

Visual Inference Specification Methods for Modularized Rulebases. Overview and Integration Proposal*

Krzysztof Kluza, Grzegorz J. Nalepa, Łukasz Łysik

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

Abstract The paper concerns selected rule modularization techniques. Three visual methods for inference specification for modularized rulebases are described: Drools Flow, BPMN and XTT2. Drools Flow is a popular technology for workflow or process modeling, BPMN is an OMG standard for modeling business processes, and XTT2 is a hierarchical tabular system specification method. Because of some limitations of these solutions, several proposals of their integration are given.

1 Introduction

Rule-Based Systems (RBS) [1] constitute one of the most powerful knowledge representation formalisms. Rules are intuitive and easy to understand for humans. Therefore, this approach is suitable for many kinds of computer systems. Nowadays, software complexity is increasing. Because describing it using plain text is very difficult, many visual methods have been proposed. For example, in Software Engineering, the dominant graphical notation for software modeling is UML (the Unified Modeling Language). Design of large knowledge bases is non trivial, either. However, in the area of RBS, there is no single visual notation or coherent method. Moreover, the existing solutions have certain limitations. These limitations are especially visible in the design of large systems.

When the number of rules grows, system scalability and maintainability suffers. To avoid this, there is a need to manage rules. Rule grouping is a simple method of rule management. However, it is not obvious how to group rules. One of the most common grouping methods involves context awareness and creation of decision tables. Another grouping method takes rule dependencies into account and creates decision trees. This leads to RBS modularization.

This paper describes three possible solutions to modularize rule bases:

- Drools Flow [2], which is a popular technology for workflow modeling,

* The paper is supported by the *BIMLOQ* Project funded from 2010–2012 resources for science as a research project.

- BPMN (the Business Process Modeling Notation) [3], which is an OMG standard [4] for modeling business processes, and
- XTT2 (EXtended Tabular Trees), which is a result of authors' research project [5] and which organizes a tabular system into a hierarchical structure.

However, these solutions have some limitations. Drools Flow is platform-dependent and not standardized. Moreover, it has some flow design restrictions. BPMN is a notation for business processes, and it is not clearly stated how processes can co-operate with rules. Furthermore, BPMN can be mapped to BPEL (Business Process Execution Language) for execution, but this mapping is non-trivial and execution is not possible for every BPMN model. XTT2, in turn, is not wide-spread, and it is not a universal method.

The general problem considered in this paper is the RBS design and modularization. The article constitutes an overview and a proposal for integration of the three presented methodologies, which can be useful in solving above-mentioned problems. The next two sections present selected rule modularization techniques and an overview of selected visual design methods for rule inference. In Section 4, a proposal of rule translation from XTT2 to Drools is described, and in Section 5, a proposal of XTT2 inference design with BPMN is introduced. Section 6, discusses future work and summarizes the main threads of this article.

2 Rule Modularization Techniques

Most classic expert systems have a flat knowledge base. So, the inference mechanism has to check each rule against each fact. When the knowledge base is large, this process becomes inefficient. This problem can be solved by providing a structure in the knowledge base that allows to only check a subset of rules [6].

CLIPS [7] allows for organising rules into so-called *modules*, that restrict access to their elements from other modules. Modularisation of the knowledge base helps rule management. In CLIPS, each module has its own pattern matching network for its rules and its own agenda. Execution focus can be changed between modules stored on the stack.

JESS [8] also provides a module mechanism. Modules provide structure and control execution. In general, although any JESS rule can be activated at any time, only rules in the focus module will fire. This leads to a structured rule base, but still all rules are checked against the facts. In terms of efficiency, the module mechanism does not influence on the performance of *conflict set* creation.

Drools Flow provides a graphical interface for modelling of processes and rules. Drools 5 has a built-in functionality to define the structure of the rule base, which can determine the order of rule evaluation and execution. Rules can be grouped in ruleflow-groups which define the subsets of rules that are executed. The ruleflow-groups have a graphical representation as nodes on the *ruleflow* diagram. They are connected with links, which determines the order of evaluation. Rule grouping in Drools 5 contributes to the efficiency of the ReteOO algorithm, because only a subset of rules is evaluated. However, there are no policies which determine when a rule can be added to the ruleflow-group.

