Decision Management in the Insurance Industry: Standards and Tools

Kimon Batoulis[1], Alexey Nesterenko[2], Günther Repitsch[2], and Mathias Weske[1]

[1] Hasso Plattner Institute, University of Potsdam, Potsdam, Germany
{Kimon.Batoulis,Mathias.Weske}@hpi.de
[2] Allianz Deutschland AG, Munich, Germany
{Alexey.Nesterenko,Guenther.Repitsch}@allianz.de

Abstract. Many organizations use business process and business decision management to handle their processes and decisions. Therefore, industry and academics have an increasing interest in exploring and developing solutions for BPM. This leads to different approaches such as open standards and proprietary process engines which in turn lead to discrepancies in terms of supported constructs, best practices, and much more. This paper assesses the model quality of a BPM project implemented by Allianz Deutschland by investigating how the open OMG Decision Model and Notation (DMN) standard and a subset of the decision management capabilities of the propietary BPM software Pega can be related. Our comparison will cover established DMN concepts as well as latest research results on the standard.

Keywords: Decision Management, DMN, Pega

1 Introduction

BPM software vendors offer solutions for companies to manage their business processes. Therefore, the requirements for this software are elicited in industrial settings. Consequently, latest research results may be overlooked in practice. We investigate the gap between academic research on decision management and decision management in the BPM software Pega by assessing the model quality of an implemented decision-heavy Pega project.

The analyzed Pega project is part of a larger project used to determine the reimbursement of a health care service filed by an insured customer. The goal of this project is to increase the rate of automation of processing customer requests. This should lead to faster processing times which entail higher customer satisfaction. Also, a more comprehensive view on a customer's history is offered. In general, the project contains a deterministic and stochastic component, both of which execute the processes fully automatically or in some exceptional cases with the help of a case worker.

The deterministic component is implemented based on a subset of the decision management functionalities of the Pega 7 software platform by Pegasystems, a platform for the model-driven development of BPM applications[3]. The functionalities implemented in Pega include checking the filed requests for correctness

[3] https://www.pega.com/

M. Brambilla, T. Hildebrandt (Eds.): BPM 2017 Industrial Track Proceedings, CEUR-WS.org, 2017.

according to formal and legal requirements and calculating the reimbursement amount according to the tariff of the customer. The use case was chosen because of the frequency of decisions in the determination process. The project contains two major ways of representing decisions for which we propose a mapping to DMN concepts to enable a comparison and to assess model quality. Moreover, it contains 65 distinct decision tables, which we analyzed for properties established in the academic decision management community. These properties include table completeness, separation of concerns from the processes and other metrics.

The remainder of this paper is organized as follows. Section 2 introduces the open DMN standard and the proprietary Pega software. In Section 3 we will compare the concepts of decision tables in Pega and DMN. Afterwards, in Section 4 we will present our analysis results regarding the quality of the Pega project's decision tables. Subsequently, in Section 5 we turn to the other method of representing decision logic in the project—when-rules—and show how they are related to DMN. Finally, in Section 6 we will conclude with the most important insights and lessons learned from this project.

2 Pega and DMN

This section briefly introduces the open DMN standard for designing and implementing decisions and the proprietary BPM software offered by Pegasystems.

2.1 Decision Model and Notation

The Decision Model and Notation [5] is an open standard published by the Object Management Group (OMG). The first version was released in September 2015, shortly after that followed by version 1.1 in June 2016.[4] It describes a modeling language for the declarative modeling and execution of decisions. DMN can be used in combination with other OMG standards such as the Business Process Model and Notation (BPMN), used to design business processes [4]. In this way, a separation of concerns of declarative decision models and imperative process models is achieved [1].

DMN allows to design decisions on two levels, called decision requirements and decision logic level. Decision requirements diagrams are displayed as a directed graph with different kinds of nodes, and edges representing dependencies (or requirements) between these nodes. An abstract requirements diagram consisting of the two basic node types is shown in Fig. 1. The rectangular nodes labeled *D1* and *D2* represent decisions that need to be taken, whereas the elliptic nodes show what input data is necessary to make those decisions. Note that a decision's input can also be another decision, e.g., *D2* is a sub-decision of *D1*. Additionally, the decision elements are associated with decision logic that prescribes how the decision must be taken, i.e., which combinations of input values lead to a specific output value. The most common and standardized way of representing decision logic is through decision tables, discussed in more detail in Section 3.

