the Concept Editor.

- Although concepts can be copied from another case into the current case, it would also be nice to be able to copy concepts (and their base attributes) within the same case.
- Triggers in the database are responsible for setting object validity. We believe there is still some optimization possible in these triggers. This could improve the performance, for example, when importing a case including columnsets and columns.
- Currently the scroll panes for the tree-view and master detail view have fixed maximum sizes. Large cases with many concepts may run into the limits of these components.
- Although it is possible to build a concept hierarchy this hierarchy is not enforced or rigidly maintained by the software. More support for constructing subConcepts would be nice and better maintenance of the concept hierarchy would be good.

Chapter 3

Using the Concept Editor

In this chapter an overview is given from the functionality of the Concept Editor and it is explained how to use it. The focus will be on the use cases for the concept editor, starting at a high level and then specifying these use cases further.

3.1 Overview of Functionality

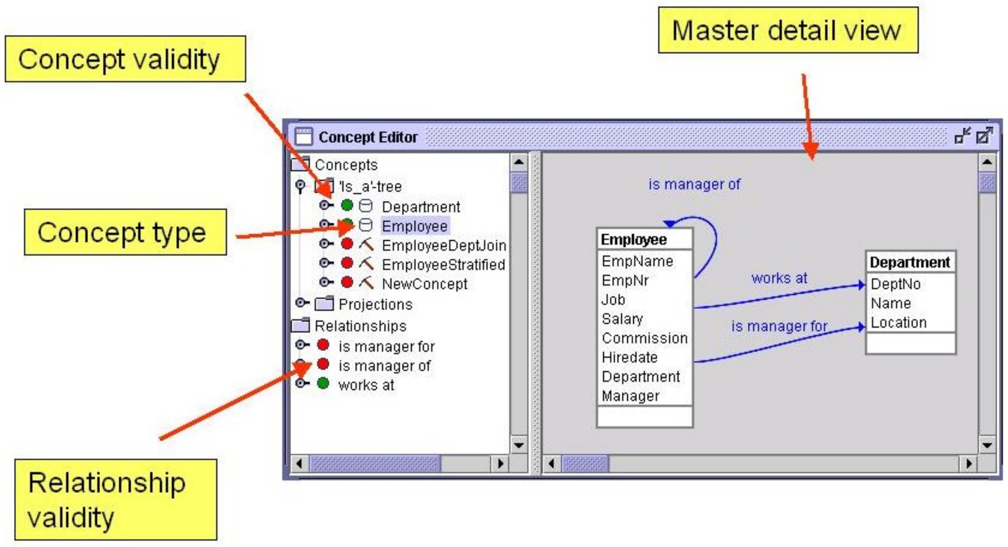
The primary goals of the Concept Editor are to build a Conceptual Data Model (Concepts, FeatureAttributes and Relationships) and map this to the Relational Data Model. The editor must provide an interface for doing this. It is also responsible for validation of Conceptual Data Model elements. The editor does not provide an interface for M4 objects that are not involved in the realization of the primary goals of the editor (e.g.: Case, Step, Operator). The editor does not access the M4 model directly for editing, but uses the M4 interface.

The following lists the use cases:

- Build Conceptual Data Model
- Map Conceptual Data Model to Relational Data Model
- Validate the Conceptual Data Model
- Viewing Concept Data
- Create and View Statistics
- Reuse of Concepts

3.2 Building a Conceptual Data Model

An important part in the work of the case designer is to build a conceptual data model. The concepts will have relationships to each other, may be



Overview of a Conceptual model in the Concept Editor.

Figure 3.1: Screenshot of the Concept Editor.

ordered in a hierarchy and will be, together with the operators, the building stones for preprocessing chains in a case.