3 Selected Visual Design Methods for Rule Inference

Efficient inference would not be possible without a proper structure and design of rule-based system. The important issues in dealing with this problem is grouping and hierarchization of rules, as well as addressing the contextual nature of the rulebase. The following subsections describe selected methods and tools, in which visual design of rule inference is possible.

3.1 Drools

Drools is a rule engine which offers knowledge integration mechanisms. The project is run by the JBoss Community, which belongs to the Red Hat Foundation. It is divided into four subprojects: Guvnor, Expert, Flow and Fusion. Each of them supports different part of integration process.

Expert is the essence of Drools. It is the actual rule engine. It collects facts from the environment, loads the knowledge base, prepares the agenda and executes rules. A modified version of the Rete [9] algorithm is used for the inference.

The *knowledge base* in Drools consists of three main elements: rules, decision tables and Drools Flow. The fundamental form of knowledge representation in Drools is a rule. This form is easy to use and very flexible. Rules are stored in text files which are loaded into the program memory by special Java classes. Rules in Drools can be supplemented with attributes which contain additional information. They have a form of name-value pairs and they describe such parameters as rule priority and provide meta information for inference engine.

Rules which have the same schema can be combined into *decision tables*. A decision table is divided into two parts: the *left-hand side*, which represents the conditions of rules and the *right-hand side*, which represent the actions to be executed. One row in a table corresponds to one rule. However, decision tables, are useful only during the design phase. The structure does not improve the performance of the inference. Decision tables are, in fact, transformed into rules. So the inference engine does not recognize which rules come from decision tables and which are just a group of unrelated rules.

Rules form a flat structure. When the inference engine matches rules against facts, it takes all rules into consideration. The user, however, can define the flow of the inference process. Drools Flow offers a workflow design functionality in the form of blocks (See Fig. 1). The user can specify exactly which rules should be executed in which order and under which conditions.

Each model in Drools Flow has to contain two blocks: *start* and *end*. Rules and rule flow are linked together inside the *ruleset* block. Each *ruleset* block has a *ruleflow-group* attribute. Similarly, each rule has the attribute with the same name. Rules belong to the ruleset block with the same values of the ruleflow-group attribute. Additionally, the process can be split and joined. Two blocks, *split* and *join*, are used for that purpose. The block *split* has different types. The AND type defines that the process follows all the outgoing connections. The *join* block also has different types. The AND type waits for all the incom-

ming subprocesses to finish. The OR type waits for the first process to finish, while the n-of-m type waits until specified number of processes finish.

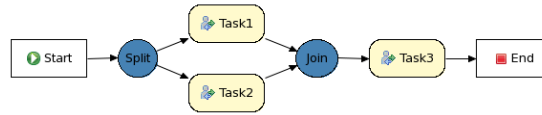


Figure 1. Sample Drools Flow diagram (drools.org)

Drools has some *limitations*. First of all, it is not a standardized solution. The form of knowledge representation still evolves. It could have been seen when version 5 was released - new block were introduced and the format of Drools Flow file has changed. Moreover, Drools does not provide any tools which can be used in the knowledge design phase. It can be problematic in large systems. What is more, the rulebase has a flat structure. Although, Drools Flow complements the structure by describing execution process, the rules still do not have a hierarchy. The last thing is that Drools is language dependent, closely related to Java. Parts of the rules and some Rule Flow blocks contain Java expressions.

3.2 XTT2

XTT2 (EXtended Tabular Trees) [5] is a hybrid knowledge representation and design method aimed at combining decision trees and decision tables. It has been developed in the HeKatE research project (hecate.ia.agh.edu.pl), and its goal is to provide a new software development methodology, which tries to incorporate some well-established Knowledge Engineering tools and paradigms into the domain of Software Engineering, such as declarative knowledge representation, knowledge transformation based on existing inference strategies as well as verification, validation and refinement.

