Extensible Design and Verification Environment for XTT Rule Bases

Krzysztof Kaczor, Grzegorz J. Nalepa

Institute of Automatics,
AGH – University of Science and Technology,
Al. Mickiewicza 30, 30-059 Kraków, Poland
kk@agh.edu.pl, gjn@agh.edu.pl

Abstract *This paper discusses practical aspects of building an extensible design environment for XTT rule bases. The environment allows for an integration of number of services. These services include the formal verification framework, as well as a logic server that runs the designed XTT rules. In the paper the architecture of the services is discussed. Main integration issues are presented, and the communication protocol is proposed. Use scenarios for the main services are given.*

1. Introduction Many intelligent systems design methods were developed in Artificial Intelligence. Such systems are based on declarative methods of knowledge representation and processing [8]. Rules are a basic knowledge representation method. The systems that use rules are known as Rule-Based Systems (RBS) [1].

The HeKatE project (hekate.ia.agh.edu.pl) aims at providing an integrated methodology for design of RBS. The main motivation behind the project is to speed up and simplify the design of RBS. The methodology divides the whole design process into three stages: conceptual, logical and physical.

The XTT method (*eXtended Tabular Trees*) [2] provides the visualization during the design phase. Rules are grouped in decision tables and presented in a graphical way. Such a representation facilitates creating clear and transparent model. The formal description enables the model quality verification during the design. This facilitates the detection and fixing of errors [2]. Having a complete XTT diagram a physical system implementation can be generated automatically.

The methodology provides not only a set of methods, but also a set of tools supporting it. The main purpose of this paper is a presentation of the Hekate Design Environment (HaDEs for short). After a short introduction to the XTT method in Section 2, the paper presents the most important information about the design tools. Section 3 describes the HQEd tool that supports XTT method. Next, in the Section 4 the paper introduces the XTT inference and verification engine called HeaRT. In Section 5 presents the way of a communication between the tools that are involved in the design. In this Section the architecture of the above mentioned services is discussed. Main integration issues are presented, and the communication protocol is proposed. Use scenarios for the main services are given in Section 6. The short summary and the future works directions are described in Section 7.

ARD is a conceptual design method. It identifies system attributes and dependencies between them. ARD is a hierarchical method that allows for gradual increasing of details level of a system description. It starts with the most general system description, where system is described by single conceptual attribute. In the next steps the system becomes more precisely defined. The last ARD level identifies the system using only the physical attributes and dependencies between them. ARD can be treated as a supportive design method for XTT. The preliminary XTT schema may be generated according to ARD diagram.

2. XTT Design Method XTT [3, 5] is a logical design method for rules. To prototype the rules the ARD method is used. ARD is a conceptual design method. It identifies system attributes and dependencies between them. ARD is a hierarchical method that allows for gradual increasing of

details level of a system description. The preliminary XTT schema may be generated according to ARD diagram.

XTT is the second and the most important stage of the whole design process provided by the methodology. This stage constitutes a logical level of design, during which all the rules are defined. Visualization is the main advantage of the method. It provides a clear model design. This is important for the complex models, which consist of a large number of rules. Such a model allows for detecting the errors and anomalies during the design phase. The formal decription supports formal verification (e.g. completeness).

The rules in XTT are formally described in an attributive logic. This logic is called ALSV(FD) (*Attribute Logic with Set Values over Finite Domains*) [5]. According to ALSV(FD), each attribute has a type, value and domain. The domain must always be defined as a finite set. ALSV(FD) provides two types of attributes: *Simple attribute* that can take only one value from the domain, and *Generalised attribute* that can take any subset of the domain as a value.

XTT tables are the basic visualization elements. XTT table is composed of rows, that correspond to rules. Each column has assigned one attribute that is displayed in a column header. Thus, each rule in the table has the same prototype. That means they have the same conditional and decision attributes. A rule consists of two parts: conditional and decision. The separation between parts is displayed as a double vertical line. See Fig. 2 for an XTT structure example.

The row may be connected to table by link. This forms a tree-like structure. Link interpretation is: inference control, partial order. The inference control examines the rules in top-down order. The rule is fired when the precondition is satisfied. Next, the inference control goes to the next rule in the table, or follows the link.

