Enterprise Architect

User Guide Series

DMN Modeling and Simulation

How do I model decisions? Use Sparx Systems Enterprise Architect to apply the Decision Model and Notation (DMN) standard constructs to model and document decisions and business rules for business and technical users, in Decision Tables created in FEEL.

Author: Sparx Systems
Date: 21/12/2018
Version: 1.0

CREATED WITH ENTERPRISE ARCHITECT

Table of Contents

DMN Modeling and Simulation

Decision Model and Notation (DMN) is a standard published and managed by the Object Management Group (OMG). It provides a standard approach for describing, modeling and implementing repeatable decisions within an organization or an initiative. It is also intended to facilitate the sharing and interchange of decision models between organizations.

The modeling notation comprises a visual grammar that allows decisions and business rules to be documented in a way that makes them readable by both business and technical audiences thus ensuring that decisions and rules are not misinterpreted. The resulting Decision Model also provides a definition of how to evaluate the logic of decisions defined in Decision Tables using the Friendly Enough Expression Language (FEEL).

The purpose of DMN is to provide the constructs that are needed to model decisions, so that organizational decision-making can be readily depicted in diagrams, accurately defined by business analysts.

In this topic, we will introduce DMN Expression, DMN Simulation Artifact and how you can use Enterprise Architect to automate the decision-making process.

DMN Expressions

- DMN Expression: Decision Table
- DMN Expression: Boxed Context
- DMN Expression: Literal Expression
- DMN Expression: Function
- DMN Expression: Invocation

DMN Data

- DMN ItemDefinition
- DMN InputData
- DataSet for InputData

DMN Simulation

- Configure a DMN Simulation Artifact
- Run, Step through or Debug the DMN Simulation

Code Generation & Connect to BPMN

- Generate the DMN Server in Java, JavaScript, C++, or C#
- Run/Debug testing of the Java version of the DMN Server
- Connect the DMN Server with the Enterprise Architect

BPSim Execution Engine

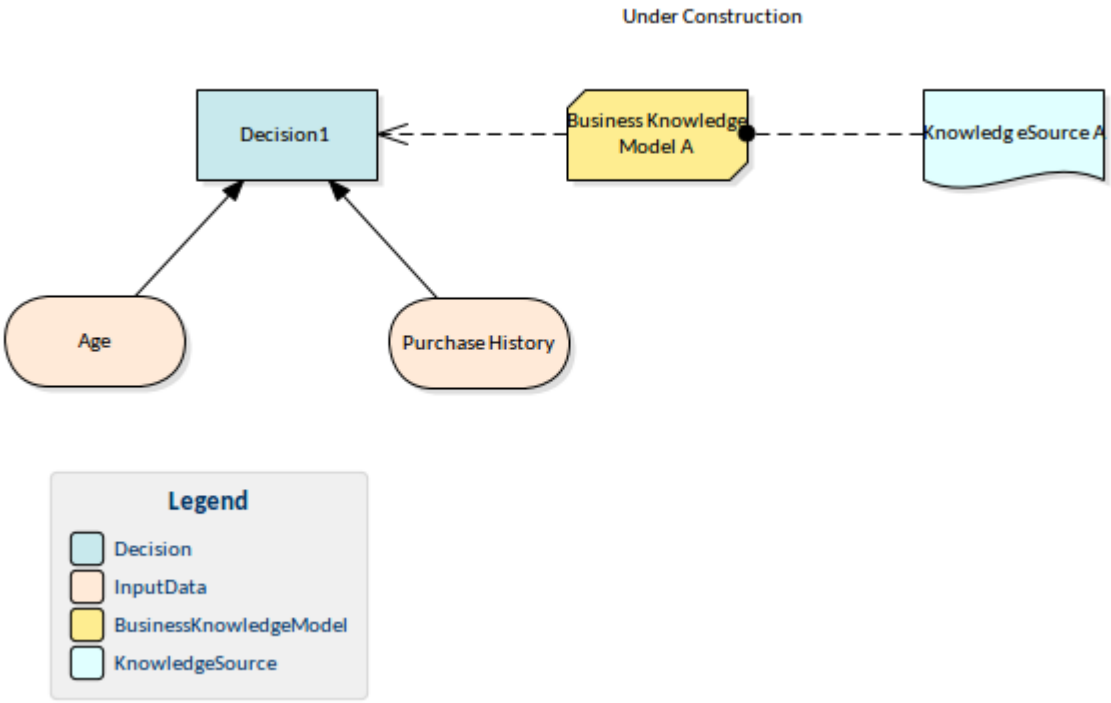
Common Errors & Solutions

- 'Run validation' will help you locate most of the modeling issues; run this before simulation and code generation
- Variable Types: as DMN models use the FEEL language (Simulate with JavaScript), typing variables is not compulsory; however, when generating code to languages that are compiled, you do have to type a variable - there are context menu options and tag values for setting a type to the variables
- Since a DMN expression allows for spaces, in order to clarify the composite Input Data there must be a space before and after the '.' in the expression; for example, 'Applicant data . Age' is valid, whereas 'Applicant data.Age' is not valid
  Note that when using the Auto Completion feature this issue will not arise

Introduction

Now more than ever, in a world turned on its head by new and innovative business and technical ideas and disruptive ways of working, does an organization need to have a clear understanding of its choices and the decisions it makes. Unmanaged complexity is the enemy, and agility the friend that heralds business success and enables an organization to respond quickly to changes in its business circumstances. Without a clear and communicable model it is almost impossible for an organization to embrace the changes that confront them daily in the digital world.

The description and implementation of decisions, which has been inexplicably and somewhat invisibly part of almost all disciplines, has now been synthesized into a rigorous and formal discipline of its own with a new way of modeling and describing decisions, Inputs, Outcomes, Rules, Business Knowledge, Authorities and more. Once you have seen the Decision Model and Notation in action and been introduced to the countless benefits it brings, you will not be able to go back to old and arcane ways of working.



Enterprise Architect has become the tool of choice for many Business and Technical leaders because of its flexible, extensible, standards-based and pragmatic approach to modeling complex systems. As a collaboration platform it is a tool for all disciplines, allowing Decision Models to be created, integrated, managed, documented, simulated and generated to programming code. The models can be visualized and integrated with a range of other models, including Business Process diagrams, Use Case models, User Stories, Test Cases, Database models, Implementation Artifacts and programming code to list just the main models.

What is Decision Model and Notation

Decision Model and Notation (DMN) is a standard published and managed by the Object Management Group (OMG). It provides a standard approach for describing, modeling and implementing repeatable decisions within an organization or an initiative. It is also intended to facilitate the sharing and interchange of decision models between organizations.

The modeling notation consists of a visual grammar that allows decisions and business rules to be documented in a way that makes them readable by both business and technical audiences, thus ensuring that decisions and rules are not misinterpreted. The resulting Decision Model also provides a definition of how to evaluate the logic of decisions defined in Decision Tables, using the Friendly Enough Expression Language (FEEL).

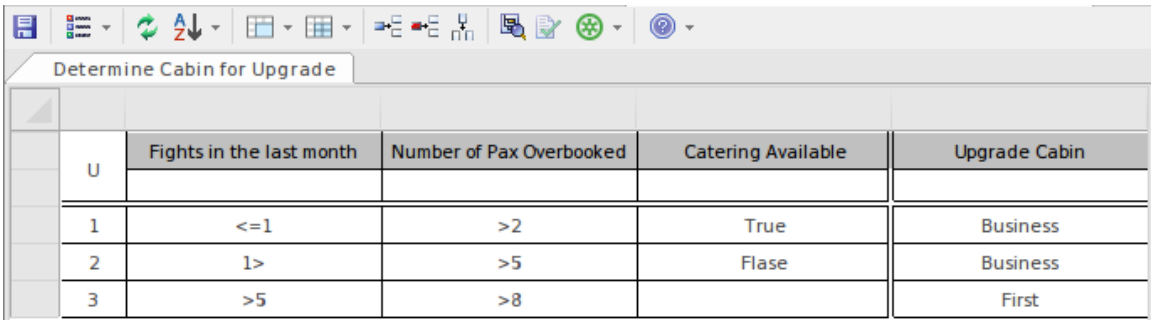
DMN defines two levels of the language, which neatly align with organizational roles:

Requirements Level

Aligns with the business people who devise, analyze and define the decision rules. This provides a mechanism for the people who own the business to model them without the need for technical knowledge of how they are implemented.

Logic Level

Aligns with implementation teams who augment the definitions to include decision logic in the form of Decision Tables that can be maintained by the business staff, or expression logic managed by technical staff.



Determine Cabin for Upgrade

U	Fights in the last month	Number of Pax Overbooked	Catering Available	Upgrade Cabin
1	<=1	>2	True	Business
2	1>	>5	Flase	Business
3	>5	>8		First

Why Use Decision Model and Notation

There are many reasons for using DMN at an Enterprise or Initiative level that will result in both business and technical value. There are many benefits described in the next topic but the most compelling of these are:

Reduced Complexity

As discussed earlier, currently business rules and decisions are commonly located in a myriad of places and in a wide variety of formats. A common place to find them is in Business Process Diagrams that are overly complex with cascading sets of Gateways that describe the outcomes. The diagrams are commonly unwieldy and do little to reveal the inputs, business knowledge, authorities or the logic of how the decisions are made.

Automation and Execution

As the Business Analysts define the decisions, inputs, business knowledge and the technologists elaborate the logic in the form of Expressions and Decision Tables, potentially creating Decision Services, the models can be used to generate programming language code that can be executed to automate the decisions and make them available to runtime systems.

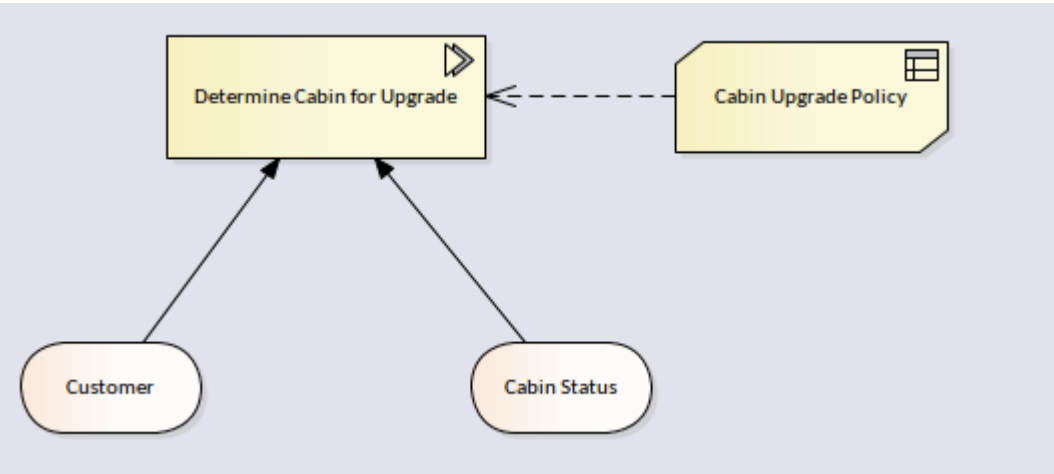
Agility and Responsiveness to Change

As a result of separating the decision models from the Business Processes models and creating an implementation and automation pipeline, the business and technology teams can respond at record speed to business change. These models then become a platform that facilitates agility and the seizing of opportunities.

A First Example

Imagine you are an Airline reservation officer working at the check-in counter for a busy domestic airline. Getting the aircraft off on-time is critical as delays can result in fees applied by the airport controllers, needing to fly at a lower altitude increasing the cost of fuel, and other penalties.

A message from the supervisor appears on your screen saying that the economy cabin is overbooked; you will need to upgrade some passengers to Business or First Class — but which passengers should be chosen and which cabin should they be upgraded to? A decision needs to be made but what factors should be considered? This can be recorded in a Decision Model using a Decision Requirements diagram.



This is helpful but the busy check-in officer would still need to weigh up all the factors and make an unbiased decision. Should a disgruntled passenger be given priority over a

Gold level frequent flyer, or should the fact that a particular passenger is connecting to an international flight take precedence. These 'rules' can all be recorded in a Decision table, making it clear which passengers should get an upgrade and to which cabin: Business or First Class. This will make it much easier to make the decision and the rules can be formulated, agreed upon and checked for consistency back at head office. In this example we have kept it simple and used two factors: firstly the number of flights the passenger has made in the last month and secondly how overbooked the cabin is.

Cabin Upgrade Policy | Input Parameter Values for Simulation

(Flights in the last month, Number of Pax Overbooked)

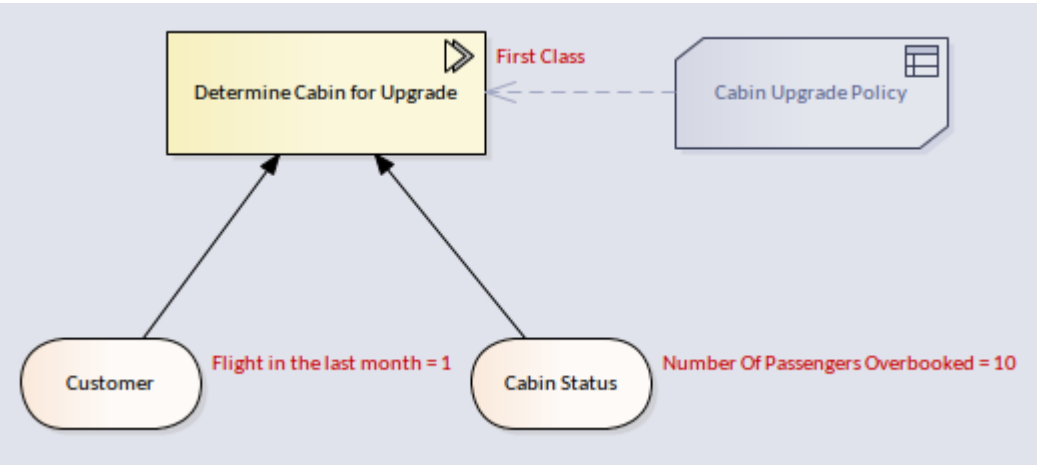
U	Flights in the last month	Number of Pax Overbooked	Upgrade Cabin
			Business Class, First Class
1	<=1	<=2	*Business Class*
2	<=1	(2..8]	*Business Class*
3	<=1	>8	*First Class*
4	(1..5]	<=2	*Business Class*
5	(1..5]	(2..8]	*Business Class*
6	(1..5]	>8	*First Class*
7	>5	<=2	*Business Class*
8	>5	(2..8]	*Business Class*
9	>5	>8	*First Class*

The table is divided into columns and rows. There are two types of columns: inputs that are required to make the decision and outputs that are the result of applying the rules.

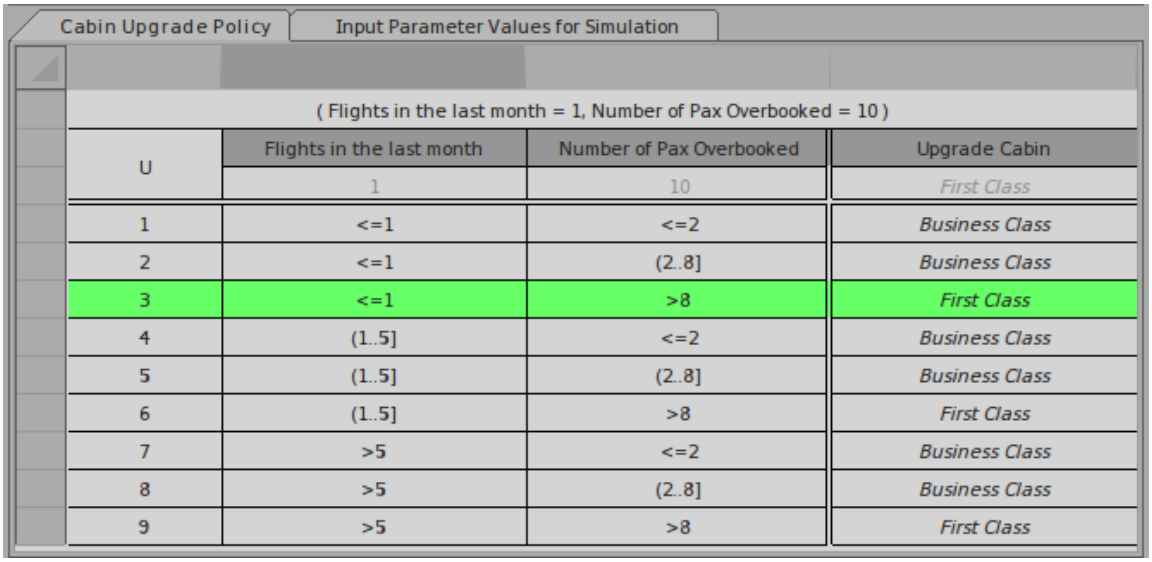
This is again very helpful but still requires the busy check-in officer to be able to source all the required information required to find the right row in the Decision table. Even if all this information were available, a wrong decision could still result from human error in selecting the wrong row in

the table.

Fortunately the Decision Models can be automated and generated to programming code that can be executed by an application. So our busy check-in officer would not need to do anything or make any decisions; as she was checking in the passengers, if a particular passenger was entitled to an upgrade it would be visible on the computer screen. In the next diagram the model has been simulated so that the business and technical staff can agree that the model has been defined correctly. Any number of user defined data sets can be used to test the model before generating out the programming code that will run in the check-in system and display the result to the end user.



When developing the models a business or technical user can step through the simulation and the system will show that user which row in the Decision table was fired to determine the output. This is very useful in models that are made up of multiple decisions.



Cabin Upgrade Policy | Input Parameter Values for Simulation

(Flights in the last month = 1, Number of Pax Overbooked = 10)

U	Flights in the last month	Number of Pax Overbooked	Upgrade Cabin
	1	10	*First Class*
1	<=1	<=2	*Business Class*
2	<=1	(2..8]	*Business Class*
3	<=1	>8	*First Class*
4	(1..5]	<=2	*Business Class*
5	(1..5]	(2..8]	*Business Class*
6	(1..5]	>8	*First Class*
7	>5	<=2	*Business Class*
8	>5	(2..8]	*Business Class*
9	>5	>8	*First Class*

It is common for the rules that govern the upgrade decision to change. For example, the Marketing Department might decide they want to reward passengers that travel on long-haul flights. The Decision Requirements diagram can be altered to include the new input, the Decision table modified, and the programming code regenerated. Once the changes have been pushed through to the airport systems, the right passengers will be automatically upgraded. The check-in officer could still view the Decision tables during a training and briefing session to understand the rules.

DecisionTable

A Decision table is a tabular representation of a set of related input and output expressions, organized into rules indicating which output entry applies to a specific set of input entries.

Access

Ribbon	Simulate > Decision Analysis > DMN > DMN Expression, then select or create a Decision or BusinessKnowledgeModel
Other	Double-click on a DMN Decision or BusinessKnowledgeModel

Toolbar Options

Option	Description
	Click on this button to save the configuration to the current Decision or BusinessKnowledgeModel.

	Click on this button to switch between Rule-as-row and Rule-as-column for the Decision table.
	Click on 'Sort By Input' to sort the rules by input columns; click on 'Sort By Output' to sort the rules by output columns. The columns can be dragged and dropped to organize the sorting order.
	Click on this button to merge adjacent input entry cells from different rules with the same content.
	Click on this button to split entry cells that have been merged.
	Click on this button to edit parameters for the Business Knowledge Model.
	Click on this button to append an input column for the Decision table.
	Click on this button to append an output column for the Decision table.
	Click on this button to append a rule for the Decision table.

	Click on this button to show or hide the allowed values fields for the 'Input' and 'Output' columns. The allowed value defined for an input or output will be used for validation and auto completion editing.
	Click on this button to validate the Decision table. Enterprise Architect will perform a series of validations to help the you locate errors in the Decision table.
	This button is enabled when a Decision table is defined for a BusinessKnowledgeModel. Select the 'Input Parameter Values for Simulation' tab, complete the fields and click on this button. The test result will be presented on the Decision table, with the runtime values of inputs and outputs displayed and valid rule(s) highlighted. You can use this functionality to unit test a BusinessKnowledgeModel without knowing the context and later on invoked by a Decision or other BusinessKnowledgeModel. Menu options are available for this tool bar button. For more information, see the

	BusinessKnowledgeModel and Test Harness Help topic.

Decision Table Hit Policy

The Hit Policy specifies the result of the Decision table in cases of overlapping rules. The single character in a particular Decision table cell indicates the table type and unambiguously reflects the decision logic.

Single Hit Policies:

- **U**nique: no overlap is possible and all rules are disjoint; only a single rule can be matched (this is the default)
- **A**ny: there might be overlap, but all the matching rules show equal output entries for each output, so any match can be used
- **P**riority: multiple rules can match, with different output entries; this policy returns the matching rule with the highest output priority
- **F**irst: multiple (overlapping) rules can match, with different output entries; the first hit by rule order is returned

Multiple Hit Policies:

- **O**utput order: returns all hits in decreasing output priority order
- **R**ule order: returns all hits in rule order

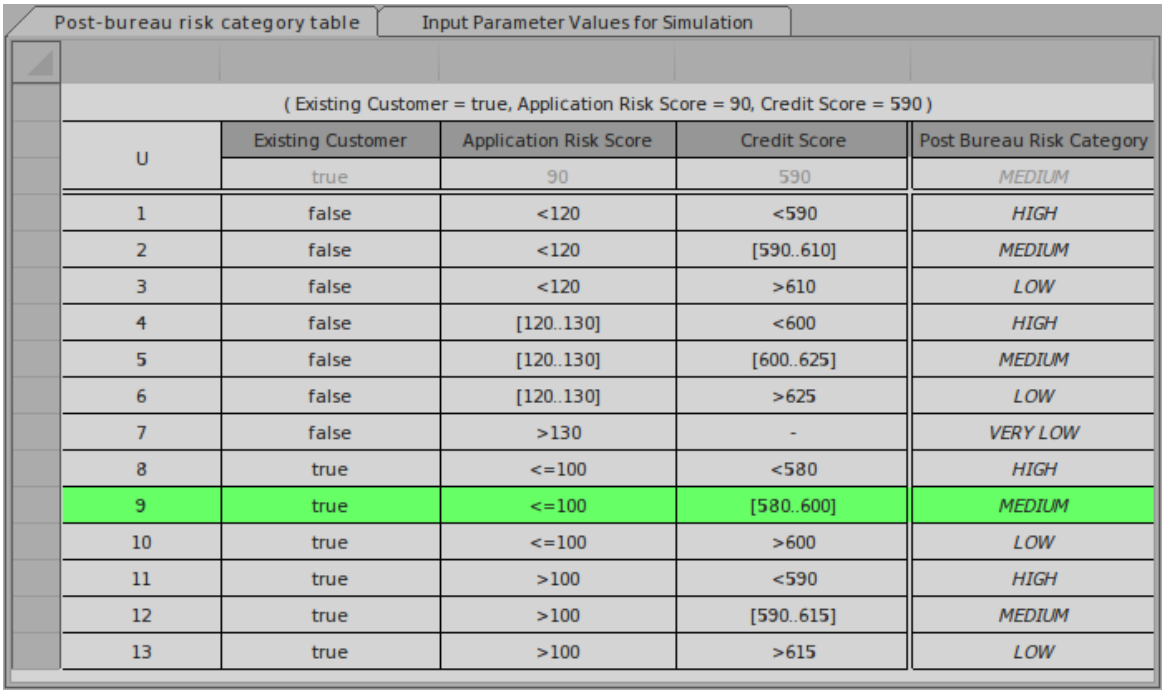
- **C**ollect: returns all hits in arbitrary order; an operator (‘+’, ‘<’, ‘>’, ‘#’) can be added to apply a simple function to the outputs

Collect operators are:

- + (sum): the result of the Decision table is the sum of all the distinct outputs
- < (min): the result of the Decision table is the smallest value of all the outputs
- > (max): the result of the Decision table is the largest value of all the outputs
- # (count): the result of the Decision table is the number of distinct outputs

Example of Unique hit policy

The 'Unique' hit policy is the most popular type of Decision table and all rules are disjoint.