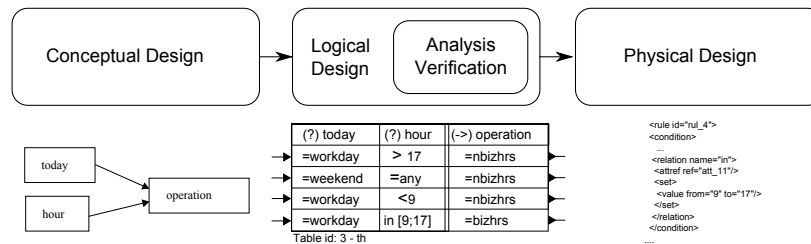


Figure 2. The HeKatE process

The HeKatE process consists of three design phases (shown in Fig. 2) [5]:

1. **The conceptual design phase**, which is the most abstract phase. During this phase, both system attributes and their functional relationships

- are identified. This phase uses ARD+ (Attribute-Relationship) diagrams as a modeling tool. It allows design of the logical XTT2 structure.
2. **The logical design phase**, in which system structure is represented as a XTT2 hierarchy. The preliminary model of XTT2 can be obtained as a result of the previous phase. This phase uses the XTT2 representation as a design tool. During this phase, on-line analysis, verification as well as revision and optimization (if necessary) of the designed system properties is provided.
 3. **The physical design phase**, in which the system implementation is generated from the XTT2 model. The code can be executed and debugged.

Some *limitations* of XTT2 can be pointed out. XTT2 provides a support for the entire process. It is used to model, represent, and store the business logic of designed systems. Rules in XTT2 are formalized with the use of the ALSV(FD) [5] logic and are supported by a Prolog-based interpretation. Although XTT2 rules are prototyped with the ARD+ method, the method is quite poor, and does not provide more advanced workflow constructs. Moreover, it is not a widely known methodology and only dedicated tools support it.

3.3 Business Rules and BPMN

BPMN [4] is a visual notation for business processes. A BPMN model defines the ways in which operations are carried out to accomplish the intended objectives of an organization. Visualization makes the model easier to understand. The goal of the notation is to provide such a notation which is easily understandable by business users. The notation provides only one kind of diagram – BPD (Business Process Diagram). There are four basic categories of BPD elements: Flow Objects (Events, Activities, and Gateways), Connecting Objects (Sequence Flow, Message Flow, Association), Swimlanes, and Artifacts. An example describing evaluation process of a student project is presented in Fig. 3.

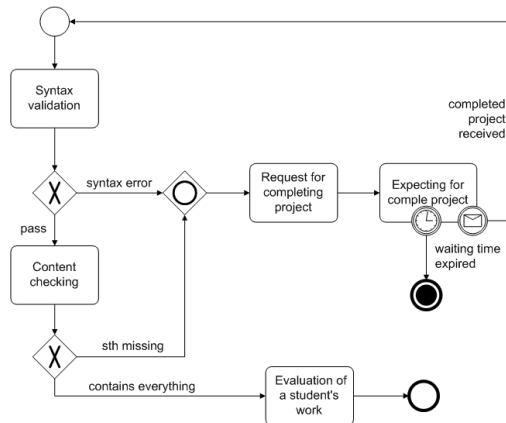


Figure 3. An example of Business Process Diagram

BPMN [4] has been developed by the Business Process Management Initiative (BPMI) and currently is maintained by the Object Management Group. Although the notation is relatively young, BPMN is becoming increasingly popular. According to OMG, there is more than 60 BPMN implementations of BPMN tools. Moreover, BPMN models can be serialized to XML and further processed e.g. into languages for execution of business processes, such as BPEL4WS (Business Process Execution Language for Web Services) [10].

Very often a Business Process (BP) is associated with particular Business Rules (BR), which define or constrain some business aspect, and are intended to assert business structure or to control or influence the business behavior [11]. According to the specification [4], BPMN is not suitable for modeling concepts, such as organizational structures and resources, data models, and business rules. There is a huge difference in abstraction level between BPMN and BR. However, BR may be complementary to the business process. In Fig. 4, an example from the classic UServ Financial Services case study has been shown. This example presents how business processes and rules can be linked.

