FIRST THREE CHAPTERS

Get the entire book from Amazon for just $9.95

DMN in Action with OpenRules

A Practical Guide for Development of Business Rules and Decision Management Applications using Decision Model and Notation (DMN) Standard and OpenRules

By Jacob Feldman, PhD

ISBN 978-1-5206053-8-8

Table of Contents

Preface 3

Dialog-Session 1: Introducing Major Concepts by Building an Executable Decision Model 10

Dialog-Session 2: Enhancing Decision Model "Determine Customer Greeting" 33

Dialog-Session 3: Integrating Tested Decision Models with IT . 43

References 58

Remaining chapters:

Dialog-Session 4: Implementing Decision Model "Patient Therapy" with Complex Formulas 58

Dialog-Session 5: Single-Hit and Multi-Hit Decision Tables 79

Dialog-Session 6: Scorecards, Accumulating Values, and Dealing with Collections of Objects 97

Dialog-Session 7: Complete Decision Model "Determine Auto Insurance Premium" 115

Preface

About Business Decision Modeling and DMN. Nowadays Business Decision Modeling is rapidly becoming major technological and methodological framework to support decision making approaches across a wide range of business problems from loan origination and insurance underwriting to clinical guidelines and recommendations. The recently introduced OMG standard "Decision Model and Notation (DMN)" [1] brings standardization to decision modeling, an emerging best practice. Decision Modeling clearly defines a business target, shows how to orchestrate decision management techniques, supports interoperability, and integrates decision models into modern enterprise architectures. DMN quickly became a mainstream approach supported by many Business Rules and Decision Management tools [6].

To Whom This Guide Is Oriented. This guide is oriented to people who want to build operational decision models for their own business environments. They've probably already looked at the DMN standard itself and quickly figured out that the text of the standard is more oriented to DMN vendors (implementers) than to practitioners. They've probably read or at least navigated through recently published DMN textbooks like [2]-[4], which are trying to cover all (!) DMN aspects and to provide methodological recommendations for how to apply DMN in the real-world decision modeling.

Thus, we expect that our readers are already convinced of the value of the decision modeling proposition. However, they still

have problems with understanding what actually should be done to build their own operational decision models.

How Readers Create Executable Decision Models. The objective of this guide is to help readers quickly learn how to apply the DMN approach to build working decision models. To achieve this objective, this book will guide readers on how to create working decision models for simple and complex business problems. More importantly, readers should be able to execute (!) their decision models and integrate them into their own IT systems. That's why we called this guide "DMN in Action"!

While DMN is supposed to take care of future interoperability (tool independence), a reader needs to choose concrete DMN tools to actually create and execute decision models. Without using such tools it's simply impossible to achieve our objective. So, as readers go through this guide, they can utilize the following tools:

1) Microsoft Excel: without doubt the most popular tool among business analysts in any industry. Our readers will mainly use Excel as the best table editor to create and enhance business decision models.
2) OpenRules: a highly popular open source business rules and decision management system. Our readers will use OpenRules to test (that assumes an ability to execute!) their decision models.

The website www.DmnInAction.com offers a reader two options: 1) open, analyze, and execute all decision models online without any download; 2) install a free evaluation version of OpenRules and analyze/modify/execute all provided or your own decision models.

Instead of Excel a reader may use Google Docs or Open Office. If a reader prefers to use a DMN implementation [6] different from OpenRules, they still can do it, but the described DMN constructs need to be adjusted to the tool-specific DMN representations.

How this Guide Is Written. This guide was written in style of dialog between the AUTHOR and the READER. The guide consists of multiple dialog-sessions, during which AUTHOR and READER are trying to build concrete decision models. They are freely discussing why and how they are actually building decision models, not avoiding possible missteps, pitfalls, and alternative approaches.

From one decision model to another the AUTHOR will lead the inquisitive and receptive READER to different decision modeling concepts and notions emphasizing and/or demystifying especially complicated aspects, and directing the reader's attention to special situations that occur in real-world decision modeling.

About the Reader. We consider the READER to be a subject matter expert or a business analyst without any programming experience. He or she may be proficient in Excel but more importantly is a person who really wants to understand how to build good decision models that can be adjusted and enhanced over time as business conditions change.

Only once, in the Dialog-Session 3, will the READER be joined by the DEVELOPER, who also will be "inquisitive and receptive" but from a purely IT integration perspective.

The actual reader of this guide will find this text to be perhaps more like a freely told story or, rather a freely conducted

discussion. Being represented by a fictional READER, the actual reader will hopefully find the answers to a lot of his/her own questions, and will eventually gain an intrinsic understanding of the DMN-based decision modeling technique.

About the Author. The fictional AUTHOR in this guide represents me, Jacob Feldman, the Chief Technology Officer of the OpenRules, Inc., a US corporation that created and maintains the open source Business Rules and Decision Management system commonly known simply as "OpenRules" [5]. Over the course of dialog-sessions I will try to share my multi-year experience as a decision modeling practitioner who has helped our customers develop and maintain production-quality decision models in various business domains.

I want to warn the readers that this guide by no means is a DMN textbook as it does not cover all the items introduced in the DMN standard. On the contrary, it concentrates on the most frequently used decision modeling constructs described in DMN and uses their particular implementations in the OpenRules formats.

As the AUTHOR, I took a liberty of choosing the DMN concepts that are most frequently used and helpful and those which should be avoided. I simply don't mention some concepts currently included in the DMN standard which I do not recommend for use. In particular, the AUTHOR will not talk about hit policies with priorities or about pure programming constructs such as loops or if-then-else statements included in the DMN Friendly Enough Expression Language (FEEL). As the AUTHOR, I promised the READER to stay away from programming, and I try to stick to this promise over the course of all (well, almost all) dialog-sessions. At the same time, the

READER and I will actively use the FEEL syntax to represent various business expressions and calculation formulas.

The nice thing about the DMN standard is that it allows vendors to compete on different representations of the same DMN concepts as long as they have the same semantic meaning. The reader may find that OpenRules graphical representations of some DMN concepts are slightly different from examples included in the current DMN specification. For instance, we do not use single-character and double-character abbreviations for hit policies which are supposed to be placed into a separate cell of every decision table. Instead, we use keywords such as "DecisionTableSingleHit" or "DecisionTableMultiHit" in decision table titles.

Finally, I believe it is extremely important to keep this guide as small as possible (no more than 150 pages) and to allow the reader to start developing their own decision models ASAP. They can do this right after going through the first 3-4 dialog-sessions and actually modifying and executing the provided examples of decision models. In general, the guide provides many examples that can be used as working prototypes. Learning by example is the quickest and easiest way to learn fundamental decision modeling concepts.

How to Use this Guide. It is not necessary to read the entire guide. Even if you read only the first two dialog-sessions, you will get a good introduction to practical decision modeling techniques. The third dialog-session "Integration with IT" allows you to invoke your IT colleagues to make sure that your future real decision models will be as easily integrated into your specific IT infrastructure as it is explained in the guide using a simple example.

Each dialog-session starts with a list of the discussed topics, and you may skip sessions that are of no interest to you. So, you can read dialog-sessions in the presented order, or you may read them in accordance with your actual interest and needs.

What is really important is not just to go through the printed examples but also download, analyze, and execute the described decision models using the supporting software. It will allow you to look at all implementation details directly in Excel including the use of dropdown lists, data validation, cell merging, outlines, and other well-known and very helpful Excel constructs. All figures may also be seen directly in Excel by downloading the related projects from www.DmnInAction.com.

Supporting Software. This guide comes with the software that includes all described decision models in Excel format. To do this, visit www.DmnInAction.com website and you will be able to download them from the page "Decision Models". You also may open OpenRules Why-Analyzer to select, analyze, and test all these decision models online without any downloads or installations.