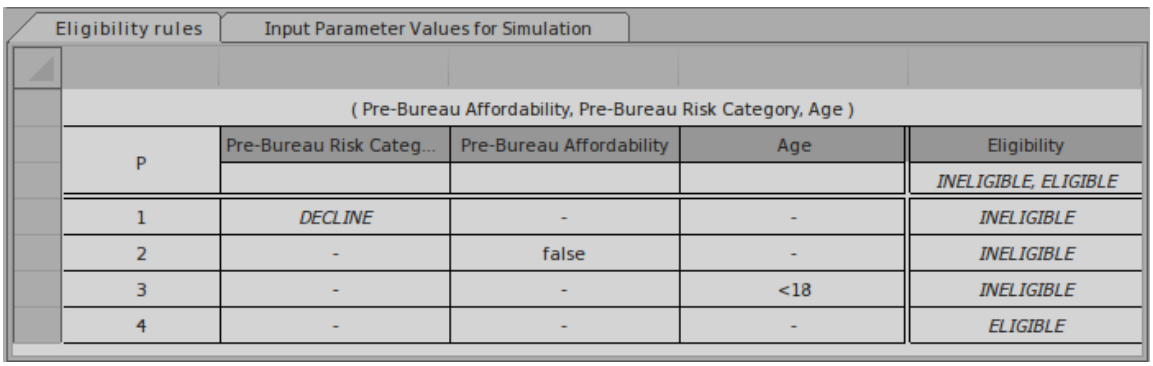
Post-bureau risk category table | Input Parameter Values for Simulation

(Existing Customer = true, Application Risk Score = 90, Credit Score = 590)

U	Existing Customer	Application Risk Score	Credit Score	Post Bureau Risk Category
	true	90	590	*MEDIUM*
1	false	<120	<590	*HIGH*
2	false	<120	[590..610]	*MEDIUM*
3	false	<120	>610	*LOW*
4	false	[120..130]	<600	*HIGH*
5	false	[120..130]	[600..625]	*MEDIUM*
6	false	[120..130]	>625	*LOW*
7	false	>130	-	*VERY LOW*
8	true	<=100	<580	*HIGH*
9	true	<=100	[580..600]	*MEDIUM*
10	true	<=100	>600	*LOW*
11	true	>100	<590	*HIGH*
12	true	>100	[590..615]	*MEDIUM*
13	true	>100	>615	*LOW*

Example of Priority hit policy

In a table with the 'Priority' hit policy, multiple rules can match, with different output entries. This policy returns the matching rule with the highest output priority.



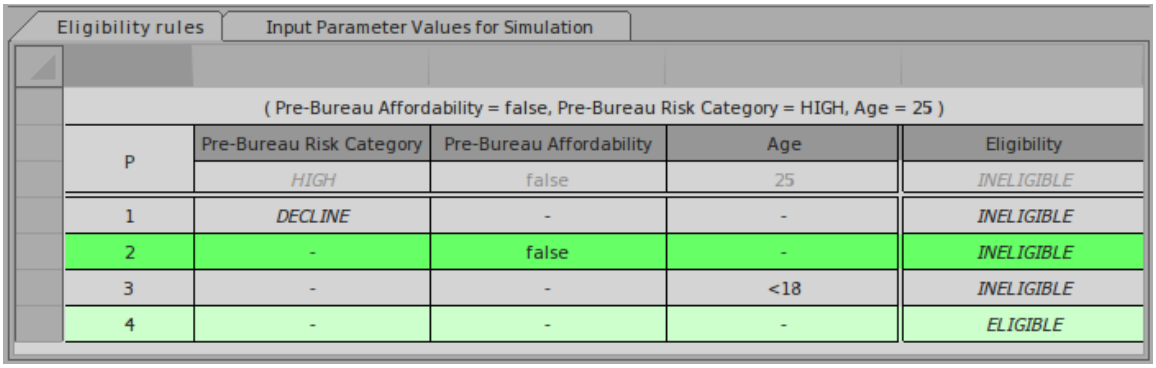
Eligibility rules | Input Parameter Values for Simulation

(Pre-Bureau Affordability, Pre-Bureau Risk Category, Age)

P	Pre-Bureau Risk Categ...	Pre-Bureau Affordability	Age	Eligibility
				INELIGIBLE, ELIGIBLE
1	*DECLINE*	-	-	*INELIGIBLE*
2	-	false	-	*INELIGIBLE*
3	-	-	<18	*INELIGIBLE*
4	-	-	-	*ELIGIBLE*

Note: The output order is defined as INELIGIBLE, ELIGIBLE; which means INELIGIBLE has higher priority than ELIGIBLE.

One possible simulation result might resemble this:



Eligibility rules | Input Parameter Values for Simulation

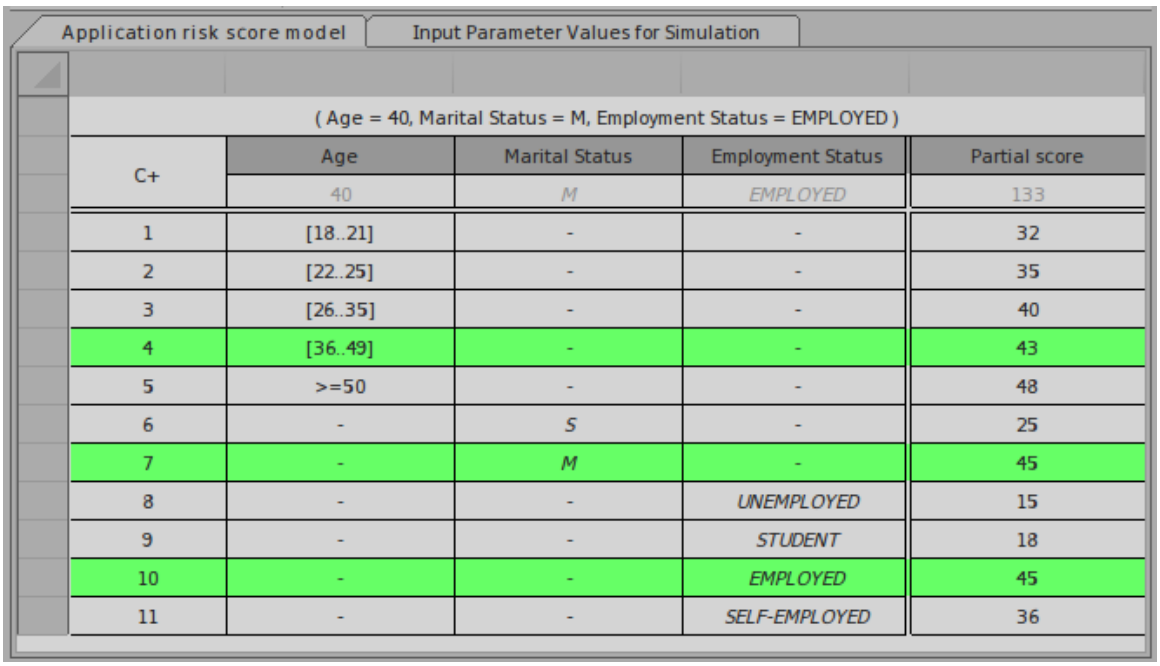
(Pre-Bureau Affordability = false, Pre-Bureau Risk Category = HIGH, Age = 25)

P	Pre-Bureau Risk Category	Pre-Bureau Affordability	Age	Eligibility
	HIGH	false	25	*INELIGIBLE*
1	*DECLINE*	-	-	*INELIGIBLE*
2	-	false	-	*INELIGIBLE*
3	-	-	<18	*INELIGIBLE*
4	-	-	-	*ELIGIBLE*

The matching rules are highlighted, but output with INELIGIBLE is the pick based on output order.

Example of Collection-Sum hit policy

For a Decision table with the 'Collect-Sum' (C+) hit policy, the result of the Decision table is the sum of all the distinct outputs.



Application risk score model | Input Parameter Values for Simulation

(Age = 40, Marital Status = M, Employment Status = EMPLOYED)

C+	Age	Marital Status	Employment Status	Partial score
	40	*M*	*EMPLOYED*	133
1	[18..21]	-	-	32
2	[22..25]	-	-	35
3	[26..35]	-	-	40
4	[36..49]	-	-	43
5	>=50	-	-	48
6	-	*S*	-	25
7	-	*M*	-	45
8	-	-	*UNEMPLOYED*	15
9	-	-	*STUDENT*	18
10	-	-	*EMPLOYED*	45
11	-	-	*SELF-EMPLOYED*	36

In this example, the output Partial Score is calculated as 43 + 45 + 45 = 13_3

Set type for Input/Output Clause

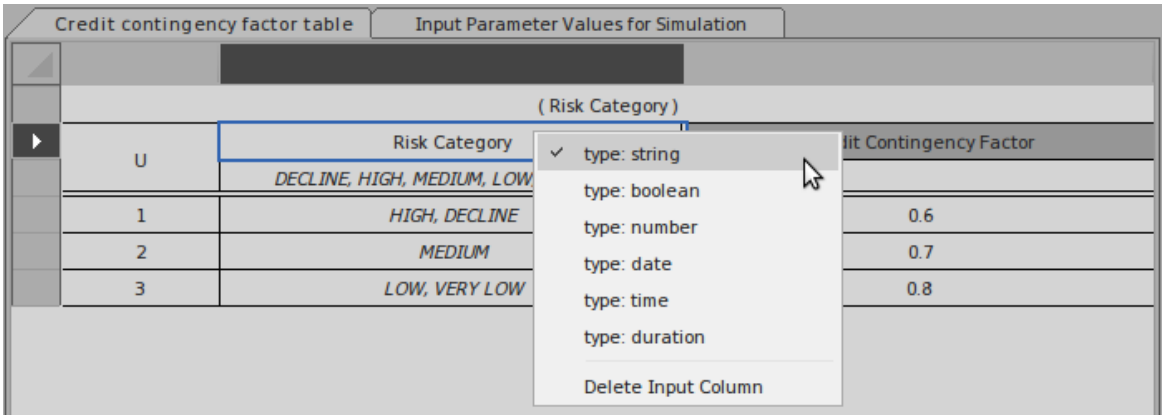
You have to set the type for Input Clause or Output Clause for these reasons:

- Display: The string will be shown in Italic font (otherwise, each string literal must be inside quotes)
- Validation: The range, gap and overlap validations are supported only on numbers
- Code Generation for typed languages such as C++, C# and Java

You might ask: since the BKM parameter already has types defined, why do we need to define types for the Input Clauses?

The answer is: the Input Clause could be an expression. For example, a parameter 'Credit Score' is a number type, but the Input Clause could be 'Credit Score > 500?', which is a boolean type.

Right-click on the Input Clause or Output Clause and select the appropriate type for it.


Credit contingency factor table
Input Parameter Values for Simulation
(Risk Category)
U
Risk Category
lit Contingency Factor
DECLINE, HIGH, MEDIUM, LOW
1
HIGH, DECLINE
0.6
2
MEDIUM
0.7
3
LOW, VERY LOW
0.8
type: string
type: boolean
type: number
type: date
type: time
type: duration
Delete Input Column

Literal Expression

A Literal Expression is the simplest form of DMN expression. It is commonly defined as a single-line statement or an if-else conditional block. When the expression is getting complicated, you might prefer a Boxed Context, or in order to improve the readability you can encapsulate some logic as a function in the DMN Library. See the example at the end of this page.

Access

Ribbon	Simulate > Decision Analysis > DMN > DMN Expression : select or create a Decision or BusinessKnowledgeModel
Other	Double-click on a DMN Decision or BusinessKnowledgeModel

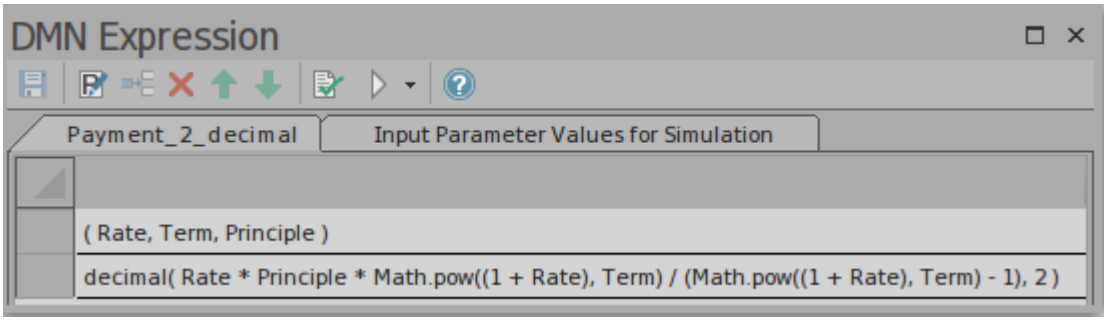
Toolbar Options

Options	Description

	Click on this button to save the configuration to the current Decision or BusinessKnowledgeModel.
	Click on this button to edit parameters for the Business Knowledge Model.
	This option is disabled for Literal Expressions.
	This option is disabled for Literal Expressions.
	This option is disabled for Literal Expressions.
	This option is disabled for Literal Expressions.
	Click on this button to validate the Literal Expression. Enterprise Architect will perform a series of validations to help you locate errors in the Expression.
	This button is enabled when the literal expression is defined for a BusinessKnowledgeModel.

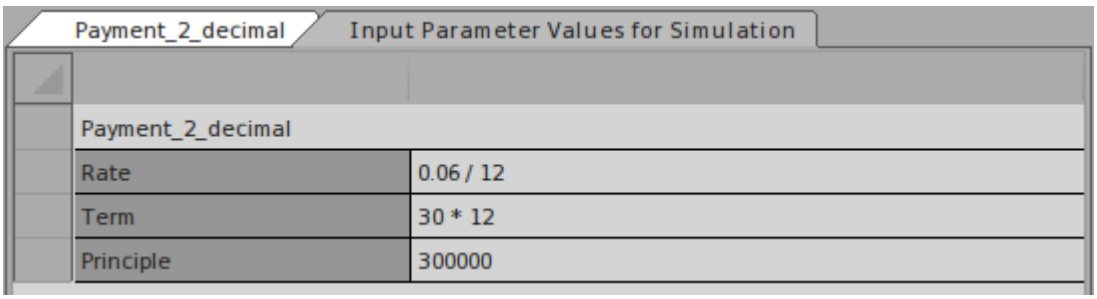
Example — Payment of 2 Decimals

This Business Knowledge Model (BKM) *Payment_2_decimal* is implemented as a Literal Expression.

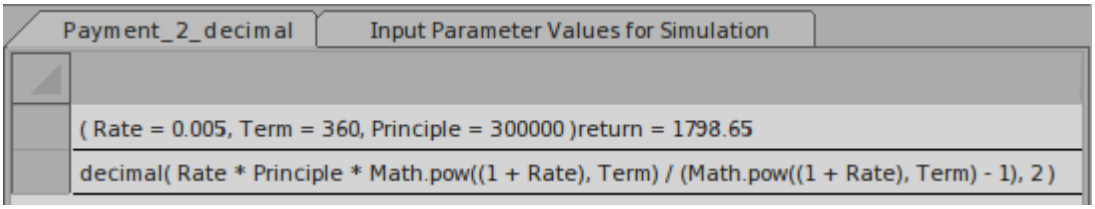


- The BKM defines three parameters: Rate, Term and Principle

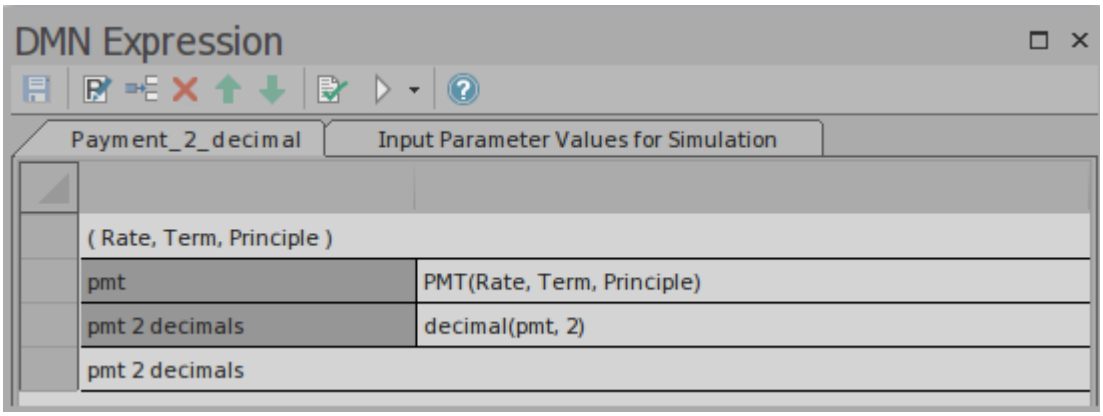
Give values for the Test Harness and evaluate the model:



- The runtime parameter value will be displayed; for example, Rate = 00.005
- The BKM's result will be evaluated by the literal expression and the value is displayed on the declaration line; for example, return = 1798.65

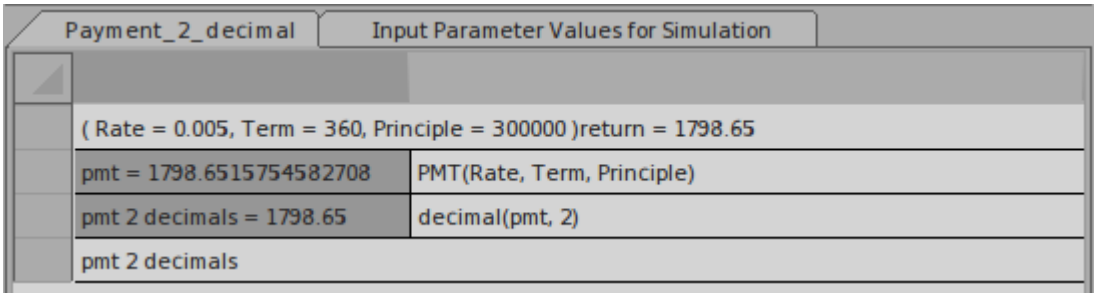


Although the implementation is one line, it is quite complicated. We can re-factor this model with Built-In function and Boxed Context to improve readability:



- The Boxed Context defines two variable-expression paired entries; these variables serve as 'local variables', which can be used in later expressions
- Return value: the expression can use the value of 'local variables'
- Any expressions in a Boxed Context can use built-in functions that are defined in the customizable Template — *DMN Library*; for example, functions PMT(...) and decimal(...) are used in this example

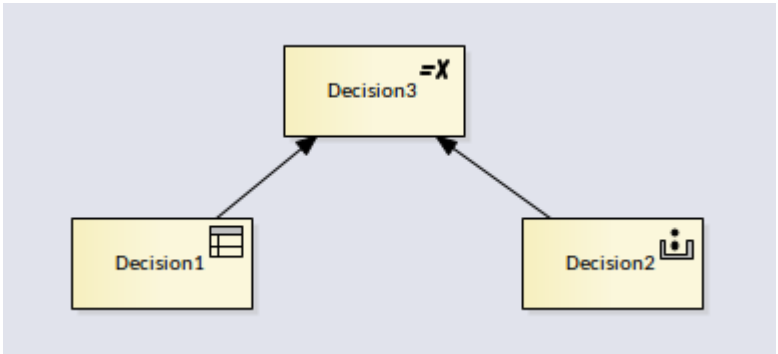
The simulation result is exactly the same as a Literal Expression:



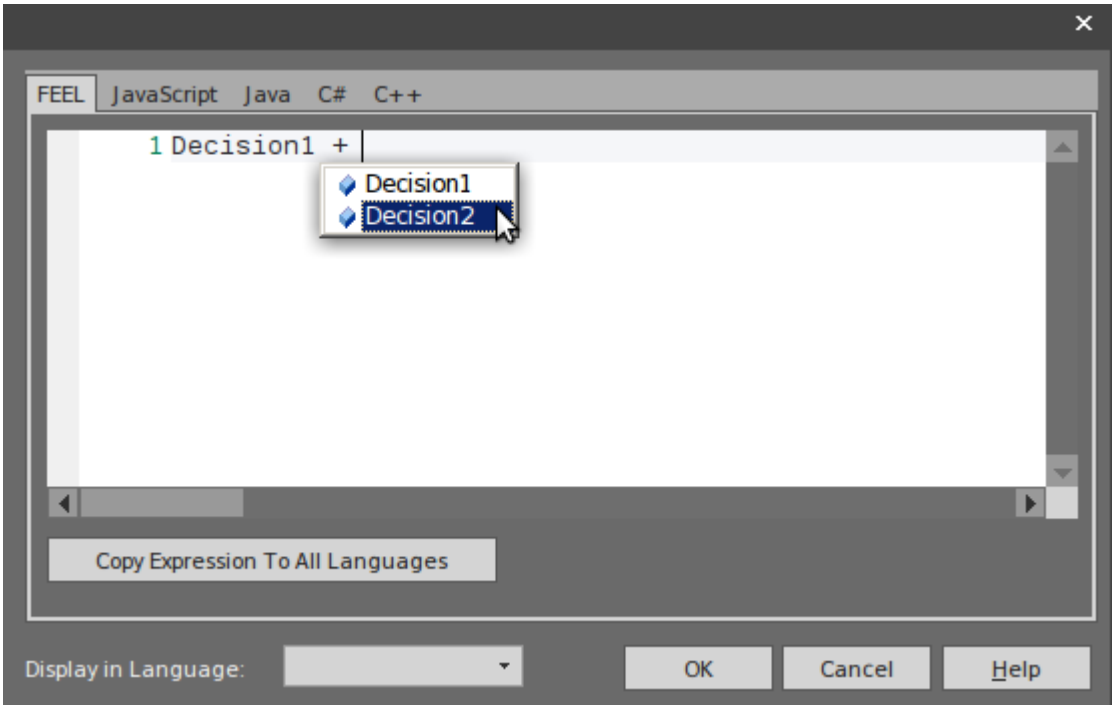
Expression Editor and Intelli-sense Support

In accordance with the FEEL language specification, the parameter names can contain spaces. This feature makes the expression easier to read. Enterprise Architect also provides Intelli-sense support for editing the expressions with minimal typing and fewer mistakes.

Given a decision hierarchy such as this, the expression in 'Decision3' should be able to use the required decisions (variables).



Right-click on the Expression and select the menu option 'Edit Expressions...' to display the 'Expression' dialog.



Press Ctrl+Space to show the Intelli-sense menu:

- For 'BKM', all the parameters will be included
- For 'Decision', all the required Decisions will be included

The DMN Model can be generated as source code in JavaScript, Java, C# or C++; since some languages might have different syntax for some expressions, Enterprise Architect provides language override pages for each language. If no override code is specified for a language, the expression defined for the FEEL language will be used.

In the generated code, the space inside a variable name will be replaced by an underscore.

Boxed Context

A Boxed Context is a collection of context entries. Each context entry consists of a variable and an expression. The Context also has a result value.

Access

Ribbon	Simulate > Decision Analysis > DMN > DMN Expression : select or create a Decision or BusinessKnowledgeModel
Other	Double-click on a DMN Decision or BusinessKnowledgeModel

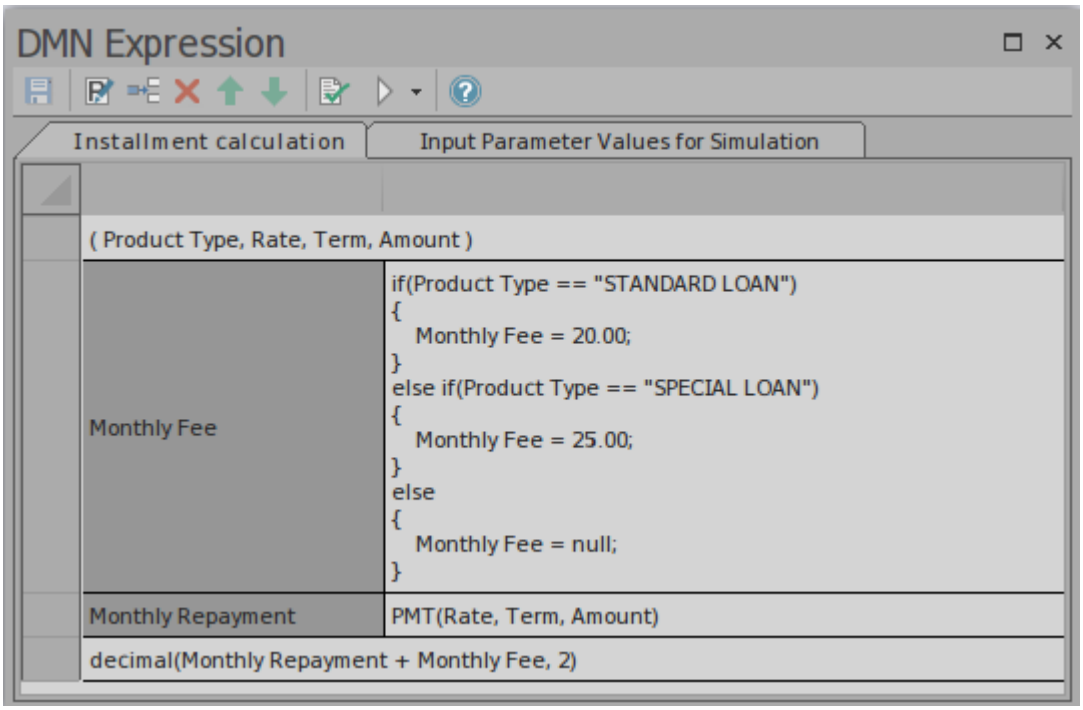
Toolbar Options

Options	Description
	Click on this button to save the configuration to the current Decision or BusinessKnowledgeModel.

