AGH University of Science and Technology
Computer Science Laboratory
Department of Automatics
Al. Mickiewicza 30
30-059 Kraków, POLAND

Design and Implementation of HQED, the Visual Editor for the XTT+ Rule Design Method

Krzysztof Kaczor, Grzegorz J. Nalepa
AGH – University of Science and Technology,
Department of Automatics,
Kraków, POLAND
{kk,gjn}@agh.edu.pl

Published online: 24.11.2008

CSL Technical Report No. 2/2008

CSL Technical Report No. 2/2008

Design and Implementation of HQED, the Visual Editor for the XTT+ Rule Design Method*

Krzysztof Kaczor, Grzegorz J. Nalepa
AGH – University of Science and Technology,
Department of Automatics,
Kraków, POLAND
{kk,gjn}@agh.edu.pl

Abstract. The report discusses the design and implementation of the HQED visual editor for the XTT structured rule design method. The editor has been developed within the HeKatE project. It supports the main logical design with XTT. It also allows for importing and visualizing the conceptual rule design with the ARD+ method. The editor has a modular architecture, that can be extended with optional plugins for system analysis. The tool supports a visual design of the XTT rulebase.

Keywords: rule-based systems, XTT, visual design, editor, HeKatE

*The paper is supported by the *HeKatE* Project funded from 2007–2009 resources for science as a research project.

Contents

List of Figures

List of Tables

1 Introduction

Rules are the most popular method of knowledge representation. They offer a transparent and intuitive knowledge representation, along with visual representation with decision tables and decision trees.

This report is dedicated to the presentation of the results of the thesis [4]. The goal of this thesis was do describe the design and implementation of a tool that supports the XTT rule design method [9]. Keeping the high quality of the XTT model during the design is a very important issue. This is why an important feature of the tool is the on-line rule analysis.

This report is organized as follows:

- The evolution of the XTT design process as well as the integration with ARD design phase are described in Section 2.
- The detailed specification of the requirements for the application is given in the Section 3.
- The Section 4 describes the issues related to the design of the application.
- The Section 5 contains the important information about the implementation of the application such as: functionality, and architecture.
- The Section 6 presents the XTT examples designed with the tool.
- In Section 7 the summary of the project as well as the issues related with the future work and the development of the application are described.

The main practical result of the thesis was the implementation of a prototype CASE tool for the XTT method called HQED.

2 The XTT approach

In this section the XTT (*eXtended Tabular Trees*) [9] hybrid rule design approach is described.

2.1 Design process, ARD, XTT, tools history

The XTT design approach is similar to *data base design approach* [2]. It consist of three phases:

1. *Conceptual design* - this is a most abstract phase. The system is identified by attributes. During this phase functional dependencies between attributes are defined. This phase introduces a new class of diagrams, which are called *Attribute-Relationship Diagrams*, ARD – for short.
2. *Logical design* - this stage involves the XTT approach. The preliminary XTT model may be obtained as result of ARD diagram translation as well as may be built from scratch. During the phase the XTT structure can be incrementally built and analysed on-line.
3. *Physical design* - the preliminary implementation of system is done by the generation of PROLOG (or any other) code based on the XTT model. Such code can be complied, executed and debugged.

The phases of the integrated design process are shown in Fig. 1. Until now a several tools supporting the design have been implemented.

Historically, the first tool is the Mirella Designer [10]. The Mirella name is an acronym for: **M**eta-level **I**ntegrated **R**BS design **E**nvironment with On-**L**ine **L**ocal **A**nalysis. The screenshot of the Mirella Designer is presented in Fig. 2. The Mirella Designer has been created to support XTT design process as well as to translate XTT to PROLOG code. This environment does not support *conceptual design* phase. The tool consist of four principal modules:

1. **Mirella Creator** – this module is responsible for definition of attributes and theirs domains.
2. **Mirella Designer** – this module is responsible for the visual design of the system knowledge base using the XTT representation.

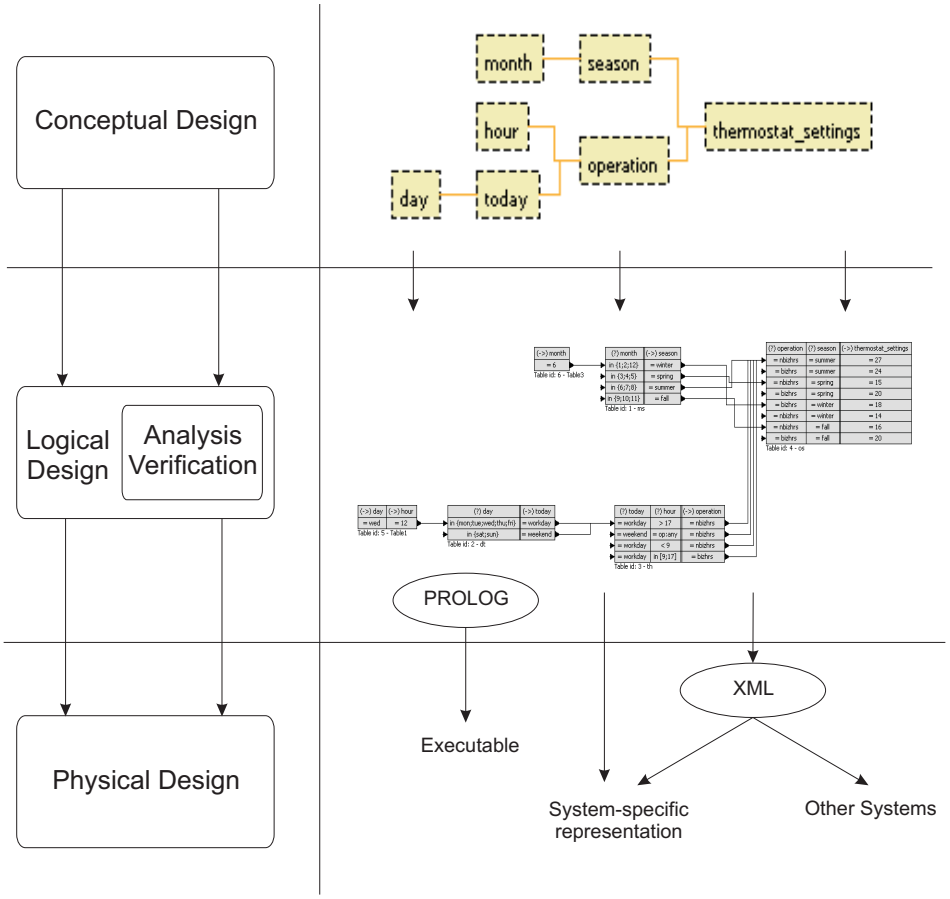

Figure 1: Phases of the integrated design process

3. **Mirella Exchanger** – which creates a framework of number of import and export plugins, exchanging data with other knowledge representation languages, and domain specific system prototypes.
4. **Mirella Validator** – this module is responsible for the on-line analysis.

The Mirella Designer has been created in ANSI C language with the help of GTK+ library for the GUI (*Graphic User Interface*). Mirella was a proof of concept and had its limits, so the development was stopped.

Two new tools for ARD were developed during the Knowledge Engineering student projects at Institute of Automatics at AGH UST in 2007: one based on the Dia editor, and another based on the Qt library. *Qt ARD editor* was another Knowledge Engineering project of 2007 [6]. The main goal of this project was to develop a standalone ARD diagram editor using the Qt library. The Qt library is a GUI library implemented in C++. The project has been successfully implemented and the results can be observed on Fig. 3. The editor allows exporting the model to ATTML that is the language based on XML. The editor allows for hierarchical ARD design, and simple attribute specification.

Dia is the universal diagram drawing software, tha uses the set of predefined object for drawing. The characteristic feature of *Dia* is the possibility to define new diagrams with XML. The goal of a Knowledge Engineering project in 2007 [8] was to develop a *Dia* library to support ARD design. The library has been successfully implemented. The result of implementations can be observed in Fig. 4.

2.2 XTT concepts

The XTT phase is the logical design phase. It involves the following stages [7]:

- Specifying the full headers of the XTT tables with appropriate attributes.