XTT enables generation of a physical implementation. The generation process produces a Hekate Meta Representation (HMR for short) of the XTT diagram [4].

3. HQEd Design Tool The tool makes the visual design on the XTT level possible. What is more, it provides a mechanizm for formal verification, which allows for checking a model against the syntax and logical anomalies. The tool has built–in engine of syntax monitoring that checks the model against such errors as: inaccessible rules, values out of domain, etc. The logical anomalies are detected by other tool that constitutes logic server (see Sect. 4).

The platform independence is the most important issue of the tool implementation. This effect has been reached by use of the Qt library. The Qt library is a cross-platform application development framework. It has a fully object-oriented architecture. Qt supports a various of technologies such as XML, OpenGL, SQL. A source code created by using Qt can be compiled on such platforms as X11, Windows, MacOS, and embeded systems based on GNU Linux.

The source code of HQEd is divided into three layers, which correspond to MVC design pattern:

- *Model* – is responsible for the internal XTT and ARD model representation.
- *View* – provides the user interface as well as is responsible for the communication with the application.
- *Controller* – provides the communication between the layers.

HQEd has a user friendly graphical interface. GUI is designed with respect to an intuitive and convenient use. Each window dialog has the appropriate input/output controls. This prevents the user from entering an incorrect data.

The Figure 1 shows the schema of the architecture. The *controller* is the most important ele-

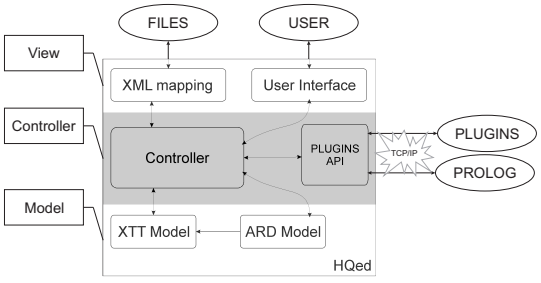

Figure 1. Architecture schema

ment of the architecture. It enables a flow of data between the layers. It corresponds to the MVC *controller* layer. The *model* consists of two layers that are responsible for the internal models representation. *ARD Model* stores an ARD model data, while the *XTT Model* stores an XTT model data. The remaining layers are included in the *view* that maps the model to the other formats:

- User Interface – the visual model representation that is appropriate for an user. This layer generates the graphical XTT diagram.
- XML mapping – translates a *model* data to a XML based language called HML (HeKatE Markup Language). The layer allow for storing and restoring the state of a model by use of the files. The HML file format supports storing both the ARD and XTT model data.
- Plugins API – provides the communication between HQEd and the other services.

The visual editor can be integrated with a custom rule interpreter HeaRT.

4. HeaRT Inference Engine HeKatE Run Time (HeaRT) is a dedicated inference engine for the XTT^2 rule bases [6]. It is implemented in Prolog in order to directly interpret the HMR representation which is generated by HQEd. HMR (HeKatE Meta Representation) is a textual representation of the XTT^2 logic designed by HQEd. It is a human readable form, as opposed to the machine readable HML format. HeaRT allows to: store and export models in HMR files, and verify HMR syntax and logic.

An example excerpt of HMR is:

```
xschm dt: [day] ==> [today].
xrule dt/1:
      [day in [mon,tue,wed,thu,fri]]
    ==>
      [today set workday]
    :th/1.
xrule dt/2:
      [day in [sat,sun]]
    ==>
      [today set weekend]
    :th/1.
```

The first line defines an XTT table scheme defining all of the attributes used in the table. Its semantics is as follows: "the XTT table *dt* has one conditional attribute: *day* and one decision attribute: *today*". Then two examples of rules are given. The second rule can be read as: "Rule with ID *2* in the XTT table called *dt*: if value of the attribute *day* belongs to the set sat, sun then set the value of the attribute *today* to the value *weekend*".

The engine implements the inference based on ALSV(FD). It supports four types of inference process, Data and Goal Driven, Fixed Order, and Token Driven [5]. Inference is based on assumption, that the system is deterministic. Conflicts should be handled during design process or detected by verification.

