Visual Design of Drools Rule Bases Using the XTT2 Method

4 authors, including:

Krzysztof Kaczor
AGH University of Science and Technology in Kraków
28 PUBLICATIONS   **300** CITATIONS

Krzysztof Kluza
AGH University of Krakow
91 PUBLICATIONS   **720** CITATIONS

Visual Design of Drools Rule Bases Using the XTT2 Method*

Krzysztof Kaczor[1], Grzegorz J. Nalepa[1], Łukasz Łysik[1], Krzysztof Kluza[1]

Abstract Drools is one of the most popular expert system frameworks. It uses rules in several formats as knowledge representation. However, Drools has several limitations. It does not support visual modeling of rules. Moreover, it does not assure quality of the knowledge base. In this paper an alternative way of knowledge base designing for Drools is proposed. The method extends the Drools design process by using the XTT2 rule representation and the HQEd visual rule editor. To deal with the differences between the XTT2 and Drools representations, the Drools Export Plugin for HQEd has been implemented. The proposed approach overcomes limitations of the Drools design methodology. Furthermore, it provides a visual rule design editor for Drools and supports formal verification of the model using an optional module.

AGH University of Science and Technology,
al. A. Mickiewicza 30, 30-059 Krakow, Poland
e-mail: {kk,gjn,llysik,kluza}@agh.edu.pl

1 Introduction

An expert system shell is a framework that facilitates a creation of complete expert systems [3]. Such systems are widely used as the Business Rules engines. They often constitute a pluggable software components, separate from the rest of the application. This allows for processing and modifying a knowledge base independently from the other parts of application. These systems can be adapted to the domain-specific problem by development of a proper knowledge base.

However, the knowledge base development is not a trivial task. To build a high quality knowledge base some mechanisms like efficient design or verification methods have to be used. There are techniques that allow for checking knowledge base on a user demand. However, fixing some errors in the knowledge base may introduce other errors. Thus, in order to provide better verification efficiency, the knowledge base should be continuously monitored during the design.

Drools [1], as an advanced framework for the Rule-Based System development, provides a unified and integrated platform for rules and workflow processing. On one hand, Drools provides a mechanism for knowledge base development and management. On the other, these mechanisms have some limitations, which make the creation of a high quality knowledge base difficult.

This paper proposes an alternative way of knowledge base development for Drools. The method uses the XTT2 visual knowledge representation and the HQEd rule editor. This approach overcomes several limitations of the existing Drools design methodology. Because of the differences between the Drools and XTT2 representations, the translation algorithm for transforming these two formats has been developed, and a new tool, Drools Export Plugin for HQEd (DEPfH), has been implemented. In the proposed approach, a rule-based system can be designed with a user-friendly tool supporting visual rule modeling. Moreover, the HQEd tool allows for an on-line formal verification of the model, what assures high quality of the Drools knowledge base. The main goal of this paper is to provide a description of the HQEd plugin (DEPfH) as a bridge for this two technologies.

The paper is organized as follows: In Section 2 a short description of the Drools business rules framework is given. Section 3 presents the motivation for this research. Section 4 provides a proposal of a new rule design method for Drools. In this section the XTT2 knowledge representation is described. To demonstrate the XTT2 integration with Drools, as well as the proposed design process a short example is presented in Section 5. The evaluation of the approach with the related research is provided in Section 6. A brief summary is given in Section 7.

2 Drools Rules Framework

Drools[2] is one of the Business Rules engines, which solves many software design problems by separating Business Logic and processes from the software source code. Currently, Drools consists of five modules: *Guvnor* (Business Rules/Processes Management System), *Expert* (Rule Engine), *Fusion* (Complex Event Processing), *Flow* (Process/Workflow) and *Planner*. Our research focuses on using two of them: *Expert*, as the main rules processing module, and *Flow*, which introduces visual modeling of a workflow.

Drools offers Eclipse IDE tools with a textual rule editor, which provides context assistance, syntax highlighting, and syntax error checking. Alternatively, rules can be created in a guided editor, which does not require the knowledge about rule syntax and is more suitable for unexperienced users. Rules, as one of the most basic form of a knowledge representation, follow the *if-then* schema. In Drools they can be complemented by meta information called attributes, and control the inference process. They can specify a rule priority, an agenda, a rule flow, etc. Moreover, a user can create Java functions, and include them into a rule base if needed. Especially, helper functions can be applied to rules or decision tables. They can be used in rule conditions inside `eval()` conditional element, e.g.: `Person(eval( validAge(age) ), sex = 'M')`.

Decision tables are one of the Drools mechanisms to manage sets of rules having the same schema. However, Drools does not provide any design tool for creating decision tables. They can be created in MS Excel, OpenOffice or as comma-separated values (CSV) file. It is important to mention that during inference process, decision tables are transformed into separate rules.

A single decision table contains rules of the same schema. Each rule, stored as a table row, consists of columns marked as "CONDITION" (left-hand side part of a rule) and "ACTION" (right-hand side of a rule). Each attribute and logical operator takes one column header while table cells contain values corresponding to appropriate table headers. Number of rows in the table corresponds to the number of rules.