	Click on this button to edit parameters for the Business Knowledge Model.
	Click on this button to add a 'Context Entry' to the boxed context. Each 'Context Entry' consists of a variable and an expression.
	This button is enabled when a ContextEntry's variable is selected. Click on this button to delete the selected Context Entry.
	This button is enabled when a 'Context Entry' variable is selected. Click on this button to move the selected 'Context Entry' up one position.
	This button is enabled when a 'Context Entry' variable is selected. Click on this button to move the selected 'Context Entry' down one position.
	Click on this button to validate the Boxed Context. Enterprise Architect will perform a series of validations to help you locate errors in the Expression.
	This button is enabled when the Boxed

	Context is defined for a Business Knowledge Model.

Example — Installment calculation



The Business Knowledge Model (BKM) *Installment calculation* is implemented as Boxed Context.

- The BKM defines four parameters: Product Type, Rate, Term and Amount
- The Boxed Context defines two variable-expression pair entries, these variables serve as 'local variables' that can be used in later expressions
- Return value: The expression can use the value of 'local

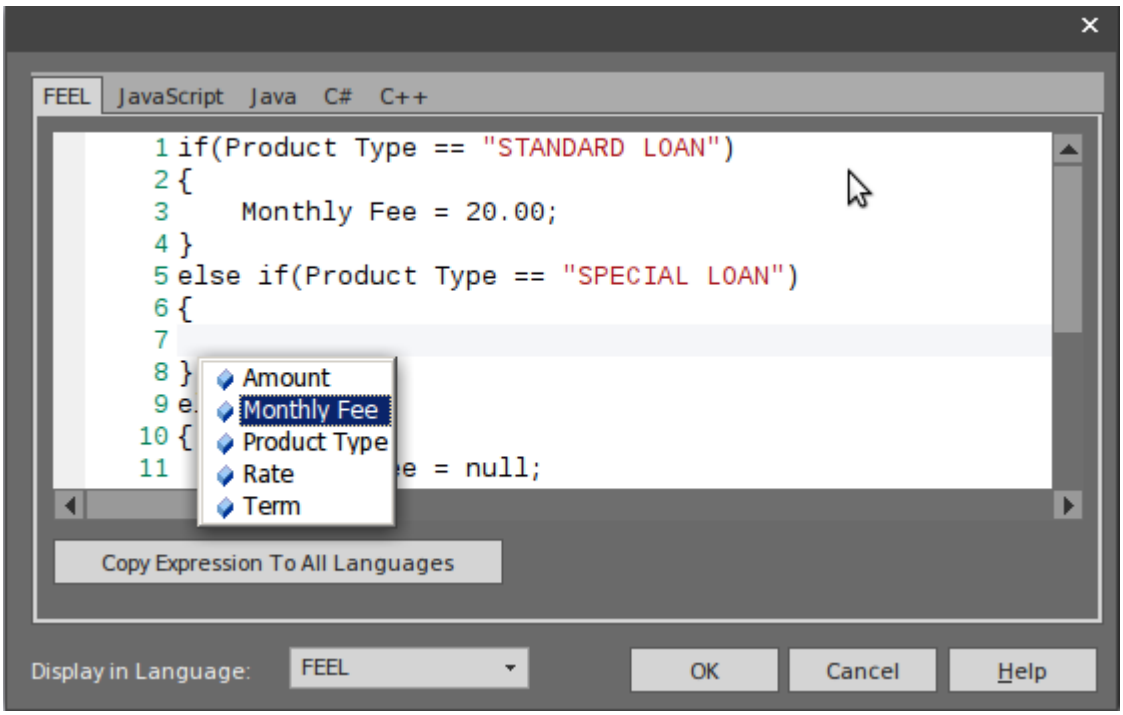
variables'

- Any expressions in a Boxed Context can use built-in functions, which are defined in the customizable Template — *DMN Library*; for example, functions PMT(...) and decimal(...) are used in this example

Expression Editor and Intelli-sense Support

The parameter and Context Entry's variable name can contain spaces, according to the FEEL language specification. This feature makes the expression easy to read. In order to help you edit the expressions with less typing and making fewer mistakes, Enterprise Architect provides Intelli-sense support for editing expressions:

Right-click on the Expression | Edit Expressions... the 'Expression' dialog displays



Press Ctrl+Space to show the Intelli-sense menu:

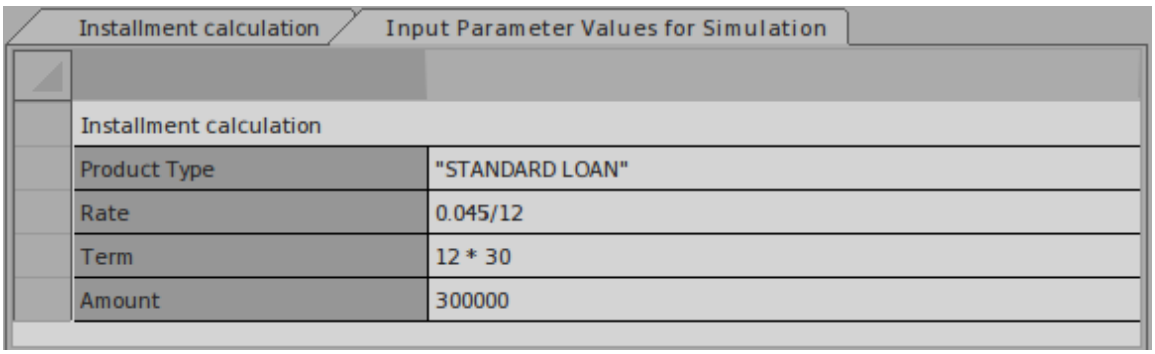
- All the Context Entry Variables earlier than the current one will be included (the context entries later than the current one are excluded)
- For BKM, all the parameters will be included
- For Decision, all the required Decisions will be included

The DMN model can be generated as source code for JavaScript, Java, C# and C++. Since some languages might have different syntax for some expressions, Enterprise Architect provides language override pages for each language. If no override code is specified for a language, the expression defined for the FEEL language will be used.

In the generated code, the space inside a variable name will be replaced by an underscore.

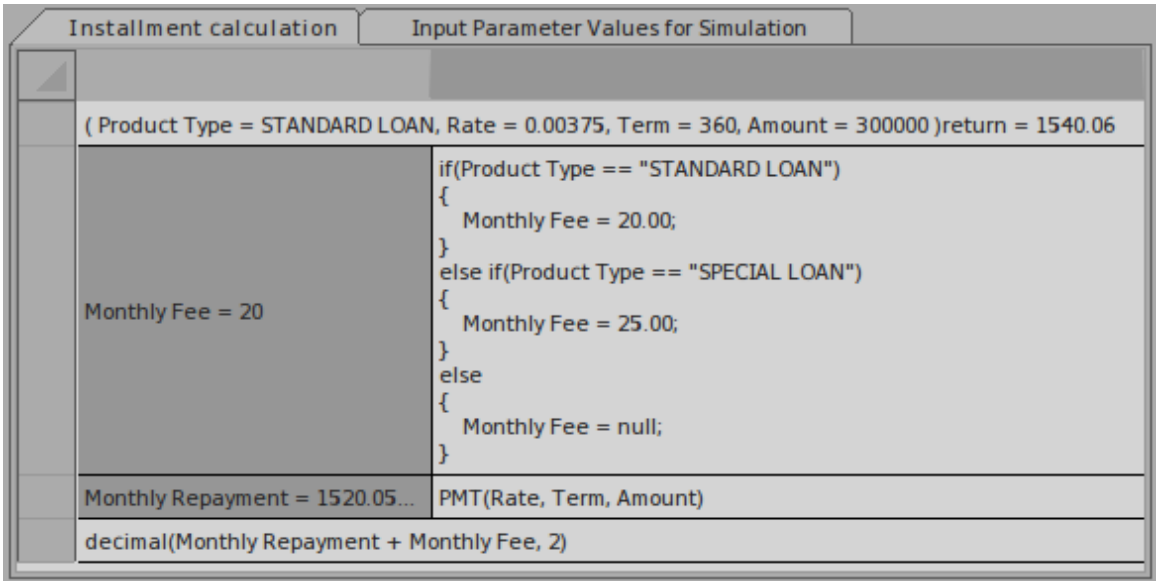
Test Harness for Business Knowledge Model

Select the 'Input Parameter Values for Simulation' tab and complete the fields.



Click on the Test Harness button on the toolbar; the test

result will be presented in the Boxed Context.



- The runtime parameter value will be displayed; for example, 'Rate = 0.00375'
- The 'Context Entry' variable's runtime value will be displayed; for example, 'Monthly Repayment = 1520.05'
- The BKM's result will be evaluated by the last entry and the values displayed on the declaration line; for example, 'return = 1540.06'

You can use this functionality to unit test a BusinessKnowledgeModel without knowing the context and later on invoked by a Decision or other BusinessKnowledgeModel.

Menu options are available for this tool bar button. For more information, see the *Business Knowledge Model and Test Harness* Help topic.

Invocation

An invocation is a container for the parameter bindings that provide the context for the evaluation of the body of a business knowledge model. There are two common use cases for an Invocation:

- Bind Input Data to Business Knowledge Model
- Bind parameters or context entry variables to Business Knowledge Model

An example of each is provided at the end of this page.

Access

Ribbon	Simulate > Decision Analysis > DMN > DMN Expression : select or create a Decision or BusinessKnowledgeModel with Invocation type
Other	Double-click on a DMN Decision or BusinessKnowledgeModel

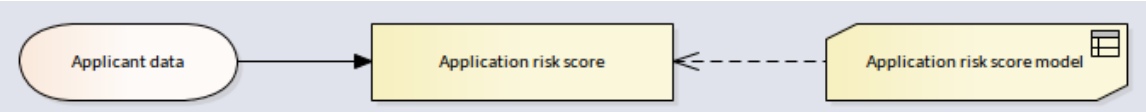
Toolbar Options

Options	Description
	Click on this button to save the configuration to the current Decision or BusinessKnowledgeModel.
	Click on this button to edit parameters for the Business Knowledge Model.
	Click on this button to synchronize with the invoked Business Knowledge Model. For example, if the Business Knowledge Model changes name, parameters, outputs or types, click on this button to synchronize these changes.
	Click on this button to set or change a Business Knowledge Model as an invocation.
	Click on this button to open the invoked Business Knowledge Model in the DMN Expression window.
	When a Business Knowledge Model is implemented as a Decision table, it could define multiple output clauses; the invocation on this Business Knowledge Model might have to specify which

	output is requested. Click on this button to list all the available outputs in a context menu; the currently configured output is checked.
	Click on this button to validate the Invocation. Enterprise Architect will perform a series of validations to help you locate errors in the Expression.
	This button is enabled when the Invocation is defined for a BusinessKnowledgeModel. Select the 'Input Parameter Values for Simulation' tab, complete the fields and click on this button. The test result will be presented on the Decision table, with the runtime values of inputs and outputs displayed and valid rule(s) highlighted. You can use this functionality to unit test a BusinessKnowledgeModel without knowing the context and later on invoked by a Decision or other BusinessKnowledgeModel. Menu options are available for this toolbar button. For more information, see the *BusinessKnowledgeModel and Test Harness* Help topic.

Example 1 — Bind Input Data to Business Knowledge Model

A full example can be created with a Model Pattern (Ribbon: Simulate > Decision Analysis > DMN > Apply Perspective > DMN Decision > Decision With BKM : Create Pattern(s))



In this example, Input Data *Applicant Data* is typed to *Applicant data Definition,* which has three components.

Applicant data : Applicant data Definition

Applicant data Definition	Marital Status : string	"M"
	Employment Status : string	"EMPLOYED"
	Age : number	40

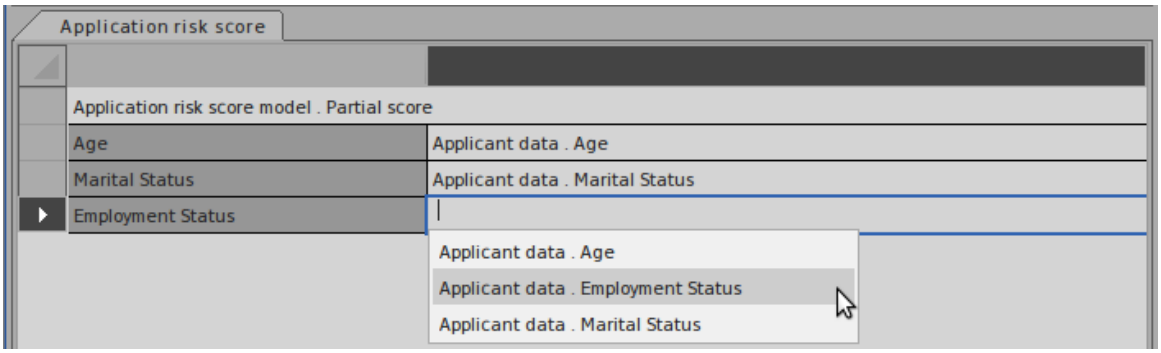
The Business Knowledge Model *Application risk score model* is implemented as a Decision table with three inputs and one output.

Application risk score model | Input Parameter Values for Simulation

(Age, Marital Status, Employment Status)

C+	Age	Marital Status	Employment Status	Partial score
	[18..120]	S,M	UNEMPLOYED,EMPLOYE...	
1	[18..21]	-	-	32
2	[22..25]	-	-	35
3	[26..35]	-	-	40
4	[36..49]	-	-	43
5	>=50	-	-	48
6	-	S	-	25
7		M	-	45
8		-	UNEMPLOYED	15
9			STUDENT	18
10			EMPLOYED	45
11			SELF-EMPLOYED	36

The Decision *Application risk score* is implemented as an Invocation to bind the Input Data's 'leaf' components to the BKM's parameters.



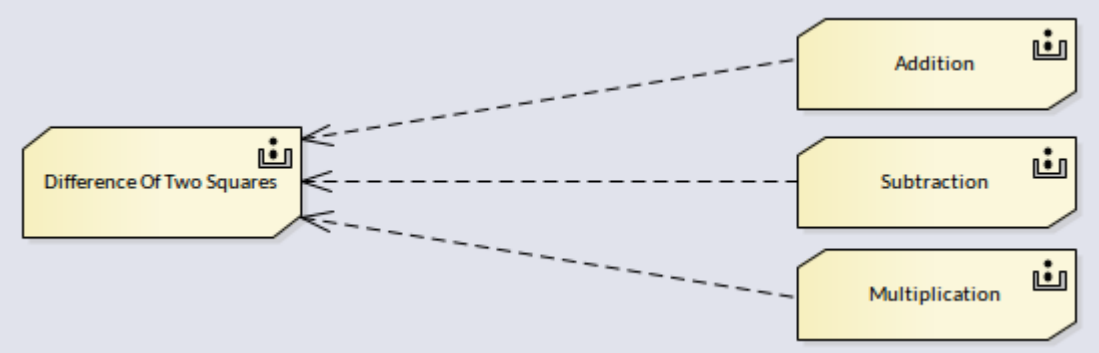
In order to make the binding easier, Auto-Completion is supported for the binding expression.

The full modeling and simulation instructions are available in the Pattern's documentation.

Example 2 — Bind Context Entry variables to

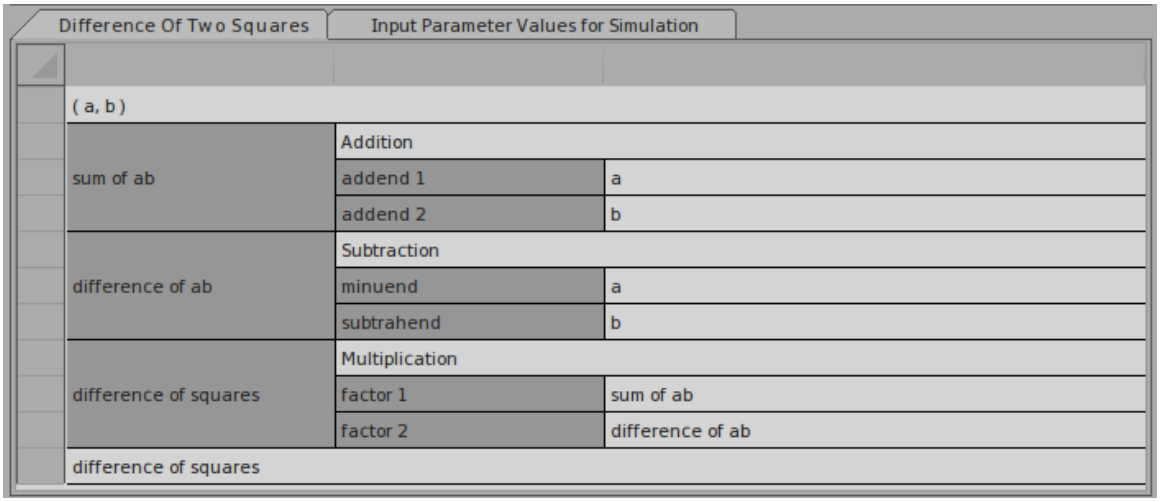
Business Knowledge Model

The full example can be created using a Model Pattern (Access - Ribbon: Simulate > Decision Analysis > DMN > Apply Perspective > DMN Business Knowledge Model > Business Knowledge Model Invocation : Create Pattern).



In this example, the BKM *Difference Of Two Squares* is implemented as Boxed Context:

- The variable *sum of ab* is implemented as an invocation by binding parameters *a* and *b* to BKM *Addition*
- The variable *difference of ab* is implemented as an invocation by binding parameters *a* and *b* to BKM *Subtraction*
- The variable *difference of squares* is implemented as an invocation by binding local variables *sum of ab* and *difference of ab* to BKM *Multiplication*



In order to make the binding easier, auto-completion is supported for the binding expression.

The full modeling and simulation instructions are available in the Pattern's document.

Item Definition

An important characteristic of data items in Decision Models is their structure.

The ItemDefinition element is used to model the structure and the range of values of the input and the outcome of decisions.

Access

Ribbon	Simulate > Decision Analysis > DMN > DMN Expression, then select or create an ItemDefinition
Other	Double-click on a DMN ItemDefinition

Toolbar Options

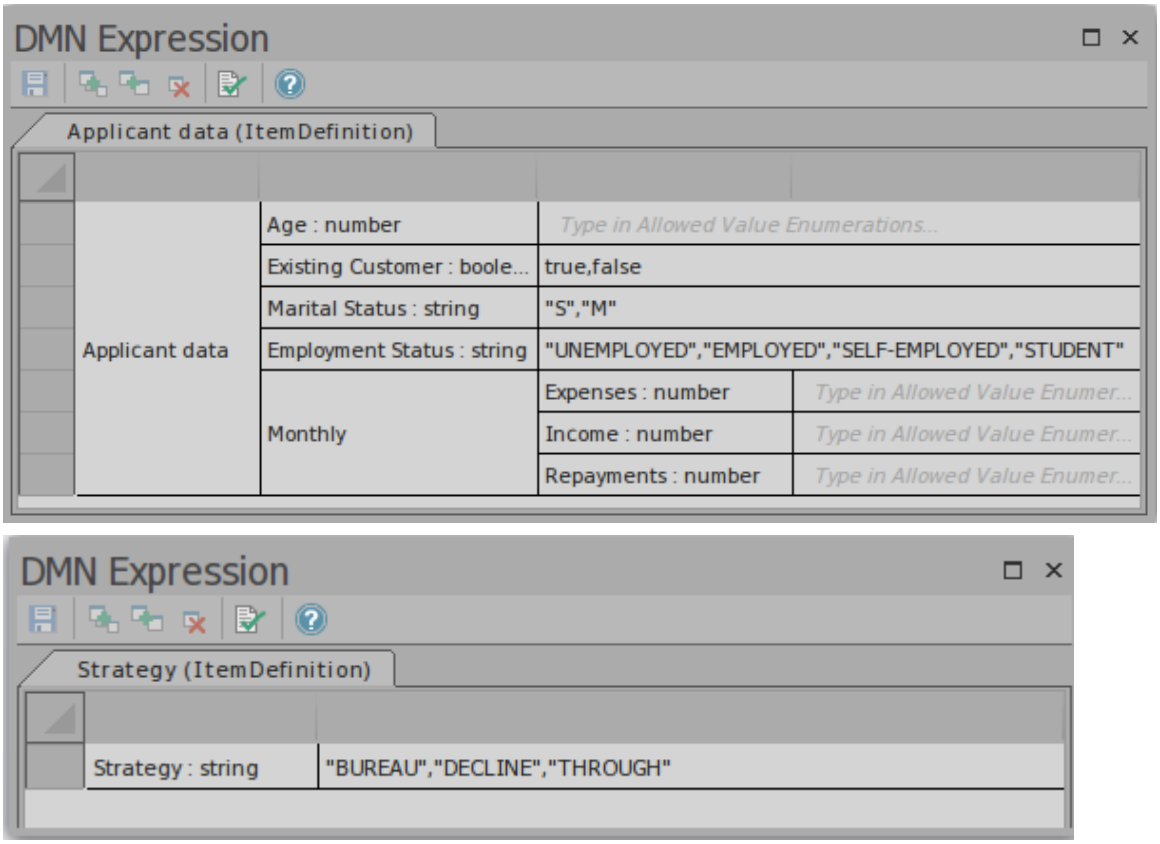
Option	Description
	Click on this button to save the configuration to the current ItemDefinition.

	Click on this button to add a child component to the selected ItemDefinition.
	Click on this button to add a sibling component to the selected ItemDefinition.
	Click on this button to delete the selected component.
	Click on this button to validate the ItemDefinition; Enterprise Architect will perform a series of validations to help you identify errors in the ItemDefinition.

Allowed Value Enumerations

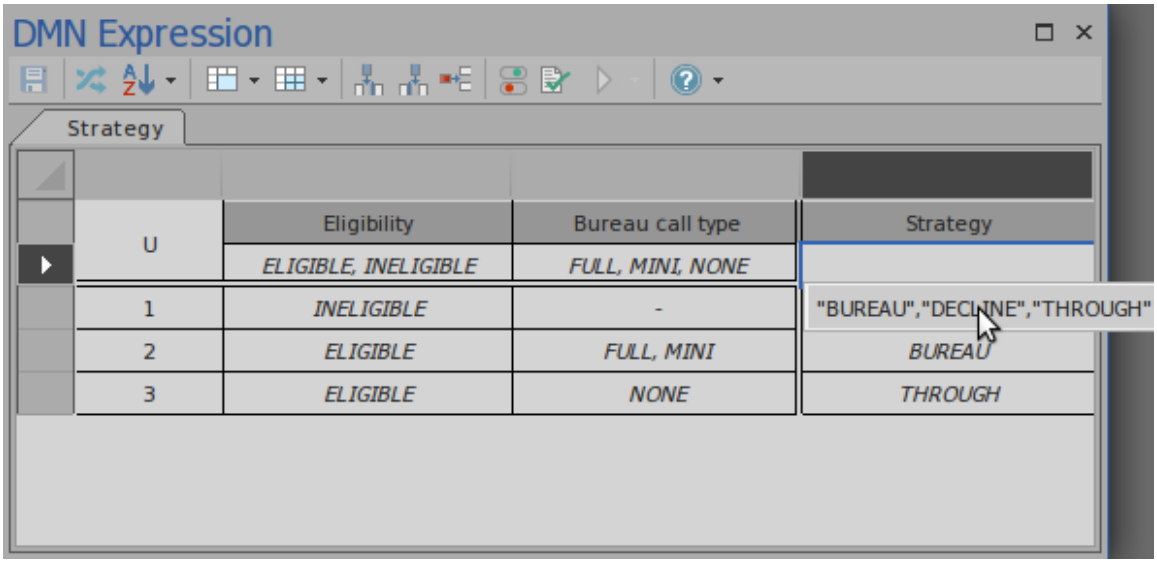
Each 'Leaf' component of the ItemDefinition can define a string of Allowed Value Enumerations.