[4] `http://www.omg.org/spec/DMN/`

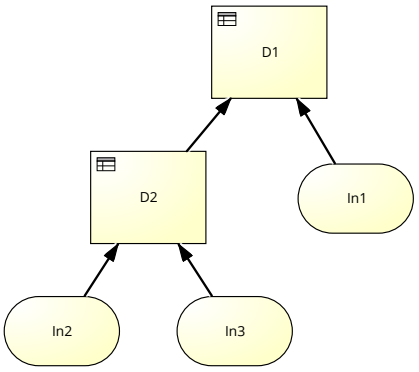

Fig. 1. An abstract DMN decision requirements diagram

2.2 Pega

With the Pega platform, the American software company Pegasystems offers proprietary software for business process management, case management and decision management. It is considered a leading vendor of BPM software since eleven years.[5] Business processes in Pega are designed in a top-down manner. First, one defines a case, which is a piece of work that delivers a business outcome. This case is then structured into stages, representing high-level milestones of that case. Lastly, stages can be implemented with process flows, representing operational steps to accomplish the milestone of the respective stage. To implement the decisions required to make during process execution, Pega offers a variety of decision making functionalities. The most common are when-rules and decision tables, but there are also other concepts such as decision trees, predictive analytics, and prioritization. The concepts used in the project of our investigation are decision tables and when-rules, which are described in more detail in Sections 3 and 5, respectively.

3 Decision Tables in Pega and DMN

This section investigates the relation between DMN's and Pega's decision management concepts regarding decision tables. This enables us to compare the two approaches. We focus on decision tables since they are standardized in the specification [5] and because this is the main way of representing decision logic in the examined Pega project.

Both Pega and DMN allow decision logic to be represented as tables, consisting of rows and columns. Each column is associated with an input or output variable, and each row denotes a rule of how to combine certain values for the input variables to arrive at a value of the output variable(s). Given this general description of decision tables, a mapping from Pega to DMN might appear straightforward. However, there are some fundamental differences in design philosophy that need to be considered, as we will we discuss later in this section.

[5] `https://www.pega.com/de/bpm`

	Conditions		Actions	
	RECHPRUEERG	DE1_STRGNRABWEICH		Return
if	5400	!=0	→	true
else if	5400	=0	→	false
else if	5500	!=0	→	true
else if	5500	=0	→	false
otherwise			→	Error

Fig. 2. Example for a Pega decision table taken from the analyzed project

An example of a Pega decision table taken from the investigated project is shown in Fig. 2. As mentioned in the previous paragraph, Pega and DMN agree on the basic way of designing decision tables. Both approaches also allow to specify how to deal with overlapping rows, i.e., rows that match for the same inputs. In Pega, the rules can be evaluated from top to bottom until the first matches whose output is then returned, or all rules are evaluated and the outputs of all matching rules are returned. These options can be directly mapped to DMN's *first* and *rule order* hit policies, that return either the output of the first matching rule, or the outputs of all matching rules in order [5].

An important difference between Pega and DMN is that Pega does not allow to specify the set of possible values the input and output variables of the table can take on. Rather, it assumes that only those values that are actually used in the table are the values that the variable can assume. For example, for the input *RECHPRUEERG* it is not clear whether or not 5400 and 5500 are the only admissible values. DMN, however, allows to specify a set of possible values for the variables as part of the column definition for that variable. This distinction is important when checking the completeness of the table, as discussed in Section 4.

The other major difference between Pega and DMN is that the former allows to define decision tables that are not free of side effects. This means that it is possible to design tables that directly act on the state of the system or the process instead of just representing a function whose sole purpose is to return a set of outputs given a set of input—a constraint that DMN requires. Fig. 3 shows such a non-functional Pega table. As you can see the result column of this table does not just set a value but calls the external function *setBusinessDataValue()* to influence the state of the system outside of the table itself.

4 Analysis of Decision Tables in the Pega Use Case

The Pega project of our investigation contains 65 distinct decision tables that are evaluated in 143 different situations. We analyzed these tables to assess the project's model quality regarding decision management and to compare it to the principles of decision management according to DMN. Therefore, we examined established correctness metrics such as table completeness and output coverage. Furthermore, we explored how often concepts are used that are not allowed in DMN such as non-functional decisions that do not only depend on the inputs but also on the system's state.