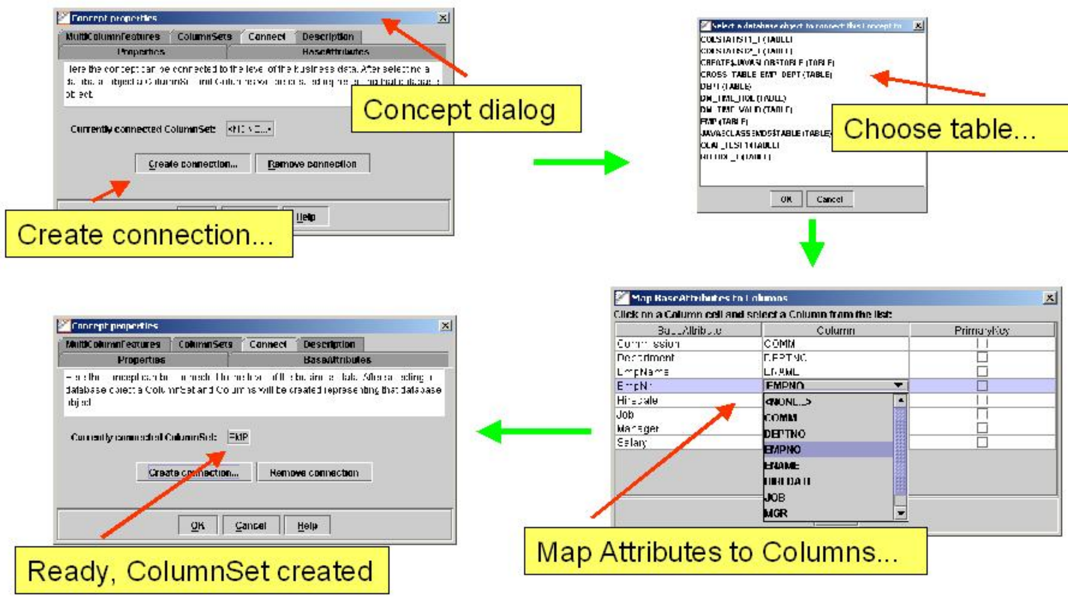
Concepts and Relationships can be created by choosing “New Concept” or “New Relationship” from the menu and filling in the properties for the Concept or Relationship. Editing and deleting existing concepts is done in a straightforward way by using the respective menu items in the Mining Mart HCI.

For an example of a Conceptual model see figure 3.1.

3.3 Mapping of Conceptual Data Model to the Relational Data Model

The Conceptual Data Model defined by the case designer will have to be mapped to the Relational Data Model (the database) in order to be able to execute a case. This is only relevant for Concepts that are indeed based on existing tables in the database (Concept type DB). For Concepts that are created in one of the Steps of a Case (Concept type MINING) the corresponding ColumnSets are created by the Compiler. For Concepts of type BASE no mapping is allowed.

For an example of mapping a concept see figure 3.2.



Click on a Column cell and select a Column from the list:

BaseAttribute	Column	PrimaryKey
Commission	COMM	☐
Department	DEPTNO	☐
EmpName	ENAME	☐
EmpNo	EMPNO	☐
Hiredate		☐
Job		☐
Manager		☐
Salary		☐

Schematic view of connecting a Concept with the Business data using the Concept Editor.

Figure 3.2: Connecting a concept.

3.4 Validity of Objects

The case designer needs to know if the current conceptual data model is valid or not. The validity of a conceptual data model can be summarized in the following way:

The Conceptual Data Model is valid if:

1. All Concepts are valid.
2. All FeatureAttributes are valid.
3. All Relationships are valid.

A Concept is valid if:

1. It is generated by an operator or based on a ColumnSet,
2. at least one included FeatureAttribute exists, and

A FeatureAttribute is valid if:

1. It is connected to a Concept.
2. It is generated by an operator or based on a Column.
3. (for MultiColumnFeature) at least two BaseAttributes exist, which belong to the same Concept as this MultiColumnFeature.

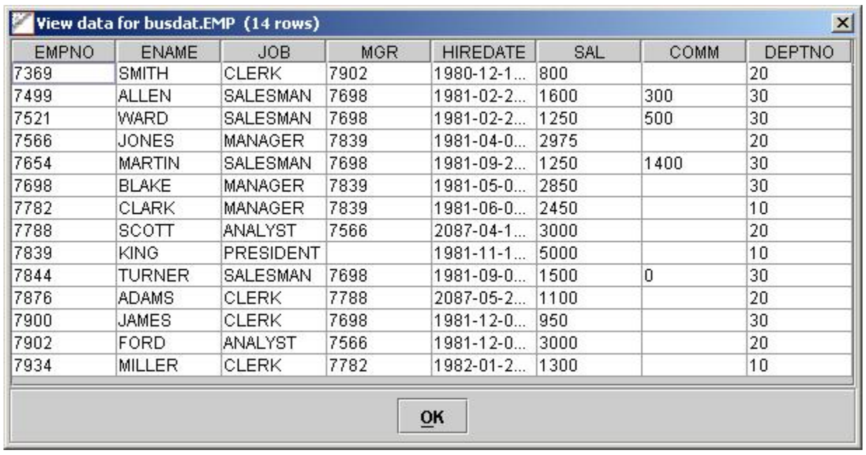
A Relationship is valid if:

1. both related Concepts exist.
2. it is based on Keys or a ColumnSet.