If you want to download these decision models to your own computer and modify and execute them locally, you may receive the supporting software for free by submitting a request to OpenRules. Alternatively, you may get an evaluation copy directly from www.OpenRules.com and use it to run the existing or to create your own decision models.

I recommend you to check www.DmnInAction.com from time to time as it is being constantly updated with new decision models not described in this guide (yet).

Acknowledgments. First of all, I want to thank the subject experts with whom I was fortunate enough to work developing complex business rules, optimization and machine learning applications over the last two decades. Secondly, I want to thank all the people who helped to bring the DMN standard to the everyday reality for many decision management practitioners worldwide. It would not have happened without previous work and the current support by key architects and actual developers of various business rules and decision management products. Special thanks should go to the authors of the "big" books that together provide methodological guidance for business decision modeling techniques in general and the DMN standard in particular.

And finally, I want to share a little story about people who had no direct acquaintance with modern decision modeling techniques but who essentially incentivized me to write this guide. Many years ago, preparing for my university entrance exams, I read a math book simply known at that time among Soviet high school students as "Tarasov and Tarasova" (by the last name of the authors). The book was written in a very unusual style of dialogs between TEACHER and STUDENT. It was a really enlightening guide that not only demystified complex mathematical concepts but gave me an initial feeling of the beauty and intrinsic integrity of mathematics in general. Thank you, Tarasov & Tarasova!

Jacob Feldman, PhD
Monroe, New Jersey
February 2017

Dialog-Session 1: Introducing Major Concepts by Building an Executable Decision Model

Discussed Topics:

- Decision Requirement Diagram (DRD)
- Tabular Decisions and Sub-Decisions
- Rules Repository
- Decision Tables
- Glossary
- Test Cases
- Executing Decision Model
- Implementation Steps

Related OpenRules Projects:

- DecisionHello
- DecisionHelloWorld

AUTHOR. Hi again. You told me that you've already had a chance to look at the DMN standard, and even looked through several of the latest decision modeling books. Now you want to start building actual decision models.

READER. Yes, and I already understand that DMN and supporting tools may help me to build decision models. However, I have difficulty grasping exactly what decision models are and how I can create workable models myself.

AUTHOR. Many business analysts or subject matter experts who start working with DMN initially face the same issue. I believe the best solution is to create a decision model that you can "touch", execute and analyze. Today we'll specify and build a relatively simple but complete decision model that demonstrates major decision modeling constructs. You will be

able not only create a decision model but also to test it using your own test cases and to evaluate the produced results.

READER. I'd like to "touch" a decision model.

AUTHOR. I hope you will at the end of this session. We will start with our own version of "Hello World", that is traditionally the first program people create when they learn a new programming language.

READER. But during our preliminary discussion, you said DMN is oriented to business people like me and you do not plan to teach me any programming language.

AUTHOR. Right, and we will stay away from programming. First, we will define a simple business problem, which I call "Greeting a Customer". Let's assume that we want our future decision model to produce a greeting like "*Good Afternoon, Mrs. Robinson!*" based on the current time of the day and some information about the customer. For example, such a decision model can be used by an IVR system…

READER. What is IVR?

AUTHOR. You usually deal with an **I**nteractive **V**oice **R**esponse (IVR) system when you make a call to any customer support system and a nice voice asks you to push "hundreds" of buttons before you can reach a live agent.

READER. I hate those systems.

AUTHOR. Me too. Anyway, the first thing that any good interactive system should do is to greet you properly as a valuable customer. The Fig. 1-1 shows different variations of possible greetings:

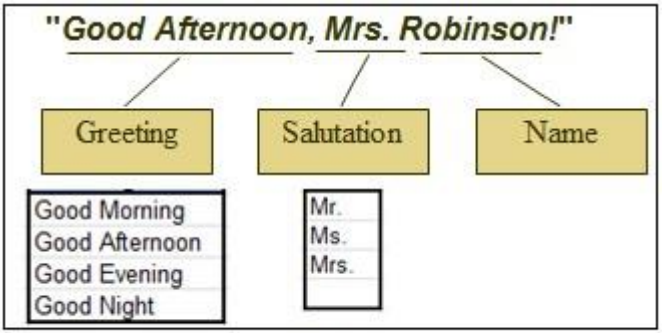

Fig. 1-1

We may assume that "Name" and other related information about the customer is already known. What we want is to automatically define a Greeting (one out of 4 alternatives) and a Salutation (one out of 3 alternatives).

READER. It looks quite simple: we can choose a Greeting based on the time of the day and we can choose a Salutation based on the customer's gender.

AUTHOR. In the real IVR system even this "simple" problem may become much more complex, but let's move forward with what you just said. So, we want to build a decision model capable of producing greetings similar to those in Fig. 1-1 based on some information about the customer. First of all, I suggest giving our decision model a name, e.g. "DetermineCustomerGreeting".

READER. Why did you omit spaces between these 3 words?

AUTHOR. Besides humans who create and maintain this decision model, this name will also be used by an information system that will invoke (execute) our decision model. So, let's consider the absence of spaces as our convention.

READER. OK. One more question: should the names of other decision models start with the word “Determine”?

AUTHOR. Not at all, it is your choice. At the same time, it is a good practice to name your decision model in accordance with what it actually does, e.g. later on we plan to consider decision models “DeterminePatient Therapy”, “CalculateTaxReturn”, “DefineLoanEligibility”, etc. OK, let’s start building our first decision model.

READER. I think we should create decision tables that define our Greeting and Salutation words.

AUTHOR. Quite good, you better call them not “words” but rather decision variables. The decision tables usually specify the decision logic or the “HOW” of your decision model. However, it’s always better to start with an answer to the question “WHAT” that shows the structure of our future decision model. For example, from looking at the Fig. 1-1 it is only natural to conclude that our decision “DetermineCustomerGreeting” consists of two sub-decisions “DefineGreeting” and “DefineSalutation”, and these sub-decisions can be defined using the proper decision tables.

DMN recommends starting with creation of a so called “Decision Requirement Diagram” or DRD. We can do it with any generic diagramming tool such as MS Visio and a specialized tool such a Decisions First Modeler. However, I promised you that our main tool over this course will be Excel. Using basic Excel “Insert+Shapes”, I can rather quickly create the first DRD for our decision model. It is presented on Fig. 1-2:

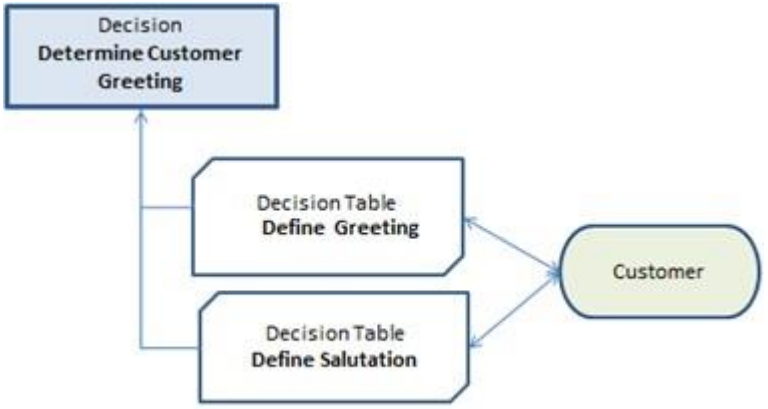

Fig. 1-2

This diagram tells us that to "DetermineCustomerGreeting" we need to "DefineGreeting" and to "DefineSalutation". We also specified a business object "Customer" that provides information for these two decision tables.

In Excel, it is easy to add a hyperlink to each element of the diagram to open to the proper file/table when you click on this element. It makes the DRD a very convenient navigation tool.