Expert, the core of Drools, has mechanisms that make declarative programming possible. It moves the responsibility for the logic from Java objects to Drools. Consequently, the knowledge maintenance process is much easier.
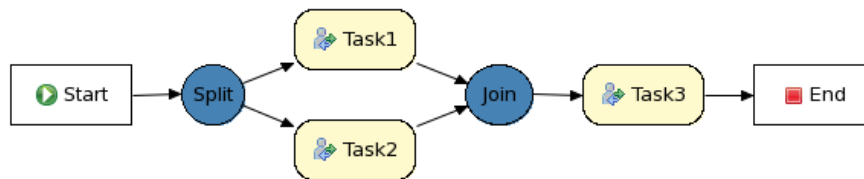
Another Drools module – Drools Flow – contains a graphical editor for designing the inference. The inference flow constitutes a graph consisting of special blocks, as well as transitions that can be joined and splitted. There is also a ruleflow-group block that contains sets of rules. When process reaches such a block, the rules in this block are evaluated. Although the rule base in Drools is generally flat, the Flow introduces some kind of structural modeling. Rules have a flat structure, but they can be complemented by *Flow*, introduces structural modeling.

The graph is used to control the inference process. The block of a "ruleflow-group" can have the rulegroups assigned.

---

[2] See: `http://www.jboss.org/drools` for Drools project from JBoss.org.

Fig. 1 Spreadsheet-based rules modeling

Fig. 2 Drools Flow diagram.

Rules, that are used in *Flow* has to be assigned to the appropriate diagram elements. It can be done, by setting *ruleflow-group* attribute.

```
1  rule "Hello World"
2   ruleflow-group "welcome"
3   when
4     Person(name == "Chuck")
5   then
6     System.out.println("Hello Chuck");
7  end
```

Listing 1 Example rule with "ruleflow-group" attribute assigned to "welcome" group.

Despite the variety of features, Drools does not provide knowledge verification. Although rule syntax is checked, many errors can still be undetected during the design process, and they can appear during the execution. This can be partially solved by using tools, such as Drools Verifier or VALENS (described in Section 6).

3 Motivation

There are several problems in the existing approaches to the rule design in Drools. They become most evident in the case of designing complex systems which contain large rule bases. It is difficult to provide the high quality design as well as the refinement of such systems. These problems are related to:

1. *Flat rule base* – the knowledge base is flat. Although Drools provides two rule representation methods, none of them is appropriate to create a structured knowledge base. They are not able to represent the complex knowledge base clearly. The large set of rules becomes unclear and the dependencies between rules are unreadable. This makes the gradual system design impossible.
2. *Lack of formalization* – the high quality rule base cannot contain errors especially on the logical level. There are some mechanisms that attempt to verify the existing knowledge. However, the lack of the formal rule definition prevents its full formal verification, and this decreases rule base quality.
3. *Lack of a dedicated visual rule editor* – Drools provides a few methods of rules development (text editor, guided editor, as well as extracting rules from the decision tables). However, it does not provide any dedicated decision table editor. A decision table have to be created with the help of a spreadsheet, and this makes Drools not user-friendly and inconvenient for this purpose.

Next section provides a proposal of a new approach to rule modeling in Drools. This method tries to solve the main problems following from the above analysis:

1. *Structured rule base* – introducing internal structure of the rule base. The similar rules that work together in a specific context are grouped into single decision table. This allows for compact and transparent knowledge base representation in case of large rule bases.
2. *Formal rule definition* – This issue is crucial for the formal verification task.
3. *Dedicated visual rule editor* – allows for gradual and visual modeling of the decision tables and the inference flow between them.

Section 4 introduces the method and tools, which are proposed as an alternative way for rule design method for Drools.

4 Solution Proposal

The problems considered in the previous section can be solved with the help of the Extended Tabular Trees (XTT2) method. This method is a part of the Hybrid Knowledge Engineering (HeKatE) [9] methodology for designing, implementing and verifying production knowledge-based systems.

Because the rule base in Drools is flat and rules are created as separated and, modeling of the systems with a large knowledge base is very complicated. The XTT2 method supports designing of a hierarchical structure of a knowledge base. Thanks to that, the design and maintenance of the complex systems is more efficient.

Although Drools provides many rule design features, rule verification is not supported. On the other hand, the XTT2 method offers on-line formal rule verification during the design process.

The integration of Drools and XTT2 is a solution to the limitations of Drools mentioned above. In this paper we propose to extend the Drools modeling process by using HQEd, a visual editor for XTT2 rule designing. A new Drools Export Plugin for HQEd has been implemented for this purpose. The XTT2 method, HQEd as well as Drools Export Plugin are described in the next subsections.

4.1 XTT2 Rule Representation

The main goals of the XTT2 representation are to provide an expressive formal logical calculus for rules as well as a compact, structural and visual knowledge representation and design. The representation allows for advanced inference control and formal analysis of the production systems.

The XTT2 method provides formal rule representation language based on the Attributive Logic with Set Values over Finite Domains (ALSV(FD)) logic [6]. The ALSV(FD)-based knowledge representation makes the XTT2 language more expressive than the propositional logic. Formal definition of the XTT2 language makes the formal verification and analysis possible. During the design process, the model of the system is continuously checked against syntax and logical errors. This allows for discovering the errors at the time they appear. On the logical level the model is checked for several logical anomalies, such as consistency, redundancy, determinism and completeness [7].