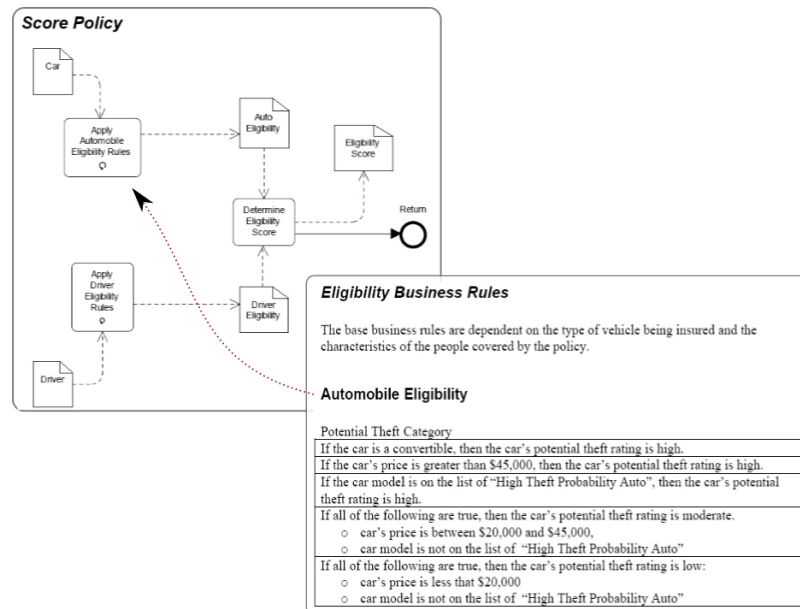


Figure 4. An example of using BR to define a process in Business Process Diagram

BPMN has some *weaknesses*. Although a specification defines a mapping between BPMN and BPEL (standard for execution languages), there is a fundamental difference between these two standards. One of the consequences of this difference is that, for instance, not every BPMN process can be mapped to BPEL and executed. Moreover, execution of the processes requires additional specification, which is not necessarily integrated with the entire design process.

Despite the fact that the BPMN model has well-defined semantics and a particular model should be clearly understood, there can be various models having the same meaning and there can be ambiguity in sharing BPMN models. Last but not least, it is difficult to assess the quality of the model.

3.4 Critical comparison

As one can see from the Table 1, each of these solutions has some pros and cons. Integration of these technologies based on their merits can bring better results than using them separately.

| | Drools 4 | BPMN | XTT2 |
|---|-----------------|-------------|-------------|
| Visual design of the rulebase | no | no | yes |
| Verification | no | some | yes |
| Workflow modeling (OR, AND etc.) | yes | yes | no |
| Runtime environment | yes | no | yes |
| Tool support | yes | yes | yes |
| Standardization | no | yes | no |

Table 1. Comparison of the three approaches

The disadvantages of the Drools Flow are platform dependency and lack of standardization. Drools Flow supports decision tables and grouping of unrelated rules. XTT2 allows multiple connections between tables. Although Drools only allows for a single connection, it provides *Join* and *Split* blocks.

The XTT2 connections are of the AND type, by default. However, the connection semantics is different than that in Drools or BPMN. In Drools and BPMN, the default inference process is forward chaining, while XTT2 provides various inference modes, e.g. forward chaining (where the connections are of the AND type) or backward chaining (where the semantics of connections varies).

BPMN is only a notation which has many elements for precise control of flow. However, this solution originally was not based on Rule-Based Systems. Therefore, it does not define the relationship between processes and rules. Although BPMN can be mapped to BPEL and executed, mapping and execution is possible only for selected groups of a BPMN model.

In case of XTT2, the entire design process is supported. What is more, formal on-line analysis can be performed during the design process, and then a prototype of the system can be generated. However, XTT2 is not a wide-spread solution, and does not pretend to be a universal method.

On the one hand, the comparison shows that XTT2 is the only one solution which supports visual modeling of the rulebase (modeling using decision tables). Moreover, only XTT2 provides formal verification. On the other hand, Drools offers workflow modeling. The integration of Drools and XTT gave the opportunity to combine these advantages. The next section describes the proposal of rule translation from XTT2 to Drools in detail, as part of the HeKatE project.