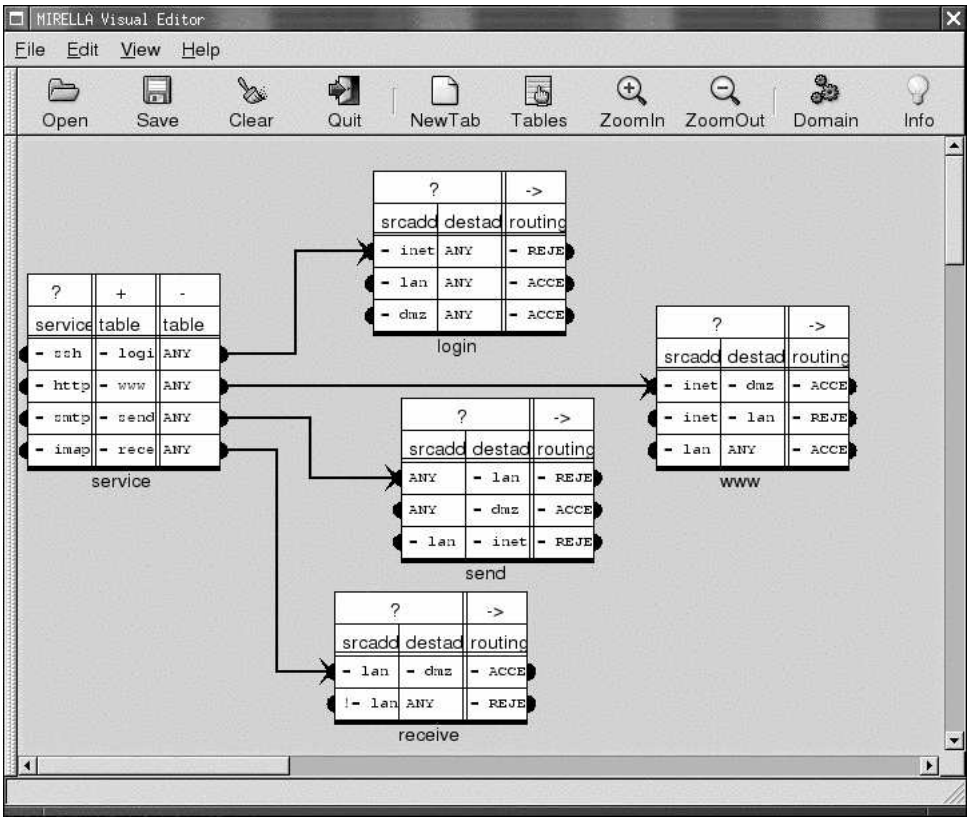

Figure 2: The Mirella Designer

- Defining the rules through filling the XTT cells in XTT rows. Each XTT table consists of XTT rows which are equivalent to rules.
- Linking XTT tables with XTT connections which determine order of rules execution.
- Building a hierarchical system design using *tree* facilities.
- Refining attribute specification including constraints.
- Generating PROLOG prototype corresponding to the XTT structure.

During the design, the on-line analysis allows to detect anomalies in the XTT structure.

XTT as described here supports some extensions, referred to as XTT+[11]. The XTT language is a hybrid of other languages and knowledge representations. The basic concepts in XTT are introduced here:

Rules – give logical representation of the system knowledge base, they introduce concepts such as: *condition*, *conclusion*, *decision*.

Decision tables – are the objects, which store knowledge. They are related with *table*, *row*, *cell*, *attribute*, *attribute value*.

Decision trees – are the objects which serves knowledge components. They introduce concepts such as: *tree*, *connection*, *node*, *branch*, *leaf*.

Attribute An *attribute* A_i, denoting a *property* of an *object* is a mapping:

$$A_i : O \rightarrow D_i$$

where: O is a set of objects and D_i is a domain of the attribute A_i [7].

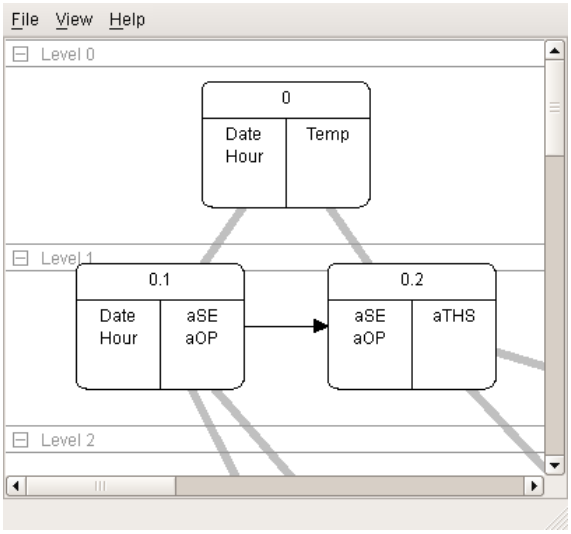

Figure 3: ARD model design in *Qt ARD editor*

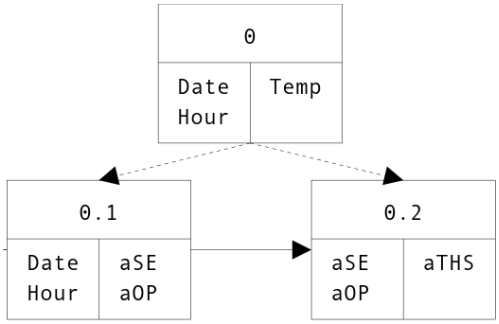

Figure 4: ARD diagram designed in Dia

Some attributes can be placed in one group to emphasize some relationships among them. The attributes grouping provides a complex structure, called *group*, which is similar to structures in C language. A *Group* can be interpreted as the set of attributes:

$$Group = \{A_1, A_2, \ldots, A_n\}$$

Attributes within a group can be referenced by their name preceded by dot and the name of the group:

Group1.Group2.attribute_name

XTT+ introduced a mechanism called *Attribute-Attribute Comparsion* which allow to compare two values of the attributes. The *Attribute-Attribute Comparsion* can be expressed as:

```
if(Attribute OPERATOR Attribute) then ...
```

The *Attribute-Attribute Comparison* enforces problems for the on-line analysis. For example: comparison of two different values of attributes that have different domains is very difficult and requires a simulation of the whole structure for checking if the value of the second attribute belongs to the domain of the first attribute. Sometimes, the XTT+ structure is incomplete and the simulation cannot be executed.

Connection A *Connection* is an ordered pair of *table/row* pairs:

$$N = ((T_w, R_x), (T_y, R_z))$$

where (T_w, R_x) is referred as *parent* row, and (T_y, R_z) as child row. A *Connection* connects two *Rows*.

Tree A *Tree* is a structure composed of two or more *Tables* connected with at least one *connection*. It can be represented as pair:

$$E = (T, N)$$

where: $T = \{T_1, T_2, \ldots, T_p\}$ is a set of *Tables* and $N = \{N_1, N_2, \ldots, N_q\}$ is a set of *Connections* between them.

The new visual representation of XTT table is shown in Fig. 5, it is more simple and intuitive than the original XTT. A simple XTT structure with several tables is shown in Fig. 6. The XTT structure may be mapped to an XML-based language called XTTML (*XTT Markup Language*).

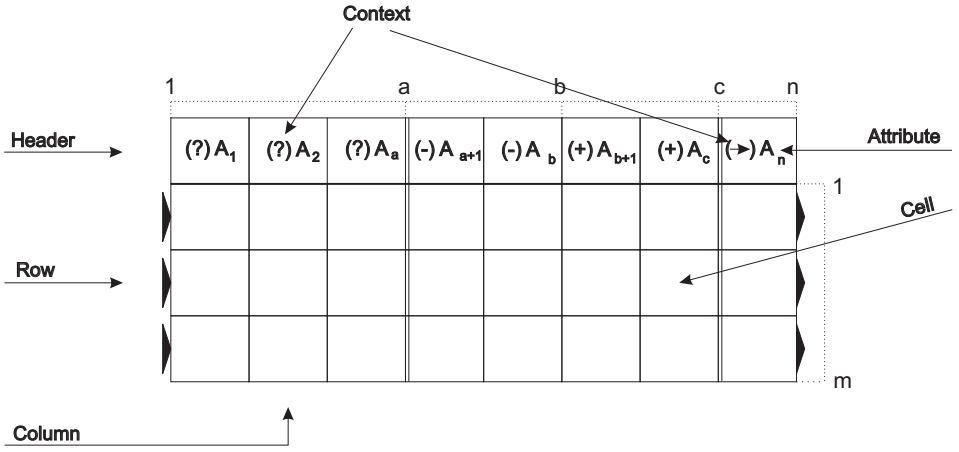

Figure 5: A simple XTT Table

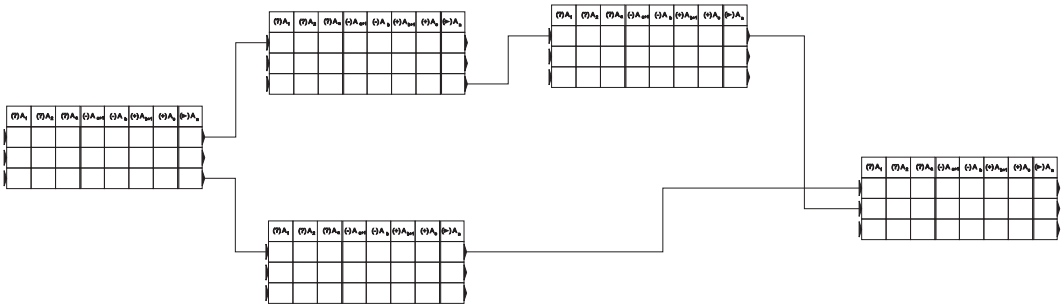

Figure 6: A simple XTT Tree

2.3 ARD+ and VARDA

ARD supports the conceptual system design and allows for generating preliminary structure of XTT. ARD allows to identify system attributes and their functional dependencies. It is a hierarchical approach, which starts from the most abstract system specification, and provides a more detailed specification. ARD has its representation in the XML language, called ARDML (*ARD Markup Language*).