READER. Probably DRDs will be useful for more complex decision model, but for such a simple case as ours, this picture probably is not necessary.

AUTHOR. In real world, DRD is useful not only to give a generic view of the decision model, but also to serve as a glue that puts different decision tables together and allows an underlying decision engine to execute our decision models. As Bruce Silver wrote, "*Decision logic is verifiable and executable. What you draw is what you execute*" [3]

To make the DRD on Fig. 1-2 executable, OpenRules allows you to present it in the equivalent tabular format:

Decision DetermineCustomerGreeting	
Decisions	**Execute Decision Tables**
Define Greeting Word	DefineGreeting
Define Salutation Word	DefineSalutation

Fig. 1-3

This is an example of an OpenRules table of the type "Decision" that starts with the proper keyword at the left top corner followed by the name of the table, in this case "DetermineCustomerGreeting". The first column contains descriptions of sub-decisions in plain English (spaces are allowed), and the second column contains the actual names of tables that implement these sub-decisions (no spaces).

READER. Do I always have to specify both a diagram and the equivalent table "Decision"?

AUTHOR. No. To execute your decision model with OpenRules it is enough to define only its tabular format "Decision". However, we are working with several diagraming tool vendors on an integration that will allow our users to automatically generate tables of the type "Decision" directly from the DRDs.

When we define the structure of our decision model, we can describe the actual business logic by creating two decision tables: "DefineGreeting" and "DefineSalutation". Let me create the first table and you will create the second one.

READER. OK. I've seen that you placed the DRD and our table "DetermineCustomerGreeting" in the Excel file "Decision.xls". Will we create these two tables in the same file?

AUTHOR. We can use the same file and different Excel worksheets. However, within real-world decision models people

place different decision modeling constructs in different files and even in different folders to simplify future maintenance of their decision models. OpenRules standard examples usually place the standard decision model in the so called “Rules Repository” that in many cases looks like the one in Fig. 1-4.

I’ve already created the folder “DecisionHello” as a placeholder for our decision model based on the standard project template provided by OpenRules. It contains folders and files similar to those described in Fig. 1-4.

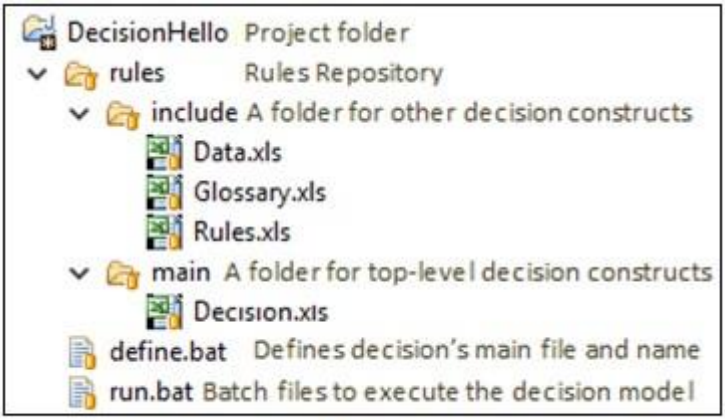

Fig. 1-4

READER. OK, I guess we will talk about files “Glossary.xls” and “Data.xls” later on. For now we will create our decision tables inside the file “Rules.xls”.

AUTHOR. Right. First, I will show you how to create a very simple decision table “Define Greeting” in Excel. After a few regular Excel manipulations, I created the following table (again utilizing some OpenRules examples):

	A	B	C	D
1				
2		DecisionTable DefineGreeting		
3		If	Then	
4		Current Hour	Greeting	
5		[0..11)	Good Morning	
6		[11..17)	Good Afternoon	
7		[17..22)	Good Evening	
8		[22-24]	Good Night	
9				

Fig. 1-5

Hopefully this decision table is self-explanatory. Could you try to explain how it is supposed to work?

READER. Yes, it looks quite intuitive to me. There are 4 rules:

1. If Current Hour is between 0 and 11, the Greeting is "Good Morning"
2. If Current Hour is between 11 and 17, the Greeting is "Good Afternoon".
3. If Current Hour is between 17 and 122, the Greeting is "Good Evening".
4. If Current Hour is between 22 and 24, the Greeting is "Good Night".

AUTHOR. The first column is a condition specified by the OpenRules keyword "If", and the second column is a conclusion specified by the keyword "Then". Could you tell me what Greeting will be produced when Current Hour is exactly 11?

READER. "Good Afternoon" because the first interval [0..11) does not include 11 while the second interval [11..17) does.

AUTHOR. Good that you remember this basic math convention from your college years. By the way, instead of [0..11) you also may write:

```
< 11
Less than 11
More or equal 0 and less than 11
```

READER. It's nice. I noticed that you merged columns in the very first "black" row. Why was that necessary?

AUTHOR. To help OpenRules to recognize the table bounds. If I had not merged the columns B and C in the Excel's row 2, OpenRules would "think" that this table has only one column (B). Have you noticed that I also left empty cells around the table?

READER. Probably you did it for esthetic considerations similar to your use of different colors.

AUTHOR. Yes and no. This particular table will work fine even if you don't surround it with empty cells. However, if we have several tables inside the same worksheet, these tables should not touch each other. Even if you want to put some comments near some rows or columns of the decision table, make sure they do not touch the table. By the way, you may insert Excel's own comments anywhere – OpenRules will ignore them, but it will help other people who will analyze your decision table.

READER. How about colors and border shapes? I've seen how precise you were making sure that a separator between columns "If" and "Then" is double-lined.

AUTHOR. Text and background colors, fonts, borders, and other visual elements are used just for consistency and standardization. For example, many OpenRules customers have for years preferred to use a black background and a white foreground in the very first "signature" row. However, some customers prefer other coloring conventions. Contrary to "merging", it does not carry any semantic meaning for OpenRules. Coloring conditions in blue and conclusions in purple comes from the DMN samples. Do does as a doubled line used to separate conditions and conclusions. Unfortunately, the current version of DMN in certain situations gives semantic meanings to italics and underlining – we, at OpenRules, believe this is a mistake and do not support such interpretations.

READER. Now it is my turn to create a similar decision table "DefineSalutation". I will simply copy/paste your worksheet "DefineGreeting" and will do some refactoring. OK, here is my version:

DecisionTable DefineSalutation	
If	Then
Gender	**Salutation**
Male	Mr.
Female	Mrs.
Female	Ms.

Fig. 1-6

Oops! I need somehow to differentiate between "Mrs." and "Ms." May I add another column with the customer's Marital Status?

AUTHOR. Of course, just insert another column before the column "Then" as you usually do in Excel.

READER. OK, this is simple… But can I simply use the word "Marital Status" as a title of my inserted column?

AUTHOR. Why not? We did not ask anybody's permission when we used the words "Current Hour" or "Gender". It will become another decision variable of our decision model.

READER. Here it goes:

DecisionTable DefineSalutation		
If	If	Then
Gender	**Marital Status**	**Salutation**
Male		Mr.
Female	Married	Mrs.
Female	Single	Ms.

Fig. 1-7

AUTHOR. Very good. This is a well-designed decision table. Can you explain why you left Marital Status empty for the first rule?

READER. Because whether Marital Status is "Married" or "Single" the Male's salutation is always "Mr."

AUTHOR. That's right. Just keep in mind that DMN requires in such cases using a hyphen instead of an empty cell. However, OpenRules correctly interprets both a hyphen and an empty cell.