The advantage of the method is the internal structure of the XTT2 knowledge representation. The rule base is not flat, and the rules that work together are grouped into contexts. Hence, the inference mechanism can work more efficient because only the rules from the focused context are evaluated. Moreover, the internal representation and structure makes the knowledge base suitable for the visual editors. The knowledge base can be depicted as the graph (see: Fig. 3). The tables constitute the graph nodes and correspond to the contexts in the knowledge base. The rules that belong to a specific context are placed in a single table in a network of tables.
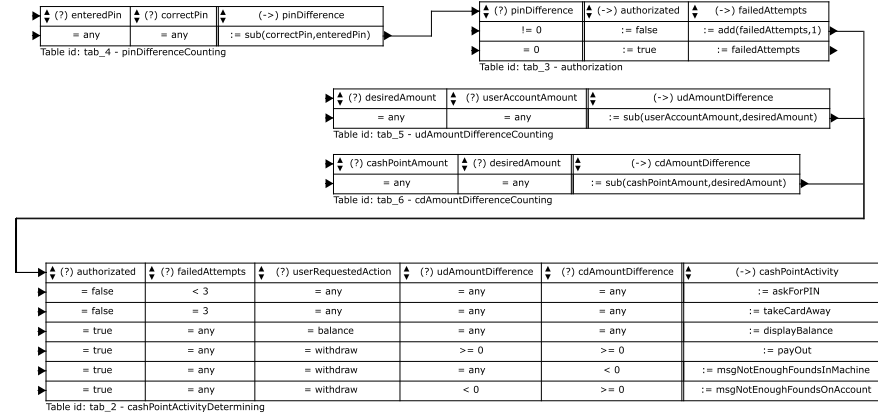
Table id: tab_4 - pinDifferenceCounting

(?) enteredPin	(?) correctPin	(->) pinDifference
= any	= any	:= sub(correctPin,enteredPin)

Table id: tab_3 - authorization

(?) pinDifference	(->) authorized	(->) failedAttempts
!= 0	:= false	:= add(failedAttempts,1)
= 0	:= true	:= failedAttempts

Table id: tab_5 - udAmountDifferenceCounting

(?) desiredAmount	(?) userAccountAmount	(->) udAmountDifference
= any	= any	:= sub(userAccountAmount,desiredAmount)

Table id: tab_6 - cdAmountDifferenceCounting

(?) cashPointAmount	(?) desiredAmount	(->) cdAmountDifference
= any	= any	:= sub(cashPointAmount,desiredAmount)

Table id: tab_2 - cashPointActivityDetermining

(?) authorized	(?) failedAttempts	(?) userRequestedAction	(?) udAmountDifference	(?) cdAmountDifference	(->) cashPointActivity
= false	< 3	= any	= any	= any	:= askForPIN
= false	= 3	= any	= any	= any	:= takeCardAway
= true	= any	= balance	= any	= any	:= displayBalance
= true	= any	= withdraw	>= 0	>= 0	:= payOut
= true	= any	= withdraw	= any	< 0	:= msgNotEnoughFoundsInMachine
= true	= any	= withdraw	< 0	>= 0	:= msgNotEnoughFoundsOnAccount

Fig. 3 An example of the XTT2 diagram for CashPoint example

Within the HeKatE project a number of tools supporting the visual design and implementation of the XTT2-based systems have been developed (see [5]). The two most important, HQEd and HeaRT, are described in next section.

4.2 HQEd Visual Editor

HeKatE Qt Editor (HQEd) [4] is a editor for XTT2-based expert systems. In the proposed approach, HQEd is a key tool, because it provides all the necessary mechanisms for the rules visual modeling as well as for integration with the external systems. This tool allows a user to design a rule base in a visual way using network of the decision tables. It also provides an interface for plugins which can extend its functionality and allow for integration with other systems. It provides all the necessary functionalities and features that facilitate creation of a high quality knowledge using user-friendly graphical interface.

The HQEd tool provides two mechanisms that assure the high quality of the knowledge base:

- *Visual design* – the visual knowledge representation makes the knowledge base more transparent what allows for more effective finding and fixing errors and anomalies.
- *Verification* – the design is supported by two types of the verification, that can discover the anomalies and errors that were omitted by designer. The HQEd tool supports two types of verification, which are provided by the XTT2 methodology:
  - *Syntax verification* – HQEd stores an internal representation of a created system. This representation is continuously monitored against syntax errors, e.g.
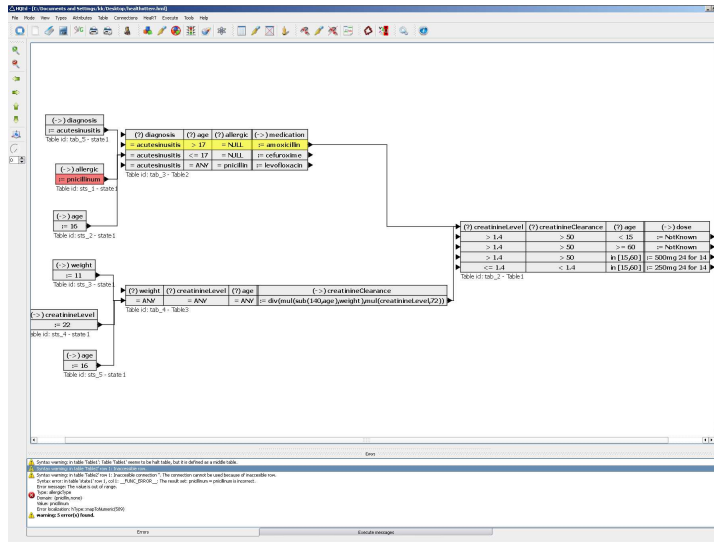
Fig. 4 HQed editing session

out of domain values, inaccessible rules, orphan tables, etc. When an error appears, the editor displays the appropriate message for the user (see Fig. 5).

– *Logical verification* – Although HQEd does not have a built-in logical verification engine, the model is verified by HeKatE Run Time (HeaRT), an external tool working as the HQEd plugin. With HeaRT [8] the model is continuously checked against logical errors, such as completeness, contradictions, determinism and redundancy.