The GUI implements validation checking when creating, editing or deleting Concepts, FeatureAttributes or Relationships. The GUI shows the validity using a red icon (invalid) or green icon (valid) (see figure 3.1).

3.5 Viewing Data

The case designer might want to see the data that is associated with a concept. This can be of importance in making decisions for preprocessing. Therefore the GUI could provide an option for viewing the data that is associated with a concept. The Mining Mart HCI provides a method for showing the data for a concept. Figure 3.3 shows an example of the dialog that is presented to the user after choosing this option.



View data for busdat.EMP (14 rows)

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	1980-12-1...	800		20
7499	ALLEN	SALESMAN	7698	1981-02-2...	1600	300	30
7521	WARD	SALESMAN	7698	1981-02-2...	1250	500	30
7566	JONES	MANAGER	7839	1981-04-0...	2975		20
7654	MARTIN	SALESMAN	7698	1981-09-2...	1250	1400	30
7698	BLAKE	MANAGER	7839	1981-05-0...	2850		30
7782	CLARK	MANAGER	7839	1981-06-0...	2450		10
7788	SCOTT	ANALYST	7566	2087-04-1...	3000		20
7839	KING	PRESIDENT		1981-11-1...	5000		10
7844	TURNER	SALESMAN	7698	1981-09-0...	1500	0	30
7876	ADAMS	CLERK	7788	2087-05-2...	1100		20
7900	JAMES	CLERK	7698	1981-12-0...	950		30
7902	FORD	ANALYST	7566	1981-12-0...	3000		20
7934	MILLER	CLERK	7782	1982-01-2...	1300		10

Viewing the data that is associated with a concept in the Concept Editor.

Figure 3.3: Screenshot of viewing data for a concept.

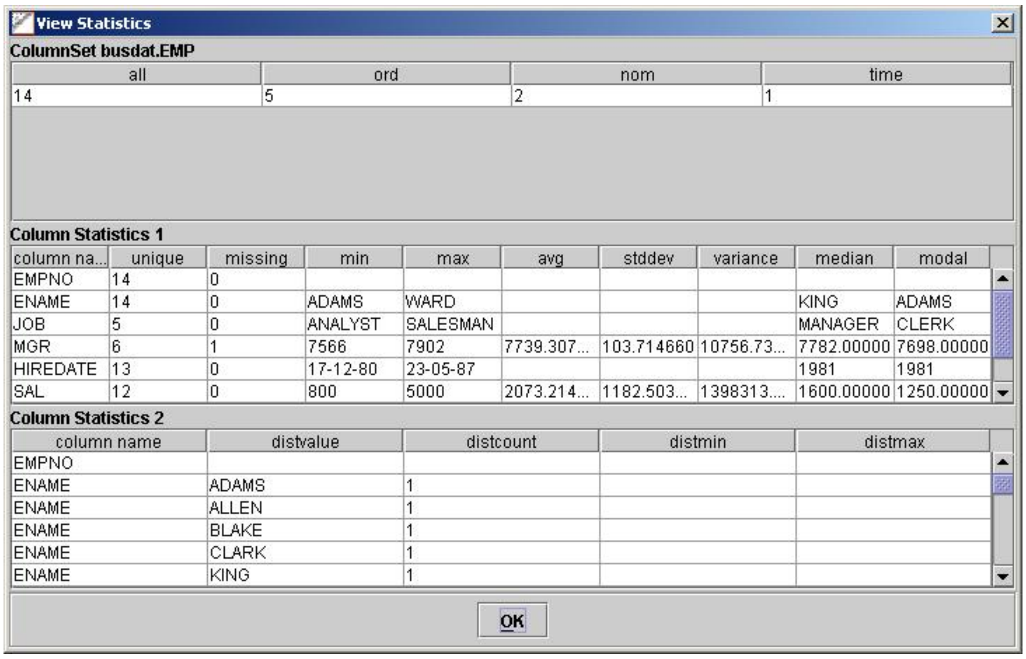
3.6 Creating and Viewing Statistics

Concept data statistics concerning cardinality, missing values, minimum, maximum, average and distribution blocks are helpful in making preprocessing decisions. These statistics can be generated by choosing the “update statistics” menu item in the HCI. They can be viewed by choosing the “view statistics” menu item. Figure 3.4 shows an example the statistics dialog.