The main concepts related to ARD (see Fig. 7) are:

A Conceptual Attribute is an attribute which describes some general aspects of the system. Such an attribute must be specified and refined. The name of *Conceptual Attribute* begins with a capital letter: `ThermostatSetting`.

A Physical Attribute is an attribute which describes a well defined and atomic aspects of system. Such attribute cannot be specified and refined further. The name of *Physical Attribute* begins with not capital: `theThermostatSetting`.

A Simple Property is described by a single attribute.

A Complex Property is described by multiple attributes.

A Dependency is a ordered pair of properties: $D = (P_1, P_2)$, where P_1 is an independent property and P_2 dependent one.

An ARD Diagram is the set of properties and dependencies.

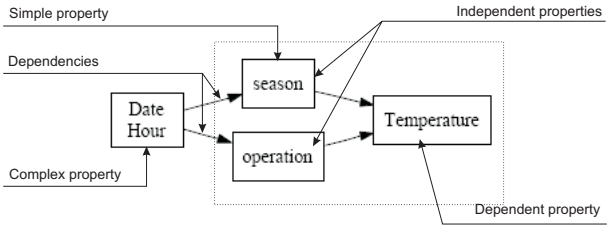

Figure 7: The concepts related with ARD

The ARD diagram is developed with two transformations. Finalization and Split.

The more detailed ARD levels create a hierarchy of levels. The purposes of hierarchical model are [12]:

- gradual refinement of a designed system, and particularly
- identification of the origin of given properties,
- ability to get back to previous diagram levels for refactoring purposes,
- the big picture perspective of the designed system.

The implementation of such a system can be done by storing the most detailed level, and all the information about transformations that is needed to recreate the more general levels. The history of changes on the levels is called *Transformation Process History* (TPH for short). The TPH diagram contains information about how a particular property has been split and how the particular attribute has been finalized.

The preliminary XTT structure is generated automatically with the *Rule Prototyping Algorithm*. The input of the algorithm is the most detailed level of ARD phase, which has all the physical attributes identified. The output is a set of *rule prototypes* for XTT.

VARDA (***Visual ARD Rapid Development Alloy***) is a prototype tool providing the ARD+ design and visualization. VARDA has been implemented in PROLOG with the help of several tools such as *GraphViz*, *ImageMagick*.

3 XTT editor requirements specification

The list of requirements for the XTT editor can be divided into three categories:

Requirements for the design and on-line quality support is the group that contains the specification of the requirements introduced by the XTT design method.

Requirements of functionality is the group that contains the specification of the requirements for the application. The design of the application should fulfil these requirements and during the implementation process the functionality that meets the requirements should be implemented.

System requirements is the group that contains the list of the minimal system requirements and dependencies. The group gives the answer to the question: what is needed to execute the application (libraries, packages, etc).

The list of requirements for each group is showed in the following sections.

3.1 Requirements for the design and on-line quality support

- The support for **ND** (not defined) and **ANY** operators.
- Defining attribute groups:
  - Defining group name, description.
  - Defining group parent, or as root.
- Defining attributes:
  - Defining attribute's name.
  - Defining attribute's acronym - an alternative name, shorter than actual name.
  - Defining attribute's type - the type can be defined as primitive type: `bool`, `integer`, `float`, `symbolic`, `enumerative`.
  - Defining attribute's domain:
    - Defining type of constraints: *no constraint*, *range*, *set* of values (set of enumerated values).
  - Defining default value for attribute.
  - Defining attribute's relation: *input*, *middle*, *output*.
  - Defining attribute's description.
  - Defining attribute's group - the group to which the attribute belongs.
- Defining tables:
  - Choosing attribute from the list.
  - Selecting condition attributes.
  - Selecting conclusion attributes.
  - Selecting attribute context (*conditional*/*assert*/*retract*/*conclusion*).
  - Defining the table's title.
  - Defining the table's description.
  - Defining the table's type (*fact*, *middle*, *halt*).
- Defining connections between tables:
  - Defining the input and output parameters.
  - Defining the type of the connection (*normal*, *cut*).
  - Defining connection's label.
  - Defining connection's description.
- Translating XTT model to executable PROLOG code.
- On-line verification of model quality:
  - Finding unused attributes.
  - Showing validation problems.

3.2 Requirements of functionality

- Visualisation:
  - Visualisation of the ARD model (ARDML file as source of information).
  - Visualisation of the simplified view of tables.
  - Visualisation of the connections:

    - Visualisation of the difference between *normal* and *cut* connection.
    - Displaying the label of the connection.
  - Marking selected items.
  - Zoom in and out.
  - Black and White view option.
  - Exporting the model visualisation as SVG or other format (BMP, JPG, PNG, XPM, PPM).
  - Printing the model visualisation.

- Groups:
  - Adding, editing, removing groups of attributes.
  - Editing group:
    - Adding attributes to the group.
    - Removing attributes from the group.
    - Moving attributes between groups.
  - Deleting group.
  - Changing the group parent.

- Attributes:
  - Adding new attributes.
  - Editing all features of attribute.
  - Removing attributes.
  - Sorting attributes within group.
  - Showing the list of unused attributes.
  - Exporting the attributes' list to ATTML file.
  - Importing the attributes' list from ATTML file.

- Tables:
  - Adding tables.
  - Editing tables.
  - Deleting tables.
  - Selecting table:
    - Selecting with the help of dialog window.
    - Direct selecting (clicking on the table).
  - Table comments.
  - Auto arrangement of tables.

- Connection:
  - Adding new connections:
    - With the help of dialog window.
    - Easy defining with the help of drag and drop technique.
  - Editing connections.
  - Deleting connections.
  - Selecting connection:
    - Direct selecting (clicking on the connection).
    - Selecting with the help of dialog window.

  - The changing the connection's curve of shape.

- ARD and XTT integration:
  - Creating XTT schema based on the ARD diagram.
- Reading and writing the XTTML file.
- Reading and writing the ATTML file.
- Reading the ARDML file.
- Search dialog:
  - Searching for attributes.
  - Searching for tables.
  - Searching for connections.
- Plugin interface:
  - The possibility to extend the application by using the external resources.
  - The possibility to data exchange between plugin and application.
- PROLOG interface:
  - The set of the predicates that allow the plugin to communicate with the application.
  - The data exchange between PROLOG and HQed.

3.3 System requirements

- The platform independence.
- Compiling the source code on different platforms without modification.
- The system requirements should be as low as possible.
- Easy maintenance.
- Portability of the application.
- Scalability.
- Dilatability (interface to plugins).

With respect to the system requirements presented in this section, the application has the following system requirements:

- Installed Qt library in version 4.3.2 or later.
- Installed C++ compiler.
- 200MB of available hard-disk space.
- 128MB of RAM (256MB recommended for complex projects).

4 The Extended Tabular Trees editor design

One of the requirements for the editor, is an open architecture that allows developing the application by creating plugins, changing the interfaces, etc. Hence, the source code of the editor must be transparent, clear, and well documented. There is no tool that allow checking if the source code is transparent and clear. To achieve this aim the special design pattern was applied. Originally, *Extended Tabular Trees editor* had been called *XTTed* (XTT EDitor), but later the name has been changed to *HQed* (*Hekate Qt EDitor*).

4.1 Model-View-Controller design pattern

The Extended Tabular Trees editor has been designed with use of the *Model-View-Controller design pattern*. Model-View-Controller (MVC for short) is an architectural pattern used in software engineering. The MVC was first described in 1979 by Trygve Reenskaug, then working on Smalltalk at Xerox PARC, and later in [1]. The main assumption of the pattern is the design of the project with three independent layers:

- **Model** – The model is the internal representation of the information on which the application operates. This representation is not available to the user and it can be interpreted as the state of the system.
- **View** – The view layer provides representation of the model for the user as well as supports the interaction between application and the user.
- **Controller** – The association between system's events. It provides communication between *Model* and *User*.

The separation on three independent layers is useful in the design process.

MVC provides the following features:

- The source code is more transparent and clear.
- The modules can be done by different programmers.
- The maintenance of the code is easier.
- The process of implementation becomes faster.

The simple diagram of the pattern is shown in Fig. 8.

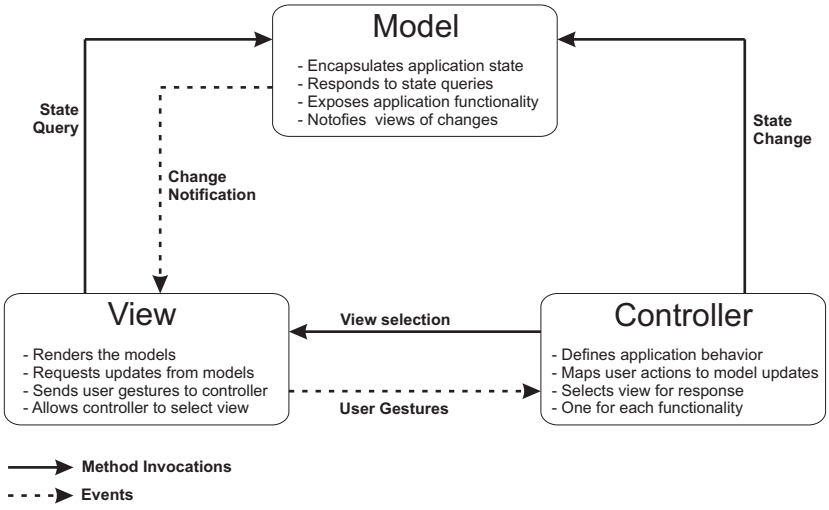

Figure 8: MVC diagram.