The discovered anomalies are showed to the user (see Fig. 6).
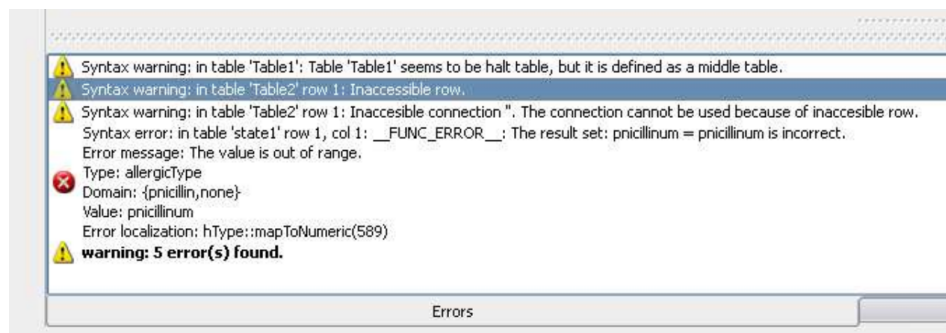
Fig. 5 Syntax errors and warnings in HQEd
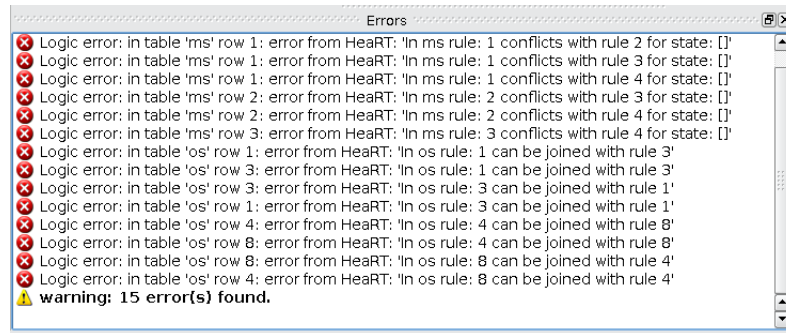
Fig. 6 Logical errors and warnings in HQEd

The architecture of HQEd consists of three layers, which correspond to Model-View-Controller (MVC) design pattern (see: Fig. 7). *Model* is responsible for the internal 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.
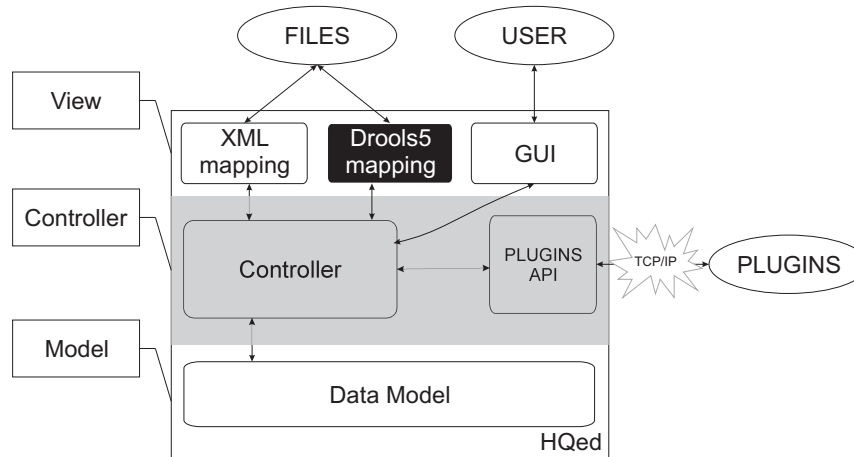
Fig. 7 HQEd architecture

In our approach, the *Data model* stores the internal representation of the XTT2 knowledge base. The remaining layers are included in the *View* that maps the *Model* to the other formats:

- GUI – the visual model representation, appropriate for a user. This layer generates the graphical XTT2 diagram.
- XML mapping – translates the *Model* data into an XML-based language and allows for saving and reading model to/from files.

- Drools mapping – translates the *Model* data into the Drools files.
- Plugins API – provides the TCP/IP-based communication between HQEd and other services.

HQEd has been implemented with the use of the Qt library[3] that makes it a cross-platform tool. Moreover, the tool allows for integration with other technologies by using plugins. Next subsection provides a description of a Drools Export Plugin, which has been implemented to integrate Drools and XTT2.

4.3 Drools Export Plugin for HQEd

In Section 3 several problems of existing approaches to the rule design in Drools have been described. These limitations can be overcome by using the XTT2 method. Although both Drools and XTT2 use rules as a knowledge representation, they store them in different formats. Therefore, a translation from XTT2 to Drools model is necessary. Drools Export Plugin for HQEd (DEPfH) implements an algorithm, which performs this non-trivial task.

The basic element of the XTT2 structure is a decision table. Such a table consist of rules of the same schema. A header of the table contains attributes, which are common for all the rules in the table. The remaining parts of the rules are stored in the table cells. On the other hand, Drools does not support decision tables directly. Sets of rules can be stored in spreadsheets in a table-like form. However, the structure differs significantly from the XTT2 tables. In this case, the logic operators are stored with attributes in table header, and are common for all the column cells.

Another difference between the XTT2 and Drools representations is the method of decision table linking. In XTT2 each rule can have a link to another rule in the network of tables. These connections are used during the inference process and indicate a rule execution order. In Drools, in turn, rules in tables can not be connected directly. The connections are allowed only between ruleflow-groups, which can contain sets of rules or spreadsheet decision tables. It is worth mentioning that the Drools decision tables are used only during the design process and they are not used during the inference process.

These differences have been taken into consideration during the DEPfH implementation. In the exported Drools model additional control elements have been introduced. Each rule has an additional action and condition, which influence the inference process. The action sets a global variable `rule_to_execute` to the ordinal number of the rule in the next table, which is to be executed. The condition provides the mechanism for executing the proper rule.