For example:

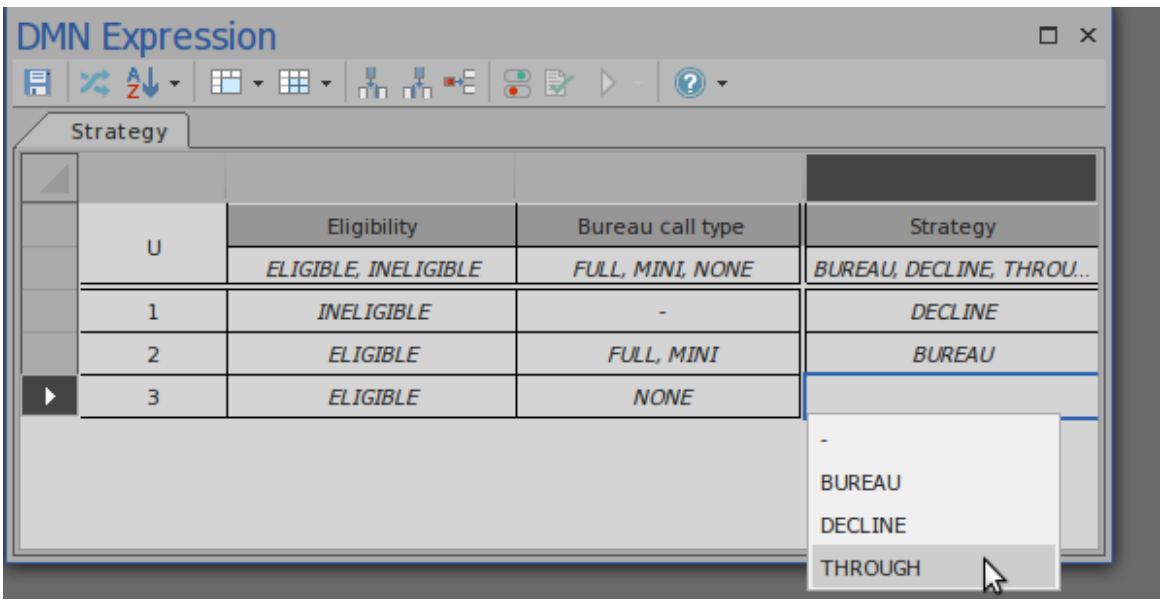


The Allowed Value Enumerations play an important role in Auto Completion. The idea is that you type these values once, and choose from a list later on.

Taking the 'Strategy' ItemDefinition as an example, we can fast fill the 'Allowed Values' field for a Decision table by selection:



Then fast fill the Decision table rules by selection:



Types of Component

An ItemDefinition element SHALL be defined using only one of these alternative methods:

- Set to a built-in type
- Composition of ItemDefinition elements

In other words, if an ItemDefinition is 'leaf' (no child components), it must be set to a built-in type; if an ItemDefinition has child components, it can not be set a built-in type.

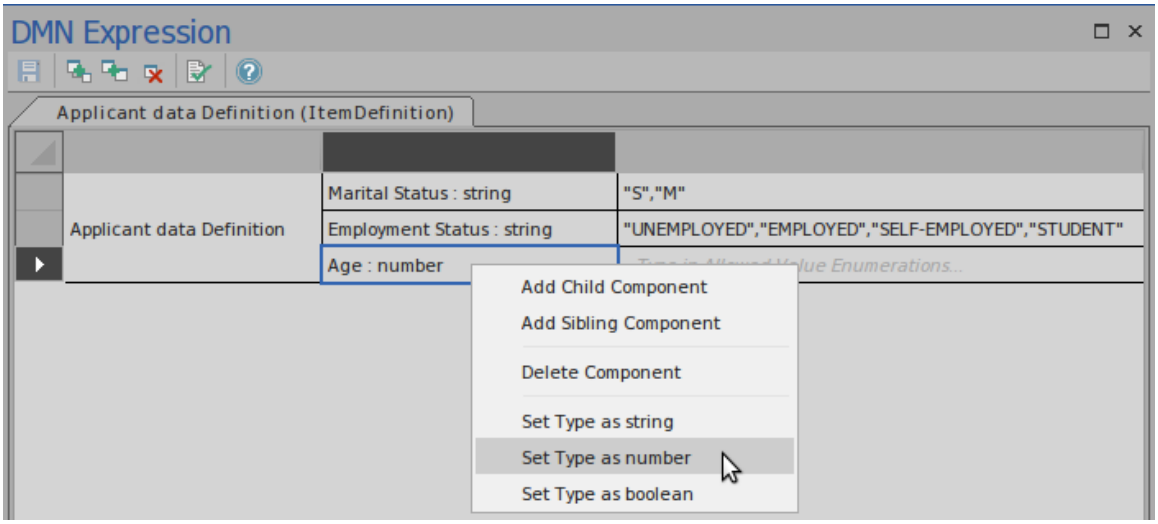
The FEEL language has these built-in types:

- **number**
- **string**
- **boolean**
- days and time duration
- years and months duration
- time
- date and time

Note: 'number', 'string' and 'boolean' are supported by Enterprise Architect for simulation.

In order to set a type for a 'leaf' ItemDefinition, you can use one of these three methods:

- Use the context menu (Recommended)
- Type in the cell after the name by appending ': string', ': boolean' or ': number'
- Input 'string', 'boolean' or 'number' in the tag 'type' for the ItemDefinition


DMN Expression
Applicant data Definition (ItemDefinition)
Applicant data Definition
Marital Status : string
"S","M"
Employment Status : string
"UNEMPLOYED","EMPLOYED","SELF-EMPLOYED","STUDENT"
Age : number
Add Child Component
Add Sibling Component
Delete Component
Set Type as string
Set Type as number
Set Type as boolean

Input Data

An InputData types to an ItemDefinition and carries values used for Decisions.

Access

Ribbon	Simulate > Decision Analysis > DMN > DMN Expression, then select / create an InputData
Other	Double-click on a DMN InputData

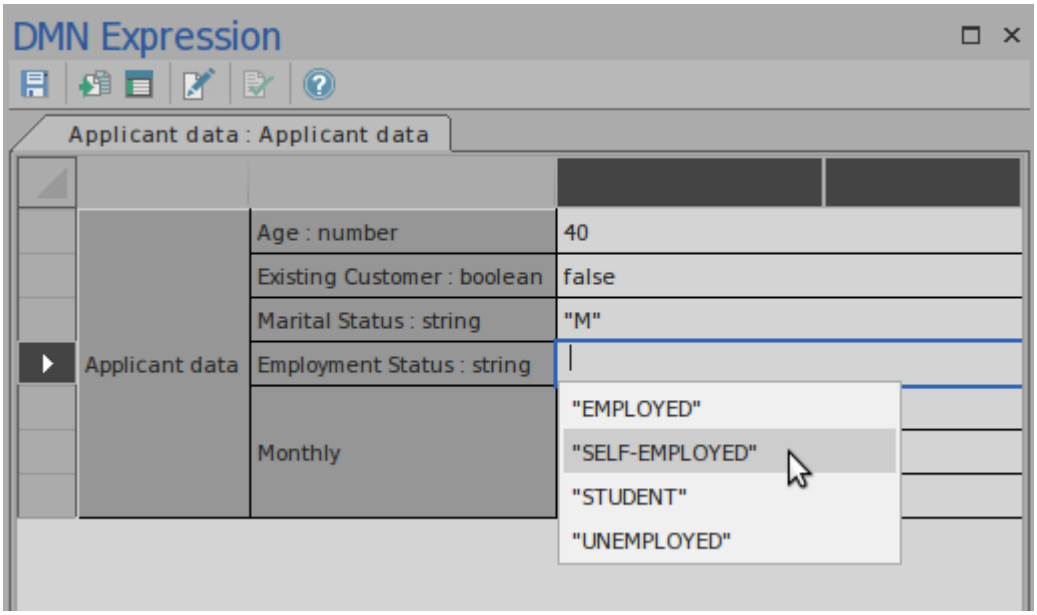
Toolbar Options

Option	Description
	Click on this button to save the configuration to the current InputData.
	Click on this button to select an ItemDefinition as the type of this InputData.

	Click on this button to open the ItemDefinition element that types this InputData.
	Click on this button to open the dialog for editing data sets for this input data. Each InputData can define multiple data sets. With this feature, the DMN Simulation can quickly test the results of a decision by choosing different data sets.
	Click on this button to validate the InputData. Enterprise Architect will perform a series of validations to help you identify errors in the InputData.

Auto Completion

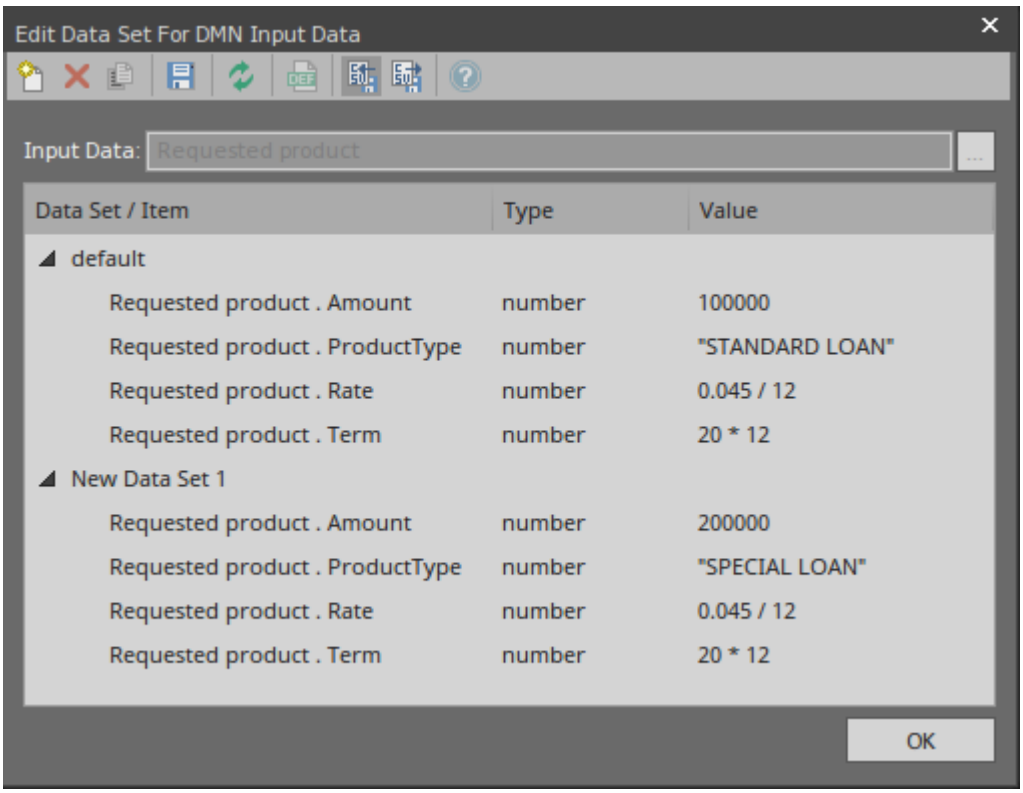
If the typing ItemDefinition has a defined 'Allowed Value', then the value for the InputData can be simply chosen from the enumerations.


DMN Expression
Applicant data : Applicant data
Age : number
40
Existing Customer : boolean
false
Marital Status : string
"M"
Applicant data
Employment Status : string
Monthly
"EMPLOYED"
"SELF-EMPLOYED"
"STUDENT"
"UNEMPLOYED"

Data Sets

Each InputData can define multiple data sets. With this feature, the DMN Simulation can quickly test the result of a decision by choosing different data sets.

- The values in the 'default' data set will show in the DMN Expression window when the InputData is selected
- You can add, duplicate or delete a dataset and set an existing dataset as default
- You can export the datasets to a CSV file and import them from a CSV file



Access

Ribbon	Simulate > Decision Analysis > DMN > DMN Expression > click on InputData item :  icon
Other	In a diagram, double-click on the DMN InputData element :  icon.

Toolbar Options

Option	Description
	Click on this button to create a new dataset.
	Click on this button to delete the selected dataset.
	Click this button to duplicate the selected dataset.
	Click on this button to save the datasets to the InputData.
	Click on this button to reload the datasets for the InputData.

	Click on this button to set the selected dataset as default.
	Click on this button to import datasets from a CSV file.
	Click on this button to export the datasets to a CSV file.

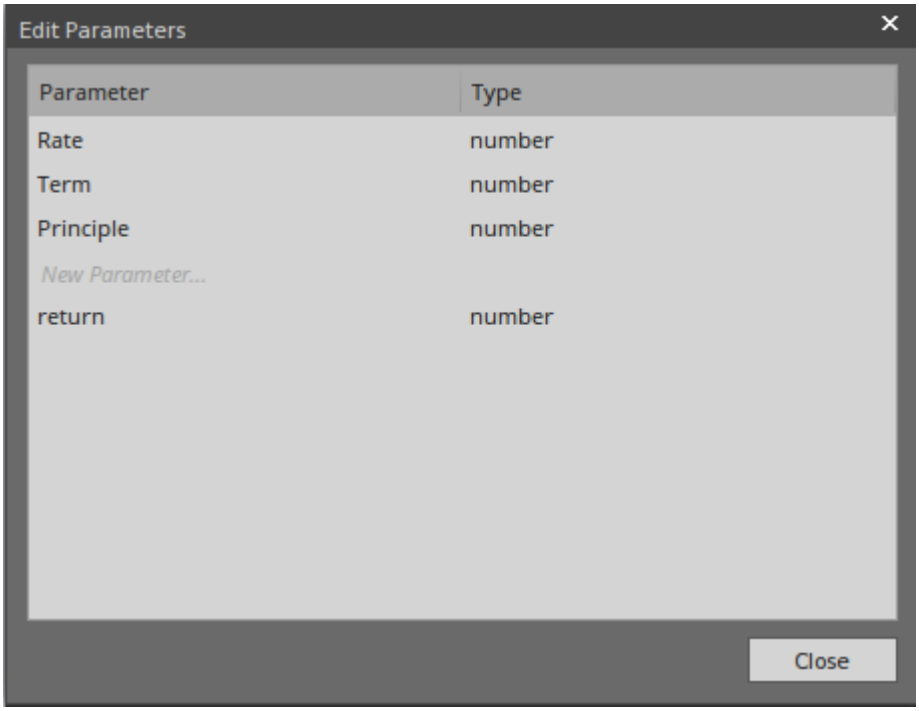
BusinessKnowledgeModel & Test Harness

A Business Knowledge Model encapsulates logic without reflecting the usage context. It is implemented as a function definition, with parameters and a DMN expression as body (such as Decision table, Boxed Context or Literal Expressions).

Because the definition of a Business Knowledge Model is self-contained, it is possible to perform 'Unit Testing' by providing arguments for the parameters. For this reason, a Business Knowledge Model will have an 'Input Parameter Values for Simulation' tab in the DMN Expression window.

Parameters of a Business Knowledge Model

In the DMN Expression window, click on the Edit Parameters button on the toolbar to open the 'Edit Parameters' dialog:



You can perform these actions on the parameters:

- Add a new parameter by typing in the 'New Parameter...' row
- Delete an existing parameter using the context menu
- Modify the name of the existing parameter by in-place editing in the cell
- Change the type of the existing parameter

Test Harness

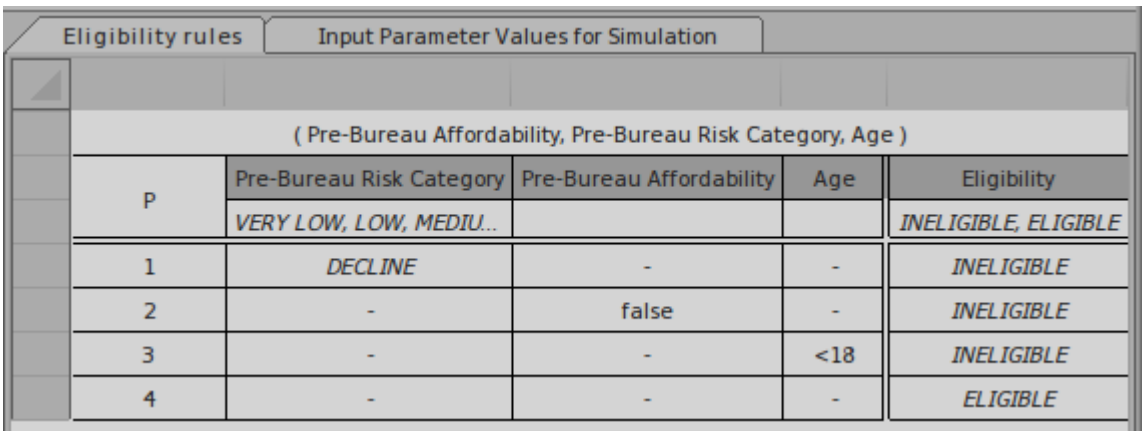
The Business Knowledge Models (BKMs) described in this section are available from the Model Wizard (Ctrl+Shift+M); select the Requirements | Decision Modeling page.

Decision Table

Example Access:

- Create pattern for 'DMN Decision | A Complete Example'
- Navigate in the Project Browser to 'A Complete Example | Business Knowledge Models | Eligibility rules'
- Double-click on the 'Eligibility rules' element to open the BKM in the DMN Expression window

When a Decision table is created for a Business Knowledge Model, we can test this BKM by binding some values:

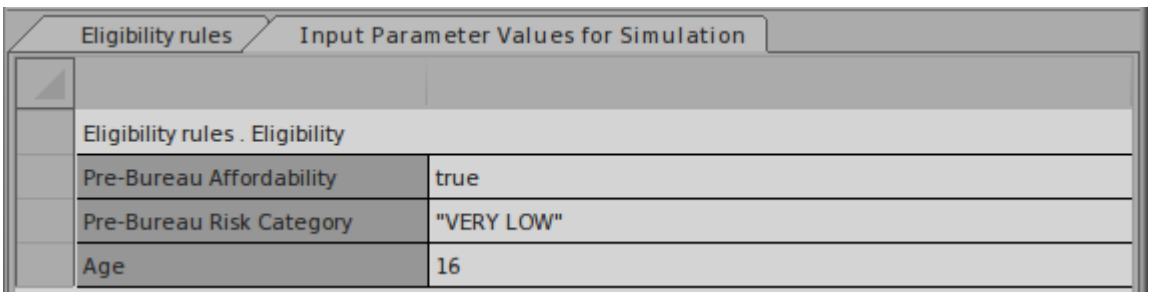


Eligibility rules | Input Parameter Values for Simulation

(Pre-Bureau Affordability, Pre-Bureau Risk Category, Age)

P	Pre-Bureau Risk Category	Pre-Bureau Affordability	Age	Eligibility
	VERY LOW, LOW, MEDIU...			*INELIGIBLE, ELIGIBLE*
1	*DECLINE*	-	-	*INELIGIBLE*
2	-	false	-	*INELIGIBLE*
3	-	-	<18	*INELIGIBLE*
4	-	-	-	*ELIGIBLE*

We can provide test values such as these:

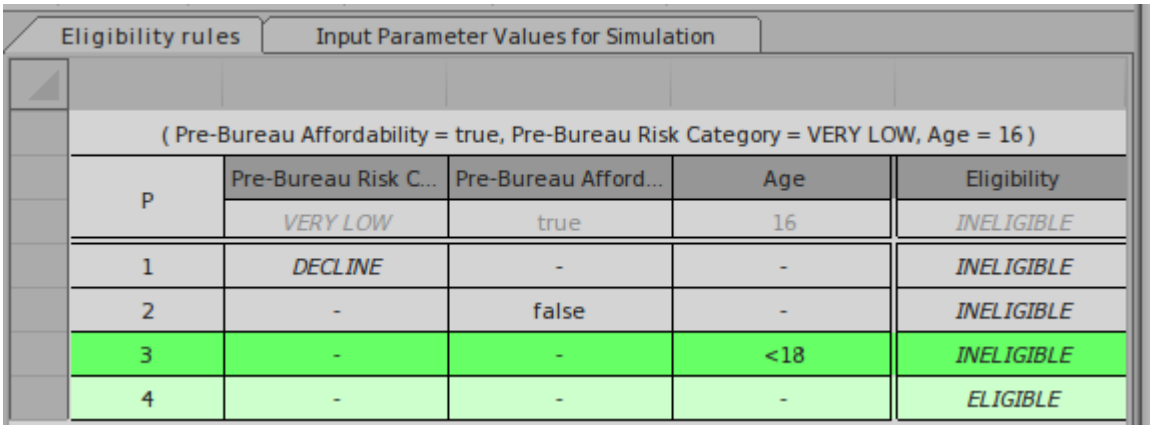


Eligibility rules | Input Parameter Values for Simulation

Eligibility rules . Eligibility

Pre-Bureau Affordability	true
Pre-Bureau Risk Category	"VERY LOW"
Age	16

Click on the Run button on the tool bar; this result is obtained:


Eligibility rules | Input Parameter Values for Simulation

(Pre-Bureau Affordability = true, Pre-Bureau Risk Category = VERY LOW, Age = 16)

P	Pre-Bureau Risk C...	Pre-Bureau Afford...	Age	Eligibility
	VERY LOW	true	16	INELIGIBLE
1	DECLINE	-	-	INELIGIBLE
2	-	false	-	INELIGIBLE
3	-	-	<18	INELIGIBLE
4	-	-	-	ELIGIBLE

- The runtime parameter value will take the place of 'Allowed Values' in simulation mode
- Valid rule(s) are highlighted
- Since this Decision table's hit policy is P (Priority) the final result is determined by the order of 'output values'; since 'INELIGIBLE' and 'ELIGIBLE' are the output values and 'INELIGIBLE' comes ahead of 'ELIGIBLE', rule #3 will give the final result and this applicant is 'INELIGIBLE'.

Boxed Context / Literal Expressions

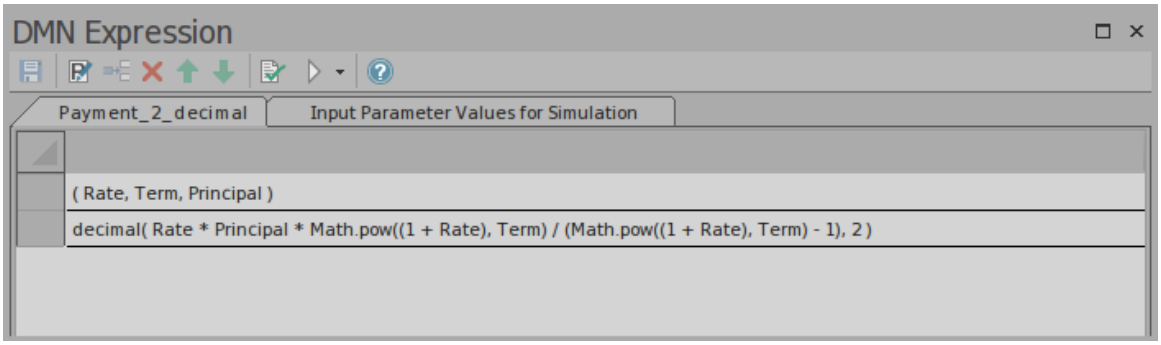
Example Access:

- Create the pattern for 'DMN Business Knowledge Model | Business Knowledge Model Literal Expression'
- Navigate in the Project Browser to 'Business Knowledge Model Literal Expression | Payment'

Double-click on the 'Payment' element to open the BKM in the DMN Expression window

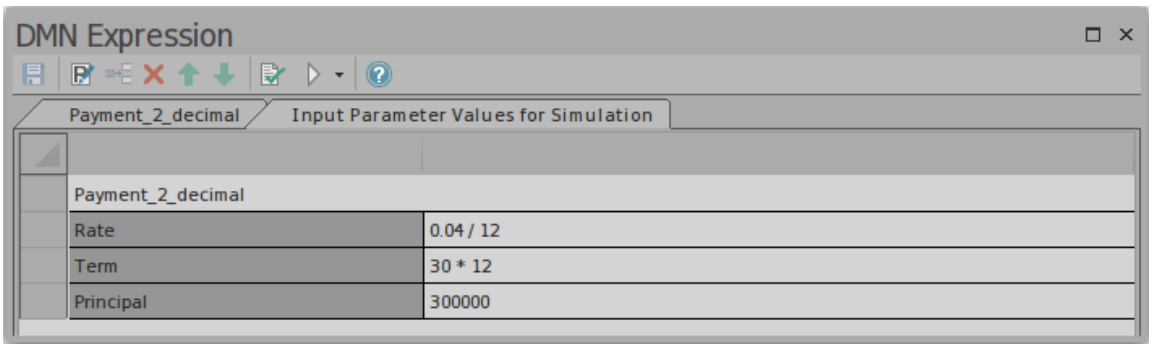
Similar to Decision table, the Business Knowledge Model implemented as a Boxed Expression can be tested as well.

Take this 'Payment_2_decimal' BKM as an example:

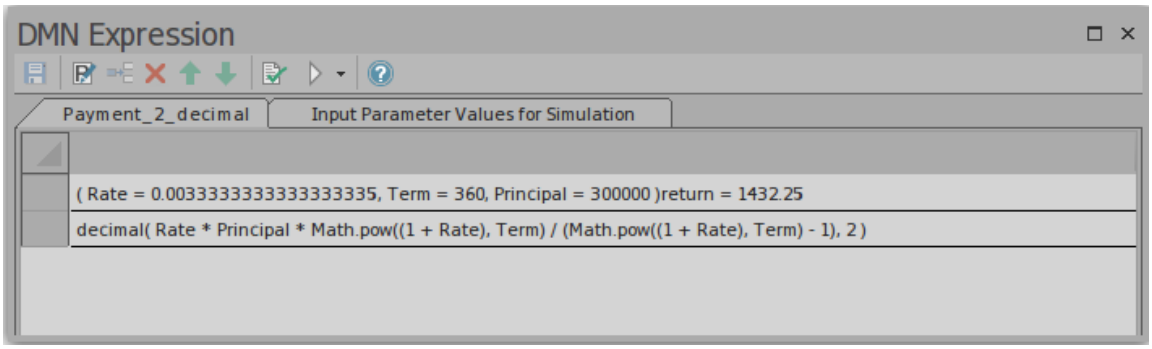


This BKM will calculate the monthly repayment based on interest rate, number of terms and principal amount.