I would like to make a few more comments about the organization of decision tables in Fig. 1-5 and Fig. 1-7. Instead of retyping the same values like Male or Female you may take advantage of Excel's ranges. For example, you may select 3 cells

with values Male, Female, Female in the first column on Fig. 1-7, and then click on the menu-item Data+Data Validation and fill out the displayed dialog as in Fig. 1-8:

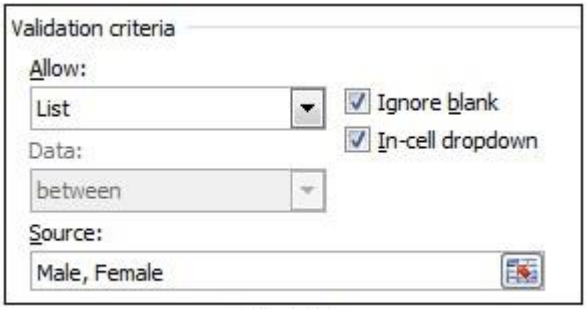

Fig. 1-8

Now you will be able to choose a value from the dropdown list. It makes sense to similarly define dropdown lists for the Marital Status, Salutation and Greeting columns.

READER. Of course I know how to use ranges in Excel, but here it really adds value to the design of our decision tables. It is probably especially useful when we have much larger lists of possible values.

AUTHOR. Yes, and later on I will show you how to organize and reuse standard value lists using OpenRules Data tables. Now I want to show you another design of the decision table “Define Salutation”:

DecisionTable DefineSalutation					
Condition		Condition		Conclusion	
Gender		Marital Status		Salutation	
Is	Male			Is	Mr.
Is	Female	Is	Married	Is	Mrs.
Is	Female	Is	Single	Is	Ms.

Fig. 1-9

As you can see, I've added special sub-columns for operators "Is" to each table column. I also changed keywords "If" and "Then" to "Condition" and "Conclusion" (they are placed in the merged cells now). While this decision table design may look a little bit more complex, it makes the table more readable when it utilizes different operators like "Is" or "=", "Is Not", "Less", "More or Equal", "Include", "Exclude", and many others. This decision table design was recommended in [7].

READER. I really like this design, even more than the one in Fig. 1-7. Can I use columns of the type "Condition" with two sub-columns for conditions, and columns of the type "Then" for conclusions?

AUTHOR. Yes, you can mix them in the way you want. By the way you can use the keyword "Action" as a synonym for "Then". Now it's time to finalize our decision model.

READER. I thought we had already defined the business logic.

AUTHOR. Yes, but to be able to test (execute) our decision model, we also need to define a business glossary and test cases. Let's start with a glossary that will list all our decision variables categorized by the business concepts they belong to.

In our case it is quite simple because we use only the following decision variables:

- **Current Hour**
- **Greeting**
- **Gender**
- **Marital Status**
- **Salutation.**

Obviously, variables Gender and Marital Status belong to a business concept that we may call "Customer". Do variables Current Hour, Greeting and Salutation also belong to a Customer?

READER. We may assume so… On the other hand, Current Hour depends on the location from where the customer is calling.

AUTHOR. We will deal with these considerations later on – now I want us to finalize and to execute our decision model ASAP. Here is the simplest glossary table for our decision model (I placed it into the file rules/include/Glossary.xls):

Glossary glossary		
Variable	**Business Concept**	**Attribute**
Gender	Customer	gender
Marital Status		maritalStatus
Current Hour		currentHour
Greeting		greeting
Salutation		salutation

Fig. 1-10

I copied and pasted all 5 decision variables from our decision tables to the first column. As we agreed, they all belong to the business concept "Customer" defined in the merged second column. OpenRules also requires us to provide technical names

(or attributes) for every decision variable in the third column. These names do not allow spaces and will be used in the future to integrate our decision model with an actual IT system.

READER. Should I always start these technical names with a small letter and then use capital letters for all consecutive words?

AUTHOR. It's better to follow this, so-called "Camel" naming convention as in real-world applications these attributes will correspond the underlying Java objects – but you do not have to worry about this.

Now we should define test cases. As we planned (see Fig. 1-4 above), I will place them in the file" rules/include/Data.xls".

READER. I guess we simply should define several customers with variables Gender, Marital Status, and Current Hour already defined. Then during the testing our decision model will produce the proper values for Greeting and Salutation.

AUTHOR. That's right. OpenRules offers you a quite powerful, yet simple mechanism known as "Test Harness". Everything is defined in special Excel tables. First we define a so-called Datatype table for our business concept Customer:

Datatype Customer	
String	gender
String	maritalStatus
int	currentHour
String	greeting
String	salutation

Fig. 1-11

This table starts with the keyword “Datatype” followed by the name of the datatype after a space – in this case “Datatype Customer”. Then we list different attributes. The first column contains attribute types, e.g. “String” for text variables, “int” for integer variables, “double” for real variables, “Date” for dates. The second column contains the names of attributes exactly as they were defined in our Glossary in Fig. 1-10.

READER. I guest capitalization is important here.

AUTHOR. Yes, very important. If you write “string” instead of “String” or “Int” instead of “int”, OpenRules will point you to a syntax error “Unknown type” (indicating the exact cell where this error occurred). Now let’s create test customers using the following table of the type “Data”:

Data Customer customers				
gender	maritalStatus	currentHour	greeting	salutation
Gender	**Marital Status**	**Current Hour**	**Greeting**	**Salutation**
Female	Married	20	?	?
Male	Single	11	?	?

Fig. 1-12

This table starts with the keyword “Data” following the datatype and the name of the array of customers – in this case “Data Customer customers”. The second row contains the technical names of Customer’s attributes, and the third column contains the business names. The business names are usually the names of variables but you can use any name as the technical names already provide mapping to the decision variables from the glossary. Then we may define as many rows as we want to specify particular customers.

READER. I got it. Do we always have to put '?' for unknown variables?
AUTHOR. In this case you may leave these cells empty, but for integer or double variables you need to put in some initial values (e.g. -1) to indicate that these variables are not defined yet.

OpenRules also allows you to define the expected results for each test case. Then it automatically checks if the actual results correspond to the expected results and shows mismatches. Here is the proper table of the type "DecisionTableTest":

DecisionTableTest testCases			
#	ActionUseObject	ActionExpect	ActionExpect
Test ID	**Customer**	**Greeting**	**Salutation**
Test 1	:= customers[0]	Good Evening	Mrs.
Test 2	:= customers[1]	Good Afternoon	Mr.

Fig. 1-13

The name "testCases" is the name used by the default test launcher. Each test has its own ID defined in the first column of the type "#". This table can use different business objects (concepts) defined in the column of the type "ActionUseObject" – in this case we have only one object "Customer". We also can add different expected test results using the columns of the type "ActionExpect" and the corresponding variable names (like Greeting and Salutation).

READER. I can see that the first test customer is expected to generate Greeting "Good Afternoon" and Salutation "Mrs." But I am a little bit confused with this strange formula:

:= customers[0]

AUTHOR. I agree that it may look confusing for a beginner. The real reason is that this is an example of a Java snippet that OpenRules allows you to put in any cell of your OpenRules table. The sign ":=" indicates that this is a Java snippet (expression) and the index [0] indicates that you want to use the very first element of the array "customers" defined in Fig. 1-12. In Java, the array indexation always starts with 0.

READER. But you promised to keep me away from programming and now we talk about Java…

AUTHOR. Sorry, but this is how the optional (!) Test Harness is currently implemented. You don't really have to know that this is an actual Java snippet -- simply follow this convention when you define test objects. However, in real-world decision models sometimes you would like to invoke predefined Java methods, and it is not so bad to know that OpenRules provides this capability by using ":=".

READER. I'd rather not have the expected values at all and not have to deal with these Java details.

AUTHOR. It may be true for this, very simple decision model. There is a way to execute all test cases directly from the array "customers" and then visually compare the results with what you expected. However, the Test Harness is very important for real-world decision modeling. We do have customers who maintain decision models with thousands of decision tables. Along with a large rule repository, they also maintain a test repository with thousands of test cases, all of which contain the expected results. When they make changes to the rule repository, they always run Test Harness to make sure that new

changes did not spoil any already tested results. It is simply impossible to do it manually. This approach is called "***Test-Driven Decision Modeling***"!