The runtime environment for Drools is Java. The DEPfH plugin uses the dedicated Java helper function to map ALSV(FD) operators to Java expressions. The function takes three parameters: a value of the attribute, a literal symbol of the logical operator, and the value to be compared to. Using this function the operator can

---

[3] See: `www.trolltech.com`.

be stored in the table cell, despite the fact that in original Drools model the logic operators are stored with attributes in table header.

The XTT2 and Drools models differ in the file structures as well. The XTT2 model stores all the model elements (rules, tables, attributes and attribute types) in a single XML-based file. Drools, in turn, keeps the model elements in separate parts: Drools Flow, decision tables, and additional Java classes (see: Fig. 8).

To deal with this problem, the DEPfH plugin splits an XTT2 model into three files: one with the XTT2 model attributes, one with the decision tables, and additional file containing the inference flow.

Due to space limitation of this paper we omit the translation algorithm description and implementation details of the DEPfH plugin.

The class is named `Workspace` and has the same name for each generated model. In the second file the decision tables can be stored. The files has a comma-separated-values (CSV) structure. This form is supported by a Drools inference engine and can be easily deploy. To overcome the logical operator placement problem, all the comparisons are made using `Compare` function, which was mentioned above. This requires generation of another Java class, `WorkspaceHelper`. The last file contains the *Flow* of the rule base. It is an XML-based file which contains blocks for decision tables and also transitions between them. The model is complemented by a starting end ending tables which are required by a Drools model. Here emerged one more difference between XTT2 and Drools structures. There are differences between XTT2 and Drools structures. XTT2 does not introduce the concept of starting end ending tables. Moreover, the XTT2 tables can have multiple incoming and outgoing connections. The solution for this is to export to Drools format additional control elements to the *Flow*, such as *Start*, *End*, as well as *Split* and *Join* elements.

Furthermore, Drools is designed for Java-based applications. Java, as an object-oriented language, requires that all the attributes are the parts of objects. To overcome this restriction, all the XTT2 attributes are wrapped in one Java class. Hence, another file, *Workspace.java* is being generated.

5 CashPoint Example

To demonstrate the design and integration scenario of XTT2 and Drools, a Cash-Point example has been selected (see Fig. 3). The example presents a simple, yet real-life system, studied as one of the benchmark cases in the HeKatE project. CashPoint is a device that allows a person to access to his/her savings. This can be achieved by inserting a card and typing in a Personal Identification Number (PIN). Then, PIN is compared with a code stored in the card. After successfully identification, a customer may withdraw a cash or display the balance.

The first phase of the integration scenario begins in the HQEd tool. The tool is used for XTT2 design process. The design starts with specification of the attribute types. (See Figure 9). Each of XTT2 attributes can be of one of the primary types:
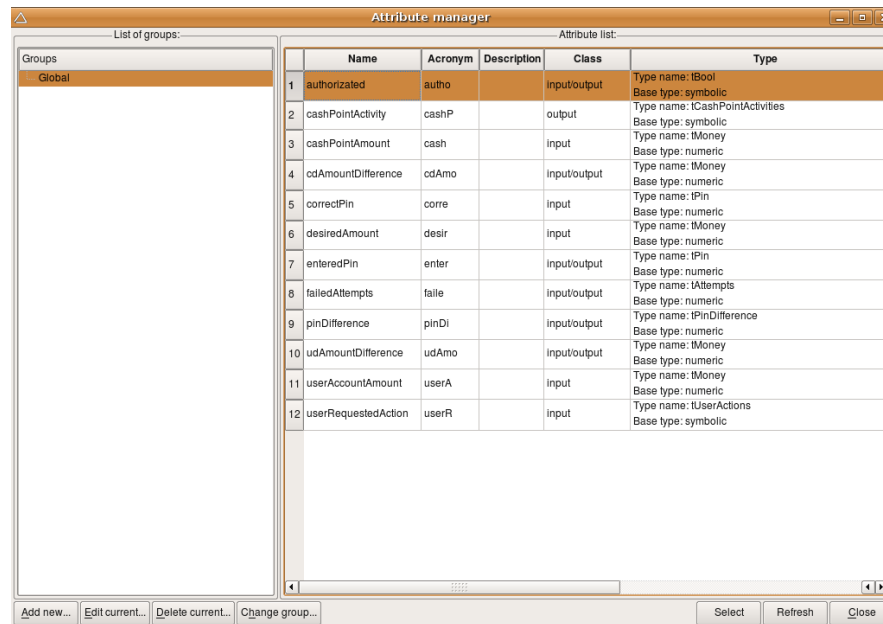
XTT Model
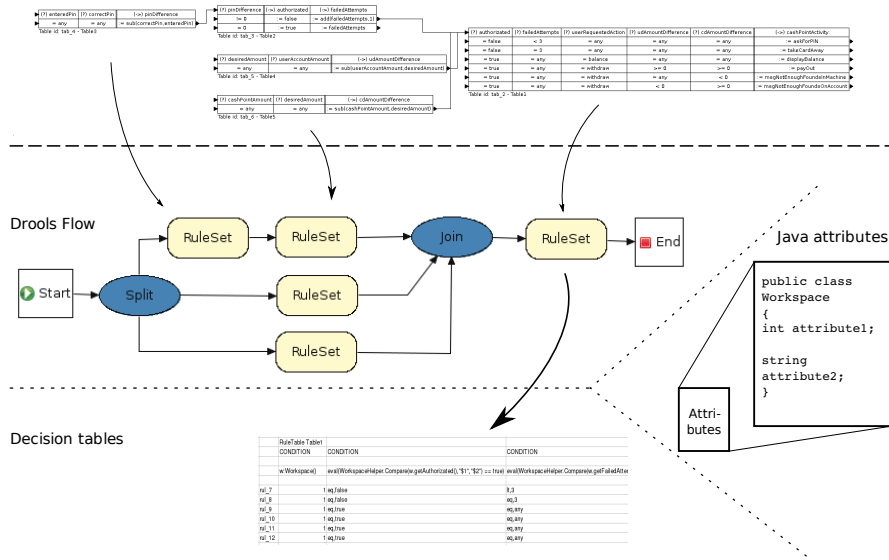
Fig. 8 An example of the XTT2 model and its Drools equivalent

(boolean, integer, numeric, symbolic, or user-defined type). The user can, however, define own types which base on the primary types. Each of the user-defined types can have the constraints defined, such as domain and, in case of numeric data type, scale and precision.