BPMN is already a well-known and standardized notation. In Drools 5, it can be used to model workflow. To facilitate workflow modeling for XTT2 and to provide an executable platform for BPMN, the integration of XTT2 with BPMN is considered. The possible scenarios are identified and described in Section 5. This research is a part of the *BIMLOQ* project (2010–2012).

4 Proposal of Rule Translation from XTT2 to Drools

Knowledge structure represented by Drools is very similar to the one represented by XTT2. In fact, that was one of the main reasons for choosing Drools as an integration platform. Both frameworks have the same goal: to provide rule-based and structured knowledge representation. On the one hand, XTT2 is a unified structure which contains both rules and inference flow. On the other hand, Drools has both of these features, but rules can exist without Drools Flow. Both solutions can be used to model business processes. Drools Flow even provides special blocks which contain Java source code to be executed. XTT2, however, is more flexible and language independent. It contains rules which do not have any dialect specific parts.

4.1 Generating Drools files

Knowledge represented in XTT2 is stored in XML form. One file contains a tree structure and rules. Drools with the Flow model, on the other hand, stores knowledge in at least two files: a file with rules and a file with a flow. The XTT2-to-Drools integration mechanism separates XTT2 rules from the structure, transforms them and puts into two separate Drools files.

Nevertheless, Drools operates on objects while XTT2 uses primitive types. In Drools, facts are instances of Java classes inserted into the working memory. When the rules are fired, values used during comparison are taken from objects using getters. The workaround would be to create one Java class which contains all XTT2 attributes. The class is called a *Workspace*. To sum up, three files are generated from one XTT2 model file: *Rule Flow* (model structure), *Decision tables* (aggregated rules), and *Workspace* (a Java class with all attributes).

The results of XTT2 into Drools translation are three files. The first one is an XML based file and represents the flow structure. It does not contain the actual rules, but only the nodes (tables' names). The second one, a CSV (Comma separated values) file, contains Decision Tables storing the rules. The last one is a single Java class which holds all the XTT2 attributes.

4.2 Structural difference

While generating Drools files from an XTT2 file, structural differences are revealed. First one was already mentioned above. It is the form of attribute types. There is, however, an easy solution. The type of every XTT2 attribute is exchanged with an appropriate Java type. All XTT2 attributes are wrapped into one *Workspace* class which does not contain any logic but getters and setters.

Another structural difference is the placement of the logical operator. An XTT2 table is translated to a Drools decision table. An XTT2 table contains logical operators in the table cell – together with the value used in comparison. This implies that in one column, many different operators can appear. In Drools, however, the logical operator is placed in a table header. This means that all cells underneath use the same operator. This problem can be solved by decomposing the XTT2 columns into one or more columns in Drools model. Table 2 is the representation of XTT rules, while Table 3 is its Drools equivalent.

| today | hour | operation |
|-----------|-----------|-----------|
| = workday | > 17 | = nbizhrs |
| = weekend | = ANY | = nbizhrs |
| = workday | < 9 | = nbizhrs |
| = workday | in [9,17] | = bizhrs |

Table 2. XTT table from the thermostat example

| condition | condition | condition | condition | condition | action |
|-------------------|-----------|-----------|-----------|-----------|---------------------------|
| Workspace | Workspace | Workspace | Workspace | Workspace | Workspace |
| Today = "\$param" | hour > | hour < | hour >= | hour <= | setOperation (" \$param") |
| workday | 17 | | | | nbizhrs |
| weekend | | | | | nbizhrs |
| workday | | 9 | | | nbizhrs |
| workday | | | 9 | 17 | bizhrs |

Table 3. Decision Table for the thermostat example

There are some structural differences in the flow structure as well. First of all, XTT2 tables allow multiple incoming connections. Furthermore, the connection can be directed to a specific row in a table. It is not possible in Drools Flow. *Ruleset* blocks can have only one incoming and one outgoing connection. This issue can be resolved by placing *split* and *join* blocks before and after the *ruleset* block. Nevertheless, the problem with row-to-row connection is still present and it is to be resolved in a future version of the integration proposal.