	Conditions		Actions
	RRZITariff		Result1
if	R65S	→	@(KIFWCommon:KIFWUtilities).setBusinessDataValue("RRE", true, Top.GeVo.gevo.steuerungsergebnis.de1_rrzi_kbez_list)
else if	R65U	→	@(KIFWCommon:KIFWUtilities).setBusinessDataValue("RRE", true, Top.GeVo.gevo.steuerungsergebnis.de1_rrzi_kbez_list)
else if	RKEXPS	→	@(KIFWCommon:KIFWUtilities).setBusinessDataValue("RRE", true, Top.GeVo.gevo.steuerungsergebnis.de1_rrzi_kbez_list)
else if	RKEXPU	→	@(KIFWCommon:KIFWUtilities).setBusinessDataValue("RRE", true, Top.GeVo.gevo.steuerungsergebnis.de1_rrzi_kbez_list)
else if	NLTN	→	@(KIFWCommon:KIFWUtilities).setBusinessDataValue("NLT", true, Top.GeVo.gevo.steuerungsergebnis.de1_rrzi_kbez_list)
else if	NLTB20	→	@(KIFWCommon:KIFWUtilities).setBusinessDataValue("NLT", true, Top.GeVo.gevo.steuerungsergebnis.de1_rrzi_kbez_list)
else if	NLTB30	→	@(KIFWCommon:KIFWUtilities).setBusinessDataValue("NLT", true, Top.GeVo.gevo.steuerungsergebnis.de1_rrzi_kbez_list)
else if	NLTB50	→	@(KIFWCommon:KIFWUtilities).setBusinessDataValue("NLT", true, Top.GeVo.gevo.steuerungsergebnis.de1_rrzi_kbez_list)
else if	NZTN	→	@(KIFWCommon:KIFWUtilities).setBusinessDataValue("NLT", true, Top.GeVo.gevo.steuerungsergebnis.de1_rrzi_kbez_list)
else if	NZTB20	→	@(KIFWCommon:KIFWUtilities).setBusinessDataValue("NLT", true, Top.GeVo.gevo.steuerungsergebnis.de1_rrzi_kbez_list)
else if	NZTB30	→	@(KIFWCommon:KIFWUtilities).setBusinessDataValue("NLT", true, Top.GeVo.gevo.steuerungsergebnis.de1_rrzi_kbez_list)
else if	NZTB50	→	@(KIFWCommon:KIFWUtilities).setBusinessDataValue("NLT", true, Top.GeVo.gevo.steuerungsergebnis.de1_rrzi_kbez_list)
otherwise		→	

Fig. 3. Example for a Pega decision table taken from the analyzed project that is not side effect free

4.1 Table Completeness

A decision table is considered complete if for every possible combination of input values at least one rule is matched [3]. If a table is incomplete, and is called with an input combination that it does not consider, the result is undefined, which may lead to undesired behavior. Hence, Pega offers an automatic check for completeness according to this definition. When doing so, as mentioned above, it assumes that only those values that are actually used in the table are the values that the variable can assume. Hence, regarding the table in Fig. 2, 5400 and 5500 would be considered the only possible values for *RECHPRUEERG*. Furthermore, the last row beginning with *otherwise*, is not regarded for the completeness check because it matches for any input, and the completeness check would always return true if it considered these otherwise-rows. Hence, the only incompleteness violations that Pega detects are those caused by missing combinations of input values. For example, if there was no row for $RECHPRUEERG = 5400$ and $DE1\_STRGNRABWEICH = 0$ the table would be declared incomplete. Yet, since the table in Fig. 2 contains a rule for every combination, it is indeed complete according to Pega.

According to DMN, however, this table is not complete, since given that *RECHPRUEERG* is of type integer, there can be many more values for this variable than in the table. We applied both completeness notions to the project's 65 decision tables and found out that according to Pega 48 (74%) of the tables are complete, but according to DMN only 5 (8%), as visualized in Fig. 4.