There are a lot of frameworks that support the MVC:

- Implementations of MVC as GUI frameworks:
  - Microsoft Foundation Classes
  - NeXTSTEP development environments encourage the use of MVC
  - Qt Toolkit since the Qt4 Release.
  - Java Swing

- Implementations of MVC as web-based frameworks:
  - ASP.NET MVC Framework
  - PureMVC Framework for ColdFusion
  - Spring MVC Framework
  - PureMVC Framework for PHP

The Extended Tabular Trees editor has been implemented according to MVC principles.

4.2 Use case diagrams

In the following diagrams the most important use cases of the editor are presented.

- In Figure 9 the most general functionality.
- In Figure 10 the functionality of the model.
- In Figure 11 the functionality of the view.

For the full discussion of the diagrams see [4].

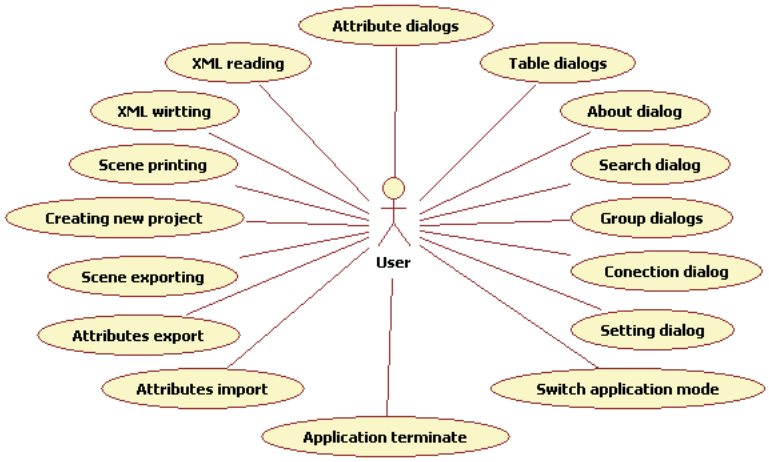

Figure 9: Use case diagram for the general functionality.

4.3 Class diagrams

In HQed implementation all the classes have been divided between *model*, *view* and *controller* layers. Additionally, *model* and *view* of ARD and XTT are also separated. In the *controller* layer, the sublayers may be distinguished (See Fig. 12).

5 HQed implementation

5.1 Architecture

HQed has been implemented according to MVC design pattern. The architecture of HQed consists of three layers: *model*, *view*, *controller*, see Fig. 13.

The *XTT model* and *ARD model* are parts of *model* layer. All the data relating with *model* are stored and processed in this layer.

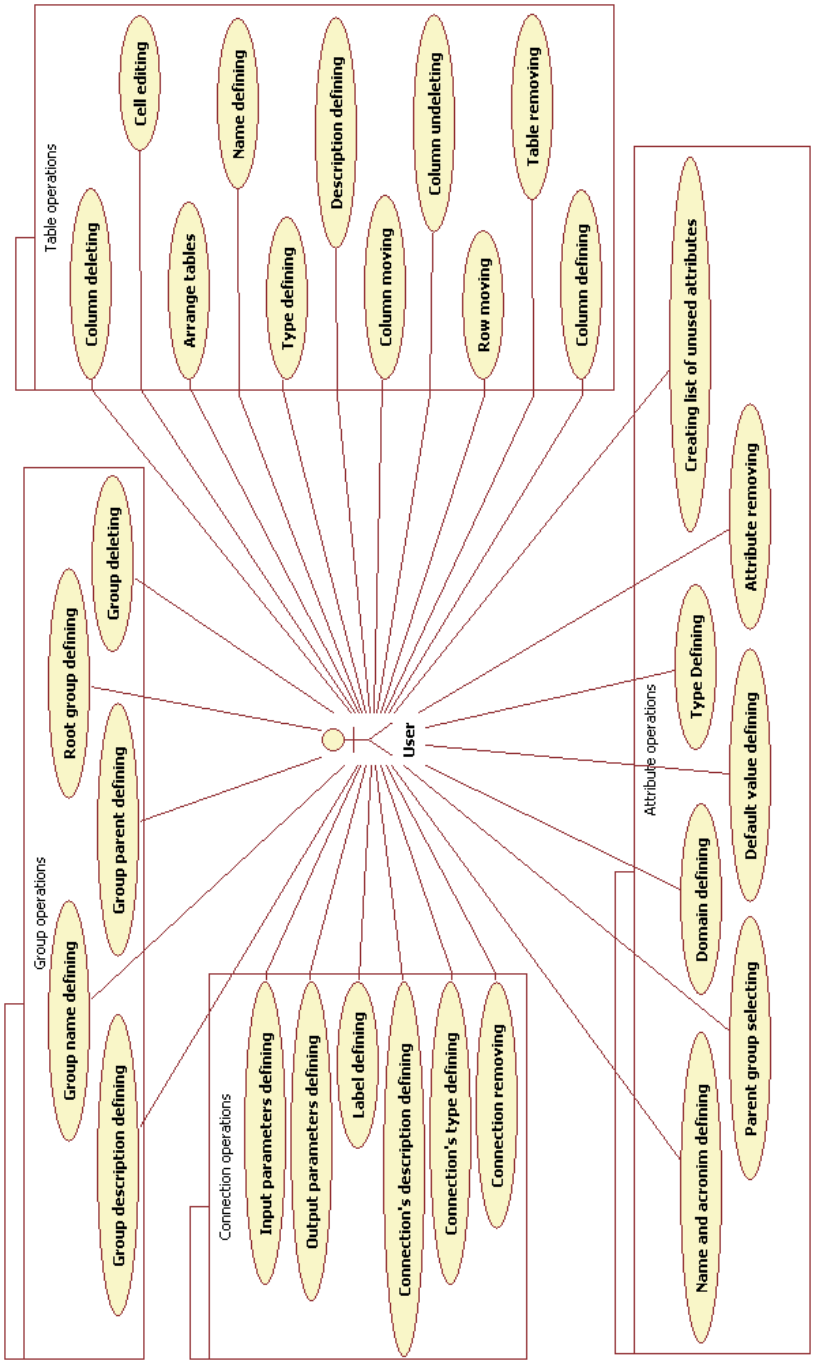

Figure 10: Use case diagram for the model.

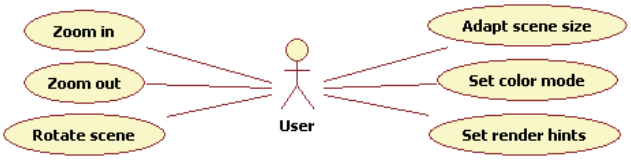

Figure 11: Use case diagram for the view.