HeaRT also provides a modularized verification framework, also known as HalVA (HeKatE Verification and Analysis). Verification and analysis module implements: simple debugging mechanism that allows tracking system's work, logical verification of models (several plugins are available, including completeness, determinism and redundancy checks), and syntactic analysis of HMR files using a DCG grammar of HMR. The verification plugins can be run from the interpreter or indirectly from HQEd using the communication protocol.

The engine has communication and integration facilities. HeaRT supports Java integration based on callbacks mechanism and Prolog JPL library. It allows direct interaction via Prolog console based on callbacks mechanism. HeaRT can operate in two modes, stand-alone and as TCP/IP server, offering TCP/IP integration mechanism with other applications. It is possible to create console or graphical user interface build on Model-View-Controler design pattern.

There are two types of callbacks are related to attributes in HMR files:

1. Input – used to get attribute value from user. This can be done by console or graphical user interface.
2. Output – used to present attribute value to user.

Callbacks can be use to create GUI with JPL and SWING in Java.

5. Service Integration The methodology provides a set of tools, which offer a different functionalities. For instance on the one hand HQEd supports the visual design on the logical level, on

the other hand HeaRT tool provides formal verification of XTT. The communication between this two tools makes the on-line verification possible. There are several ways to provide communication between two separated applications: network-based communication, file-based communication, CORBA, standart I/O redirection, etc. Owning to common support for network technologies, HQEd communicates with HeaRT through a TCP/IP-based protocol. The editor sends a request for a model verification and receives a feedback message that contains the information about result.

The logic server is not the only tool that HQEd communicates with. The other tools can provide other services that may concern data representation, GUI modification, etc. All the tools that provide services are servers. The clients connect to a server and use the services.

A single service that is offered by a server constitutes an additional functionality of a client. Thus the servers can be treated as plugins.

The whole communication goes through the dedicated protocol. The set of commands has been divided into four groups:

- Verification commands – responsible for performing a formal verification of XTT and sending result to the connected clients.
- Simulation commands – used to execute and report of a XTT model simulation.
- Presentation commands – allow for changing value presentation format.
- Translation commands – allow for the translation between different formats of a knowledge representation.

The protocol does not support the session mechanizm. The clients are not constantly connected to the servers. They establish a connection only when they want to send a request. The connection is closed after server response or timeout.

The format of the protocol messages is very similar to the format of lists in the Prolog language. The elements are separated by commas and grouped with the help of the square brackets. An example of simple protocol commands is:

```
[model, getlist].
```

This command is a request for list of all the stored models in a server repository. The response of a server can look as follows:

```
[true,
 [
  [username, modelname],
  [otherusername, othermodelname]]].
```

The first element indicates if the message contains a result of a request (`true`), or error message (`false`). The second element is a list of pairs. Each pair contains a name of the client (`username`) that is a owner of model given as `modelname`.

The protocol provides a very simple error handling mechanizm. When request is incomplete, has incorrect format etc, then the server sends the information about error to the client. The error command has the following format:

```
[false, 'Error message'].
```

The first element must be set to `false`, in order to indicate error occurence. The second element is a quoted string containing an error message.

6. Rule Modeling with HQEd The complete XTT diagram consists of: tables, rules, connections. The design starts with the types definition. The sequence defining policy is not specified. However, the most convenient defining sequence is as follows: types, attributes, tables, rules, and links. The tool forces this sequence partially. For instance: a table can be defined when exists at least one attribute.

A definition of an attribute starts with a domain specification. The *Type Editor* window dialog is designed for the domain definition. The action *Type manager* from the *Types* menu calls this dialog. The *New* button allow for creating a new type. The action *Attributes* → *New attribute* calls the *Attribute editor* dialog. The dialog allows for creating the attributes.

The next step relies on the tables and rules defining. Firstly, the table schema must be defined. It assigns the attributes to conditional or decision part of a rule. The *Table editor* dialog allows for definition of a table schema. Additionally dialog enables a table title and type defining. The last stage of a table definig relies on a rules specification. Each row corresponds to a rule. The rule

is defined when all the cells in the row are specified. *Cell editor* window dialog allows for creating and editing the rows and cells. The action *Cell editor* form the *Tables* menu calls this dialog.