4.2 Usage of the Otherwise-Rule

As mentioned above, Pega allows the designer of decision tables to specify an *otherwise*-rule that is the last rule of the table, and that only matches when none of the previous rules have matched. We analyzed all tables of the project that contain such an otherwise-rule to find out in which situations they are used. For example, one might assume that they are only used in error situations, such as in in the table in Fig. 2, where the last rule returns an *Error* value. However, we also

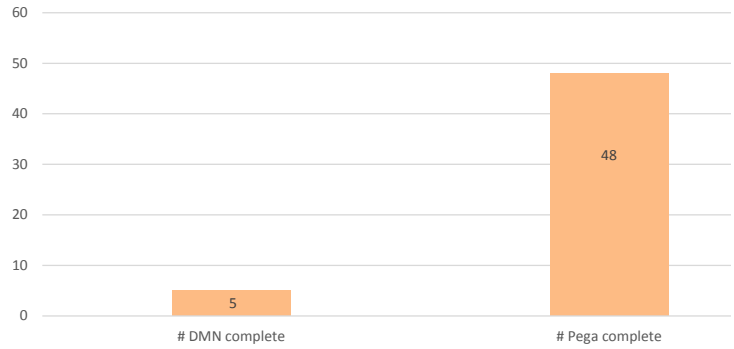

Fig. 4. Number of decision tables of the project that are complete according to DMN and Pega

found tables such as the one in Fig. 5 where the otherwise-row is used to return a "normal" decision result. Consequently, there seem to exist inconsistencies regarding the usage of the otherwise-row and it is unclear in general whether or not its behavior is intended and expected to happen or whether such a situation can actually never occur.

Having analyzed the decision tables of the project regarding this matter, we found out that in 36 (55%) of the cases, the otherwise-row returns an unexceptional result, in 21 (32%) it returns an error value and in 8 (13%) cases the result is undefined. This is summarized in the pie chart in Fig. 6.

4.3 Overlapping and Unreachable Rules

As explained in Section 3, given a set of inputs, Pega will evaluate the rules of the table from top to bottom. If the rules of the table overlap, depending on the configuration, only the first matching rule will be considered, or all matching rules. This means that there may be cases in which a rule is overlapping with another (previous) rule and will thus never be executed because the previous rule is always matched first.

In total we found 52 overlapping rules, 14 (27%) of which can never be executed because of the reasons explained. While unreachable rules are not

	Conditions		Actions
	Current Receipt auftrGrund	Historical Receipt auftrGrund	Return
if	DREC	DREC	→ Yes
	DRECD	DRECD	
	UNT	UNT	
	UNTD	UNTD	
else if	KT	KT	→ Yes
else if	KHT	KHT	→ Yes
else if	EKHT	EKHT	→ Yes
otherwise			→ No

Fig. 5. Pega decision table of the project in which the *otherwise*-row is used to return an unexceptional decision output

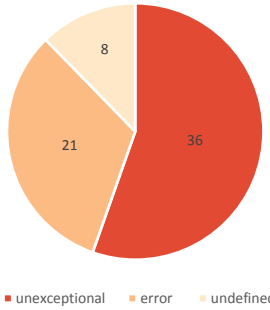

Fig. 6. Proportions of tables in the project using the *otherwise*-row for normal, error and undefined workflow

necessarily harmful, it may be the case that the modeler did not recognize that the rule will never be executed and thus the table will never return the respective value although it is supposed to in certain situations. Pega does not offer an automatic check regarding overlapping and unreachable rules. The results are summarized in the chart in Fig. 7.

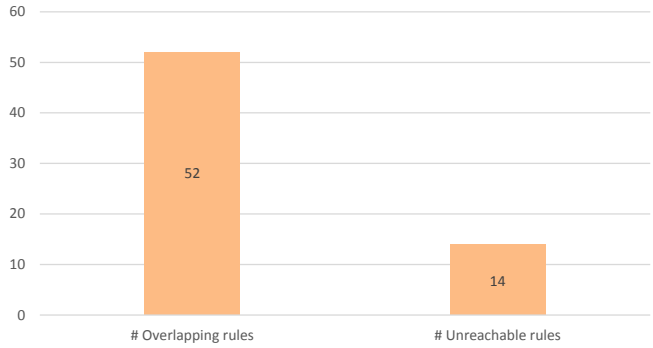

Fig. 7. Number of rules of the project that are overlapping and unreachable

4.4 Output Coverage

When integrating decision tables with process models that act on the outcome of the table, it must be ensured that the process can actually handle every possible output. Otherwise, a deadlock will occur. To prevent such a situation, the output coverage criterion can be applied [2]. An example of a violation of this criterion is illustrated in Fig. 8 and Fig. 9. The table in Fig. 8 can produce three outputs: *2*, *3* and *Error*. However, the split gateway of the process in Fig. 9 from which the decision is called only has two outgoing edges for the first two options—the *Error* case is not considered.