Fig. 9 Defining attributes' types in HQEd model

Next, the model attributes are identified. An attribute is a variable which stores the process values and is used by the rules. When creating the attribute, the user has to define: name, acronym, description and attribute's type.
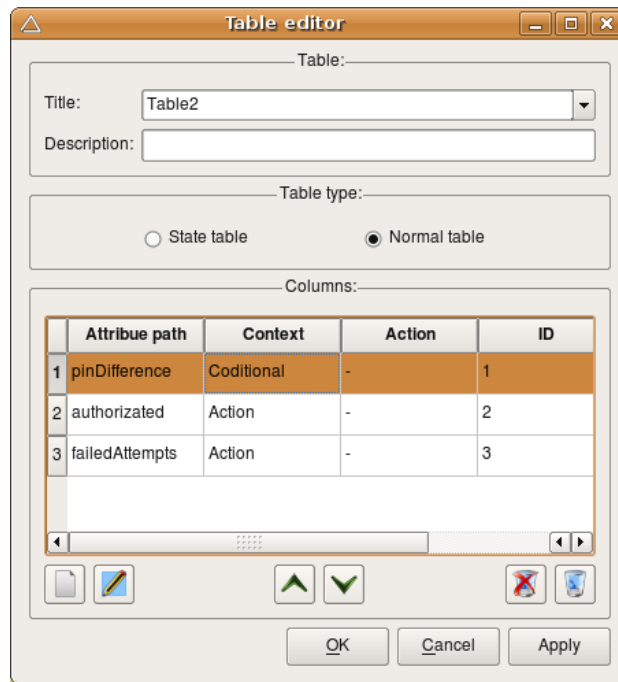
Fig. 10 Attributes list in Cash Point

When all the attributes are specified, the user can define the XTT2 tables. First, the table schema has to be indicated. The user chooses XTT2 attributes and decides if the attribute is to be the part of the condition or action of the rule.

Then, the table can be filled in with rules (each row of the table corresponds to one rule). In the Figure 12 the cell editor contains two rules. The first column on the left with the question mark signifies that this column belongs to the conditions. The latter two column, with arrows signs, are the actions to be taken, when the condition is true.

A table cell contains two items: the comparison sign and the value to be compared to. This can be changed in the expression editor (See Figure 13). The logical operators supported by HQEd are the following: *equal, not equal, greater than, lower than, greater or equal, lower or equal, in, not in*.

Some cells can contain actions, such as assigning constant value to the attribute, assigning the other attribute value to the attribute, or assigning the result of the complex expression to the attribute. On the other hand, the cell that contains actions, can do the following tasks:

1. They can assign constant value to the attribute.
2. They can assign the other attribute value to the attribute.
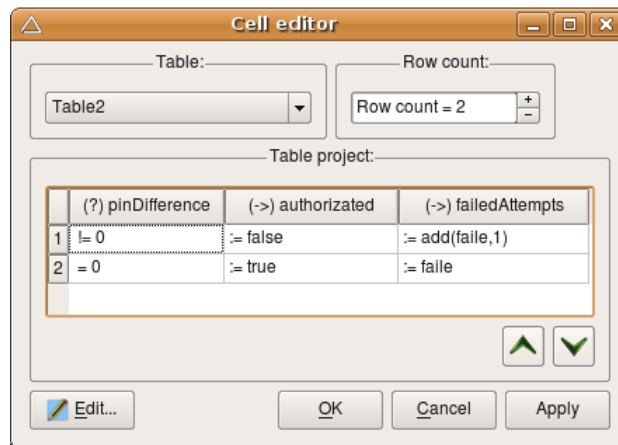3. They can assign the result of the complex expression to the attribute.

Fig. 11 Table editor in HQEd

Fig. 12 Cell editor in HQEd

The expression can be a combination of the following mathematical functions: *add, sub, mul, div, sin, cos, tan, log, abs, fac, pow, root*. The functions are supported only for integer and numeric attribute types.

Fig. 13 Expression editor

After filling the XTT2 tables with rules, connections between tables can be defined. Then, the model is complete (see: Fig. 3). A complete model can be observed in Fig. 3. Full XTT2 model is presented on the Figure 14. It consists of five tables and 11 rules. The table *CheckPIN* calculates the difference between given and correct PIN number. Another table, *CountIncorectPINs*, increases, if necessary, the counter of the incorrect PIN numbers entered. The table *CheckMoneyInAcount* count the difference between the requested amount of money and the amount on the customer's account. The difference between the cash stored in the cash point and the user requested amount is counted in the table *CheckMoneyInCashMachine*. The last, and the most significant table decides about the action to be taken. The possible actions are:

Ask for PIN   - when the customer types in an incorrect PIN less than 3 times.
Take card away   - when the customer types in an incorrect PIN more then 3 times.
Display balance   - when PIN is correct and the customer asked for the account balance.
Pay out   - when PIN is correct, the customer asked for money withdrawal and there is enough money both on the account and in the cash machine
Denial of the withdrawal   - when there is not enough money on the customer's account or in the cash point.