We could provide test values such as these:



Click on the Run button on the tool bar; this result is obtained:



The runtime parameter and return value will be displayed with an equals sign '=', followed by the runtime value.

In this example, given an annual Rate of 4% for 30 years

and a principal of $300,000, the monthly repayment is $1,432.25

Note: The DMN Library already has a PMT function defined; this example mainly demonstrates how Literal Expression works and how to test it with a set of arguments.

DMN Expression Auto Completion

DMN defines many expressions, such as FunctionDefinition, DecisionTable, Boxed Context, Invocation and Literal Expression. The parameters, arguments and logic of expressions is implemented largely by 'text'.

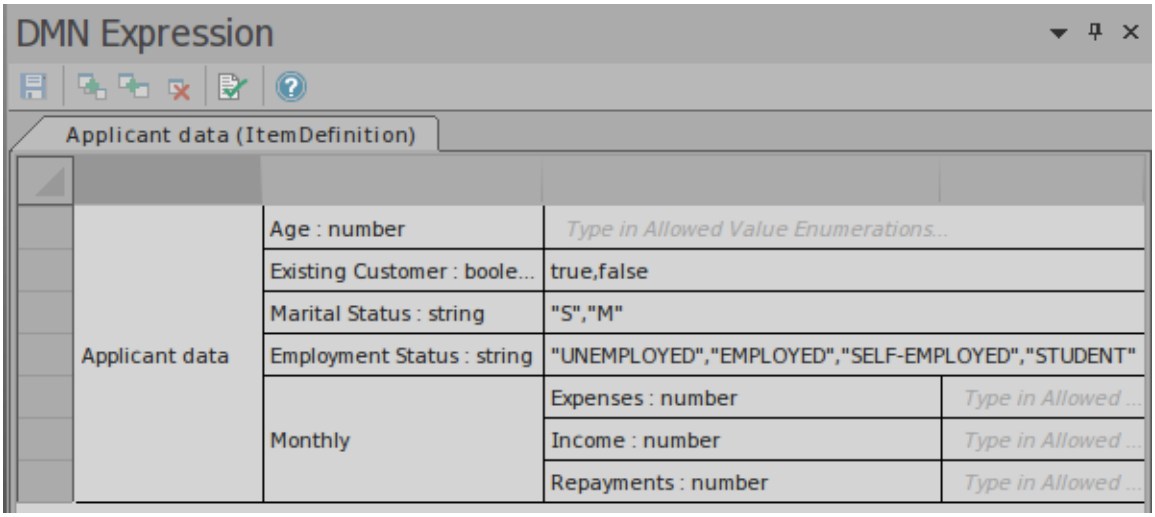
To make modeling easy and reliable, Enterprise Architect provides an Auto Completion facility, helping provide the:

- Allowed Values of ItemDefinition
- Input/Output Entries of a Decision Table
- InformationRequirement

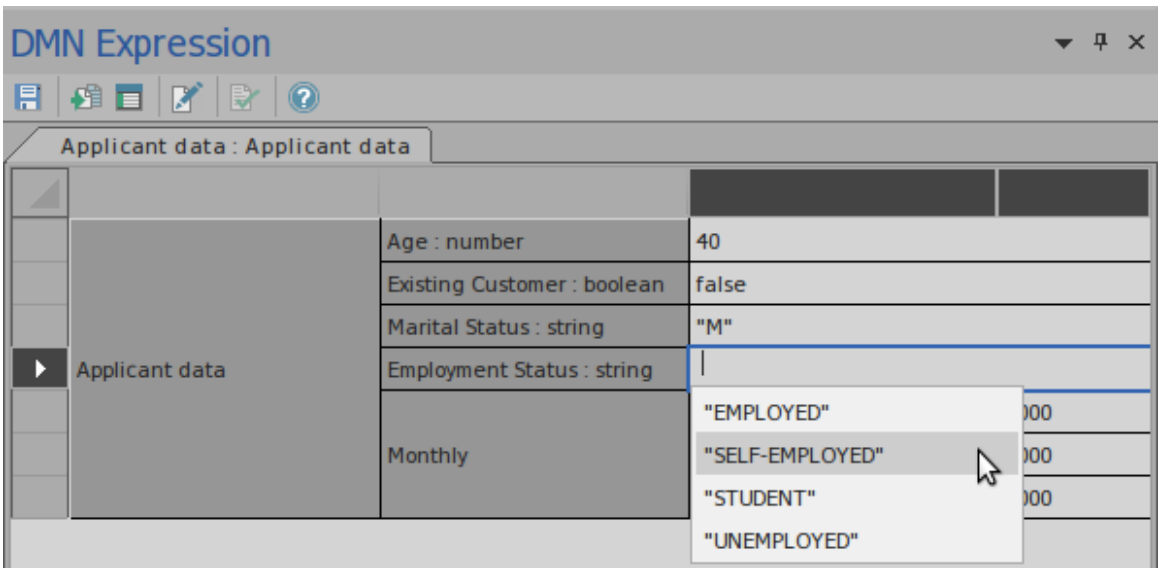
Allowed Values of ItemDefinition

The idea is to define allowed value enumerations in ItemDefinition, then compose a list for selection whenever these values are requested.

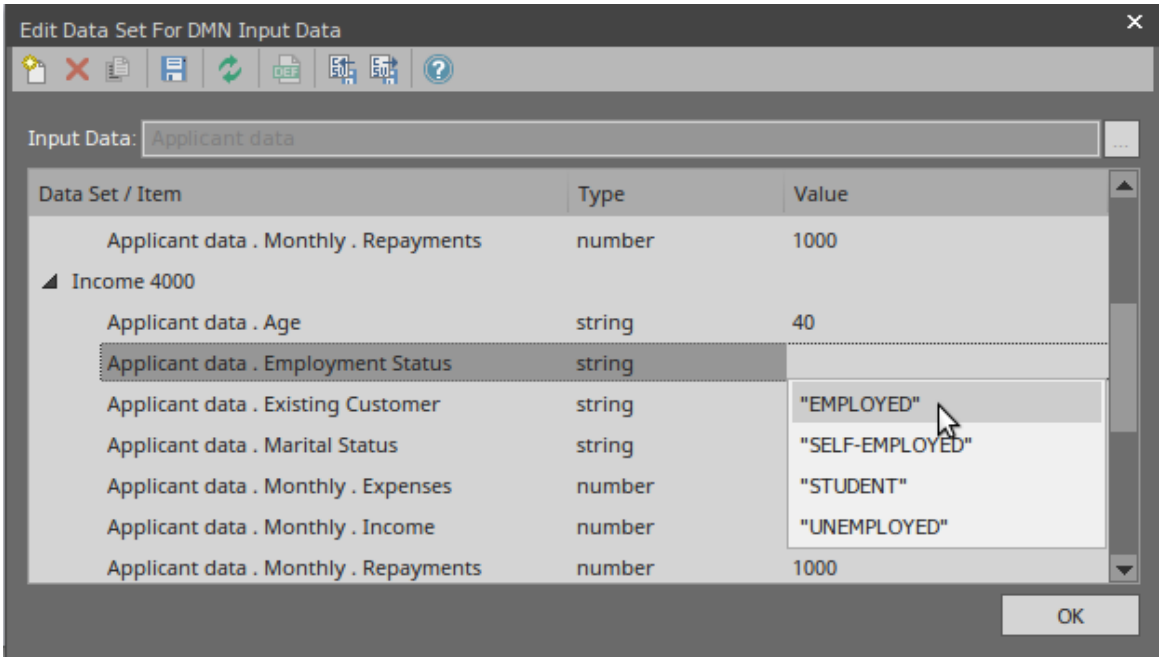
In this example, ItemDefinition 'Applicant data . Employment Status' defines an enumeration of allowed values.



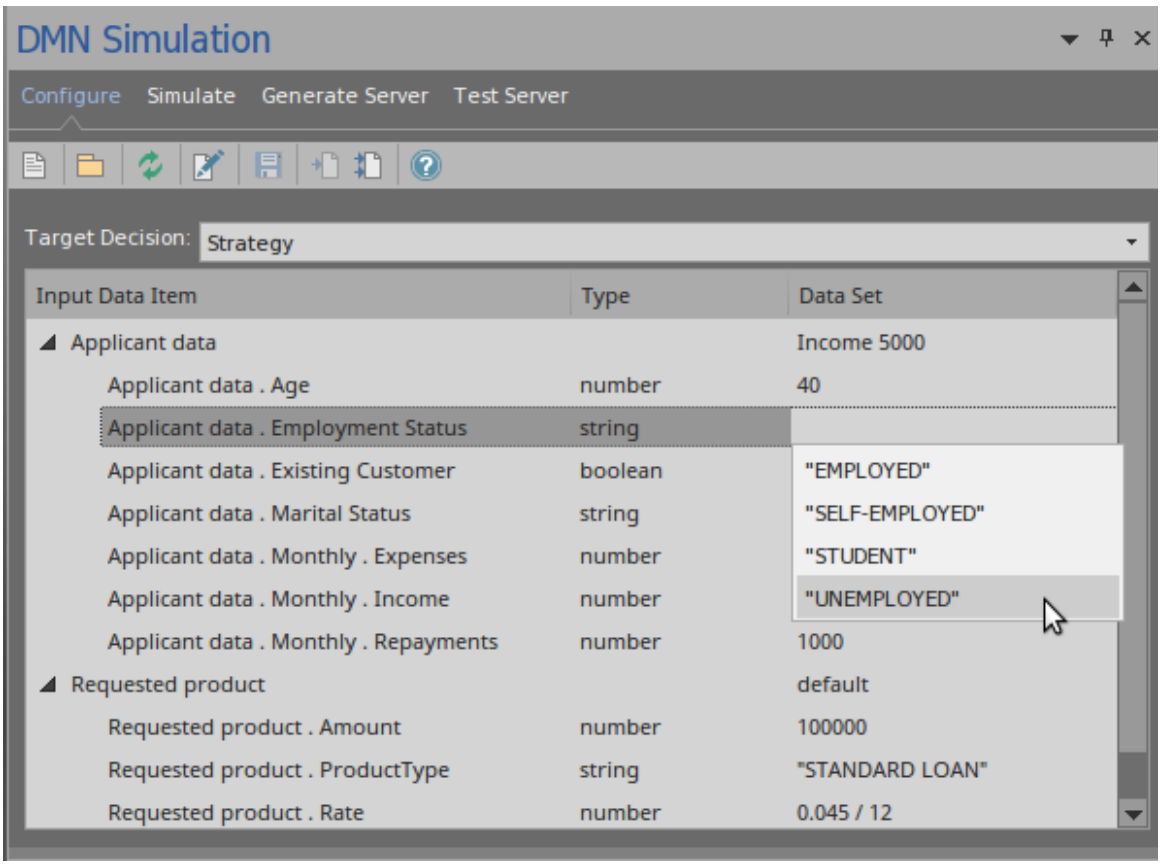
When editing values for the InputData typed to this ItemDefinition, press the Spacebar on the keyboard to display a list of values to select from.



We could also define multiple data sets for the InputData, as the Auto Completion feature is available on this dialog.

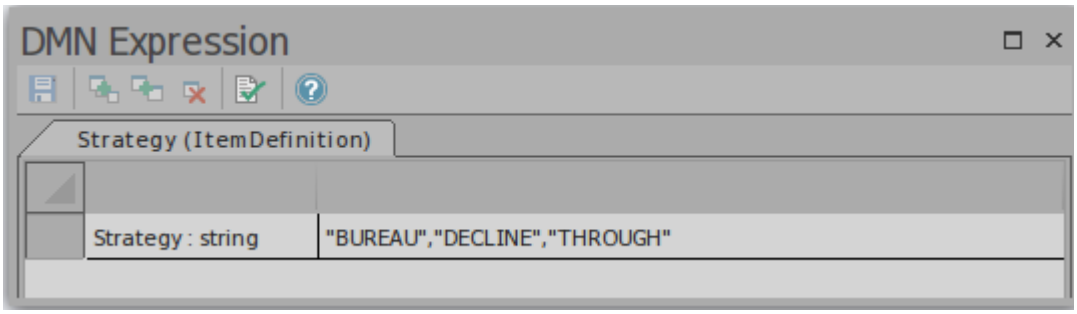


On the Simulation window, you could change the test value to simulate the model; the Auto Completion feature is available on this list as well.

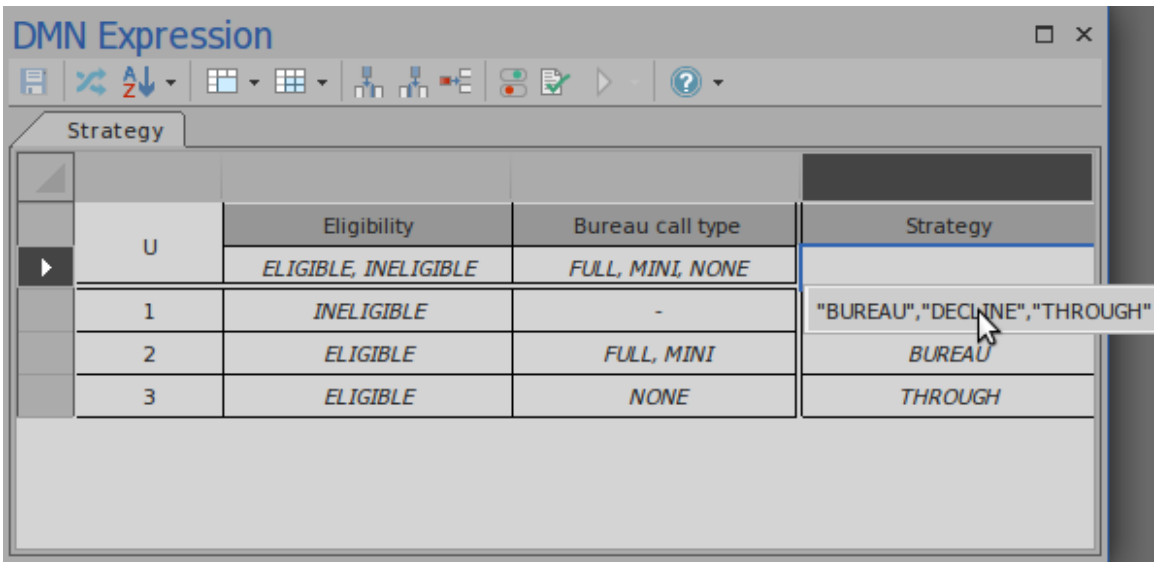


Input/Output Entries of a Decision Table

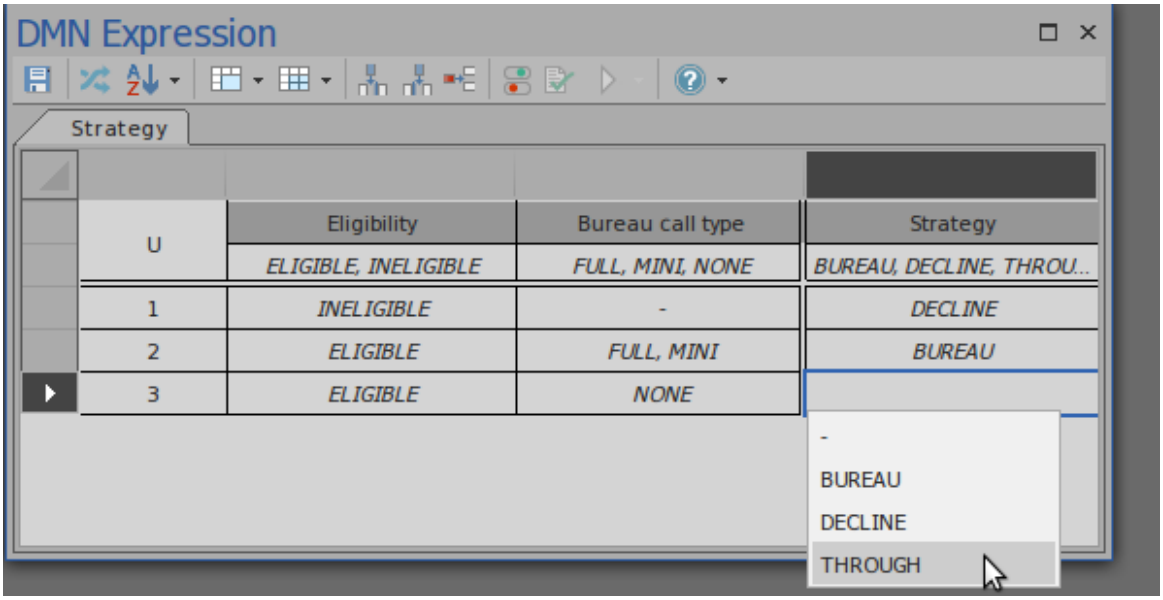
Take the 'Strategy' ItemDefinition as an example:



We can quickly fill the 'Allowed Values' field for a Decision table by selection:

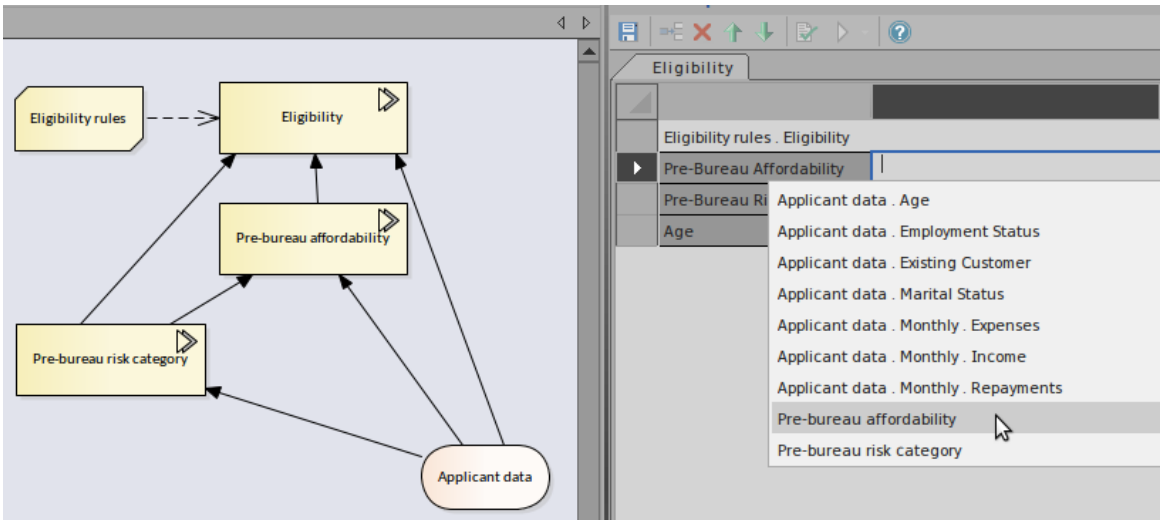


Then we can quickly fill the Decision table rules by selection:



Information Requirement

On a decision hierarchy, a decision might access required decisions and input data; these required elements form a list of variables that can be used by the decision.



In this example, Decision 'Eligibility' requires two decisions - 'Pre-bureau risk category' and 'Pre-bureau affordability' - and one Input Data item 'Applicant data'.

When setting the binding values for the invoked BusinessKnowledgeModel 'Eligibility rules', an Auto Completion list will prompt for selection.

In this list, there are sub-decision names - leaf components of the input data.

With this feature, you can easily set up an invocation.

DMN Expression Validation

DMN defines many expressions, such as FunctionDefinition, DecisionTable, Boxed Context, Invocation and Literal Expression. The parameters, arguments and logic of expressions are implemented largely by 'text'.

To make modeling easy and reliable, Enterprise Architect provides two features: Auto Completion and Validation.

- Auto Completion: You can select a text string from a list of enumerations rather than type the text in
- Validation: This identifies modeling errors caused by typos, logic incompleteness, inconsistency, and so on

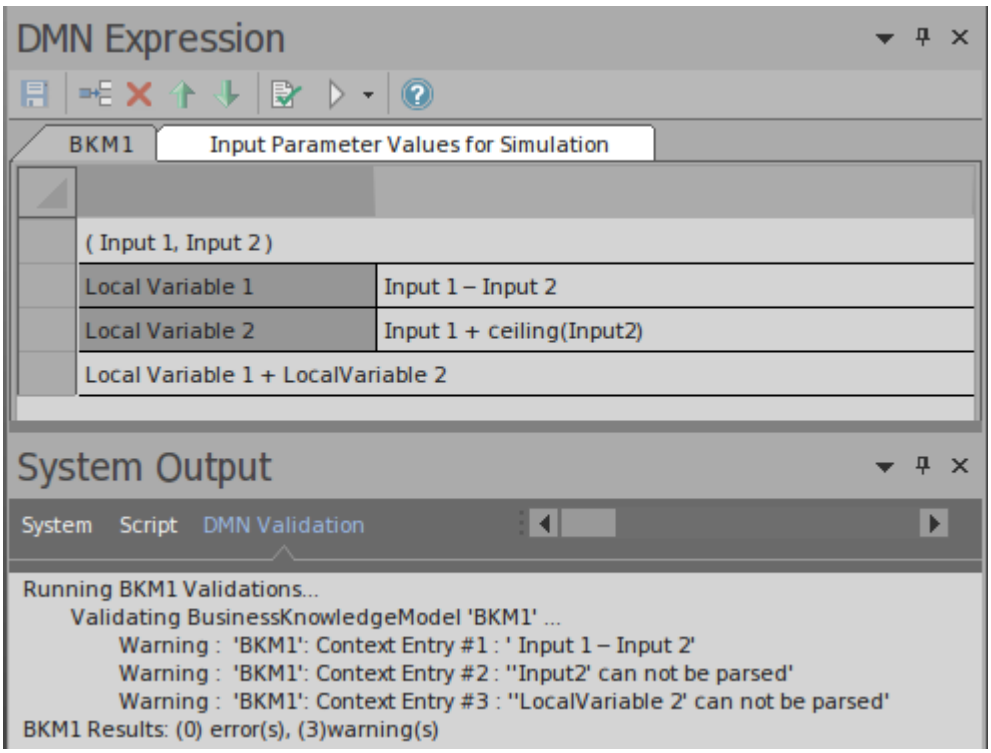
In this topic, we will show you how to validate a DMN Expression.

Access

DMN Expression Window	Simulate > Decision Analysis > DMN > DMN Expression : Validate button
DMN Simulation Window	Simulate > Decision Analysis > DMN > Open DMN Simulation > Simulate : Validate icon

Common validations

Variable Name Validation



In this example, BusinessKnowledgeModel BKM1 defines two parameters, 'Input 1' and 'Input 2', and two local variables, 'Local Variable 1' and 'Local Variable 2'.

- Context Entry #1 failed because there is a typo: it should be operator '-', but the user typed in '–' instead
- Context Entry #2 failed because there is no space between 'Input' and the number 2; note that the function ceiling is defined in DMN Library so it can be successfully parsed
- Context Entry #3 failed because there is no space between 'Local' and 'Variable'

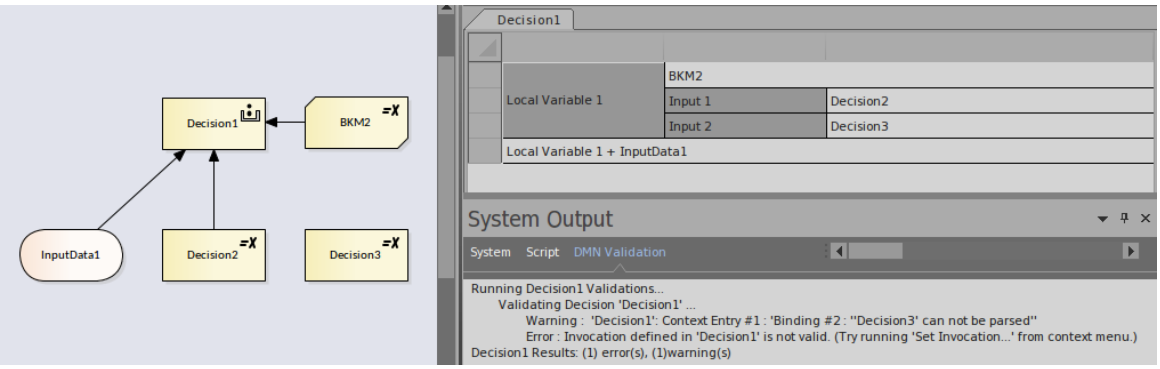
It is hard to identify these kinds of errors by eyesight,

running validation can help identify errors and then you can perform an easy fix.