The analyzed Pega project contains 86 processes like the one in Fig. 9 which calls a decision table from a split gateway and then processes the output with

	Conditions		Actions
	BlockNumber		Return
if	2	→	2
else if	3	→	3
otherwise		→	Error

Fig. 8. Pega decision table of the use case that can produce more outputs than the associated process in Fig. 9 can handle

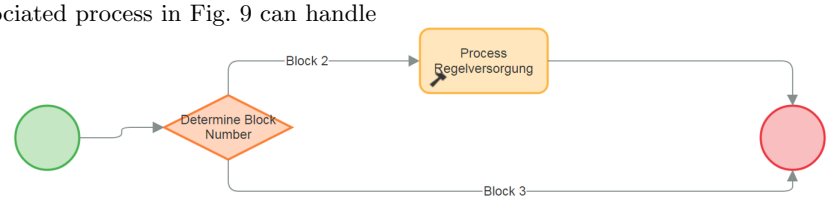

Fig. 9. Pega process model of the use case that does not contain a branch for the output *Error* of the table in Fig. 8

two or more outgoing edges following the gateway. Out of these 86 processes, 26 (30%) violate the output coverage criterion and may therefore lead to deadlocks.

4.5 Non-Functional Decision Tables

As explained in Section 3, Pega allows to design tables that are non-functional or side effect free. This means that Pega tables are not guaranteed to behave like a function, which given a specific input will always return the same output and will not change the state of the system in any way. In the analyzed Pega project we found that 13 (20%) of the 65 tables are non-functional. One of these tables we already showed above in Fig. 3. Decision tables like these violate the principle of the separation of concerns of process and decision logic, leading to lower readability, maintainability and changeability [1]. This principle requires that there is a clear separation between the tasks of decision making and process execution. Therefore, the results of decisions should only be processed by the process, not by the decision.

When taking a closer look at the table in Fig. 3, one can see that the output differs only by the first parameter with which the function *setBusinessDataValue()* is called. Tables like these can be easily refactored into functional tables that have the same decision logic but are side effect free. Table 1 shows the refactored table as a DMN decision table. The table now only outputs the parameter for the function call. The call itself should then be done by the process that receives the decision result.

5 Mapping Pega When-Rules to DMN Decision Requirements Diagrams

Besides decision tables, very many decisions of the analyzed Pega project are implemented using *when-rules*. When-rules in Pega are used to represent simple

Table 1. Side effect free variant of the table in Fig. 3 expressed in DMN

U	Input	Output
	RRZITariff	Result1
	string	{RRE, NLT}
1	R655	RRE
2	R65U	RRE
3	RKEXPS	RRE
4	RKXEPU	RRE
5	NLTB20	NLT
6	NLTB30	NLT
7	NLTB50	NLT
8	NZTN	NLT
9	NZTN	NLT
10	NZTB20	NLT
11	NZTB30	NLT
12	NZTB50	NLT

Boolean decision logic. A when-rule consists of one or more variables connected by logical *or* and *and* operators. Each variable represents a Boolean expression, such as a string or integer comparison, a boolean value, or another when-rule. Fig. 10 shows an example of a Pega when-rule from the use case, which consists of three variables, A, B and C. The truth values for A and B are each determined by other when-rules, while C is given by a comparison testing for an empty string. At the bottom of the figure, you can see the Boolean expression that must be fulfilled in order for the when-rule to be true: *(A or B) AND C*.

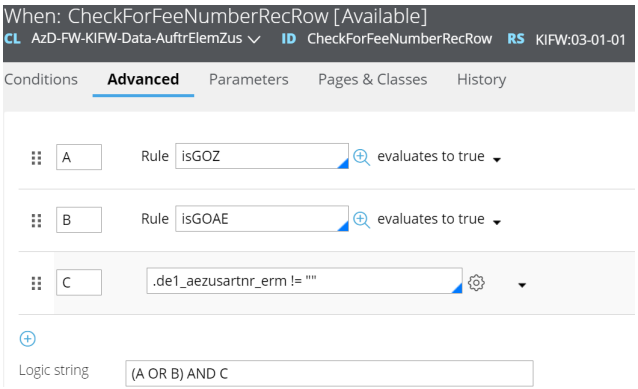

Fig. 10. Example of a Pega when-rule from our use case

Table 2 shows the resulting DMN decision table. The derived DMN decision table is shown in Fig. 10. The mapping underlying this derivation is straightforward, as discussed in the following. The number of input columns of the table is determined by the number of variables of the when-rule, while the number of rows is given by the number of disjunctions in the when-rule. Therefore, each row of the decision table is an assignment of truth values to the variables such