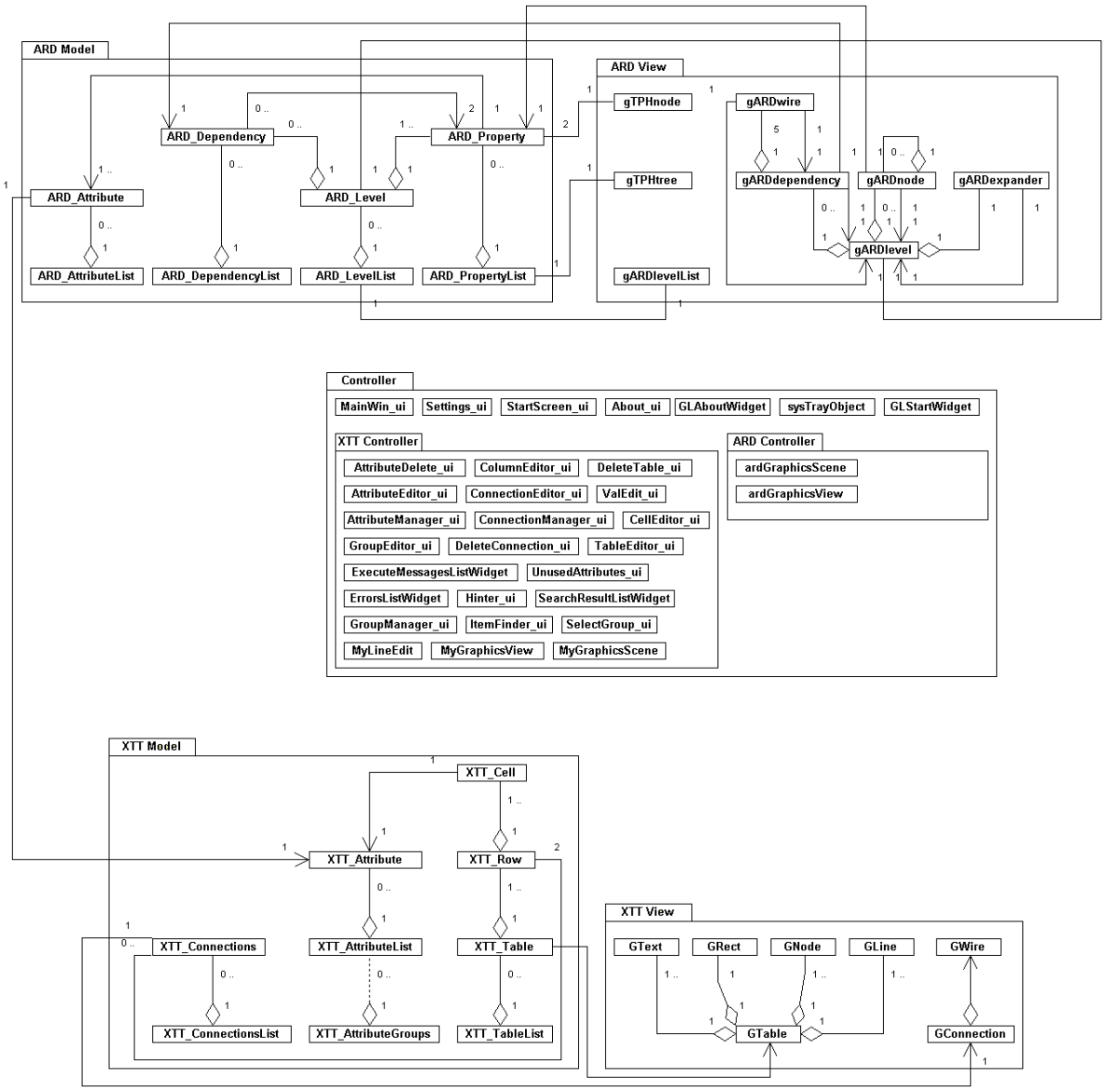

Figure 12: Class diagram for HQed tool

The *controller* layer in the sense of MVC, combines the following layers of the architecture: *Controller*, *PROLOG API*, *PLUGINS API*.

The *PROLOG API* layer The task of this layer is to make functions and data available to the PROLOG engine. As the result, of implementation such a layer, is the possibility to the control of HQed from the PROLOG level. This is a very important because there are PROLOG tools supportingthe analysis of the logical quality of the model.

The *PLUGINS API* layer is responsible for communication between HQed and external plugins. The task of this layer is very similar to *PROLOG API* layer. The difference lies in the fact that this layer making data and functions available to external plugins. The way of communication can be different: (DLL, TCP/IP). The TCP/IP seems to be the best solution because it is supported by the vast majority of the operating systems, and it makes possible to use the plugins over the network.

The *view* layer in the sense of MVC, combines the following layers of the HQed architecture: *User Interface* and *XML mapping*. These layers are responsible for representation of the model.

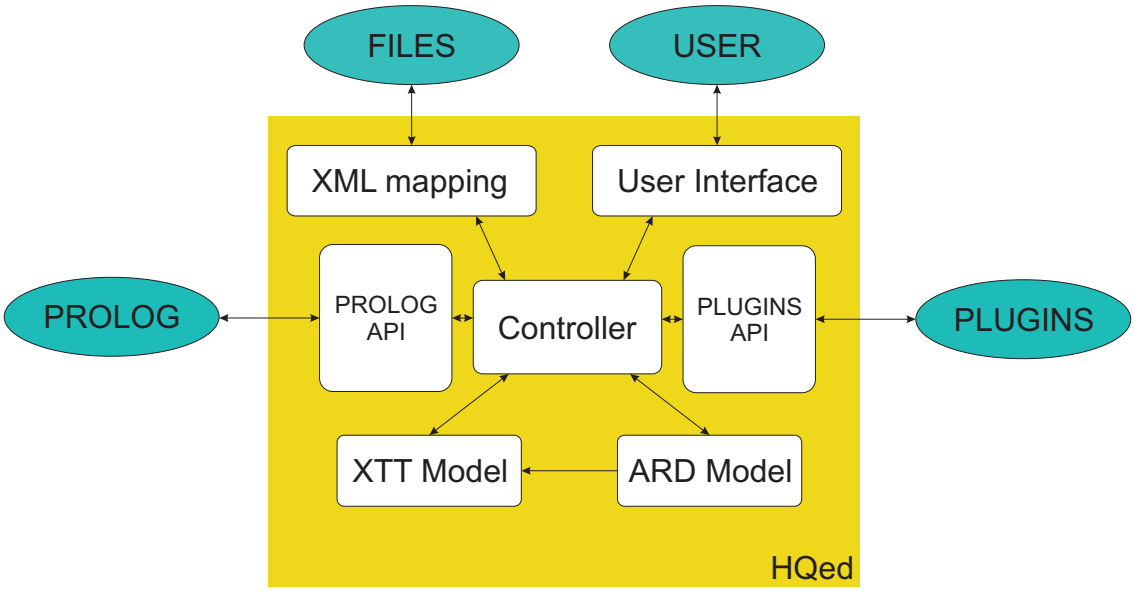

Figure 13: The architecture of HQed

The *User Interface* layer is responsible for communication between user and application. The first task of this layer is the representation of the *model* layer in the graphical way and suitable for the user form. The second task of this layer, is intercepting all the user events and the dispatching them to the *controller* layer.

The *XML mapping* layer allows to map the model as the XML-based language XTTML and vice versa. This is a very important layer, because with the help of it, the designer can save his work and later load it.

As a summary of the implementation some most important technical information is given:

- The source code consists of:
  - 70k lines of source code.
  - 173 files that contains the source code (*.cpp, *.h).
  - The size of source code is equal to 2.89MB (it depends on the file system).
- The implementation contains:
  - 145MB as a size of the root directory.
  - 104 classes.

5.2 Development platform

With respect to the platform independence requirement for HQed, the special programming tool must be selected. Additionally, other requirements, such as: PROLOG Interface, Graphic User Interface, Plugin Interface, narrow down the number of tools. The used platform to implementation of HQed is Qt library. Qt is a cross-platform rich application development framework. It includes [14]: an intuitive, easy to use class library, integrated development tools, cross-platform development on desktop and embedded platforms, support for C++ and Java development. The current 4.4 Qt library consists of over 400 classes, which encapsulate all infrastructure needed for end-to-end application development. The API is separated into 13 modules (See Fig. 14).

Qt is Cross-Platform library. The source code written once can be compiled on such platforms as: Windows, Mac, and X11.

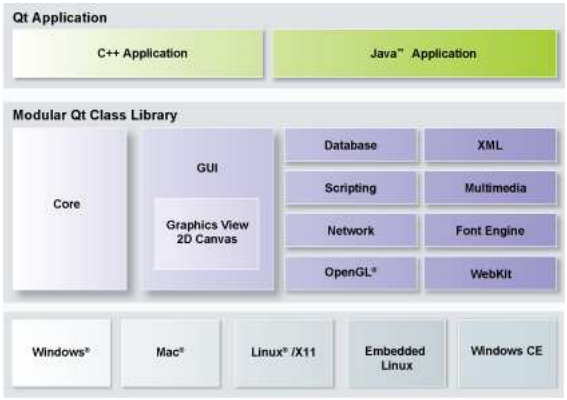

Figure 14: Qt architecture diagram. (See [14])

The version of library used to implementation HQed is 4.3, with optional compatibility with version 4.2.3 avaialbe in some GNU/Linux distributions.

HQed is an open source and free software. It is licensed by third version of GPL licence from the Free Software Foundation.

5.3 Requirements fulfillment

Design support's requirements fulfilment:

- *The support for **ND** (not defined) and **ANY** operators* – The operators are supported by the application. The form of the operator **ND** is `op:nd` and the operator **ANY** is `op:any`.
- *Defining attribute groups* – The functionality was implemented and it allows the user to perform all the actions that are required. The user can define the tree of the groups and for the each group the name as well as the description. The application enables the user to change the structure of the tree.
- *Defining attributes* – The functionality requirements are fulfilled. The user can define all the required properties of the attributes as well as can define the constrained domains. The constraints can have the range form or the set form.
- *Defining tables* – The functionality is also available from the application's level. The user can define the structure of the tables and freely modify it later. The properties such as the title, the description, the type can be defined too.
- *Defining connections between tables* – There are two ways to define the connection between the tables: with the help of window dialog. The properties of the connection can be defined by using the *editor of connections* window dialog.
- *Translating XTT model to executable PROLOG code* – This functionality has **not** been implemented yet. It may be considered as the future work.
- *On–line verification of the model quality* – The on–line verification is implemented. Due to the lack of possibilities for a complete verification of the model during the design process the implemented functionality does not meet with the requirements precisely. The application controls the quality of the model in the range of possibilities.

Functionalities's requirements fulfilment:

- *Visualisation* – The application supports the visualisation of the tables, the connections as well as the graphical user interface. All the operations can be executed with the use of the window

dialogs. The visualisation's requirements demand "the lack of the user freedom". The process of defining the properties of objects, values, should be performed with the help of forced possibilities (i.e. well-defined lists of values, etc.). All the requirements, which are described in the Section 3.2, are fulfilled.