READER. OK, I got it. The use of ":=" is not a big price to pay for this functionality. Are we now ready to run these test cases?

AUTHOR. Only one more thing is required. We need to explicitly tell OpenRules which business objects correspond to business concepts defined in the Glossary. It can be done using the following table of type "DecisionObject" (I recommend keeping it in the file Decision.xls):

DecisionObject decisionObjects	
Business Concept	**Business Object**
Customer	:= decision.get("Customer")

Fig. 1-14

Yes, here we are again using a Java snippet

:= decision.get("Customer")

However, your IT colleagues would really appreciate this capability as it dramatically simplifies the future integration of your tested decision model with their actual business objects that will be used in the production environment.

And finally, we should specify the names for our main file Decision.xls (FILE_NAME) and our main decision name from the table "Decision" (DECISION_NAME). These names are defined in the file "DecisionHello/**define.bat**" contains as follows:

```
set FILE_NAME=rules/main/Decision.xls
set DECISION_NAME=DetermineCustomerGreeting
```

Now you are able to execute your decision model against our test cases. Simply double-click on the file "**run.bat**" that is located in the same folder "DecisionHello".

READER. Finally! Wow, it ran really fast. Here are the results:

```
*** Decision DetermineCustomerGreeting ***
Decision has been initialized

RUN TEST: Test 1
Decision Run has been initialized
Decision DetermineCustomerGreeting: Define
Greeting Word
Assign: Greeting = Good Evening [Good Evening]
Decision DetermineCustomerGreeting: Define
Salutation Word
Conclusion: Salutation Is Mrs. [Mrs.]
Decision has been finalized
Validating results for the test <Test 1>
Test 1 was successful

RUN TEST: Test 2
Decision Run has been initialized
Decision DetermineCustomerGreeting: Define
Greeting Word
Assign: Greeting = Good Afternoon [Good Afternoon]
Decision DetermineCustomerGreeting: Define
Salutation Word
Conclusion: Salutation Is Mr. [Mr.]
Decision has been finalized
Validating results for the test <Test 2>
Test 2 was successful
All 2 tests succeeded!
```

Fig. 1-15

AUTHOR. This is the execution protocol of your testing. It shows execution steps and results for each Test Run.

READER. I see that first OpenRules executed our sub-decision "Define Greeting Word" and our decision table assigned "Good Evening" to the variable Greeting. Then the second decision table made a Conclusion: `Salutation Is Mrs.` This is exactly what was expected. And we got similar results for the second test. Can I add another test-customer?

AUTHOR. Be my guest.

READER. OK, I've just added the third test customer who is a single female that calls at 22:00:

Data Customer customers				
gender	maritalStatus	currentHour	greeting	salutation
Gender	**Marital Status**	**Current Hour**	**Greeting**	**Salutation**
Female	Married	20	?	?
Male	Single	11	?	?
Female	Single	22	?	?

DecisionTableTest testCases			
#	ActionUseObject	ActionExpect	ActionExpect
Test ID	**Customer**	**Greeting**	**Salutation**
Test 1	:= customers[0]	Good Evening	Mrs.
Test 2	:= customers[1]	Good Afternoon	Mr.
Test 3	:= customers[2]	Good Afternoon	Mrs.

Fig. 1-16

Let me run test-cases again:

```
RUN TEST: Test 3
Decision Run has been initialized
Decision DetermineCustomerGreeting: Define Greeting
Word
Assign: Greeting = Good Night [Good Night]
Decision DetermineCustomerGreeting: Define
Salutation Word
Conclusion: Salutation Is Ms. [Ms.]
Decision has been finalized
Validating results for the test <Test 3>
MISMATCH: variable 'Greeting' has value 'Good Night'
while 'Good Afternoon' was expected
MISMATCH: variable 'Salutation' has value 'Ms.'
while 'Mrs.' was expected
Test 3 was unsuccessful
```

Fig. 1-17

Note that I intentionally made errors in the expected Greeting and Salutation. As a result, we received two mismatches which I just highlighted. It seems I really can "touch" our decision model.

AUTHOR. Congratulations! Now I can say that you have gotten the first taste of how to build a decision model using OpenRules-based DMN implementation.

Let's summarize our **major implementation steps**:

1. **Create DRD** (Decision Requirement Diagram) and its tabular representations of the type "Decision" - Figures 2 and 3
2. **Create Decision Tables** – Figures 1-6 and 1-9
3. **Create Glossary** – Fig. 1-10
4. **Create Test Cases** – Fig. 1-16
5. **Executed Test Cases** – Figures 1-15 and 1-17.

While we used these five steps for a simple decision model, they represent a quite generic implementation schema for much more complex decision models as well.

Dialog-Session 2: Enhancing Decision Model "Determine Customer Greeting"

Discussed Topics:

- Adding New Decision Variables
- Single-Hit Decision Tables
- Comparing Dates
- Commenting Out Unused Tables
- Importance of Testing

Related OpenRules Project:

- DecisionHelloWithDates

AUTHOR. Usually the first working decision model is just a good starting point and business people continue to enhance their decisions by adding more concepts and related decision logic. During this session we also will try to enhance our decision model "DecisionHello" to show new DMN capabilities while making it look closer to real-world decision models.

READER. Good, because when I am thinking about decision models we want to use in my organization, they would have many more decision variables, and the decision tables would need to cover many exceptions.

AUTHOR. OK, but we will add some enhancements one by one making sure that our model keeps working. First, I want our model to make exceptions for young customers by producing greetings like "**Good Afternoon, Young Robinson!**"

READER. While I think it is not right to mess with somebody's age, let's do it as an example. First, we probably need to define "young customers".

AUTHOR. Let's say they should be 7 years old or younger.

READER. In this case we just need to add one more condition to the decision table "DefineSalutation". It should check if the customer's Age is less than 8.

AUTHOR. Sounds good. Do you want to try to make the proper changes?

READER. No problem. Assuming that we will define the variable "Age", I will simply add one more rule at the end of this table – see my new table in Fig. 2-1:

DecisionTable DefineSalutation							
Condition		Condition		Condition		Conclusion	
Gender		Marital Status		Age		Salutation	
Is	Male					Is	Mr.
Is	Female	Is	Married			Is	Mrs.
Is	Female	Is	Single			Is	Ms.
Is				<	8	Is	Young

Fig. 2-1

AUTHOR. Adding a new column "Age" so quickly, shows you really have a good grasp of Excel. Now, could you explain how this decision table will work for a male customer of the age 5?

READER. Easy… Wait a second! Do you want to tell me that the first rule will be satisfied and the produced Salutation will be "Mr'" and not "Young"?

AUTHOR. Yes, it will. And the actual reason is that by default our decision table is **single-hit** as specified by the keyword "DecisionTable".

READER. Single-hit? What does it exactly mean?

AUTHOR. It means that when the first rule (counting from top-down) is satisfied, its conclusions will be executed and all remaining rules will be ignored.

READER. WOW! I could have expected that the decision table behaves this way. What if I add the condition “Age >= 8” to the first 3 rules?

AUTHOR. This will fix the problem. Just for your information, there are other types of decision tables, such as multiple-hit decision tables. But we will talk about them during the future sessions. By the way, you may merge the same operations and values to make your table look better.