Table 2. DMN Decision table derived from the when-rule in Fig. 10

F	Input			Output
	A	B	C	return
	boolean	boolean	string	boolean
1	true	–	= “”	true
2	–	true	= “”	true
3	–	–	–	false

that the when-rule becomes true. The table has a *first-hit* policy, meaning that the rows are evaluated from top to bottom until one becomes true. Therefore, if none of the conditions of the when-rule are fulfilled, the last row of the table will return false, irrespective of the inputs.

Since the variables A and B each refer to another when-rule, their values are also determined by decision tables whose outputs will be used as inputs for the table in Fig. 2. This leads to the DMN decision requirements diagram (DRD) shown in Fig. 11. In general, simple DRDs consist of two elements, namely rectangular decision shapes and elliptic input data shapes. The decision shapes are associated with decision logic such as a decision table and the input data shapes correspond to the inputs that are required by the decision logic.

Hence, in the diagram in Fig. 2, the top-level decision is associated with the decision table in Table 2, which was derived from the when-rule in Fig. 10. The top-level decision has three inputs, corresponding to the variables A, B and C of the when-rule. Since A and B each refer to another when-rule, they are also represented as decision elements. These decisions are associated to the decision tables derived for these when-rules, the details of which are not shown here. All other elements of the DRD are “primitive” inputs and therefore represented as ellipses.

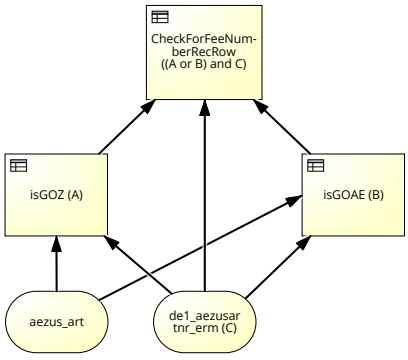

Fig. 11. Decision requirements diagram derived from then when-rules contained in Fig. 10

The visual presentation of decisions and their dependencies in DRDs allows to introduce an abstraction from the implementation. This means that dependent decisions implemented separately can be brought back together in the DRD. In the same regard, DRDs can support the implementation of decisions by disclosing

their dependencies. With such information made obvious, duplicating decision logic can be avoided.

6 Conclusion

In this paper, we presented our results from a model analysis project conducted on an IT project of Allianz Deutschland developed to improve the automatic reimbursement of health care service bills. We showed that there is a different understanding of model quality and correctness in the academic and the industrial BPM community. Still, both worlds agree on the fact that BPM is a useful concept to achieve business goals in an organized manner.

Through working with Allianz' Pega project we got valuable insights on working with industrial BPM projects. The most important lesson we learned is the necessity of domain knowledge. Without a certain amount of domain knowledge it is not that easy to understand the processes and decisions implemented in the system. This is especially true when labels of activities or decision inputs/outputs are based on domain-specific vocabulary and abbreviations. Moreover, only a few simple concepts for designing and implementing decisions offered by Pega are used in the project. These are decision tables and when-rules. When-rules are even simpler ways to express decision logic than decision tables. They consist of one or more variables connected by logical *or* and *and* operators. However, Pega offers other, more advanced means of specifying decision logic such as decision trees and predictive analytics. Yet, these concepts are not applied in the project at hand, hinting that they are not really necessary to implement decision management applications.

References

1. Batoulis, K., Meyer, A., Bazhenova, E., Decker, G., Weske, M.: Extracting decision logic from process models. In: CAiSE. pp. 349–366 (2015)
2. Batoulis, K., Weske, M.: Soundness of decision-aware business processes. In: BPM Forum (2017)
3. Calvanese, D., Dumas, M., Ülari Laurson, Maggi, F.M., Montali, M., Teinemaa, I.: Semantics and analysis of dmn decision tables. In: BPM (2016)
4. OMG: Business Process Model and Notation, Version 2.0.2 (January 2014)
5. OMG: Decision Model and Notation, Version 1.1 (May 2016)