- *Groups* – The operation such as adding, editing, deleting group can be executed with the help of the window dialogs, which are designed to this end. The operations of adding, editing, deleting can be done by using *Group manager*.
- *Attributes* – All the required operations related with the attributes can be done. The special dialogs are designed. The *Attribute editor* dialog makes adding and the editing the properties of the attributes possible. The *Attribute delete* dialog makes deleting of the attributes possible. All the required properties, related with the attributes, can be defined with the help of these dialogs.
- *Tables* – The requirements with regard to the tables are fulfilled. The application allows the user to define, edit, delete the tables. The properties, such as title, description, as well as the type of the table can be defined. There are several window dialogs that support these actions.
- *Connection* – The fulfilment of requirements which concern connections has been described in the previous section. Additionally, the application allows user to define the type of connection ("normal", "cut"). Regarding the visualisation of the connections, the connection is presented as broken line. The form of the connection (the lengths of each part) can be changed manually.
- *ARD and XTT integration* – The algorithm of the ARD and XTT integration was implemented successfully. According to the final level of ARD the XTT schema is generated automatically. Additionally, the application allows the user to configure some aspects of algorithm.
- *XML support* – The application allows user to read and write the XML files. The information about the XTT model are stored as the *XTTML* file. There are two versions of *XTTML* and the application supports both of them. The information about attributes and groups are stored as *ATTML* file. The application allows to export and import data from/to *ATTML* file. The ARD model is stored as *ARDML* file. The application supports only the reading the *ATTML* file.
- *Search dialog* – The application has built-in the search engine and the dialog that displays the results. The scope of search can be changed manually. The search dialog tries to find, the entered phrase in the properties' values of tables, attributes and connections. The results are displayed in the window dialog.
- *Interfaces* – The interface to the plugins has not been implemented yet. The development of interface to the plugins can be the considered as the subject of the future work.

System's requirements fulfilment:

- *The platform independence* – There is no possibility to create an executable version of application that is platform independent. In case of the source code this problem does not exist. Thanks to the Qt library. the source code that has been written can be compiled on the different platforms without modification, including: GNU/Linuix, Windows, OSX.
- *The minimal system requirements* – The application should use as few system's resources as it is possible. In case of HQed the system requirements are following:
  - Installed Qt library in version 4.3.2 or later.
  - Installed C++ compiler.
  - 200MB of available hard-disk space.
  - 128MB of RAM (256MB recommended for complex projects).

- *Scalability* and *dilatability* of application can be available with the help of using the plugins. Due to the fact that the interface to the plugins is on very early stage of the development, the application can not be extended by the external resources.

6 HQed Use cases

This section presents two examples of the editor use:

- *Thermostat* that presents the problem of creating a temperature control system for an office.
- *Cash Point* that presents the design of the behaviour of an ATM system.

They are designed using a classic expert systems approach [3].

6.1 Thermostat

The main problem described by *Thermostat* use case is the problem of creating a temperature control system for an office. The temperature control system sets the temperature in the office depending on the current season and the working time. The detailed description of *Thermostat* system problem is presented in [13]. The input to the system consists of current time. The system's response is the most appropriate temperature at the current time.

The Table 1 shows the specification of the attributes that are used to solve *Thermostat* problem. The Figure 15 shows the steps of the ARD design process. The Figure 16 shows the ARD transformations process history. According to the last level of ARD diagram, where all the physical attributes are identified, the schema of XTT is generated. The Figure 17 shows the full XTT model.

Name	Type	Domain	Description
day	enumerative	{monday, tuesday, wednesday, thursday, friday, saturday, sunday}	The name of the current day. The attribute defines the today's value.
time	integer	<0; 24>	The value of the hour. The attribute defines the operation's value.
month	enumerative	{january, february, march, april, may, june, july, august, september, october, november, december}	The name of the current month. The attribute defines the season's value.
season	enumerative	{spring, summer, autumn, winter}	The name of the season.
today	enumerative	{workday, weekend}	The type of the day.
operation	enumerative	{bizhrs, nbizhrs}	This attribute describes if the current hour is the working time (bizhrs) or not (nbizhrs).
thermostat	integer	{14, 15, 16, 18, 20, 24, 27}	The value of the thermostat setting.

Table 1: XTT Thermostat attributes specification.

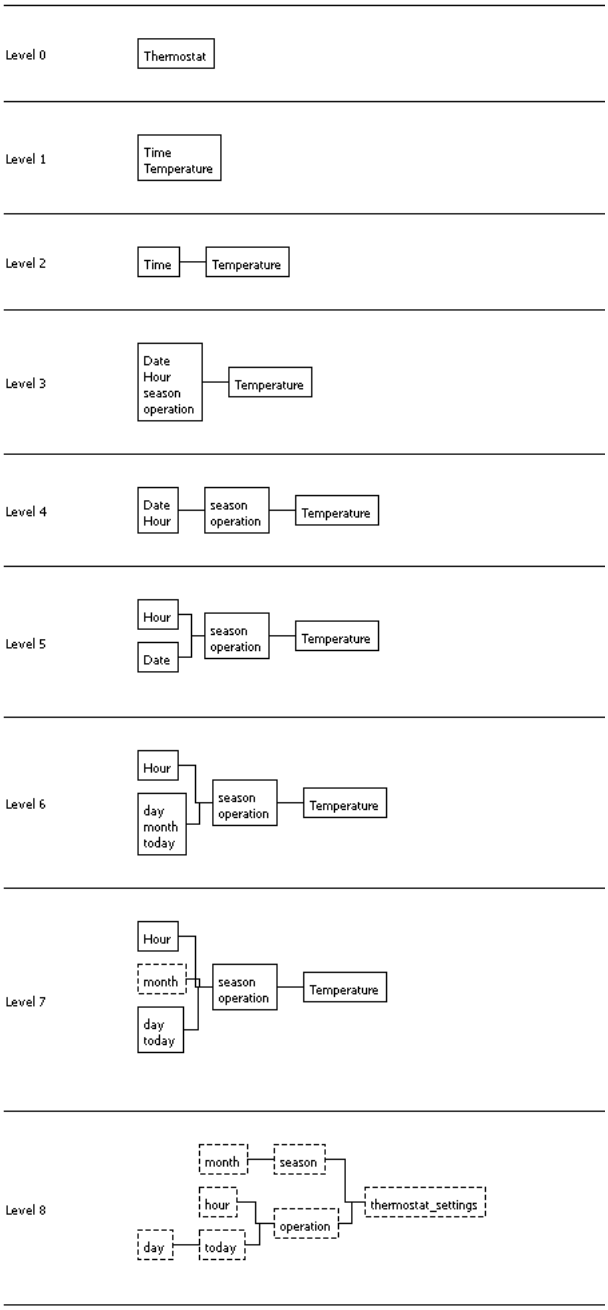

Figure 15: The ARD model of the Thermostat (rendered by the HQed).

The original *Thermostat* system rule base follows[1]:

```
Rule: 1
if     'the day' is 'Monday'
or     'the day' is 'Tuesday'  or 'the day' is 'Wednesday'
or     'the day' is 'Thursday' or 'the day' is 'Friday'
then   'today' is 'a workday'
```

[1]The rules 7–10 are correct from the author's point of view, because he comes from Australia.

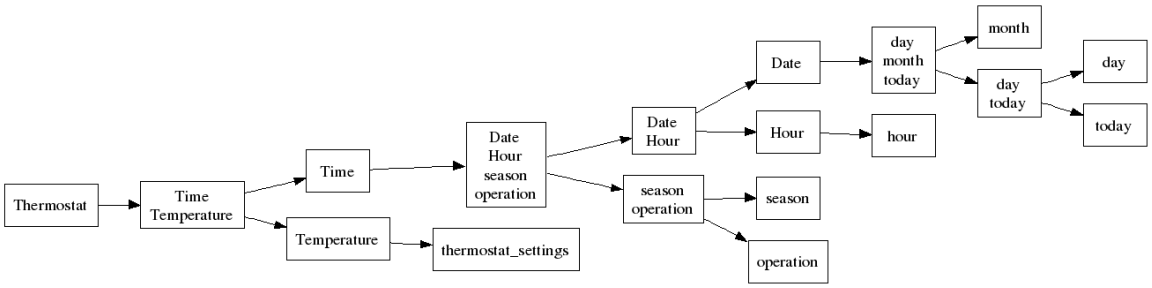

Figure 16: The TPH diagram of the Thermostat (rendered by the VARDA).