Dependency Validation

A decision might require other decisions, input data and business knowledge models; these relationships are identified by informationRequirement and KnowldegeRequirement connectors.

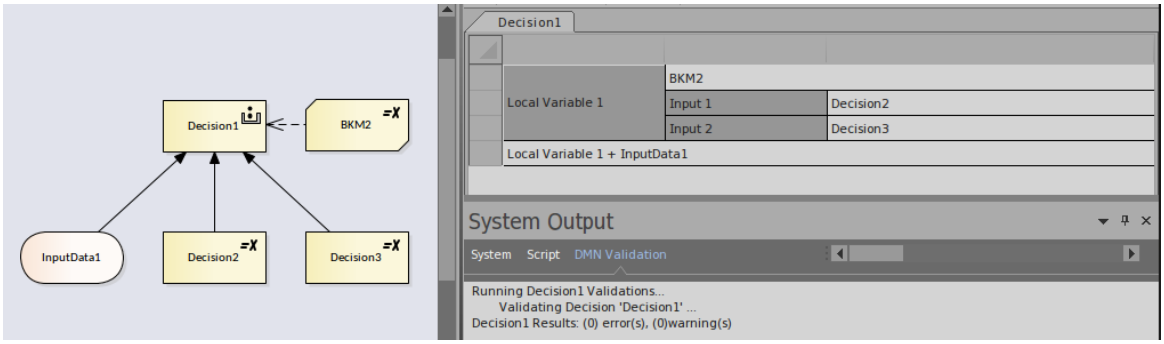
When the graph is getting complex, it is quite possible that some connectors are missing or the wrong connector type is being used.



In this example, click on the Validate button, Enterprise Architect will show that:

- 'Decision3' is used by 'Decision1' by binding to a parameter of the called BKM2; however, it is not defined - an InformationRequirement connector is missing
- The Invocation defined in 'Decision1' is not valid; the connector type from 'BKM2' to 'Decision1' should be a KnowledgeRequirement

After fixing these problems, run the validation again:


Decision1
BKM2
InputData1
Decision2
Decision3
Decision1
BKM2
Local Variable 1
Input 1
Input 2
Decision2
Decision3
Local Variable 1 + InputData1
System Output
System
Script
DMN Validation
Running Decision1 Validations...
Validating Decision 'Decision1' ...
Decision1 Results: (0) error(s), (0)warning(s)

DMN Decision Table Validation

A Decision table is one of the most common and powerful DMN Expressions to express the decision logic. However, modeling a Decision table can also be complicated, especially if multiple input clauses are used in combination for many Decision table rules. Luckily, Enterprise Architect provides a feature to validate a Decision table; this topic explains how to apply this feature.

Access

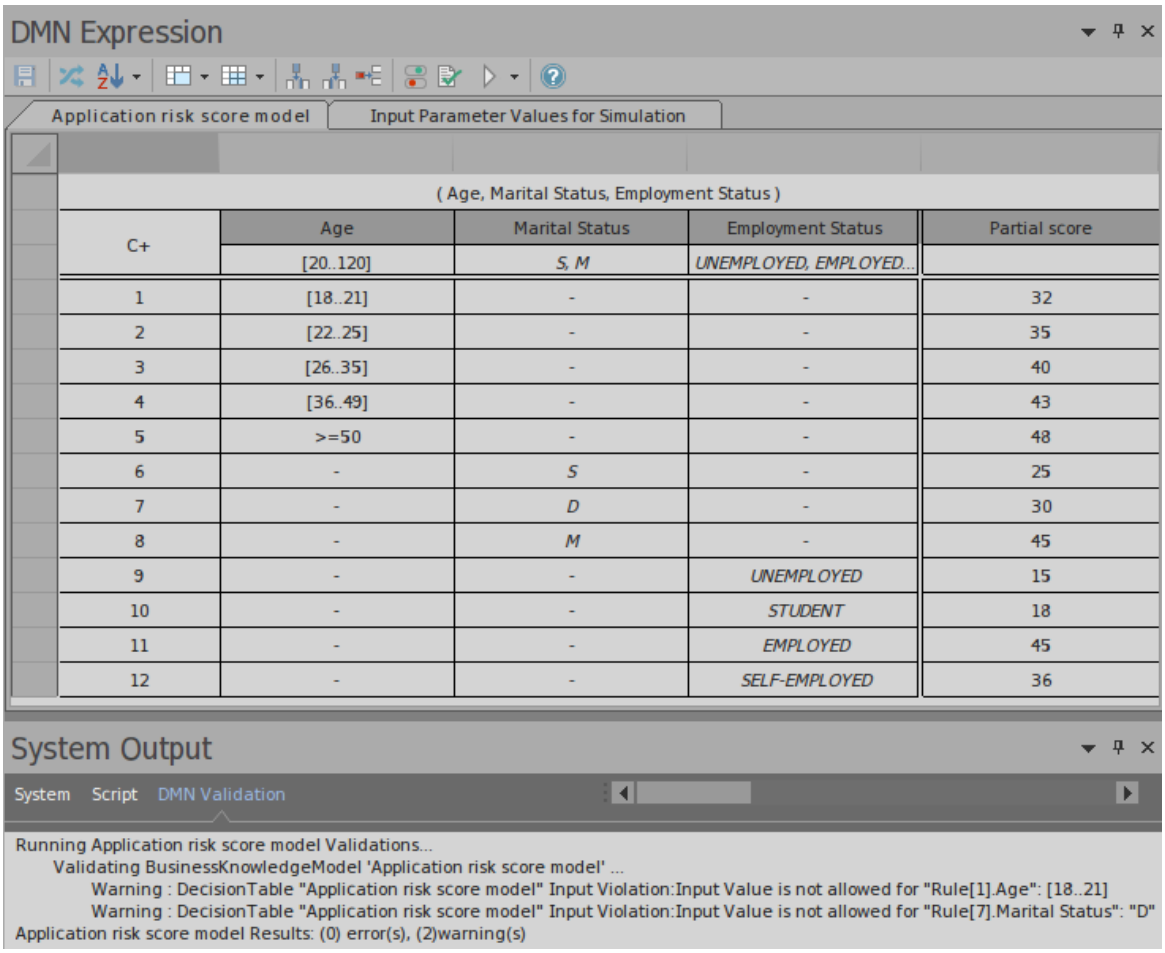
DMN Expression Window	Simulate > Decision Analysis > DMN > DMN Expression : Validate button
DMN Simulation Window	Simulate > Decision Analysis > DMN > Manage > Open DMN Simulation > Configure : Validate button

Entries out of range detection

It is good practice to define 'allowed values' for the input clause and output clause of a Decision Table.

Firstly, the enumerations (comma separated string) will enable the Auto Completion feature, where you can choose the value from a list rather than typing it in the entry field.

Secondly, the 'allowed values' enable Enterprise Architect to perform a range check for the entry values.



In this example:

- The 'Age' input clause defines a range of [20..120], causing the first rule [18..21] to fail
- The Marital Status clause defines an enumeration of 'S, M', causing rule #7 with value 'D' to fail

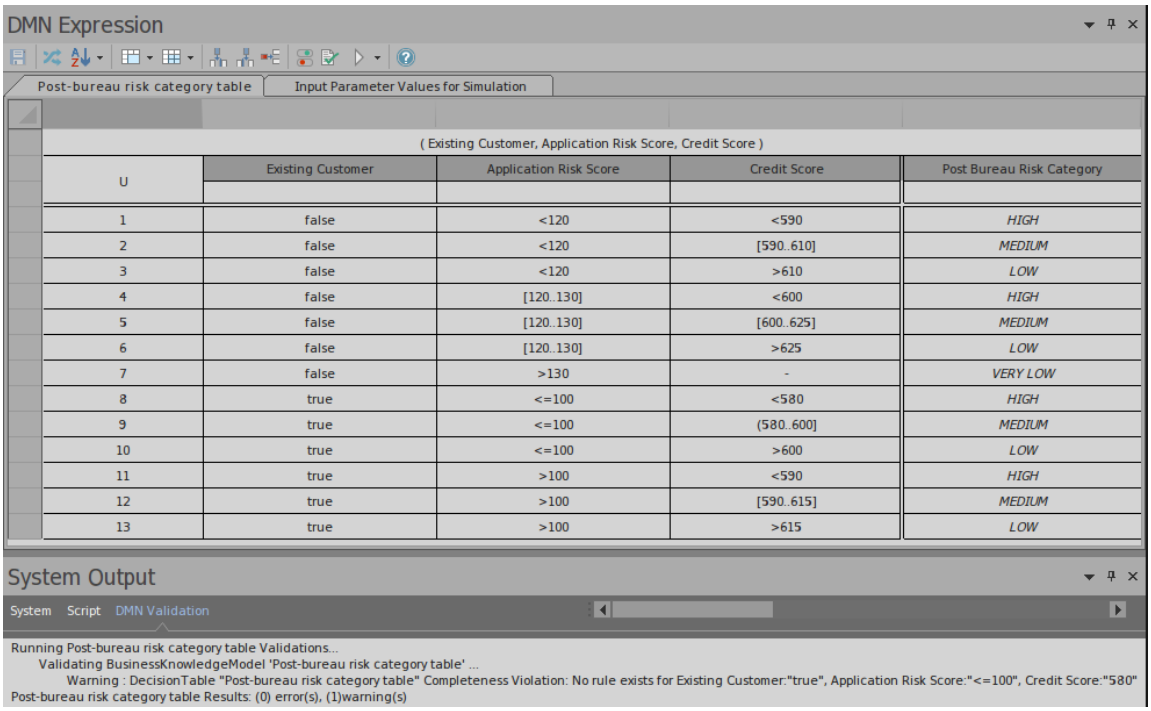
You can fix this issue by changing the 'allowed value' range or the rule entry, based on the actual business rules.

Completeness detection — report gaps in the rules

The gaps in rules for a Decision table mean that, given a combination of input values, no rule is matched. This indicates that some logic or rule might be missing (unless a default output is defined).

When the Decision table defines rules with lots of number ranges, it is hard to detect gaps by eyesight and too hard to compose test cases.

For example,



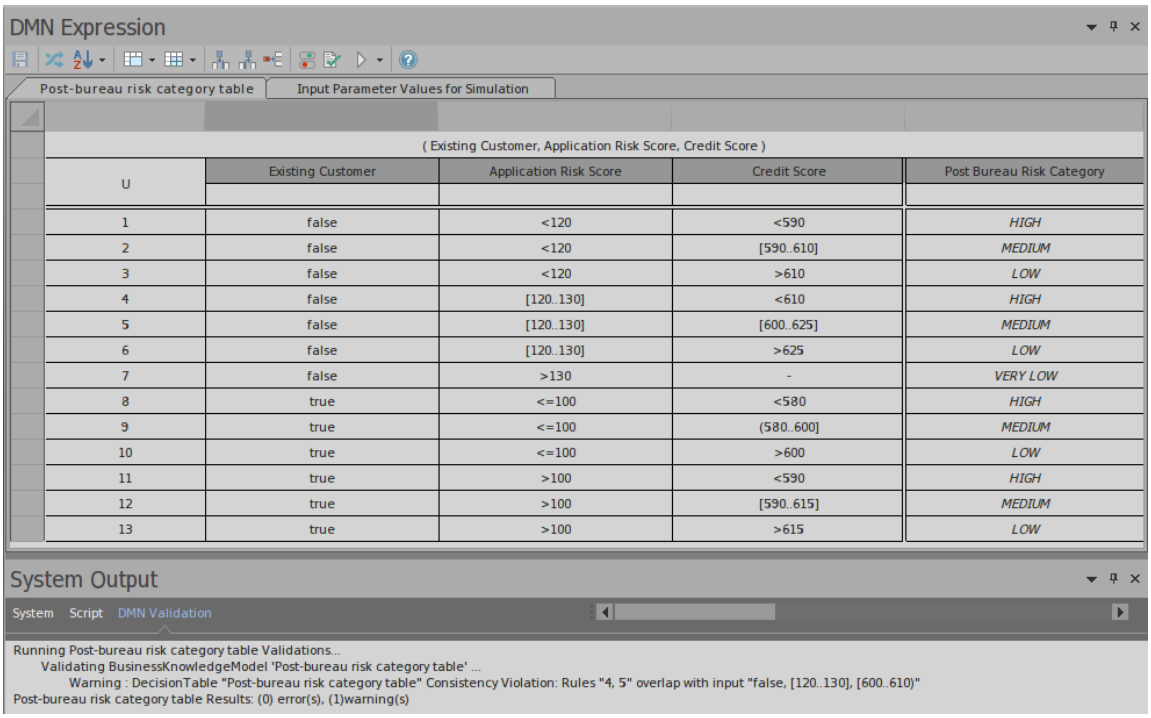
The validation reports a gap in the rules. You might first perform a merge on the Decision table to focus on the ninth input 'Credit Score' and easily detect the error in input entry (580..600], which should be [580..600].

Rule Overlaps detection for Unique Hit Policy

When rules overlap, given a combination of input values, multiple rules are matched. This is a violation if the Decision table specifies 'Unique' as its Hit Policy.

When the Decision table defines rules with lots of number ranges, it is hard to detect gaps by eyesight and too hard to compose test cases.

For example:



The validation reports an overlap in the rules. You might first perform a merge on the Decision table to focus on the 3rd input 'Credit Score' and easily detect the overlap between '<610' and '[600..625]'. You could fix this issue

either by changing rule #4 to '<600' or by changing rule #5 to '[610..625]', based on the actual business rules.

DMN Simulation

After a Decision Model is created, you can configure a DMN Simulation Artifact and Validate, Run, Step Through or Debug the model.

By switching data sets, you can do what-if analysis to ensure the model meets the requirements of the business.

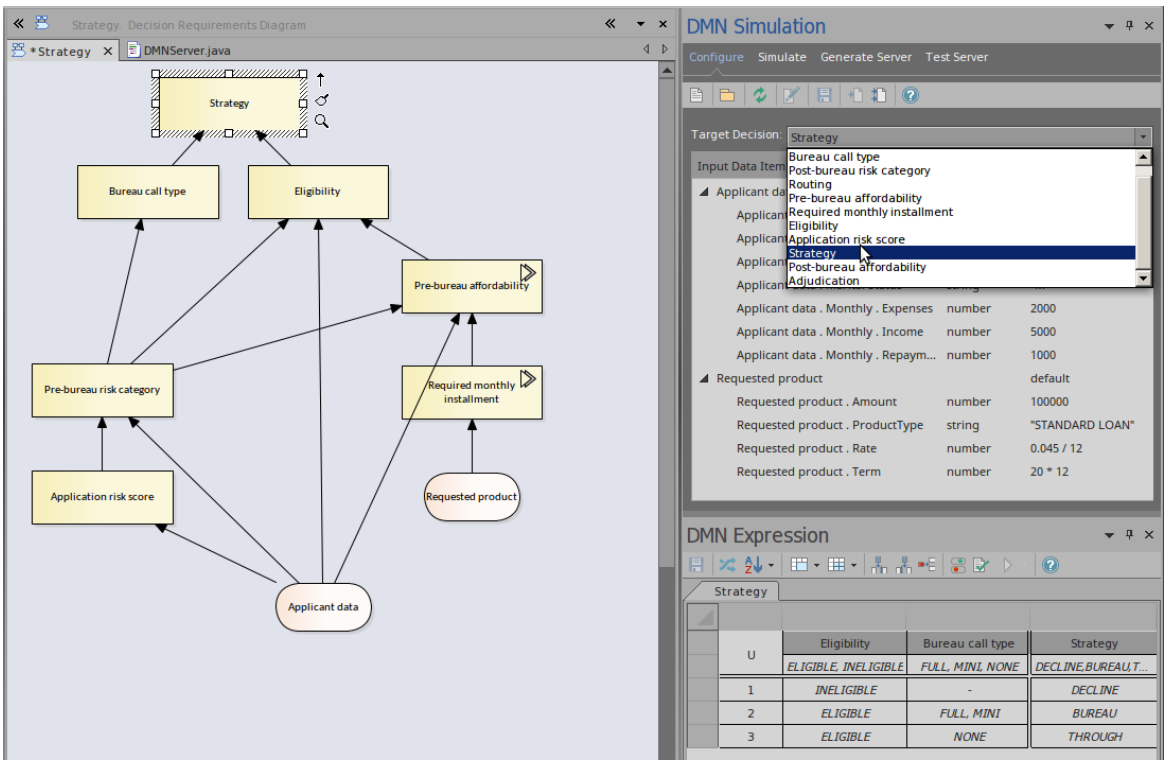
You can also generate code for the DMN Server. Currently, four languages are supported: Java, JavaScript, C++ and C#.

In order to simulate BPMN and DMN together, Enterprise Architect's BPSim Execution Engine accepts a DMN Server in Java language. Before the BPSim Execution Engine executes, ensure that the DMN Server is tested. Luckily, we can Run or Debug the testing for the Java DMN Server easily in Enterprise Architect.

Configure DMN Simulation

- Create a DMNSimConfiguration element in a Package and double-click on it to open it; all DMN elements in this Package (Decision, BusinessKnowledgeModel, InputData ItemDefinition) will be loaded to the Simulation window
- The 'Target Decision' combo box will be filled with all the decisions; choose a target decision - the dependent InputDatas will be filled in the list
- Choose a defined dataset by clicking on the Dataset cell in the list (for example, choose Dataset 'Income 5000' for

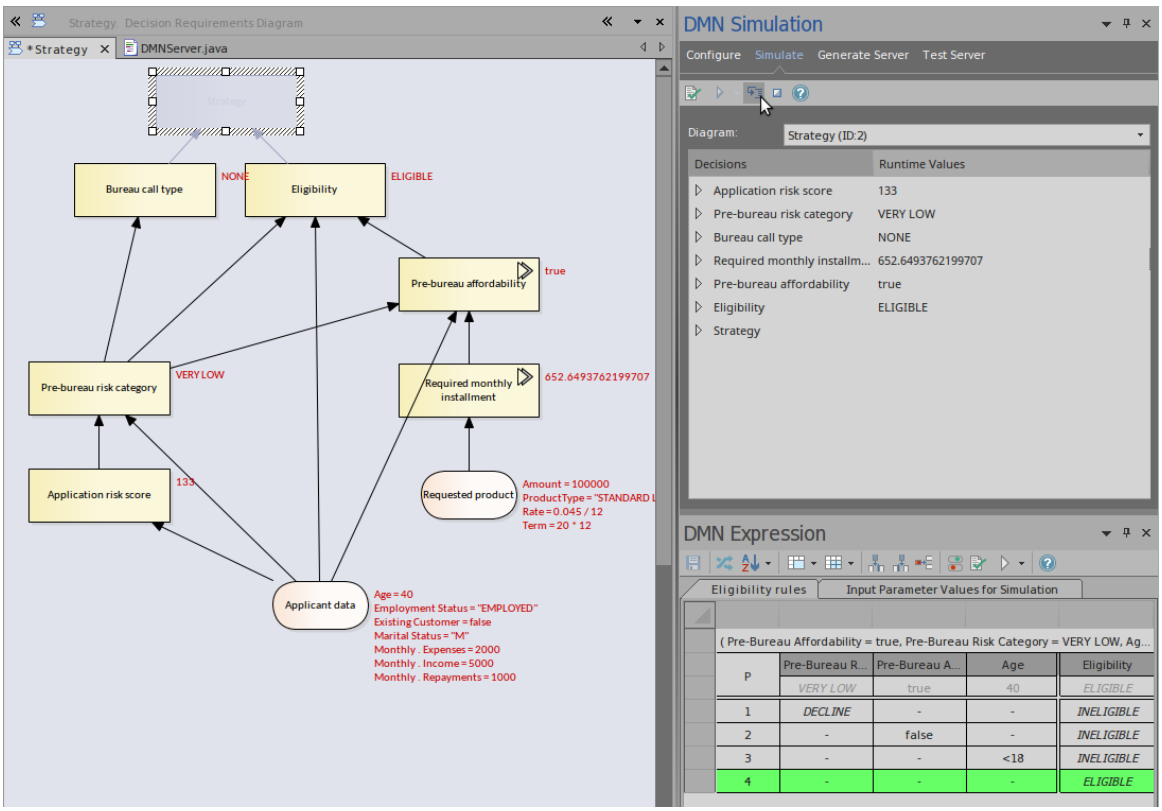
InputData 'Applicant data'; choose 'default' for InputData 'Requested product')



Simulate DMN Model

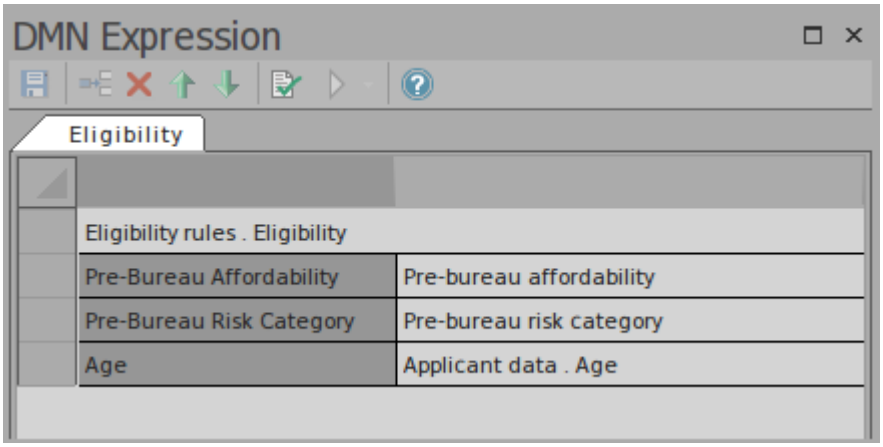
- When a Target Decision is specified, the 'Simulation' page will be filled with the decisions, in dependency order
- Each decision row can be expanded to show the invoked BusinessKnowledgeModel
- Click on the Run button to evaluate all the decision values based on the values for Input Data
- Click on the Step button to evaluate a single decision and watch the DMN Expression window, which clearly shows the input value for the decision and output based on the input; the diagram containing the decision hierarchy will

highlight the executed decisions and show the runtime results on a label



In this example, the decision 'Eligibility' returns a string 'ELIGIBLE' and invokes BusinessKnowledgeModel 'Eligibility rules' by binding the parameters as shown:

- Bind 'Pre-Bureau Affordability' to the dependent decision 'Pre-bureau affordability' (runtime value: True)
- Bind 'Pre-Bureau Risk Category' to the dependent decision 'Pre-bureau risk category' (runtime value: VERY LOW)
- Bind 'Age' to the field 'Age' in dependent input data 'Applicant data' (runtime value: 40)



The BusinessKnowledgeModel 'Eligibility rules' has a Hit Policy P (Priority), meaning that multiple rules can match, but only one hit should be returned; the ordering of the list of output values is used to specify the (decreasing) priority.

In this run time case ('Pre-Bureau Affordability' = true, 'Pre-Bureau Risk Category' = VERY LOW, 'Age' = 40), only one rule with output 'ELIGIBLE' matches.

Configure DMN Simulation

A DMNSimConfiguration Artifact contains information to simulate a DMN model depicted by Decision Requirements diagrams.

Access

Ribbon	Simulate > Decision Analysis > DMN > Open DMN Simulation > Configure Page
Other	Double-click on a DMNSimConfiguration element

Toolbar Options

Option	Description
	Click on this button to select or create a DMNSimConfiguration element.
	Click on this button to set a Package for the DMNSimConfiguration Artifact. All

	DMN elements under this Package or its sub-Packages will be loaded.
	Click on this button to reload DMN elements from the configured Packages. For example, when some DMN elements are modified, run this command to reload the Package so that the changes will be taken into account for DMN Simulation.
	Click on this button to open the dialog for editing data sets for the selected input data.
	Click this button to save the DMN Simulation window information to the DMNSimConfiguration element, including the: • Target Decision • Selected Dataset for each dependent InputData
	Click on this button to export the 'name = value' records for all the InputDatas and their specified DataSets to a BPMN 2.0 DataObject (write to the Notes).
	Click on this button to export the 'name = value' records for the selected InputData

	and its specified Dataset to a BPMN 2.0 DataObject (write to the Notes). The values defined in the BPMN 2.0 DataObject will be read by the Enterprise Architect BPSim Execution Engine.

DMNSimConfiguration Artifact

You can create a DMNSimConfiguration element in one of these ways:

- In the DMN Simulation Window, click on the Artifact button on the toolbar
- On a Decision Requirements Diagram, drag the DMNSimConfiguration element from the toolbox

By default, all DMN elements in the current Package (Decision, BusinessKnowledgeModel, InputData ItemDefinition) will be loaded to the Simulation window.