Another difference with the representation form of Drools Flow appeared when version 5 of Drools was released. In version 4.0.7, the Drools Flow structure could only be created in a dedicated Eclipse plugin. This is because the file which contained the structure was a Java class serialized using the XStream library (<http://xstream.codehaus.org>). The programmer was dependent on the class, which was included into the plugin. In version 5 however, the file storing the Flow structure was slimmed and now contains only the most important information: blocks defined in the flow and connections between them.

5 Proposal of XTT2 Inference Design with BPMN

The integration of XTT2 with BPMN faces two main challenges.

Different goals: BPMN provides a notation for modeling business processes. Such processes define the order of tasks to accomplish the intended objectives of an organization. Although in BPMN one can define very detailed description of the particular task, it is rather not the proper use of the notation. The XTT2 methodology, in turn, is not only a notation. It provides well-founded systematic and complete design process [5]. This preserves the quality aspects of the rule model and allows gradual system design and automated implementation of RBS.

Different semantics Apart from goals, the semantics of both notations is also different. BPMN describes processes while XTT2 provides the description of rules. Although the semantics of each BPMN element is defined, the implementation of some particular task is not defined in pure BPMN. XTT2 provides a formal language definition and therefore enables automatic verification and execution. Therefore, BPMN and XTT2 operate on different abstraction levels.

Several **integration scenarios** for XTT2 and BPMN are considered:

- **BPMN integration with XTT2**

This scenario assumes that BPMN and XTT2 have some intersecting parts, in which the integration of the two solutions can be performed. The general idea is as follows: BPMN is responsible for inference specification and hierarchization of the rulebase, and rule tables for some part of the system are designed in XTT2. Another example is a BPMN model of a cashpoint, shown in Fig. 7 and 8.

- **BPMN as a replacement of ARD+**

Because the abstraction level of ARD+ and BPMN seems to be similar, in this scenario BPMN is proposed to be used instead of present solution – ARD+. This assumes that mapping between BPMN tasks and XTT2 tables is one-to-one. A prototype example of this approach is shown in Fig. 5.

- **BPMN representation of XTT2 table**

This is not a primary goal of integration. However, this could enable BPMN design of the whole XTT2 methodology, including single tables and rules. An example of this approach can be seen in Fig. 6.

Because the assumed mapping in the first scenario may be not one-to-one, this scenario is highly complex. It requires well-prepared analysis and specification of both solutions as well as a detailed specification of the integration proposal. However, this is the best scenario for real-world cases. In the second one, in turn, the mapping is very simple, because each task is mapped to exactly one table. However, this solution does not provide the table schema, as it was in the case of ARD+. The third scenario is a rather academic one, because tables are already an efficient method of presenting rules, and their visual representation in another form may not be so useful [12].

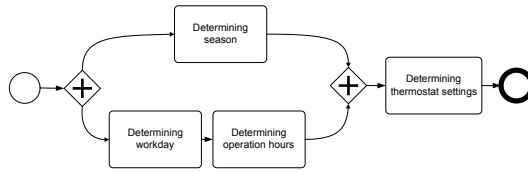


Figure 5. An example of using BPMN instead of ARD+

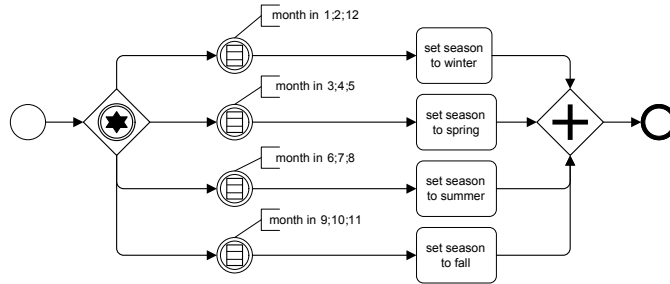


Figure 6. BPMN representation of XTT2 table

6 Conclusions and Future Work

The general problem considered in this paper is the RBS design and modularization. The paper considers possible solutions to modularize rule bases with Drools, BPMN and XTT2. However, these solutions have some limitations. The paper constitutes an overview and a proposal for integration of the three presented methodologies, which can be useful in solving the identified problems.