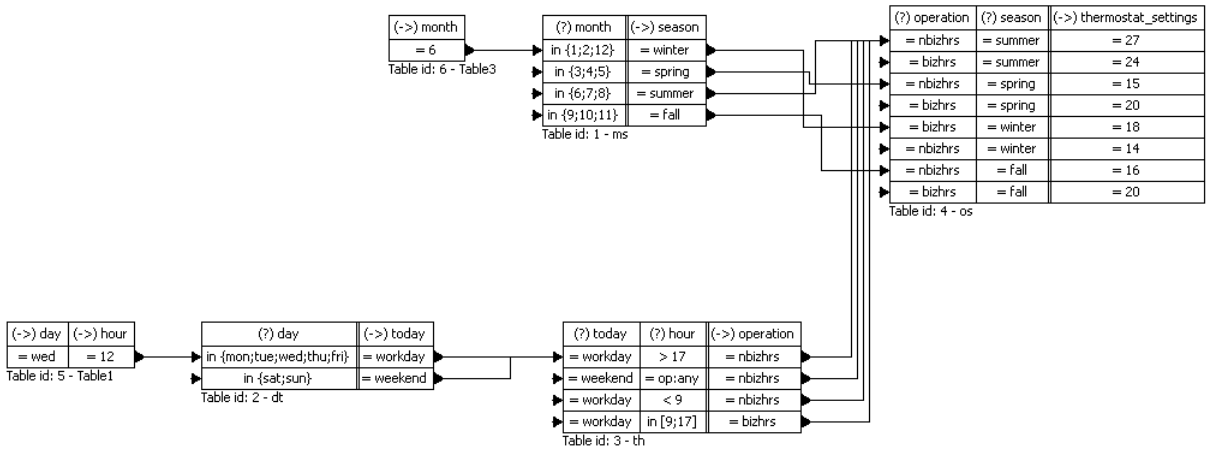

Figure 17: The XTT model of the Thermostat (rendered by the HQed).

```
Rule: 2
if    'the day' is 'Saturday'
or    'the day' is 'Sunday'
then  'today' is 'the weekend'

Rule: 3
if    'today' is 'workday'
and   'the time' is 'between 9 am and 5 pm'
then  'operation' is 'during business hours'

Rule: 4
if    'today' is 'workday'
and   'the time' is 'before 9 am'
then  'operation' is 'not during business hours'

Rule: 5
if    'today' is 'workday'
and   'the time' is 'after 5 pm'
then ' operation' is 'not during business hours'

Rule: 6
if    'today' is 'weekend'
then  'operation' is 'not during business hours'

Rule: 7
if    'the month' is 'January'
or    'the month' is 'February or the month is December'
then  'the season' is 'summer'
```

```
Rule: 8
if    'the month' is 'March'
or    'the month' is 'April or the month is May'
then  'the season' is 'autumn'

Rule: 9
if    'the month' is 'June'
or    'the month' is 'July' or 'the month' is August'
then  'the season' is 'winter'

Rule: 10
if    'the month' is 'September'
or    'the month' is 'October or the month is November'
then  'the season' is 'spring'

Rule: 11
if    'the season' is 'spring'
and   'operation' is 'during business hours'
then  'thermostat_setting' is '20 degrees'

Rule: 12
if    'the season' is 'spring'
and   'operation' is 'not during business hours'
then  'thermostat_setting' is '15 degrees'

Rule: 13
if    'the season' is 'summer'
and   'operation' is 'during business hours'
then  'thermostat_setting' is '24 degrees'

Rule: 14
if    'the season' is 'summer'
and   'operation' is 'not during business hours'
then  'thermostat_setting' is '27 degrees'

Rule: 15
if    'the season' is 'autumn'
and   'operation' is 'during business hours'
then  'thermostat_setting' is '20 degrees'

Rule: 16
if    'the season' is 'autumn'
and   'operation' is 'not during business hours'
then  'thermostat_setting' is '16 degrees'

Rule: 17
if    'the season' is 'winter'
and   'operation' is 'during business hours'
then  'thermostat_setting' is '18 degrees'

Rule: 18
if    'the season' is 'winter'
and   'operation' is 'not during business hours'
then  'thermostat_setting' is '14 degrees'
```

The process of designing the *Thermostat* system was accomplished successfully. It consist of the six tables, including two *fact* tables. The *fact* tables are used for setting the initial values of attributes.

6.2 Cash point

This example presents a cash point system that is equivalent to an ATM. The example shows the typical operations that are executed on the ATM. The user that wants to use the ATM, can choose one of the two types of operations:

- Withdraw the cash.
- Inquire about balance.

Both of cases require the PIN authentication. When the PIN is correct the operation will start, otherwise the authentication operation will be repeated until the moment when the PIN code will be correct or the limit of the attempts will be exceeded. There are only three attempts to enter the correct PIN code, after that the card will be kept in the cash point.

The goal of this example is the design the cash point system with the help of XTT. The example comes from [5]. The first step is to identify the use cases of the cash point. In the second step it must be defined what is input and output. In the third step the parameters should be specified and in the last step the rules should be defined.

- The inputs to the system: PIN, the user's actions.
- The outputs: confirmation, cash, message.

The system, designed with use of the XTT, consists of the attributes that are described in Table 2. The last level of ARD diagram as well as the transformation process history diagram are showed in Figures 18 and 19. The XTT model of Cash Point is presented in Figure 20.

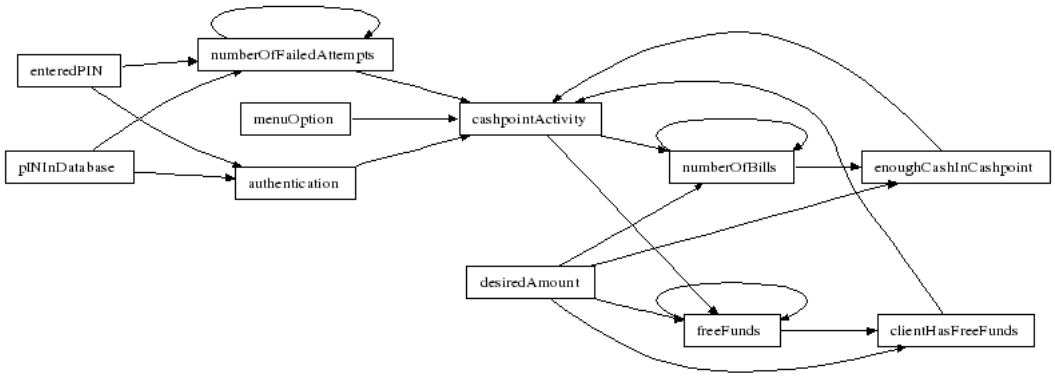

Figure 18: The ARD model of Cash Point.

Each row represents one rule. The rules can be written as follows:

```
Rule: 1
if    numberOfBills is greater or equal to desiredAmount
then  enoughCashInCashpoint is true

Rule: 2
if    numberOfBills is less than desiredAmount
then  enoughCashInCashpoint is false

Rule: 3
if    desiredAmount is greater or equal to freeFunds
then  clientHasFreeFunds is true
```

Name	Type	Domain	Description
numberOfBills	float	> 0	The value of the bills in the cash point.
desiredAmount	float	> 0	The amount that the user wants to withdraw.
enoughCashInCashpoint	boolean		Denotes if there is enough bills in the cash point to pay the desired amount.
freeFunds	float	> 0	The value of the free funds of the user.
clientHasFreeFunds	boolean		Denotes if the user has free funds.
enteredPIN	integer	<0, 9999>	The PIN number, entered by the user.
pINInDatabase	integer	<0, 9999>	The correct PIN number.
authentication	bool		Denotes if the user has entered the correct PIN code.
numberOfFailedAttempts	integer	<0, 3>	The number of attempts of entering the PIN number.
menuOption	enumerate	{payout, balanceInquiry}	Denotes the action that user wants to execute.
cashpointActivity	enumerate	{Ask for PIN(), Take away card(), Pay out(), Not Enough Funds In Machine(), Not Enough Funds On Account(), Not Enouch Funds(), Display Balance()}	This attribute is an output from the system. It denotes the action, which the cash point should executed.

Table 2: XTT Cash Point attributes specification.

```
Rule: 4
if    desiredAmount is less than freeFunds
then  clientHasFreeFunds is false

Rule: 5
if    enteredPIN is pINInDatabase
then  authentication is true

Rule: 6
if    enteredPIN is pINInDatabase
then  authentication is false
and   numberOfFailedAttempts is numberOfFailedAttempts + 1

Rule: 7
if    authentication is false
and   numberOfFailedAttempts is less than 3
then  action: 'ask_for_pin'
```