3.7 Re-using Concepts

An important functional requirement is to be able to reuse an existing Conceptual Data Model from another case. The user must be able to select a Conceptual Data Model from another case, import it into the Concept Editor and adapt it to his wishes. For adapting the imported Concepts, FeatureAttributes and Relationships he can use the functionality which has been mentioned in “build Conceptual Data Model”(see Section 3.2).

Cases can be exported by the HCI to a file using the export option in the file menu. Via the import menu the user is able to import a case from a file (from another database) or import concepts from another case (in the same database).



ColumnSet busdat.EMP

all	ord	nom	time
14	5	2	1

Column Statistics 1

column na...	unique	missing	min	max	avg	stddev	variance	median	modal
EMPNO	14	0							
ENAME	14	0	ADAMS	WARD				KING	ADAMS
JOB	5	0	ANALYST	SALESMAN				MANAGER	CLERK
MGR	6	1	7566	7902	7739.307...	103.714660	10756.73...	7782.00000	7698.00000
HIREDATE	13	0	17-12-80	23-05-87				1981	1981
SAL	12	0	800	5000	2073.214...	1182.503...	1398313....	1600.00000	1250.00000

Column Statistics 2

column name	distvalue	distcount	distmin	distmax
EMPNO				
ENAME	ADAMS	1		
ENAME	ALLEN	1		
ENAME	BLAKE	1		
ENAME	CLARK	1		
ENAME	KING	1		

Viewing the statistics from data that is associated with a concept in the Concept Editor.

Figure 3.4: Screenshot of viewing statistics for data from a concept.

Chapter 4

Conclusions

The Concept Editor is part of the Mining Mart HCI. It allows to build a Conceptual Model, map concepts to business data, show the validity status of objects, view data, view statistics and re-use concepts. With this functionality all the functional requirements have been fulfilled. In some areas improvements are possible. These are mainly improvements to make things more user-friendly and better scalable.

We have also described aspects of the architecture, design and implementation of the Concept Editor. For more specific information about Java classes used in the Concept Editor we refer to the API documentation.

Our experience with the Editor is that it provides a convenient way of working with concepts and also partners have worked with it successfully in their cases.

Bibliography

[Eul02] Euler, T., *Compiler Constraints and Operator Parameters*, Technical report TR 12-02, October 30, 2002.

[KZV00] Kietz, J.-U., Zücker, R., Vaduva, A., “Mining Mart: Combining case-based reasoning and multistrategy learning into a framework for reusing kdd-applications”, in *Proc. of the Int. Conf. on Multi-Strategy Learning, MSL-2000*, 2000.

[LaR02] Laverman, B., Rem, O., *Description of the M4 Interface used by the HCI of WP12*, Deliverable D12.2, July 16, 2002.

[MoS02] Morik, K., Scholz, M., *The MiningMart Approach.*, Workshop Management des Wandels der 32. GI Jahrestagung, 2002, to appear.

[PeF02] Perot Systems Netherlands, Fraunhofer Institute AiS, *Mining Mart Human Computer Interface*, Technical report TR 12-01, March 15, 2002.

[VKZ01] Vaduva, A., Kietz, J.-U., Zücker, R., Dittrich, K., Morik, K., Marco, B., Luigi, P., *M4 - The MiningMart Meta Model.*, Deliverable, D8/9, IST Project MiningMart, IST-11993, 2001.

[Zu01a] Zücker, R., *Description of the M4-Relational Metadata-Schema within the Database.*, Deliverable D7a, IST Project MiningMart, IST-11993, 2001.

[Zu01b] Zücker, R., *Description of the Metadata-Compiler using the M4-Relational Metadata-Schema.*, Deliverable, D7b, IST Project MiningMart, IST-11993, 2001.

[ZKV01] Zücker, R., Kietz, J.-U., Vaduva, A., “MiningMart: Metadata-Driven Preprocessing.”, in *Proceedings of the ECML/PKDD Workshop on Database Support for KDD*, 2001.