READER. OK, here is a new version:

DecisionTable DefineSalutation							
Condition		Condition		Condition		Conclusion	
Gender		Marital Status		Age		Salutation	
Is	Male			>=	8	Is	Mr.
Is	Female	Is	Married			Is	Mrs.
Is	Female	Is	Single			Is	Ms.
				<	8	Is	Young

Fig. 2-2

AUTHOR. Looks good. By the way, instead of the operator “<” you may write “Is Less” and instead of the operator “>=” you may write “Is More Or Equal”. Now, what we should do to test your new decision table?

READER. We need to add the variable Age to our glossary. Here is the modified glossary will look like:

Glossary glossary		
Variable	**Business Concept**	**Attribute**
Gender	Customer	gender
Marital Status		maritalStatus
Age		age
Current Hour		currentHour
Greeting		greeting
Salutation		salutation

Fig. 2-3

AUTHOR. How about test cases?

READER. Of course. First I need to add "age" to the Datatype:

Datatype Customer	
String	gender
String	maritalStatus
int	age
int	currentHour
String	greeting
String	salutation

Fig. 2.4

And here is the Data table "customers" with an additional column for "Age":

Data Customer customers					
gender	maritalStatus	currentHour	greeting	salutation	age
Gender	**Marital Status**	**Current Hour**	**Greeting**	**Salutation**	**Age**
Female	Married	20	?	?	45
Male	Single	11	?	?	20
Female	Single	15	?	?	5

Fig. 2-5

AUTHOR. Very good. I noticed that you correctly made sure that the first "black" row is correctly merged covering a new row as well.

READER. I've also corrected the expected results in the table "testCases":

DecisionTableTest testCases			
#	ActionUseObject	ActionExpect	ActionExpect
Test ID	**Customer**	**Greeting**	**Salutation**
Test 1	:= customers[0]	Good Evening	Mrs.
Test 2	:= customers[1]	Good Afternoon	Mr.
Test 3	:= customers[2]	Good Afternoon	Young

Fig. 2-6

The third case should produce the salutation "Young". Correct?

AUTHOR. Double-click on "run.bat" and we will see.

READER. Yes, we got Salutation Young in the third test case, and the first two cases are also correct:

```
RUN TEST: Test 3
Decision Run has been initialized
Decision DetermineCustomerGreeting: Define
Greeting Word
Greeting := Good Afternoon
Decision DetermineCustomerGreeting: Define
Salutation Word
Conclusion: Salutation Is Young
Decision has been finalized
Validating results for the test <Test 3>
Test 3 was successful
All 3 tests succeeded!
```

Fig. 2-7

AUTHOR. Good. What if we want to produce the salutation "Young" only for boys?

READER. I will simply add a check for Male in the last rule of the table "DefineSalutation".

AUTHOR. In this case what will happen when a customer is a 7 years old female?

READER. You are right -- no rules will be satisfied and the Salutation will remain undefined. Should I add one more rule (the default) to explicitly assign '?' to Salutation?

AUTHOR. It certainly will be better than "do nothing"…

READER. OK. Here is my updated decision table:

DecisionTable DefineSalutation							
Condition		Condition		Condition		Conclusion	
Gender		Marital Status		Age		Salutation	
Is	Male			>=	8	Is	Mr.
Is	Female	Is	Married			Is	Mrs.
Is	Female	Is	Single			Is	Ms.
Is	Male			<	8	Is	Young
						Is	?

Fig. 2-8

Let me run it again (just in case). Here we go:

```
RUN TEST: Test 3
Decision Run has been initialized
Decision DetermineCustomerGreeting: Define
Greeting Word
Greeting := Good Afternoon
Decision DetermineCustomerGreeting: Define
Salutation Word
Decision has been finalized
Validating results for the test <Test 3>
MISMATCH: variable 'Salutation' has value '?'
while 'Young' was expected
Test 3 was unsuccessful
1 test(s) out of 3 failed!
```

Fig.2-9

READER. Oops! What did happen? It says:

```
MISMATCH: variable 'Salutation' has value '?'
while 'Young' was expected
```

But why?

AUTHOR. Look at your third customer. This is a 5 years old female. So, naturally your first 4 rules failed, and then the default rules assigned '?' to the variable "Salutation" – exactly as it was supposed to do.

READER. Aha! Let me change the gender to "Male" and run our decision model again. Finally I got it right:

```
Conclusion: Salutation Is Young
```

AUTHOR. As you can see, you cannot overestimate the importance of testing! Even small changes could lead to results which are quite different from what you expected. **Never assume that your modified decision model will produce the correct results – TEST IT!**

READER. Now I see how the addition of even one rule can cause serious trouble. I think this example gives me a better understanding of how the decision model actually works.

AUTHOR. Good. Let's move forward with another enhancement. So far, we used only text variables and integer numbers. In addition, you may use real variables by defining their type as "double" instead of "int" in the Datatype tables. However, now I want to show you **how to deal with dates**.

READER. That would be great. Our own decision models would then be able to apply different rules on different dates.

AUTHOR. Actually DMN allows you to directly compare dates and to make basic arithmetic operations on them. To enhance our model with dates, let's assume that we do not know the customer's age but rather his/her Date of Birth. So, we may replace the condition "Age" in decision table "DefineSalutation" (see Fig. 2-8) with a condition "Date of Birth".

Instead of making changes in this table directly, I will first create a copy of the table in the same worksheet using Excel's Copy/Paste. Then I will comment out the previous table by adding "**//**" in front of the keyword "//DecisionTable".

READER. Why do you do that?

AUTHOR. Sometimes you want to try different implementations of your business logic. Who knows, maybe later on you would like to get back to the previous table. In general, people use standard version control systems such as SVN or GIT for their rule repositories. However, commenting out some decision constructs in the same Excel files is also a useful practice.

So, our new decision table "DefineSalutation" will look as follows:

DecisionTable DefineSalutation							
Condition		Condition		Condition		Conclusion	
Gender		Marital Status		Date of Birth		Salutation	
Is	Male					Is	Mr.
Is	Female	Is	Married	<	1/1/2010	Is	Mrs.
Is	Female	Is	Single			Is	Ms.
Is	Male			>=	1/1/2010	Is	Little

Fig. 2-10

READER. I guess we need to add the new variable "Date of Birth" to the glossary. Let me make the changes myself.

AUTHOR. Yes, go ahead.

READER. Here is the modified glossary:

Glossary glossary		
Variable	**Business Concept**	**Attribute**
Gender	Customer	gender
Marital Status		maritalStatus
Date of Birth		dob
Age		age
Current Hour		currentHour
Greeting		greeting
Salutation		salutation

Fig. 2-11

Now, I will add the attribute “dob” to the Datatype table “Customer”. What type should I use for this attribute?

AUTHOR. You may simply use the type “Date” which is a valid Java type but you do not really have to know about it.

READER. OK, here is the corrected table:

Datatype Customer	
String	gender
String	maritalStatus
Date	dob
int	age
int	currentHour
String	greeting
String	salutation

Fig. 2-12

And now I will add the column “Date of Birth” to the Data table “customers”. I guess I still may keep the attribute “Age” in it.

AUTHOR. Correct, even if our new decision table doesn't use it.

READER. Here is the updated table "customers":

Data Customer customers						
gender	maritalStatus	currentHour	greeting	salutation	dob	age
Gender	**Marital Status**	**Current Hour**	**Greeting**	**Salutation**	**Date of Birth**	**Age**
Female	Married	20	?	?	1/15/1972	45
Male	Single	11	?	?	10/19/1917	20
Male	Single	15	?	?	5/15/2012	5

Fig. 2-13

And, now I believe I can double-click on "run.bat"… YES! The results are the same as when we used the variable "Age". I must say it was quite a natural way to handle dates.

AUTHOR. You also can use predefined functions (or Java methods) to automatically calculate Age based on the Date of Birth, but we will talk about that later. See you next time.