Decision Hierarchy

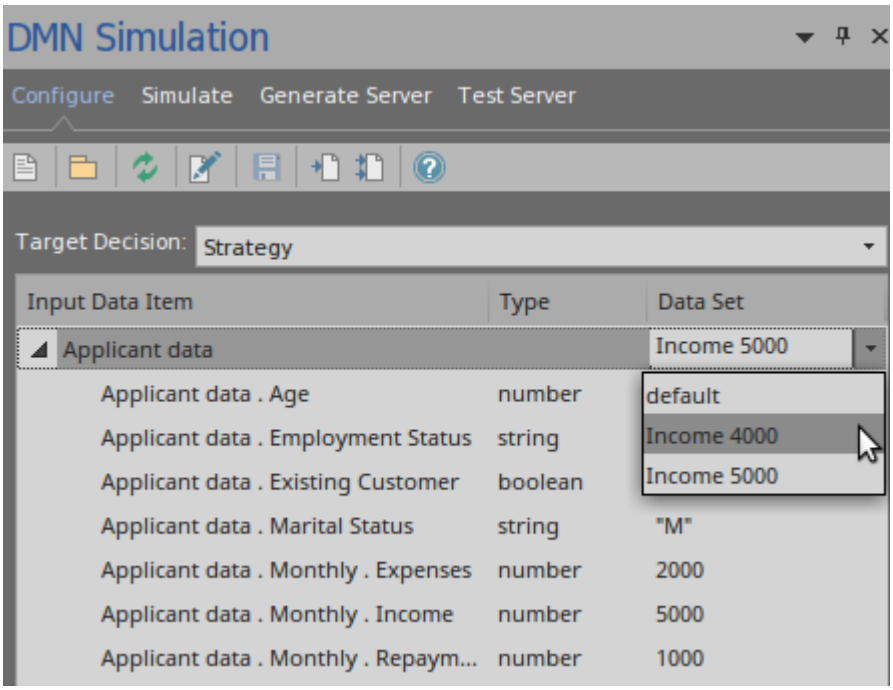
When a Package is loaded, a Decision Requirements Graph (DRG) and decision dependency hierarchy are created. The DMN InformationRequirement connectors determine the hierarchy.

- All the decisions will be listed in the 'Target Decision' combo box
- When a 'Target Decision' is specified, all the dependent Decisions, invoked BusinessKnowledges, InputDatas and the typed ItemDefinitions will be determined

DataSet & Input Data

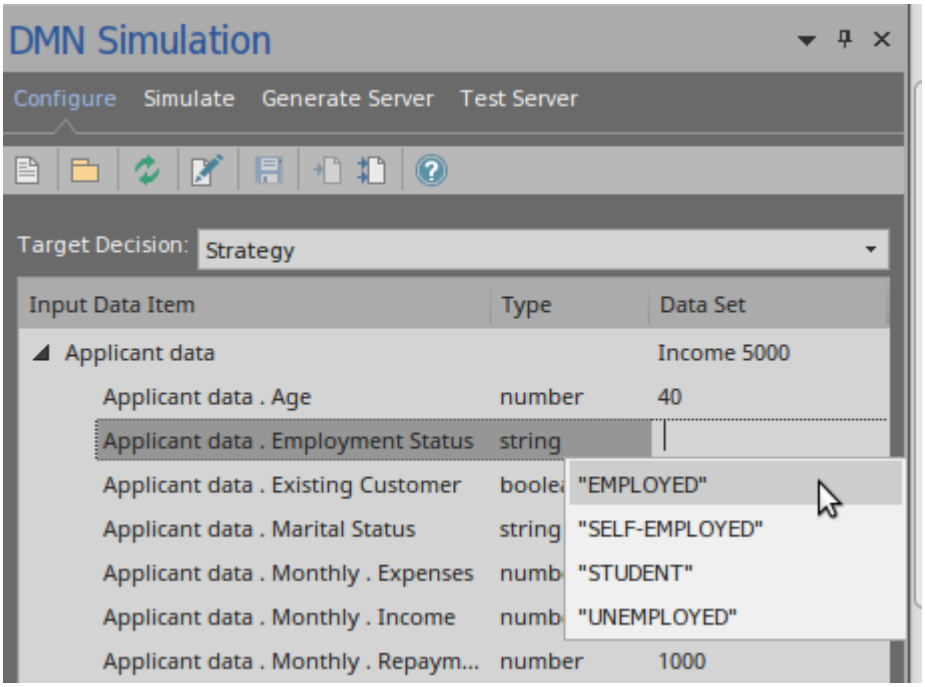
When the Target Decision is selected, all the dependent InputDatas are added to the list.

You can choose a dataset for simulation from the list of defined datasets of an InputData.

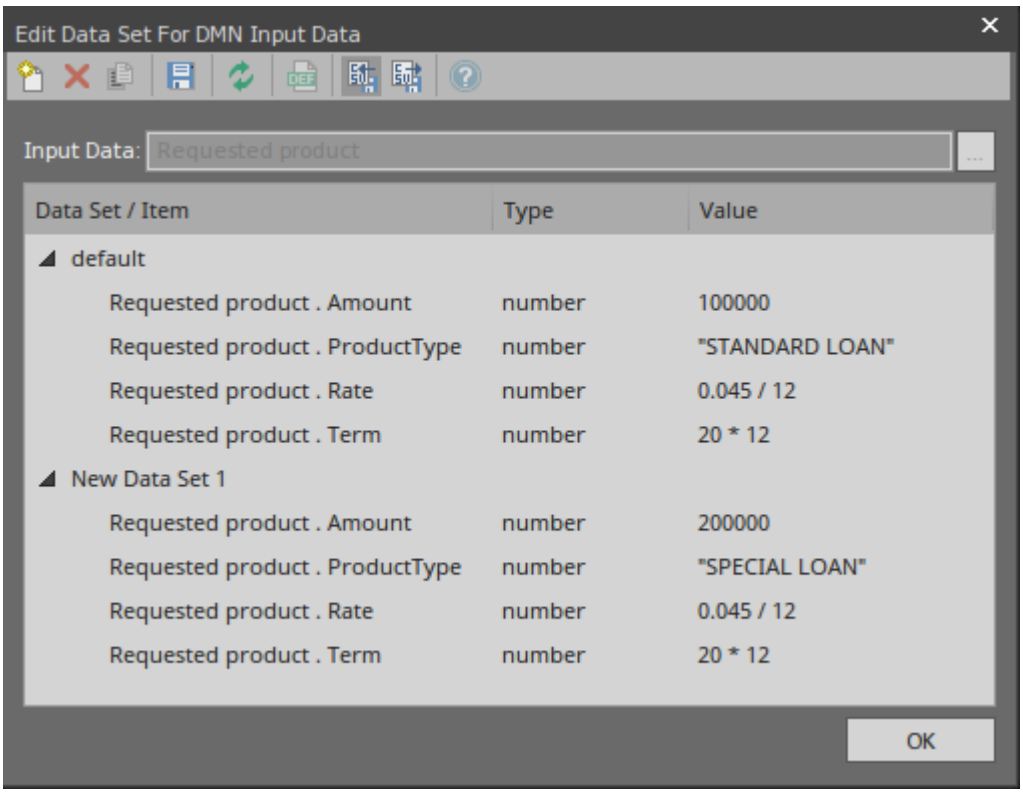


You can also change the dataset value in the list, if the ItemDefinition specifies an Allowed Value Enumeration. The auto-completion feature is supported and you can

simply choose from a list of enumerations.



You can also edit the datasets for an InputData by clicking on the Dataset button on the toolbar to open the 'Dataset Edit' dialog.



Simulate DMN Model

A DMNSimConfiguration Artifact contains information to simulate a DMN model depicted by Decision Requirements diagrams.

Access

Ribbon	Simulate > Decision Analysis > DMN > Open DMN Simulation | Simulate Page
Other	Double-click on a DMNSimConfiguration element |Simulate Page

Toolbar Options

Option	Description
	Click on this button to validate all the dependent DMN elements based on the Target Decision. Note: A

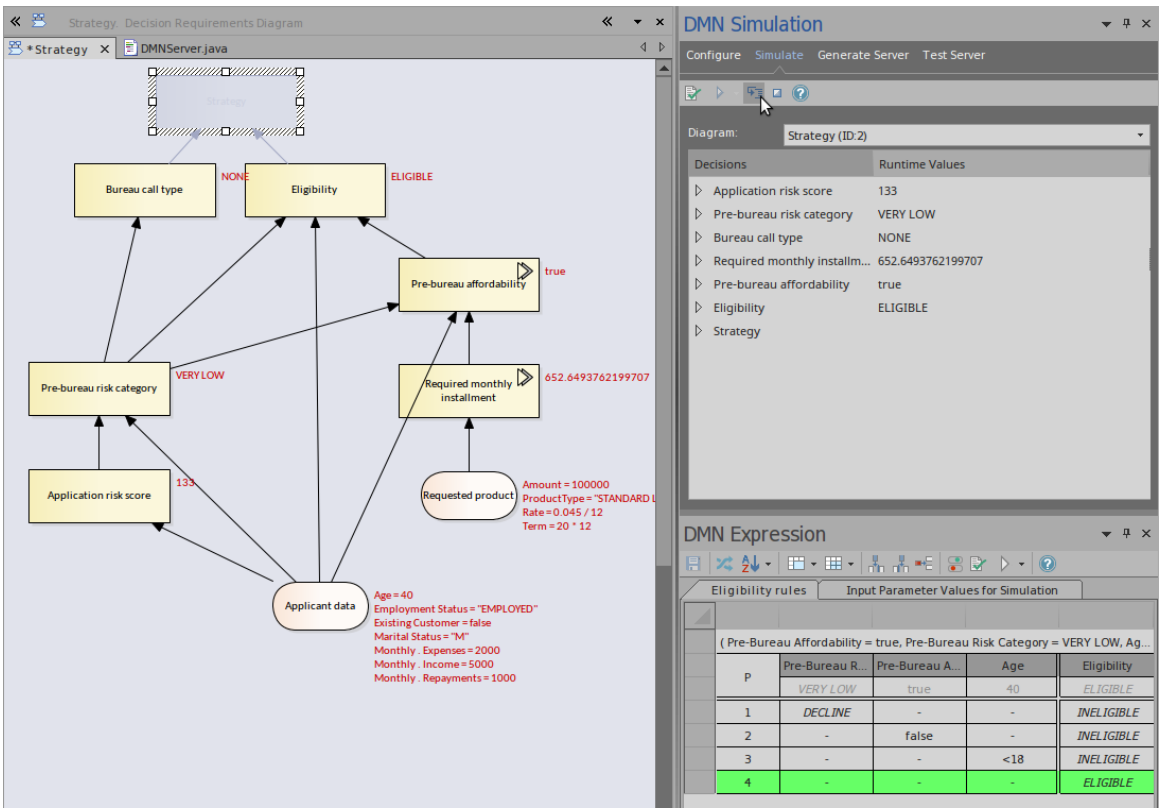
	Decision/BusinessKnowledgeModel/InputData/ItemDefinition that is on the Target Decision hierarchy will not be considered. For example, you have some unfinished Decision elements in the Package; as they have no relationship to the Target Decision, they will not impact the simulation.
	Click on this button to execute the decision hierarchy in order. The results will be represented on the diagram and shown in the 'Runtime values' column.
	Click on this button to step through the decision hierarchy in order. One click will evaluate one decision element. With this feature, you will be able to see the decision-making process; the decision logic and runtime values will be displayed clearly in the DMN Expression window.
	Click on this button to exit the simulation mode.

Simulation Overview

When a Target Decision is specified, the 'Simulation' page will be filled with the decisions, in dependency order.

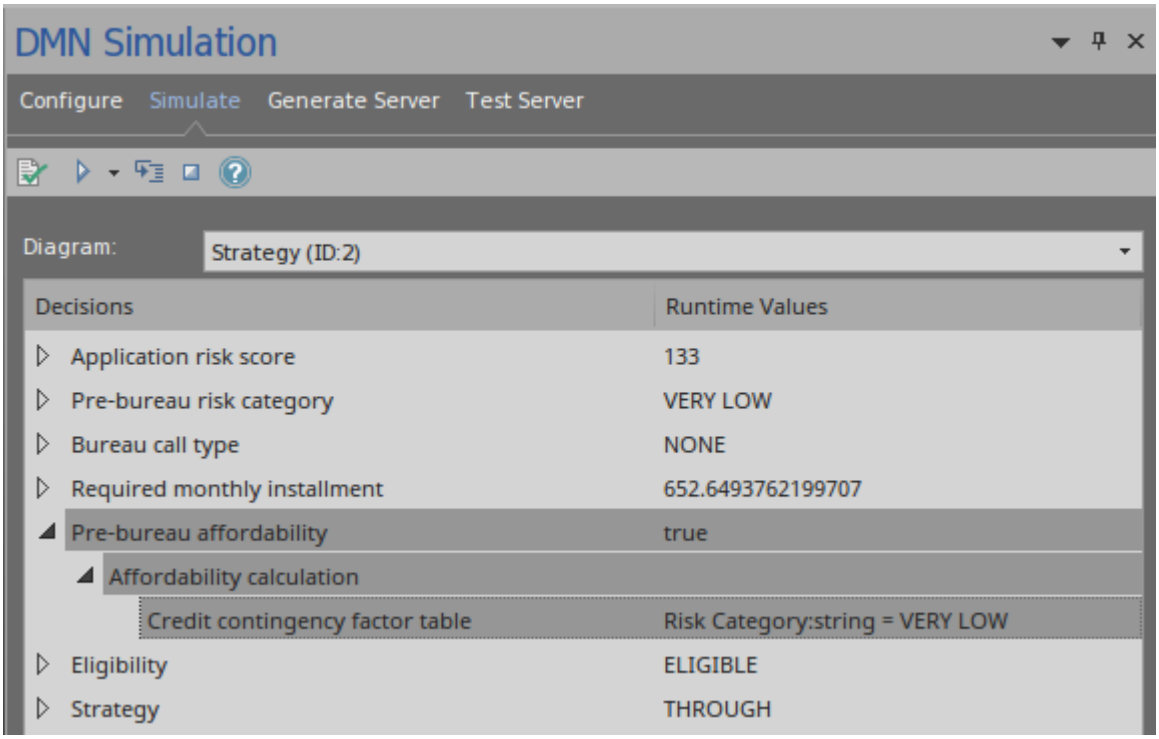
When Executing or Stepping Through the Decision Hierarchy, the decisions will be evaluated in order:

- The runtime result will be showing in the Decision row
- The runtime result will be displayed on the diagram
- The decision logic and input/output data will be presented in the DMN Expression window



Invocation Hierarchy

You can expand the Decision in the list to see the invocation hierarchy.



For example:

- Decision 'Pre-bureau affordability' invokes BusinessKnowledgeModel 'Affordability calculation'
- BusinessKnowledgeModel 'Affordability calculation' further invokes another BusinessKnowledgeModel 'Credit contingency factor table'

Advanced Debugging

Although Enterprise Architect provides a validation feature to help you locate many modeling issues and DMN expression issues, the simulation might still fail (rarely but possible) due to uncaught issues.

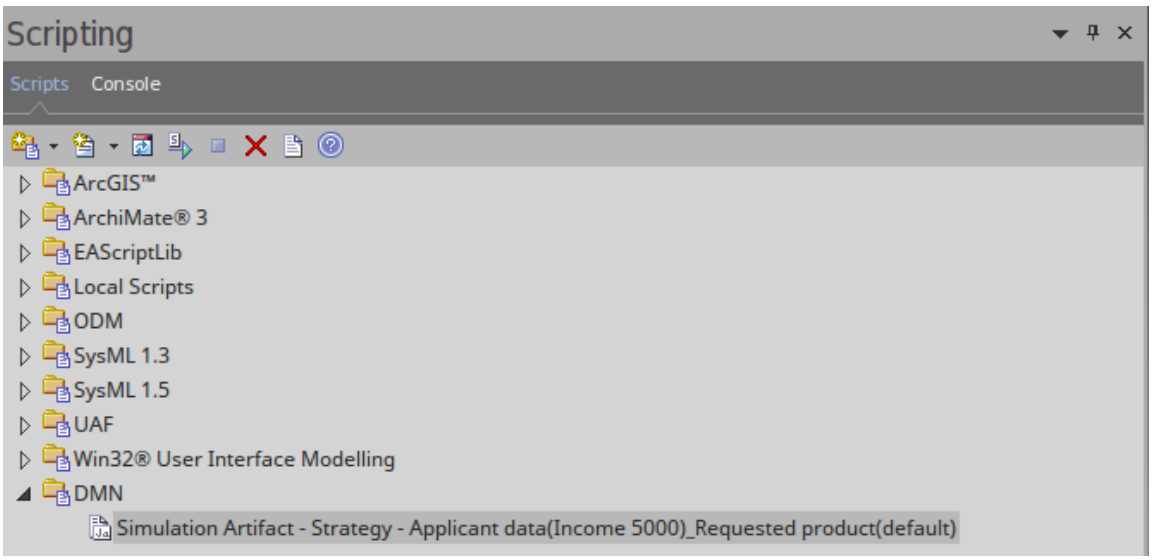
However, Enterprise Architect provides the ability to debug

the code that is running behind the simulation. You can also modify the code and run it in cycles until the issue is found and fixed.

The Execute button on the toolbar displays a menu with these options:

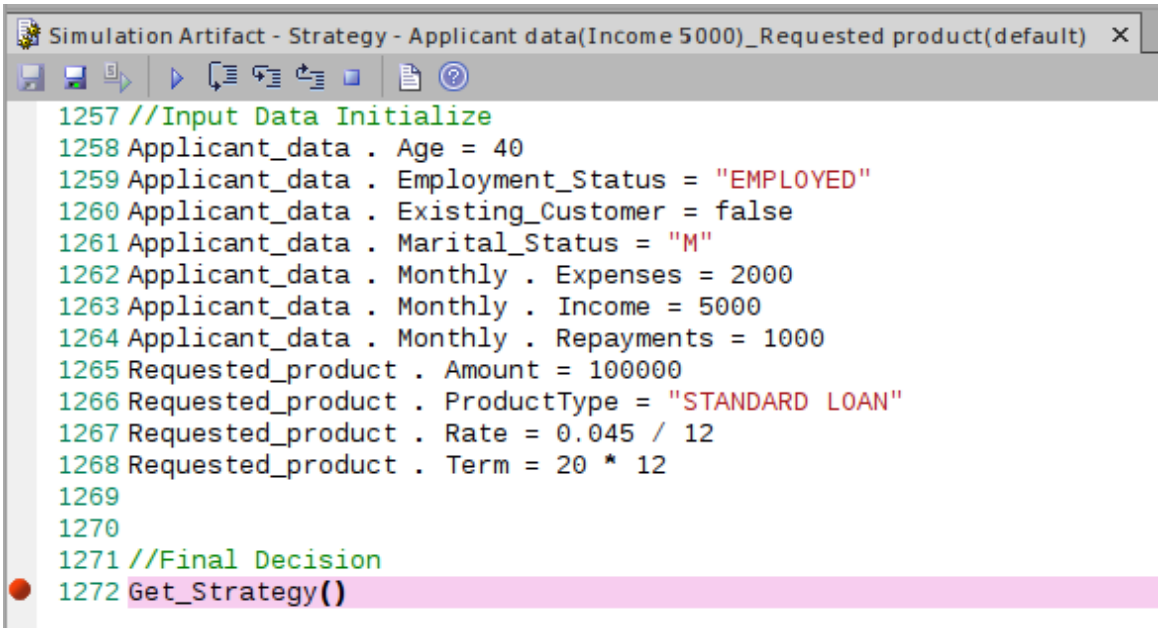
- Generate New Script (Scripting Window)
- Update Selected Script (Scripting Window)
- Run Selected Script (Scripting Window)
- Edit DMN Template

If you select 'Generate New Script (Scripting Window), the Scripting window displays showing a script created in a Package named 'DMN'.



- The default script name is composed of these parameters: 'ArtifactName - TargetDecision - InputData1(DataSet)_ InputData2(DataSet)_...'

Double-click on this file to open it in the Enterprise Architect Script editor, set a breakpoint, and debug the file.



By selecting the script in the Scripting Window, and if the script matches the model (by the 'Simulation Script Identifier' in the script), you enable the menu option 'Run Selected Script'.

You can customize the DMN Template to generate the correct script for simulation.

DMN Module Code Generation & Test

After a Decision Model is created and simulated, you can generate a DMN Module in Java, JavaScript, C++ or C#. The DMN Module can be used with the Enterprise Architect BPSim Execution Engine, Executable StateMachine, or your own project.

Enterprise Architect also provides a 'Test Module' page, which is a preprocess for integrating DMN with BPMN. The concept is to provide one or more BPMN2.0::DataObject elements, then test if a specified target decision can be evaluated correctly or not.

If any error or exception occurs, you can create an Analyzer Script to debug the code of the DMN Module and Test Client.

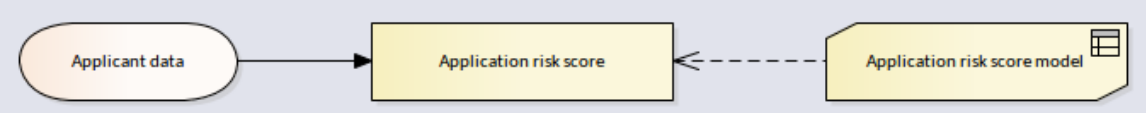
After this 'Test Module' process, Enterprise Architect guarantees that the BPMN2.0::DataObject elements work well with the DMN Module.

You then configure BPSim by loading DataObjects and assigning DMN Module decisions to BPSim Properties, which will be further used as conditions on the Sequence Flows outgoing from a Gateway.