Appendix A

Programming standards

Copyrights

The source code will be copyrighted by the company where it originated, unless otherwise decided during the course of the MiningMart project. Due to the sensitive nature of copyright protection however, it should be noted that any material produced should be protected by copyright (default will be a copyright owned by the producer of that material) at all times. If it is decided to place material in the public domain, then experience shows that an Open Source style license such as the GPL or LGPL provides adequate protection coupled with full disclosure and availability. Note however that such a license does not preclude a copyright ownership by the MiningMart partners. Perot Systems Nederland will include the following code snippet at the top of all of its code: // Copyright 2002 by Perot Systems Nederland // All rights reserved.

Java Package Name Choices

Java package naming has solved many of the problems faced by integrators of code from different sources. We would however propose to standardize the names of the packages themselves as follows:

- All code which is common for the entire MiningMart system should reside in packages under the global package miningmart.
- All code which is internal to parts developed by an individual partner should reside in either a package under the MiningMart package mentioned above, or else in a package named according to that partners own standards. For example, the implementation of the Concept Editor will reside under the package com.syllogic.miningmart.concepteditor.

This approach, when combined with the Factory and Facade design patterns, will allow partners to interface to each others code, without ever having to known partner specific package names. In fact, one would never know of the com.syllogic packages unless one browsed the jar files or performed getClass().getName() calls on objects received from the factory methods. This is the same approach also followed by Sun Microsystems for large parts of the standard Java library.

Naming Standards

Names are generally defined as is common for Java code:

- Package names are all lowercase.
- Classes and interfaces start with an uppercase letter, while methods, attributes, and variables start with a lowercase letter. Names are generally chosen to describe value (Classes, interfaces, and attributes) or function (methods) and not abbreviated. If the name is a concatenation of several words, then the start of each word is highlighted by putting the first letter in uppercase.
- Accessor methods are named getXxx() or setXxx(), where Xxx is the capitalized name of the attribute involved.
- Single letter variables are only used in loops:
  - Counters are called n, m,
  - Indexes in arrays are called i, j,
  - Characters (typically retrieved from strings are read from streams) are called c.

  For other uses descriptive names should be used.
- In a few cases suffixes will be used:
  - Enterprise Java Beans will be referred to by the name of the Remote interface. The Home interface will have this name suffixed with Home, and the implementation will have suffix EJB.
  - Value objects will have suffix V.
  - When several methods are needed with the same name and parameter list, but with different return types, a single letter suffix may be used to distinguish between them. For example, if an integer value is sometimes needed in a wrapper object, and sometimes as an int, the Integer returning method will get a suffix O to signify as object.

Version Control Related Standards

Perot Systems uses CVS for its version control. Since CVS relies on RCS code for the actual processing of the files, RCS style markers are used and substituted. We normally use *Id* at the top of the file to provide identification, and *Log* at the bottom to keep track of the change history.

Source Code Layout

Unfortunately layout is a subject which can easily waste a lot of time. When working with code produced by others, the simple but hard rule is to follow the style already in use. For newly crafted code, Professor Andrew Tanenbaum of the Amsterdam Free University once stated (in relation to contributions to the Minix Operating System) that he would not look at any piece of code before it had been reformatted by cb using his standard settings. We do not propose to go that far, but will use the following settings:

- ASCII TAB characters will be presumed to expand to 8 spaces.
- Indentation is 2 spaces.
- If/while/for/etc will always have use braces around their sub-statements.
- The opening brace will preferably be placed at the end of the line preceding the block, while the closing brace comes on a line by itself.
- Every statement is on its own line.

Other Standard Elements

Perot Systems Nederland will use Together 5.5 as its primary design/development tool, Together ControlCenter will allow us to develop EJBs using a simplified interface to the three source files involved, and integrates seamlessly with CVS. Together stores most of its integration information in the source files themselves, using the javadoc style comments. Some of these comments are used to control display of UML diagrams, while others control the links to the J2EE deployment tools. As a result of this, several @¡keyword¿ style lines will be present in the source files, which do not come from the javadoc standard. Some of the more common ones are: @notProperty This signifies that a method looks like an accessor function, but should not imply the presence of a JavaBean property. @ejbHome Specifies the EJB Home interface. @ejbRemote Specifies the EJB Remote interface. @ENV-REF Starts an environment reference. @ENV-TYPE Java type of the environment value. @ENV-VALUE The actual value involved. @RESOURCE-REF Starts a resource reference. @RES-JNDI-NAME JNDI name of the resource.