Dialog-Session 3: Integrating Tested Decision Models with IT

Discussed Topics:

- Collaborative Rules and Decision Management
- Business Concepts and Java Decision Objects
- Request-Response
- Concatenating Strings
- Invoking Decision Models from Java Applications
- OpenRules Decision API
- Use of Java Inside Decision Tables

Related OpenRules Project:

- DecisionHelloWithRequestResponse
- DecisionHelloWithJava

AUTHOR. The DMN standard and supporting tools (such as OpenRules) are oriented mainly to business analysts or subject matter experts who are usually not software developers. While business people create and test their decision models, at some point of time they still want to transfer already tested models to their IT department to be integrated with actual business applications. During this session I will explain you how such integration can be achieved.

READER. Yes, even if I already understand how to create my own datatypes and test cases, I know that in our production environment real data is coming from a database and the produced results should be saved back to the database. However, I really have no idea how our IT has organized the database.

AUTHOR. That's exactly why we need to address the IT integration problems ASAP to make sure that decision model

development does not become just a theoretical exercise. You, as a subject matter expert, will continue to stay in charge of the business logic by doing decision model maintenance and testing. However, we need to involve your IT for the integration. So, I suggest the following plan for today:

1) First we will slightly modify our decision model "Determine Customer Greeting" to make it look more like a real-world decision service;
2) Second, we will invite a software developer from your IT (let's call this person "DEVELOPER") to join this session and discuss with us all integration issues using the already tested decision model "Determine Customer Greeting".

READER. Sounds like a good plan. Let me make a quick call, so the DEVELOPER will join us remotely over the web.

AUTHOR. Please do it. During our webinar with the DEVELOPER, we may use Google Docs instead of local Excel. It will be an example of so-called "Collaborative Rules and Decision Management" – see the corresponding schema at this OpenRules web page.

OK, let's start with the first part. In the real-world, application decision models are usually used as services with clearly specified Requests (for input) and Responses (for output). Probably your DEVELOPER expects to see such objects first and is less interested in your business logic. So, let's add Request and Response objects to our decision model "Determine Customer Greeting".

READER. You keep saying “objects” while so far I only remember one object “Customer” to whom you referred as a “Business Concept”.

AUTHOR. You are right – we defined the business concept “Customer” in our Glossary, then specified a corresponding Datatype “Customer”, and then defined specific test-customers in the Data table “customers”.

READER. Probably real customers may be quite different from our definition of the Customer and contain much more information.

AUTHOR. Yes, but this is a good thing that we really did not worry how objects of the type “Customer” are actually represented in a database or how are they defined in a Java application that will use our decision model. In a way, our business concept “Customer” may be considered as our (decision model’s) view of the actual object “Customer”.

Now, I want to split out business concept “Customer” into 3 business concepts by modifying our Glossary in the following way:

Glossary glossary		
Variable	**Business Concept**	**Attribute**
Name	Customer	name
Gender		gender
Marital Status		maritalStatus
Current Hour	Request	currentHour
Location		location
Greeting	Response	greeting
Salutation		salutation
Result		result

Fig. 3-1

READER. I can see that you moved “Current Hour” to the new business concept “Request” and decision variables “Greeting and “Salutation” to another business concept “Response”.

AUTHOR. Here is my reasoning for doing that. The “Current Hour” doesn’t really belongs to the concept “Customer” as we may receive phone calls from the same customer during different hours of the day. Greeting and Salutation are also specific for a particular request, and I placed them in the separate concept “Response” that contains only output data.

READER. What about new variables “Location” and “Result”?

AUTHOR. We may use “Location” i.e. from where the phone call was received to define the Current Hour at this location. And I added “Result” to demonstrate how we can combine the generated Greeting, Salutation and customer’s Name together.

READER. OK. Should not we modify our Data.xls file now?

AUTHOR. Yes, please do it yourself.

READER. No problem. Here are my new Datatype tables:

Datatype Customer	
String	name
String	maritalStatus
String	gender

Datatype Request	
String	location
int	currentHour

Datatype Response	
String	greeting
String	salutation
String	result

Fig. 3-2

And below are the properly modified Data tables:

Data Customer customers		
name	gender	maritalStatus
Customer Name	**Gender**	**Marital Status**
Robinson	Female	Married
Smith	Male	Single
Brown	Female	Single

Data Request requests
currentHour
Current Hour
20
11
14

Data Response responses		
greeting	salutation	result
Greeting	**Salutation**	**Result**
?	?	?
?	?	?
?	?	?

Fig. 3-3

AUTHOR. You didn't specify any location in the table "requests" but that is fine as we will not use this attributes in our model at this time. How about the table "testCases"?

READER. I am not sure how to modify it. We used to have only one column of the type "ActionUseObject" for the object "Customer"…

AUTHOR. You can just add two additional columns of the same type but for the objects Request and Response.

READER. Yes, of course! Here it goes:

DecisionTableTest testCases					
#	ActionUseObject	ActionUseObject	ActionUseObject	ActionExpect	ActionExpect
Test ID	**Request**	**Customer**	**Response**	**Greeting**	**Salutation**
Test 1	:= requests[0]	:= customers[0]	:= responses[0]	Good Evening	Mrs.
Test 2	:= requests[1]	:= customers[1]	:= responses[1]	Good Afternoon	Mr.
Test 3	:= requests[2]	:= customers[2]	:= responses[2]	Good Afternoon	Ms.

Fig. 3-4

Can I run this model now?

AUTHOR. Not yet. Remember we use a special table "decisionObjects" to map our business concept "Customer" to decision objects of the type "Customer? We need to modify this table as follows:

DecisionObject decisionObjects	
Business Concept	**Business Object**
Request	:= decision.get("Request")
Customer	:= decision.get("Customer")
Response	:= decision.get("Response")

Fig. 3-5

READER. OK. Now after running “run.bat” I received exactly the same results.

AUTHOR. Which is good as we didn’t change anything in the business logic or test-customers. Now we are just better prepared for the integration with IT. We still have a few minutes before our web session with your DEVELOPER. Let’s add one more table to fill out the output variable “Result”.

READER. Let’s do it. You said the Result should look like “Good Afternoon, Mrs. Robinson!”.

AUTHOR. And here is a decision table “DefineResult” that has only one Action to define the variable “Result”:

DecisionTable DefineResult
Action
Result
::= ${Greeting} + ", " + ${Salutation} + " " + ${Name} + "!"

Fig. 3-6

As you can see, here we use a special formula that concatenates 6 different strings:

- ${Greeting} – the generated Greeting
- ”, “ – a comma following by a space
- ${Salutation} – the generated Salutation
- “ “ – a space
- ${Name} – Customer’s name
- “!” – the concluding exclamation sign.

In the actual DMN, you should be able to simply use a so-called FEEL expression that looks friendlier:

```
Greeting  + ", " + Salutation + " " + Name + "!"
```

However, the above formula demonstrates how the current OpenRules DMN implementation handles the concatenation of strings (as well as any other Java expression). It will be improved in future releases.

To make sure that this table will be executed by the decision model, we should add it to the table "Decision":

Decision DetermineCustomerGreeting	
Decisions	**Execute Decision Tables**
Define Greeting Word	DefineGreeting
Define Salutation Word	DefineSalutation
Define Result	DefineResult

Fig. 3-7

READER. When I tried to run this decision model, it additionally produced lines like:

```
Result: Good Afternoon, Ms. Brown!
```

AUTHOR. Very good. And now it's time to start our web-session with your DEVELOPER. Please upload our files Decision.xls, Glossary.xls and Data.xls to the Google's server to share them with the DEVELOPER using Google Docs. Don't worry, we will delete them from the server when we are done.

DEVELOPER. Hello!

AUTHOR. Nice to meet you.

READER. Hi. As I told you ahead of time, we've completed our simple decision model called "Determine Customer Greeting".