The last XTT design stage relies on linking tables. There are two ways to define a link: using the *Connection editor* dialog, or the *drag&drop* technique.

The following example is adapted from [7] and shows a design process of ATM behaviour: cash withdrawing, and balance inquiring. Each operation requires the correct PIN number. The user has three attempts to enter the correct PIN number. After three unsuccessful attempts the card is returned to the user. If an user tries to withdraw some money, the ATM checks:

- if the required amount does not exceed the ATM resources,
- if the user has enough free funds.

The input attributes are: PIN number and user actions. The system outputs are: cash and displayed information. The detailed description of the case can be found in [7].

HQEd enables the design of described system. The complete XTT diagram for the ATM case is shown in Fig. 2. The tool checks the model quality during the whole design process. It immediatelly notifies the user of the discovered anomalies. HQEd supports a syntax checking. It tries to detect all the anomalies besides the logical ones. It is very hard to enter incorrect value, because after each step HQEd checks entered values if they are compatible with the defined domain. However, when tool detects an incorrect value then it shows appropriate message to the user, which can be similar to the given:

```
Argument value: pay_out is not
compatible with type cashpointActivity
Type error message: The value is out
of range.
Type: cashpointActivity
Domain: Ask_for_PIN/1 U
Take_the_card_away/2 U
Pay_out/3
Value: pay_out
```

As it was mentioned, the logical verification is performed by HeaRT tool, which constitutes a logic server. HeaRT communicates with HQEd through TCP/IP protocol. When the XTT model

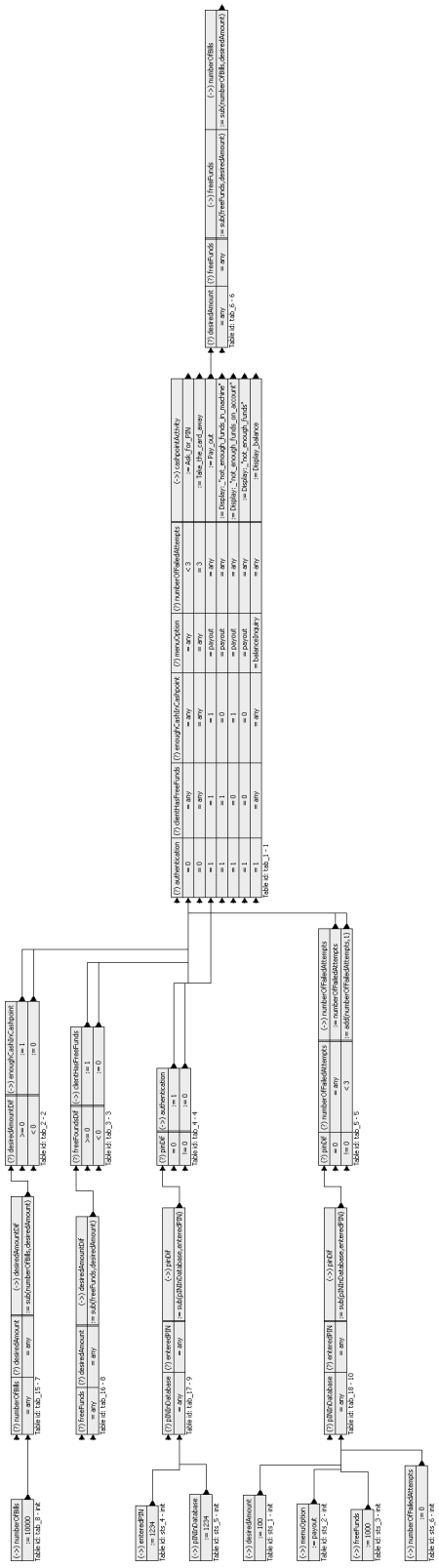

Figure 2. XTT model of the ATM use case

is changed, HQEd sends a request for a verification of the model. The examples of requests are:

```
[model,verify,vreduce,h1,m691,t1].
[model,verify,vcontradict,h1,m691,t2].
[model,verify,vcomplete,h1,m691,t3].
```