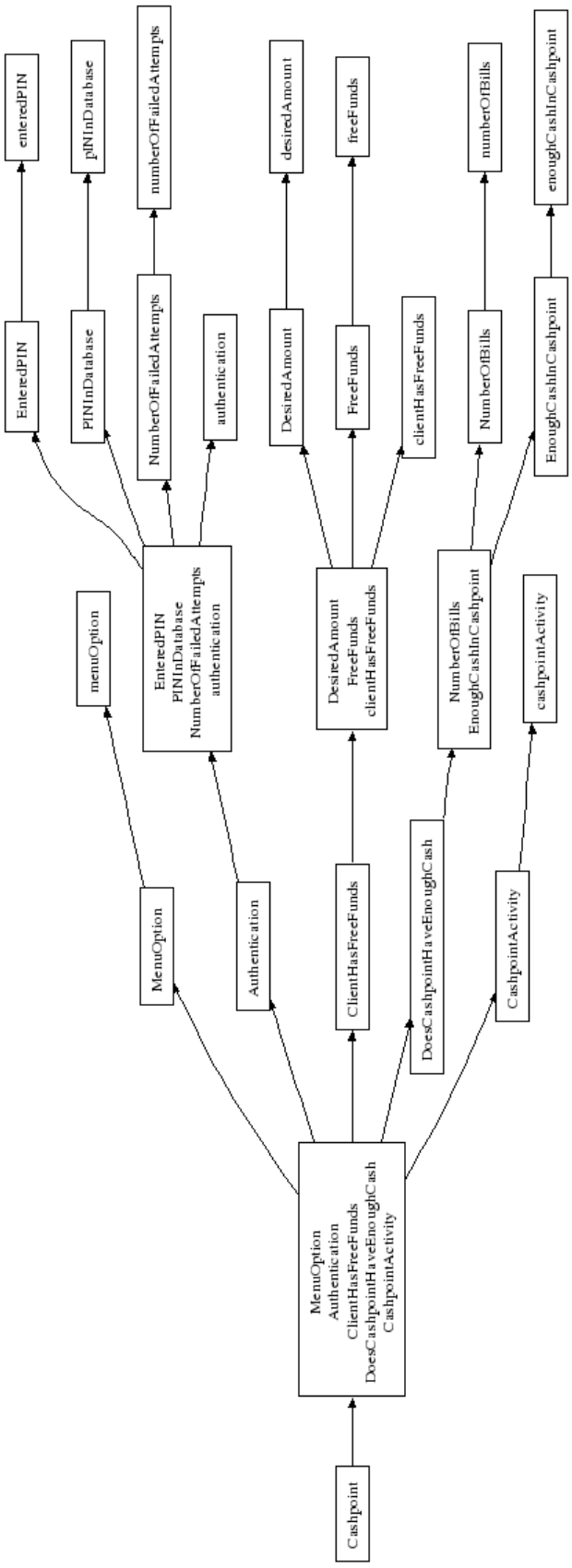

Figure 19: The transformation history of Cash Point.

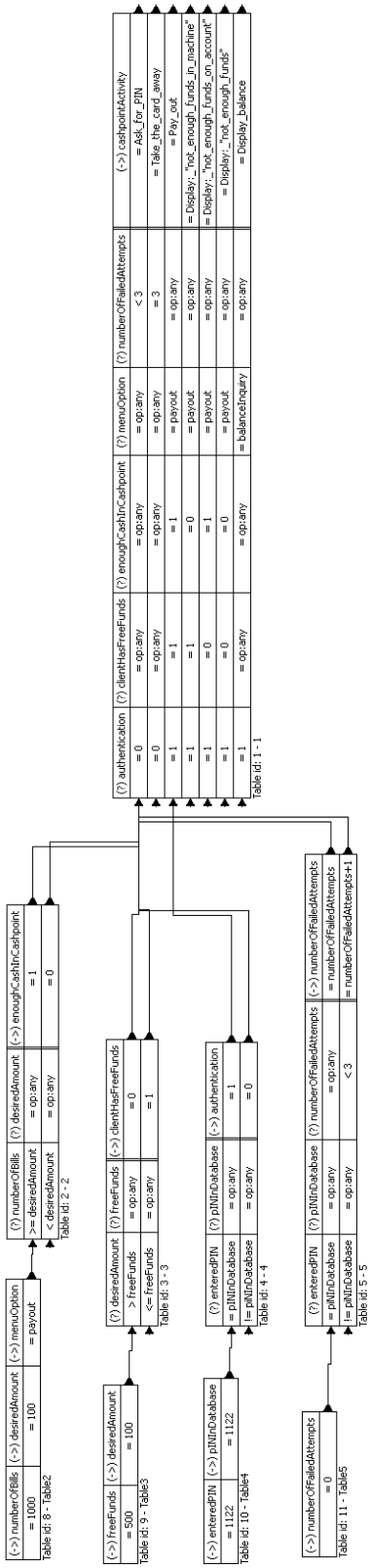

Figure 20: The XTT model of Cash Point.

```
Rule: 8
if    authentication is false
and   numberOfFailedAttempts is 3
then  action: 'take_away_card'

Rule: 9
if    authentication is true
and   clientHasFreeFunds is true
and   enoughCashInCashpoint is true
and   menuOption is 'payout'
then  action: 'payout'

Rule: 10
if    authentication is true
and   clientHasFreeFunds is true
and   enoughCashInCashpoint is false
and   menuOption is 'payout'
then  action: 'not enough funds in machine'

Rule: 11
if    authentication is true
and   clientHasFreeFunds is false
and   enoughCashInCashpoint is true
and   menuOption is 'payout'
then  action: 'not enough funds on accound'

Rule: 12
if    authentication is true
and   clientHasFreeFunds is false
and   enoughCashInCashpoint is false
and   menuOption is 'payout'
then  action: 'not enough funds'

Rule: 13
if    authentication is true
and   menuOption is 'balanceInquiry'
then  action: 'display_balance'
```

The process of the designing the *Cash point* system was accomplished successfully. The XTT model consists of nine tables including the four *fact* tables. Similarly to the first example, the *fact* tables are used to set the initial values of the attributes. The simulation of the XTT model runs without any problems. The value of the `cashpointActivity` attribute, is the system response to the initial values of the attributes.

7 Future work

In the report the problem of the design and implementation of a CASE tool which supports the XTT design process has been discussed. The implementation process of the tool has been accomplished successfully. The tool effectively supports the XTT design method. The graphical user interface as well as the visualisation of the model have been implemented. Additionally the application allows the user to view the ARD model. The ARD model can be loaded from a ARDML file and then reconstructed and displayed. Moreover the algorithm of the generating the XTT schema according to ARD model has been implemented as well.

HQed is a work in progress. Two main directions for the future work include: enhanced Plugins interface, and an API for Prolog

Plugins interface Plugins allows the application to add new elements providing new functionality. To use plugins an interface allowing the data exchange between the tool and the plugin should be designed. The enhancements of this part of the application are planned.

API for Prolog Plugins can extend the application's functionality, but they are mainly implemented in of the languages such as JAVA, C++. PROLOG is the language of choice for implementing high level analysis modules for the XTT rulebase, to formally verify model features, such as: Redundancy, Reduction, Consistency, and Completeness.

So far HQed has been successfully used in teaching by number of students in 2007/2008 class of Knowledge Engineering for practical design of XTT-based systems. Moreover, the tool plays an important role as the XTT modeling tool of choice for the HeKatE project (`hekate.ia.agh.edu.pl`).

References

[1] S. Burbeck. Applications programming in Smalltalk-80(tm): How to use Model-View-Controller (MVC). Technical report, Department of Computer Science, University of Illinois, Urbana-Champaign, 1992.

[2] T. Connolly, C. Begg, and A. Strechan. *Database Systems, A Practical Approach to Design, Implementation, and Management*. Addison-Wesley, 2nd edition, 1999.

[3] J. Giarratano and G. Riley. *Expert Systems. Principles and Programming*. Thomson Course Technology, Boston, MA, United States, 2005.

[4] K. Kaczor. Design and implementation of a unified rule base editor. Master's thesis, AGH Univerity of Science and Technology, June 2008. Supervisor: G. J. Nalepa.

[5] M. Kamiński. Design of control system cases with ARD+/XTT+. Knowledge Engineering Project (MIW), G. J. Nalepa supervisor.

[6] T. Kania and T. Szypenbejl. Metody inżynierii wiedzy – Projekt: Edytor diagramów ARD. AGH UST, 2006. Supervised by G. J. Nalepa, Ph. D.

[7] A. Ligęza. *Logical Foundations for Rule-Based Systems*. Springer-Verlag, 2006.

[8] D. Marynowski. Metody inżynierii wiedzy – Projekt: Mirella_DiaARD. AGH UST. Supervised by G. J. Nalepa, Ph. D.

[9] G. J. Nalepa and A. Ligęza. A graphical tabular model for rule-based logic programming and verification. *Systems Science*, 31(2):89–95, 2005.

[10] G. J. Nalepa and A. Ligęza. A visual edition tool for design and verification of knowledge in rule-based systems. *Systems Science*, 31(3):103–109, 2005.

[11] G. J. Nalepa and I. Wojnicki. Proposal of visual generalized rule programming model for Prolog. In D. Seipel and et al., editors, *Proceedings of the 17th International Conference on Applications of Declarative Programming and Knowledge Management (INAP 2007) and 21st Workshop on (Constraint) Logic Programming (WLP 2007)*. Technical Report 434, pages 195–204, Wurzburg, Germany, October 4–6 2007.

[12] G. J. Nalepa and I. Wojnicki. Towards formalization of ARD+ conceptual design and refinement method. In D. C. Wilson and H. C. Lane, editors, *Proceedings of the 21st International Florida Artificial Intelligence Research Society Conference (FLAIRS-21)*, pages 353–358, Coconut Grove, Florida, USA, 15–17 May 2008. AAAI Press.

[13] M. Negnevitsky. *Artificial Intelligence. A Guide to Intelligent Systems*. Addison-Wesley, 2002.

[14] Trolltech. Qt product. `http://trolltech.com/products/qt`, 2008.