So far we used Excel only to represent our business logic and created and ran different test cases. So, the model works and is being tested.

Now we want to integrate this model with a Java application. My ultimate objective is to make sure that you would be able to help me to similarly integrate our own decision models in the future.

DEVELOPER. What information does your model require as an input and what will it return as an output?

AUTHOR. This is the right question. As a developer, you may consider the decision model as a "black-box" that takes some input and produces some output.

READER. We've already defined all business concepts in the Excel file "Glossary.xls" – you can see it in the Google Docs now (Fig. 3-1).

AUTHOR. The glossary is actually a map between our business concepts Customer, Request, and Response and Java objects that should be provided by your Java application. The first column contains the business names of our decision variables, and the third column contains their technical counterparts. Preparing to this discussion, I've already opened our OpenRules's project "DecisionHelloWithJava" inside Eclipse IDE, and added 3 Java classes:

- Customer.java
- Request.java
- Response.java

They are basic Java beans and their attributes have exactly the same names as defined in the file Glossary.xls and the same

types as defined in the file Data.xls – see the Datatype tables (Fig. 3-2).

DEVELOPER. OK, this is clear. I can see that you've already added all getters and setters to these Java classes.

AUTHOR. I simply used Eclipse Source +"Generate Getters and Setters…" to do it. Now I should show you a Java launcher that will create test-instances of these 3 classes and will pass then to our decision model using OpenRules Java API. Here is the proper code from the class Main.java:

```
public static void main(String[] args) {

    String fileName = "file:rules/main/Decision.xls";
    String decisionName = "DetermineCustomerGreeting";
    Decision decision = new Decision(decisionName,fileName);
    decision.put("FEEL", "On");
    decision.put("report", "On");

    Customer customer = new Customer();
    customer.setName("Robinson");
    customer.setGender("Female");
    customer.setMaritalStatus("Married");

    Request request = new Request();
    request.setCurrentHour(20);
    request.setLocation("NY");

    Response response = new Response();

    decision.put("Customer", customer);
    decision.put("Request", request);
    decision.put("Response", response);
    decision.execute();

    decision.log("\nDecision produced: " + response.getResult());
}
```

Fig. 3-8

AUTHOR. As you can see, first I created an instance of the OpenRules class Decision:

```
String fileName = "file:rules/main/Decision.xls";
```

```
String decisionName = "DetermineCustomerGreeting";
Decision decision =
                new  Decision(decisionName,fileName);
```

It refers to the main file "Decision.xls" located in the sub-folder "rules/main". Look at this file in Google Docs. The "decisionName" corresponds to the name of the decision table on Fig. 3-7.

Then I created test-instances of the classes Customer, Request, and Response. The class Decision is a HashMap and you should put these 3 instances into the decision using names specified in the table "decisionObjects" of the file Decision.xls (Fig. 3-5). Now we are ready to execute the decision using the method

```
decision.execute();
```

You may print the produced result by displaying any Response's attributes, e.g.:

```
decision.log(response.getResult());
```

DEVELOPER. So, let me summarize what you said. I am just creating an object of the class Decision, adding my input and output objects using the put-methods, and then calling Decision's method execute(). It's a very straightforward API!

AUTHOR. I am glad you like it. This API is well-documented in the User Manual, where you will find more specific features. Of course, you may get your data instances from a database, GUI, or other sources, convert them to Java objects, pass them to the decision for processing, and save the decision's results back to the original sources.

DEVELOPER. Do you provide any database interface?

AUTHOR. Yes, OpenRules comes with a simple JDBC interface, but you may use any 3rd party interface as well.

DEVELOPER. What if my application is not Java-based but rather .NET?

AUTHOR. No problem. You may deploy your OpenRules decision project as a web service and the proper WSDL interface for your .NET application will be automatically generated – read more here.

DEVELOPER. Can I try your application myself?

AUTHOR. I would recommend that you download the OpenRules Evaluation Version and open your own Eclipse with the provided workspace "openrules.dmn". Then you will be able to run and debug the discussed decision model from the project "DecisionHelloWithJava" using Eclipse Run/Debug "As Java Application". You will also find many more complex decision models in this workspace.

READER. It seems you feel comfortable supporting my decision modeling efforts, don't you?

DEVELOPER. I think so. Anything else should I know?

AUTHOR. That's enough to integrate and run decision models. Also please make sure that your business people (decision modelers) utilize your standard version control system for maintenance of their Excel-based rule repositories.

DEVELOPER. Of course, I understand that we should keep all versions of their Excel files in-tact along with versions of our software that utilize these decision models.

AUTHOR. And finally, before you go, I want to share a few general thoughts. A good decision modeling practice frequently deals with what we, technical people, call "separation of concerns" or **correct distribution of knowledge between decision model and the code**. The DMN standard (and its OpenRules implementation) allows decision modelers to put a lot of formulas and even Java code directly into Excel. For example, the READER and I have already discussed how to add a customer's Date of Birth and Age to our model. With OpenRules, we can define a formula that calculates the current customer's Age based on the date of birth. For example, consider the following decision table:

Fig. 3-9

As you may guess, this decision table directly calls the static Java method "yearsToday" of the class Dates and passes to it the value of a Date variable specified by the macro $D{Date of Birth}. The result will be assigned to the variable "Age".

DEVELOPER. I guess OpenRules uses "::=" as an indicator of a Java expression.

AUTHOR. Yes, use "::=" in action-columns and ":=" in condition–columns.

DEVELOPER. Can they use my own Java methods or 3rd party Java libraries directly in Excel?

AUTHOR. Yes. To do that you should make sure that your jar-files are in the project's classpath and simply add a statement "import.java" to the standard Excel table "Environment" as in this example:

Environment	
include	../include/Rules.xls
	../include/Data.xls
	../include/Glossary.xls
	../../../openrules.config/DecisionTemplates.xls
import.java	com.openrules.tools.Dates

Fig. 3-10

DEVELOPER. Got it.

READER. Why didn't you tell me about these capabilities? Are they too technical for me?

AUTHOR. Not only for that reason. The knowledge of how to calculate a person's age based on his/her date of birth is not really business knowledge and does not belong to your business (!) decision model. Your developers can (and should!) guarantee that the object Customer already comes with the attributes Date of Birth and Age being synchronized. That's why the best decision models I've seen in my multi-year consulting practice were created by companies in which business specialists and software developers work in concert. These considerations complete our session today. Thank you to both of you and good luck developing and integrating good decision models together!

Suggested Exercises.

1. Consider a more complex decision model when various customer attributes including gender and marital status

are missing or unknown – see e.g. this DMCommunity's Challenge "Greeting a Customer with Unknown Data"

2. Exercise for DEVELOPER. Analyze and execute the decision model "DecisionInsurancePremium" that will be discussed in the Dialog-Session 7. Instead of the business concepts Driver, Car, and Client defined in Excel, you should use the corresponding Java classes with the same names. Execute Main.java directly from Eclipse – it uses the same decision model but with a different main decision described in the file "DecisionJava.xls"

References

1. Standard "Decision Model and Notation (DMN)", Object Management Group
2. Real-World Decision Modeling with DMN by James Taylor and Jan Purchase, 2016
3. DMN Method and Style: The Practitioner's Guide to Decision Modeling with Business Rules by Bruce Silver, 2016
4. Knowledge Automation: How to Implement Decision Management in Business Processes by Alan N. Fish, 2012
5. OpenRules, Open Source Business Rules and Decision Management System, http://openrules.com
6. Catalog of DMN Supporting Tools http://openjvm.jvmhost.net/DMNtools/
7. The Decision Model by Barbara von Halle, Larry Goldberg, 2010
8. The History of Modeling Decisions using Tables by Jan Vanthienen, 2012