The command requests for a verification of a table *t1* in terms of reduction, contradiction, and completeness. The examples of responses are:

- `[true, []].` – there are no logical anomalies in the given table.
- `[true, [[4, 'In t2 rule: 4 can be joined with rule 8'], [8, 'In t2 rule: 4 can be joined with rule 8'], [8, 'In t2 rule: 8 can be joined with rule 4'], [4, 'In t2 rule: 8 can be joined with rule 4']]]` – there are two rules that can be merged into a single one.
- `[true, [[2, 'In t3 rule: 2 conflicts with rule 3 for state: (menuOption) = (Pay_out)'], [3, 'In t3 rule: 2 conflicts with rule 3 for state: (menuOption) = (Pay_out)']]]` – a contradiction for value `Pay_out` has been detected. There are two different conclusions for this value.
- `[true, ['In t4 uncover state: (menuOption) = (Pay_out)']]]` – there is no rule that defines a conclusion for the given value.

7. Future work The main topic of this paper is the presentation of HeKatE design and verification framework. The crucial tools HQEd and HeaRT have been presented and their the most important functionalities have been discussed. What is more, the issues of the tools integration have been introduced and the short description of the communication protocol have been presented.

The future work involves larger tools integration. This can be done by implementation of the new services, development and improvements of the communication protocol. The other ways of communication should also be considered. Tools testing is also a very important issue. The tools are to be tested during a design of the new, more complex cases. A set of use cases that can be compared with the same cases designed in other technologies (CLIPS, Drools) is in the works. This is important from the methodology point of view, because it makes the identification of the methods limitations possible as well as allows for benchmarking the HeKatE methodology.

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

References

[1] Joseph Giarratano and Gary Riley. *Expert Systems. Principles and Programming*. Thomson Course Technology, Boston, MA, United States, fourth edition edition, 2005. ISBN 0-534-38447-1.

[2] Antoni Ligęza and Grzegorz J. Nalepa. Visual design and on-line verification of tabular rule-based systems with xtt. In Klaus P. Jantke, Klaus-Peter Fähnrich, and Wolfgang S. Wittig, editors, *Marktplatz Internet: Von e-Learning bis e-Payment : 13. Leipziger Informatik-Tage, LIT 2005*, Lecture Notes in Informatics (LNI), pages 303–312, Bonn, 2005. Gesellschaft fur Informatik.

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

[4] Grzegorz J. Nalepa and Antoni Ligęza. Prolog-based analysis of tabular rule-based systems with the "xtt" approach. In Geoffrey C. J. Sutcliffe and Randy G. Goebel, editors, *FLAIRS 2006 : proceedings of the nineteenth international Florida Artificial Intelligence Research Society conference : [Melbourne Beach, Florida, May 11–13, 2006]*, pages 426–431, FLAIRS. - Menlo Park, 2006. Florida Artificial Intelligence Research Society, AAAI Press.

[5] Grzegorz J. Nalepa and Antoni Ligęza. Hekate methodology, hybrid engineering of intelligent systems. *International Journal of Applied Mathematics and Computer Science*, 2009. accepted for publication.

[6] Grzegorz J. Nalepa, Antoni Ligęza, Krzysztof Kaczor, and Weronika T. Furmańska. Hekate rule runtime and design framework. In Gerd Wagner Adrian Giurca, Grzegorz J. Nalepa, editor, *Proceedings of the 3rd East European Workshop on Rule-Based Applications (RuleApps 2009) Cottbus, Germany, September 21, 2009*, pages 21–30, Cottbus, Germany, 2009.

[7] Pascal Poizat and Jean-Claude Royer. Kadl specification of the cash point case study. Technical report, IBISC, FRE 2873 CNRS - Universite d'Evry Val d'Essonne, France, Genopole Tour Evry 2, 523 place des terrasses de l'Agora 91000 Evry Cedex, January 2007.

[8] Frank van Harmelen, Vladimir Lifschitz, and Bruce Porter, editors. *Handbook of Knowledge Representation*. Elsevier Science, 2007.