When the XTT2 model is complete it can be exported to the Drools format. The complete XTT2 model (shown in Fig. 3) can be exported using the DEPfH plugin to the Drools format (as depicted in Fig. 8). The result are three files containing Drools Flow, decision tables and Java classes (see: Fig. 8). In the output folder, three files are created. One file contains Rule Flow model (See Figure 15).

The number of block of "ruleflow-group" type is the same as the number of tables in the XTT2 model.

The second exported file contains the rules in a form of decision table (See Figure 16). The file has a comma-separated values format (CSV). The file contains the name of the package in which the rules will be placed. In the initial phase of Drools inference process, the rules are transformed into Java objects. The *RuleSet* attribute contains the name of the package.
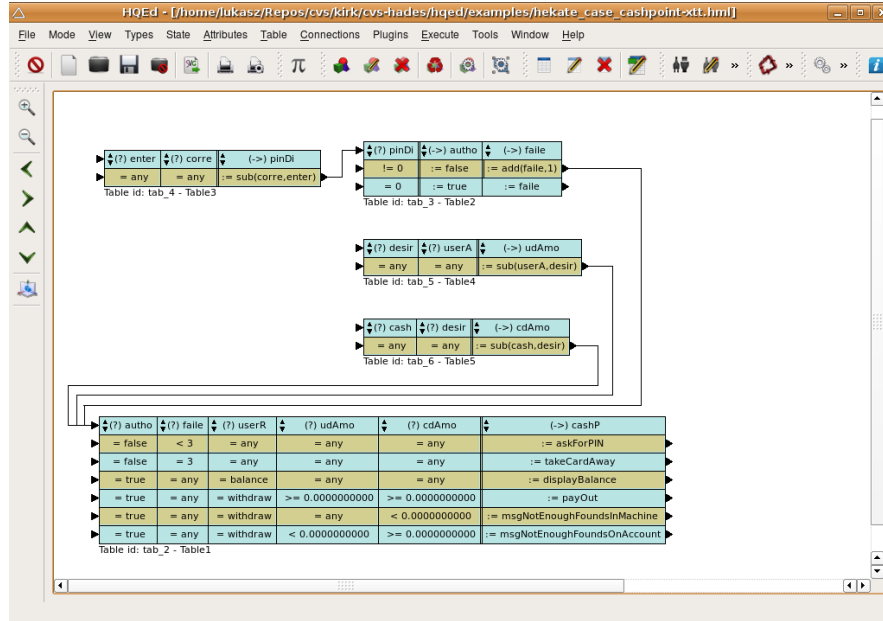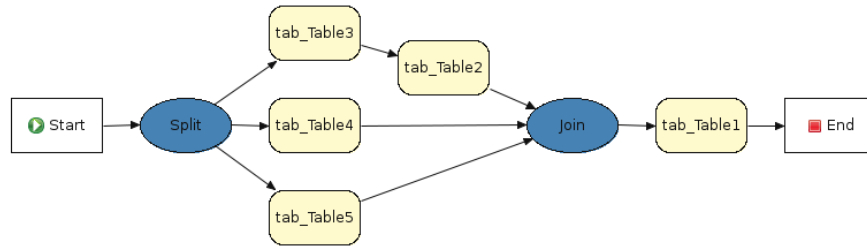
Fig. 14 Cash point model in HQed

Fig. 15 Rule Flow model of cashpoint

Each decision table begins from the *RuleTable* keyword. The structures of XTT2 table and Drools decision table is slightly different. In the XTT2 model the logical operator is placed in the table cell with the value, while in the Drools, the operator is placed in the table's header. This structure difference can be resolved by using helper function. This means that each Drools tables' cell contains also logical operator. The example header of the table's column it presented in the Listing 2. The *WorkspaceHelper.Compare* function takes three arguments. The first is the current value of the attribute. The latter two are the logical operator and the value used in the comparison - both taken The function performs the comparison. The given value can be a single value, range or set.

Fig. 16 Decision tables for cash point example

```
1  eval(WorkspaceHelper.Compare(w.getAuthorizated(),"$1","$2") ==
       true)
```

Listing 2 The example header of the Drools decision table.

The last generated file is the Java class which wraps all the attributes in the model. The last step of the integration requires to create knowledge base loader in the application. This can be done by using Drools classes. The example code which creates the Java object which accesses the knowledge is presented in Listing below.

```
private static KnowledgeBase readKnowledgeBase() throws Exception {
  KnowledgeBuilder kbuilder = KnowledgeBuilderFactory.newKnowledgeBuilder();

  DecisionTableConfiguration config = KnowledgeBuilderFactory.newDecisionTableC
  config.setInputType(DecisionTableInputType.CSV);

  kbuilder.add(ResourceFactory.newClassPathResource("RuleFlow.rf"), ResourceTyp
  kbuilder.add(ResourceFactory.newClassPathResource("DecTable.txt"), ResourceTy

  KnowledgeBase kbase = KnowledgeBaseFactory.newKnowledgeBase();
  kbase.addKnowledgePackages(kbuilder.getKnowledgePackages());

  return kbase;
}
```

Two Java classes have to be included into the application: *Workspace* and *WorkspaceHelper*. In this point, Then, the application is ready for running an inference process. When the workspace is created and all facts are specified, the workspace is loaded into the working memory and the inference process can be started. Later, the workspace have to be inserted into the working memory, and finally the inference process can be started.