The work described here is partially in progress. The rule translation from XTT2 to Drools is being developed and implemented. The design is the result of the comparison of both semantics (XTT2 and Drools) while the translation is achieved by the module to HQEd, written in C++. Drools 5 has some differences from its predecessor. The most important thing is that Drools Flow focuses more on a process management, rather than on the rule hierarchisation. In the previous version the main part was the block which refers to the rules in the knowledge base. In the new version there are much more blocks which provide strict integration with Java programming language.

Moreover, several issues concerning BPMN as an end-user notation are considered. Future work will be focused on integration of the three described solutions. The plan involves analysis of the BPMN notation for the purpose of Rule-Based Systems, which can be useful for implementation and application of the integrated methodology. In a more distant future, the plan involves running selected BPMN models in the rule engine, and comparison of the analysis of BPMN models via rule engine to executable BPEL4WS.

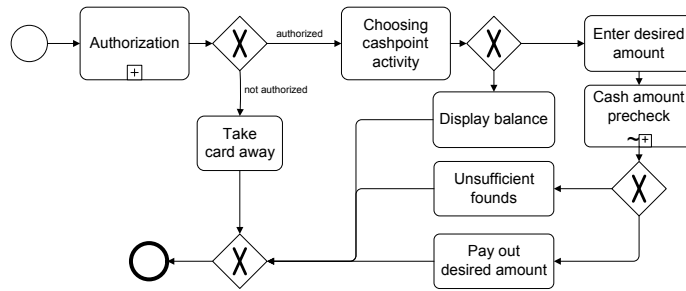


Figure 7. Example of BPMN representation of cashpoint

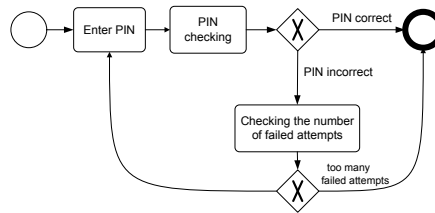


Figure 8. Example of BPMN representation of cashpoint *Authorization* subactivity

References

1. Ligeza, A.: Logical Foundations for Rule-Based Systems. Springer-Verlag, Berlin, Heidelberg (2006)
2. Browne, P.: JBoss Drools Business Rules. Packt Publishing (2009)
3. Owen, M., Raj, J.: BPMN and business process management. Introduction to the new business process modeling standard. Technical report, OMG (2006)
4. OMG: Business process modeling notation (bpmn) specification. Technical Report dtc/06-02-01, Object Management Group (February 2006)
5. Nalepa, G.J., Ligeza, A.: HeKatE methodology, hybrid engineering of intelligent systems. International Journal of Applied Mathematics and Computer Science **20**(1) (March 2010) 35–53
6. Bobek, S., Kaczor, K., Nalepa, G.J.: Overview of rule inference algorithms for structured rule bases. (2010) to be published.
7. Giarratano, J., Riley, G.: Expert Systems. Principles and Programming. 4th edn. Thomson Course Technology, Boston, MA, United States (2005) ISBN 0-534-38447-1.
8. Friedman-Hill, E.: Jess in Action, Rule Based Systems in Java. Manning (2003)
9. Doorenbos, R.B.: Production Matching for Large Learning Systems. Carnegie Mellon University, Pittsburgh, PA, United States of America (2005)
10. Sarang, P., Juric, M., Mathew, B.: Business Process Execution Language for Web Services BPEL and BPEL4WS. Packt Publishing (2006)
11. Hay, D., Kolber, A., Healy, K.A.: Defining business rules - what they really are. final report. Technical report, Business Rules Group (July 2000)
12. Kluza, K., Nalepa, G.J.: Analysis of UML representation for XTT and ARD rule design methods. Technical Report CSLTR 5/2009, AGH University of Science and Technology (2009)