ODL CODE Workspace ws = new Workspace(); ws.setMonth(1); ws.setHour(15); ws.setDay("mon"); ksession.insert(ws);

Map<String, Object> parameters = new HashMap<String, Object>(); parameters.put("workspace", ws); ksession.startProcess("com.sample.ruleflow",parameters); ksession.fireAllRules();

```
Workspace ws = new Workspace();
ksession.insert(ws);
Map<String, Object> parameters = new HashMap<String, Object>();
parameters.put("workspace", ws);
ksession.startProcess("com.sample.ruleflow",parameters);
ksession.fireAllRules();
```

There are two main advantages of using the presented XTT2 approach for modeling a knowledge base for Drools. The model has been designed in a convenient visual way, using a user-friendly tool. Moreover, the formal verification of the Cash-Point model has been provided by the HeaRT plugin. Therefore, the exported model has high quality assured.

6 Related research and evaluation

Although Drools does not provide any built-in rule verification mechanisms, there are some tools, which can be helpful in checking Drools rule base, such as Drools Verifier[4] and VALENS [2] from LibRT[5].

Drools Verifier is a dedicated tool for verifying the Drools knowledge base. It can discover several kinds of errors, such as rules redundancy, subsumption, equivalence, incoherence, or always true expressions. It can work continuously during the design process, as well as on a user demand. The detected errors are simultaneously reported with information about the quality of a knowledge base.

Unlike Drools Verifier, VALENS is not dedicated only for Drools. It allows for maintaining and verifying rule bases in different formats. In addition, a user can also import knowledge in the various representation formats such as production rules and decision trees. The tool is run as a standalone application, and allows for checking rules against consistency, completeness and correctness.

---

[4] See: http://community.jboss.org/wiki/DroolsVerifier.

[5] See: http://www.librt.com.

The tools mentioned above verify an original Drools rule base. However, they are not based on formal methods, so any formal verification of the rule base cannot be performed. In our approach, a Drools rule base is designed with the help of the XTT2 method. Thanks to formal rule representation language based on ALSV(FD) logic the formal verification and analysis during the design process is possible.

When it comes to rule visual designing, Drools does not provide any dedicated tool for creating the decision tables. Although Drools Flow supports visual modeling of the inference process, and decision tables can be created in spreadsheets or text files, the whole design process is not really visualized.

There are also attempts to use some rule visual modeling language, such as URML, however, these techniques allow for modeling only single rules, not the decision tables and the inference structure. HQEd supports the gradual visual design method for decision tables, which is more convenient to use than spreadsheets or text files used for Drools. Then, the DEPfH plugin, which implements a translation algorithm from XTT2 to Drools representation, can be used. This allows for executing the created knowledge base in Expert, the Drools rule engine.

The advantage of the proposed approach of knowledge base development for Drools is that the system is created with a user-friendly dedicated tool, which uses the formally defined method, and supports the visual design process. In this case the quality of the knowledge is assured by formal verification.

7 Summary

In this paper an alternative way of knowledge base development for Drools using the XTT2 method is proposed. The new modeling approach uses the HQEd rule editor, which overcomes limitations of the Drools design methodology. To deal with the differences between the XTT2 and Drools representations, the Drools Export Plugin for HQEd has been implemented.

There are two main advantages of the proposed approach. The system can be designed with a user-friendly tool that supports visual modeling of the knowledge base. Furthermore, the HQEd tool allows for on-line formal verification of the model. This assures high quality of the Drools knowledge base.

References

1. Browne, P.: JBoss Drools Business Rules. Packt Publishing (2009)
2. Gerrits, R., Spreeuwenberg, S.: Valens: A knowledge based tool to validate and verify an aion knowledge base. In: ECAI 2000, Proceedings of the 14th European Conference on Artificial Intelligence, Berlin, Germany, August 20-25, 2000, pp. 731–738 (2000)
3. Giarratano, J.C., Riley, G.D.: Expert Systems. Thomson (2005)
4. Kaczor, K., Nalepa, G.J.: Design and implementation of HQEd, the visual editor for the XTT+ rule design method. Tech. Rep. CSLTR 02/2008, AGH University of Science and Technology (2008)
5. Kaczor, K., Nalepa, G.J.: HaDEs – presentation of the HeKatE design environment. In: J. Baumeister, G.J. Nalepa (eds.) 5th Workshop on Knowledge Engineering and Software Engineering (KESE2009) at the 32nd German conference on Artificial Intelligence: September 15, 2009, Paderborn, Germany, pp. 57–62. Paderborn, Germany (2009)
6. Ligęza, A.: Logical Foundations for Rule-Based Systems. Springer-Verlag, Berlin, Heidelberg (2006)
7. Ligęza, A., Nalepa, G.J.: Proposal of a formal verification framework for the XTT2 rule bases. In: R. Tadeusiewicz, A. Ligęza, W. Mitkowski, M. Szymkat (eds.) CMS'09: Computer Methods and Systems: 7th conference, 26–27 November 2009, Kraków, Poland, pp. 105–110. AGH University of Science and Technology, Cracow, Oprogramowanie Naukowo-Techniczne, Kraków (2009)
8. Nalepa, G.J.: Architecture of the HeaRT hybrid rule engine. In: L. Rutkowski, [et al.] (eds.) Artificial Intelligence and Soft Computing: 10th International Conference, ICAISC 2010: Zakopane, Poland, June 13–17, 2010, Pt. II, *Lecture Notes in Artificial Intelligence*, vol. 6114, pp. 598–605. Springer (2010)
9. Nalepa, G.J., Ligęza, A.: HeKatE methodology, hybrid engineering of intelligent systems. International Journal of Applied Mathematics and Computer Science **20**(1), 35–53 (2010)