Modeling & Simulation

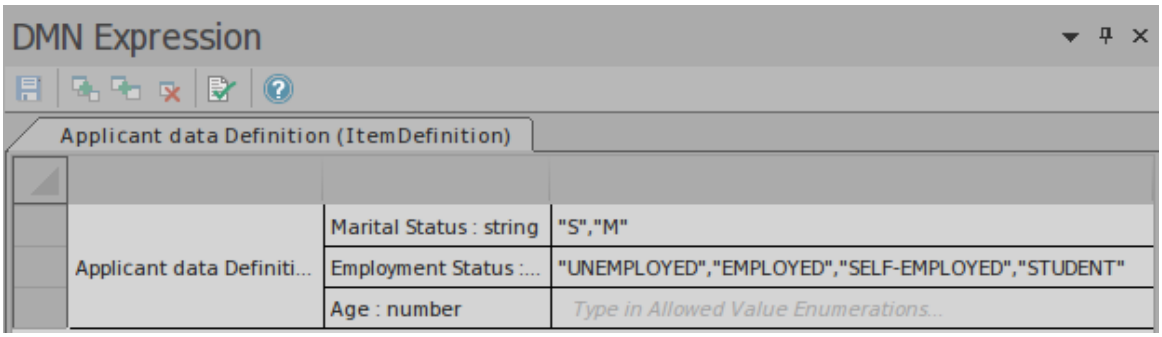
We took the example available from Model Patterns - Ribbon: Simulate | DMN | Apply Perspective | DMN

Decision | Decision with BKM

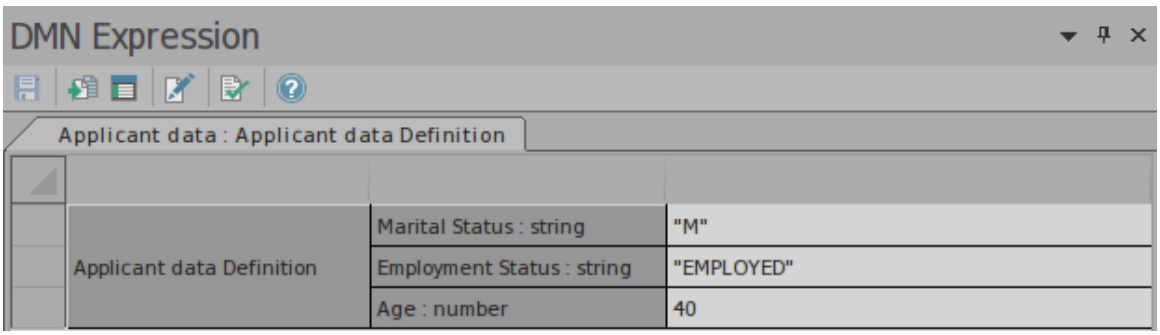


Here is a brief overview of the model:

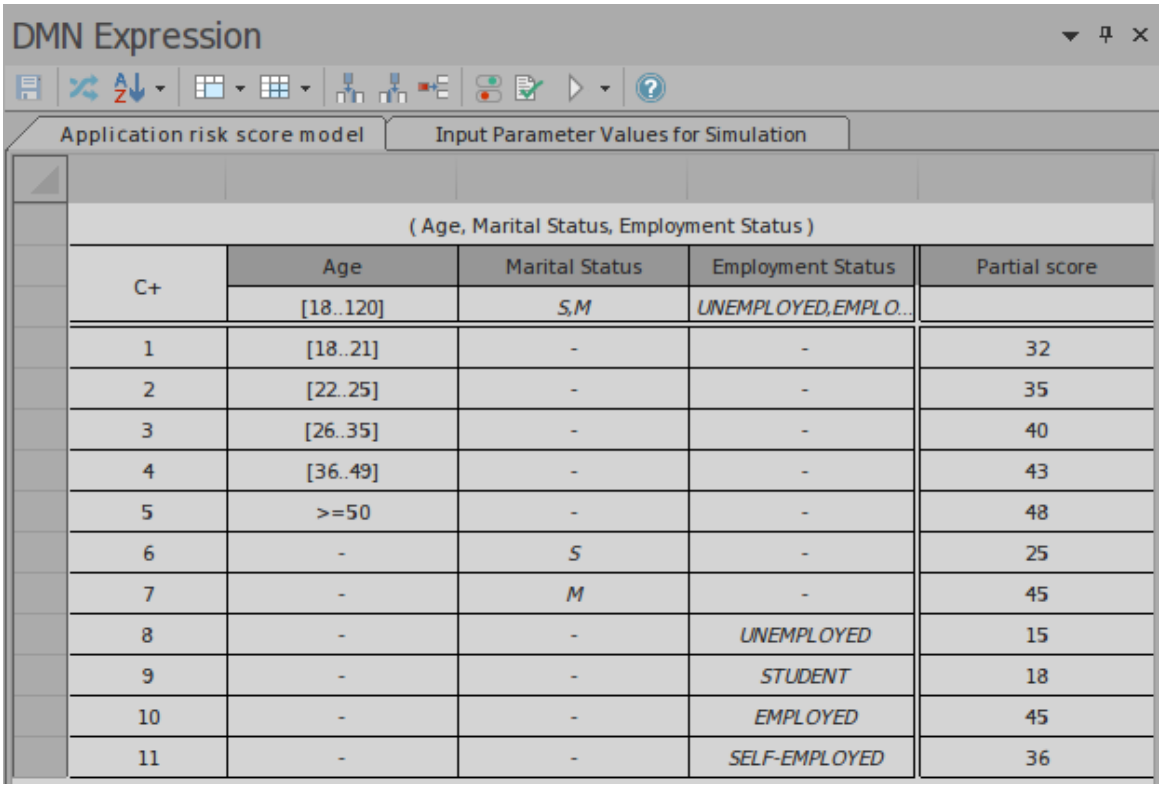
- Item Definition



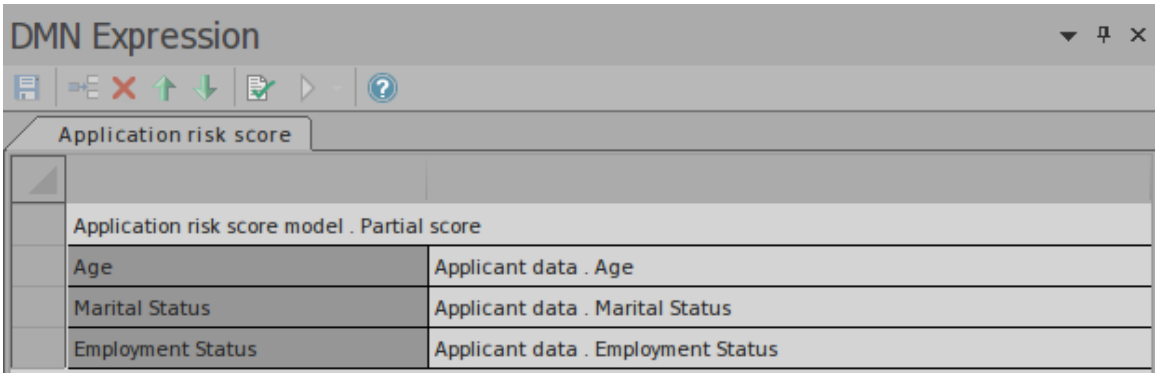
- Input Data



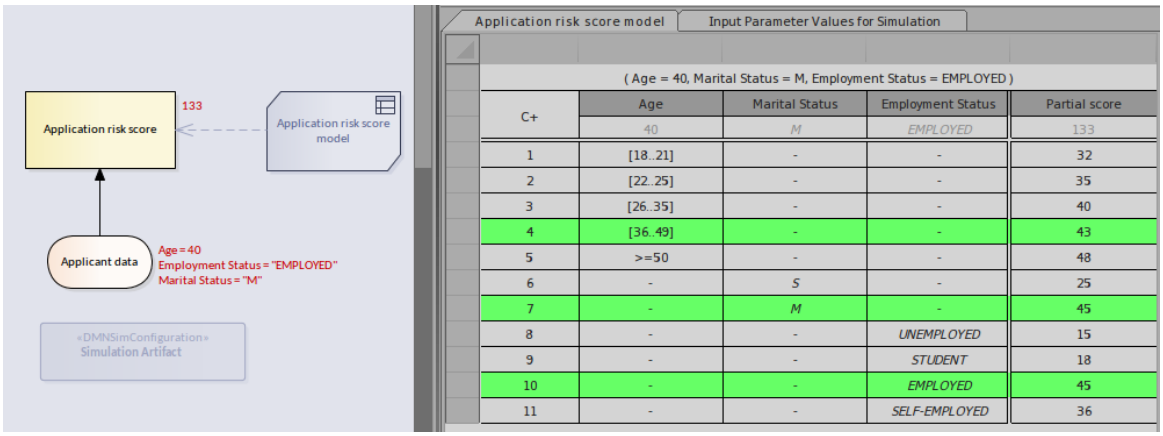
- Business Knowledge Model (Decision Table)



- Decision (Invocation)



- Simulation



- Export InputData to a BPMN2.0 DataObject

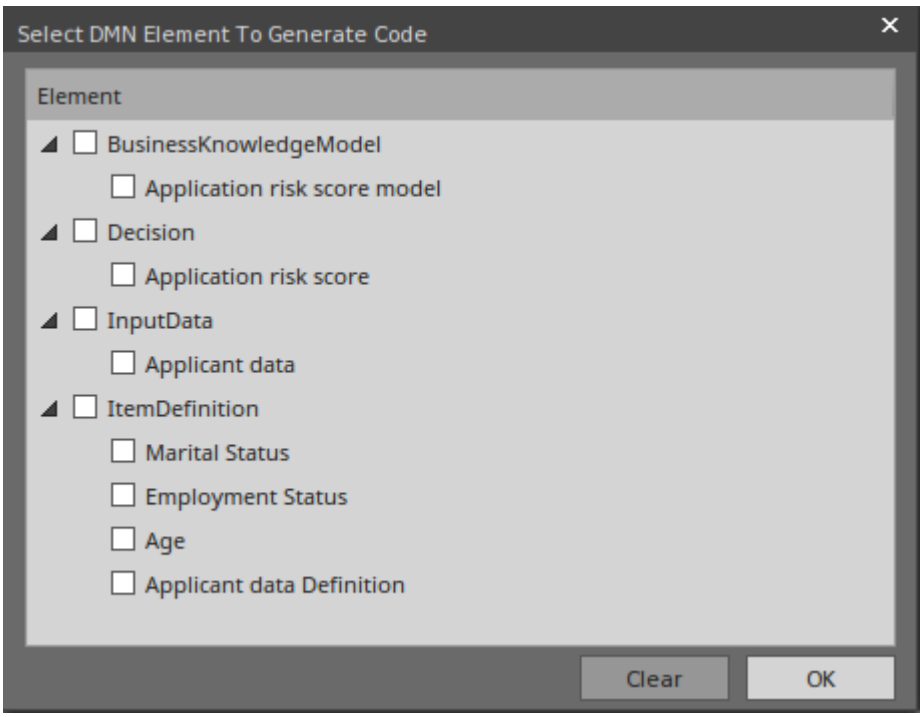
Activate the 'Configure' page, and click on the Export DataObject button on the toolbar.

DMN Module: Code Generation

Activate the 'Generate Module' tab of the DMN Simulation window, select the DMN elements you want to generate to the server, specify the file path and language, then click on the Generate button.

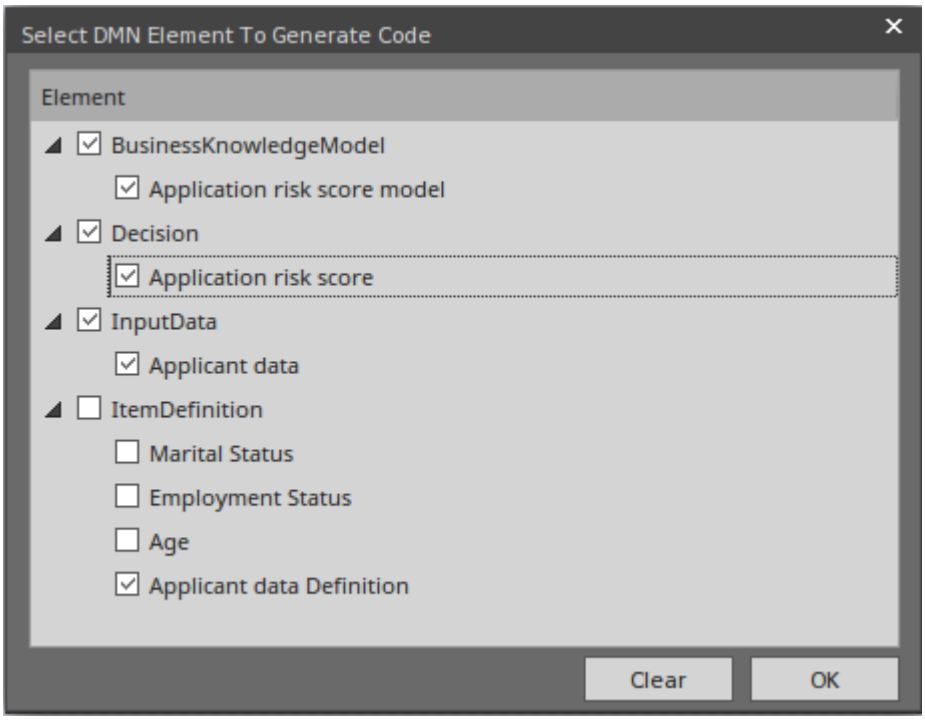
(Note: For Java, the path has to match the Package structure.)

- Add the elements to the module, then click on the Add button on the toolbar to open the 'DMN Element Selection' dialog

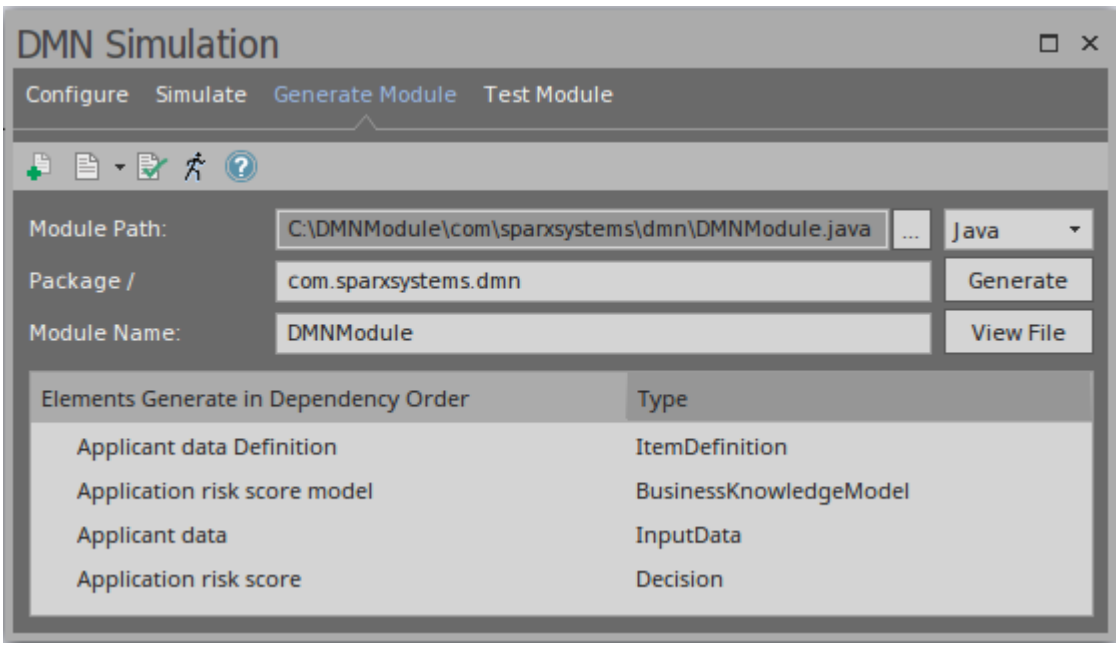


- Click on the decision you want to generate to the server; in this example we select the decision 'Application risk score'

Note: All the dependencies will be selected automatically.



- Generate the DMN Module; give the Java file a Package name, then click on the Generate button



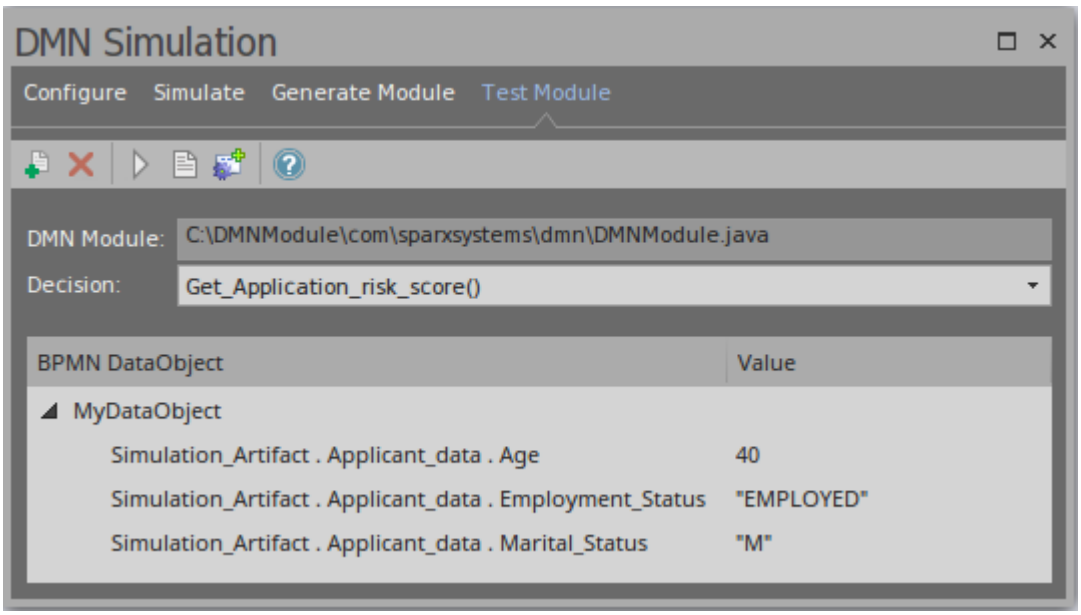
Note: In this example, the Module Path ends with '\com\sparxsystems\dmn', which matches the Package 'com.sparxsystems.dmn'.

- Click on the Test button to access the 'Test Server' page

DMN Server: Test client

If this page was activated from the 'Generate Server' page, the 'DMN Module' field will be filled automatically with the generated DMN Server's path. Otherwise, click the ... button to browse for a DMN Server file.

Click the Add DataObject button to add one or more BPMN2.0 DataObject(s) to the list, choose a decision from the combo box, then click the Run button on the toolbar.



In the System Output window, this message indicates the DMN Server and BPMN2.0 DataObject can work well with each other to evaluate the selected decision.

Running Test Client for DMN Server...

dmnServer.Application_risk_score: 133.0

Result : 133.0

The Running completed successfully.

If there are errors, create an Analyzer script by clicking the toolbar button and fix the issue.

Important: This 'Test Module' step is recommended before integrating DMNServer.java to the Enterprise Architect BPSim Execution Engine.

Integrate DMN Module Into UML Class Element

After a Decision Model is created and simulated, you can generate a DMN Module in Java, JavaScript, C++ or C# and test it.

The DMN Module can be integrated with a UML Class element, so the code generated from that Class element can reuse the DMN Module and be well-structured. Since a Class element can define a StateMachine, after integration with the DMN module the Executable StateMachine simulation will generically be able to use the power of the DMN Module.

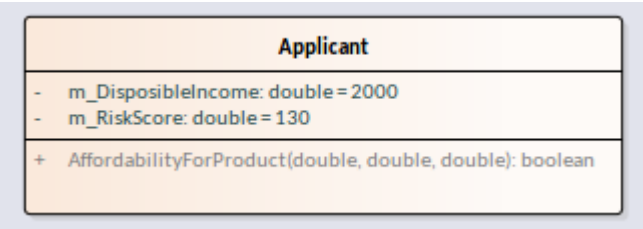
In this topic, we will explain the process of integrating a DMN Model with a UML Class element:

- Class element
- DMN Model(s)
- DMN Binding to Class & Intelli-sense
- Code Generation on Class Element

Class Element's Requirement

Suppose we have a Class *Applicant* with an operation *AffordabilityForProduct* that evaluates whether the applicant can afford a loan product.

A simplified model resembles this:



The Class *Applicant* contains two attributes, which are actually calculated from more basic data such as the applicant's monthly income, expenses, existing repayments, age and employment status.

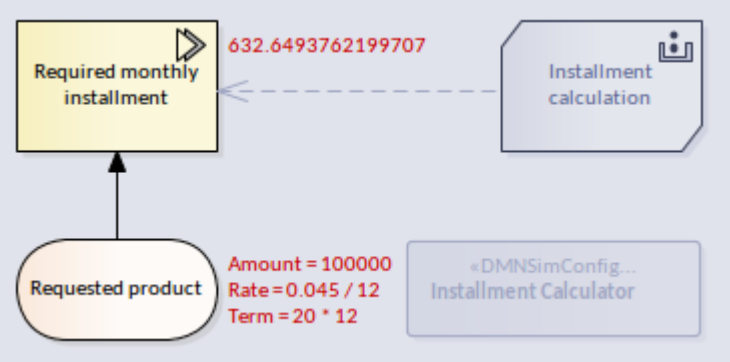
In this example, however, we simplify the model by skipping these steps and providing disposable income and risk score directly. In the 'DMN Complete Example' (Model Patterns), you can see all the more detailed steps.

DMN Model(s)

In this example, we have two disjoint DMN Models to show that a UML Class can integrate multiple DMN Models.

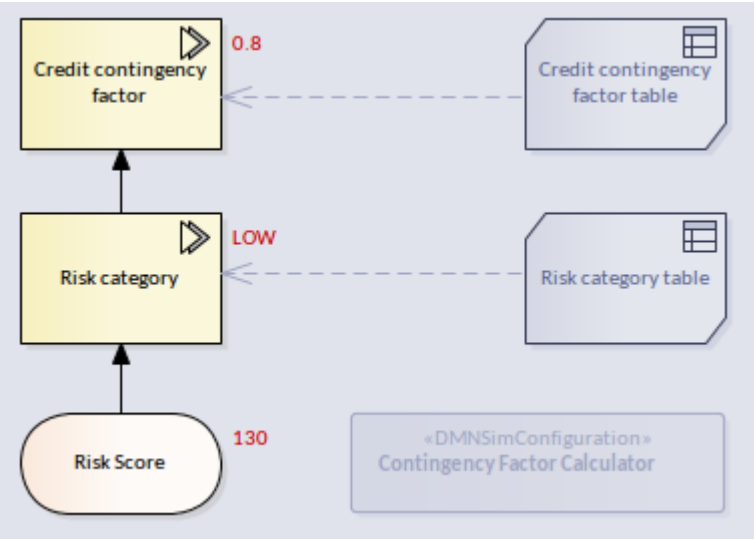
- Installment Calculator

This DMN Model computes the monthly repayment based on amount, rate and terms. It is composed of an InputData, a Decision and a Business Knowledge Model.



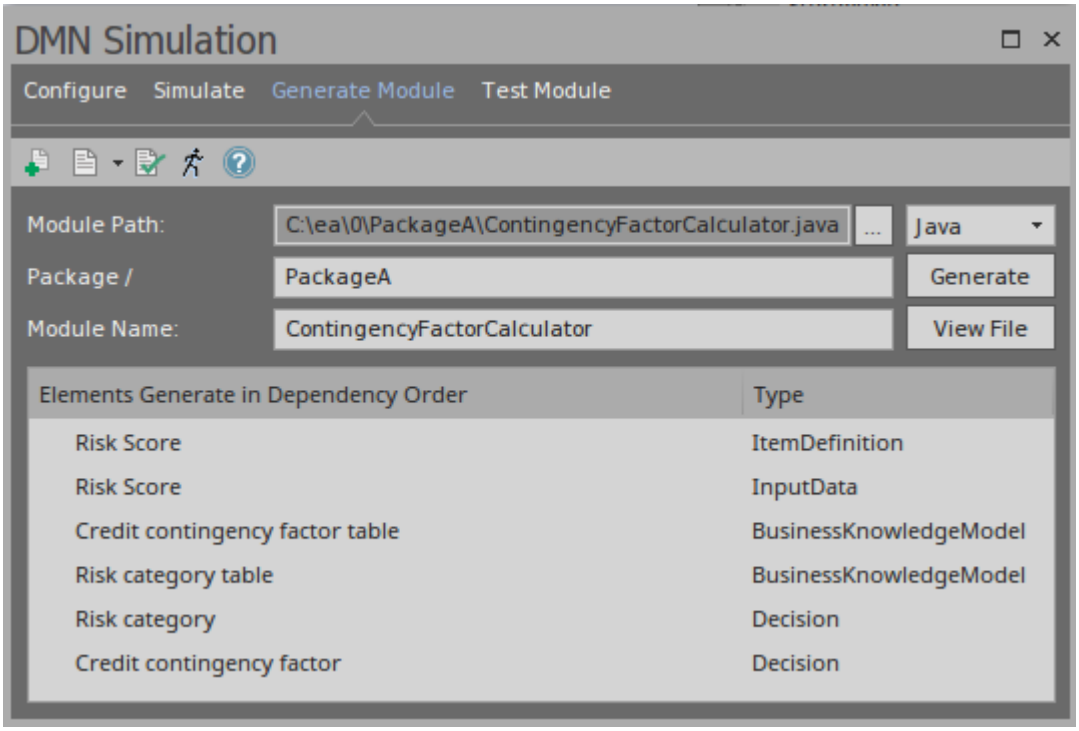
- Credit Contingency Factor Calculator

This DMN Model computes the credit contingency factor based on the applicant's risk score. It is composed of an InputData, two Decisions and two Business Knowledge Models.



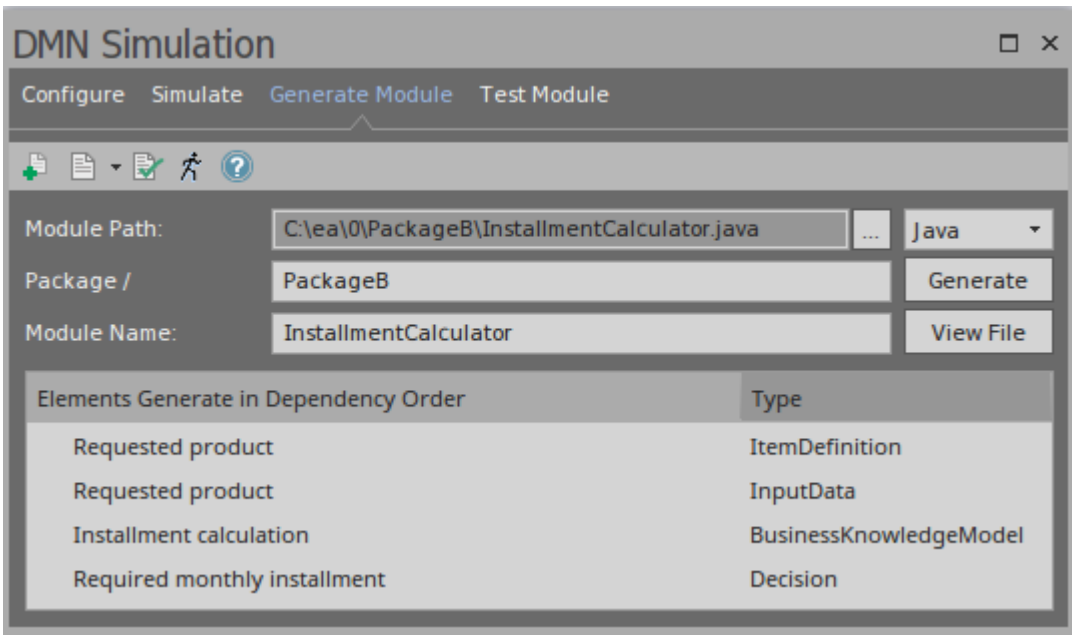
Note: In this example, we focus on how to integrate DMN Modules into a Class Element; the DMN element's detail is not described here. The full example is available in Model Patterns and the EAExample model.

- Generate code for both DMN Models

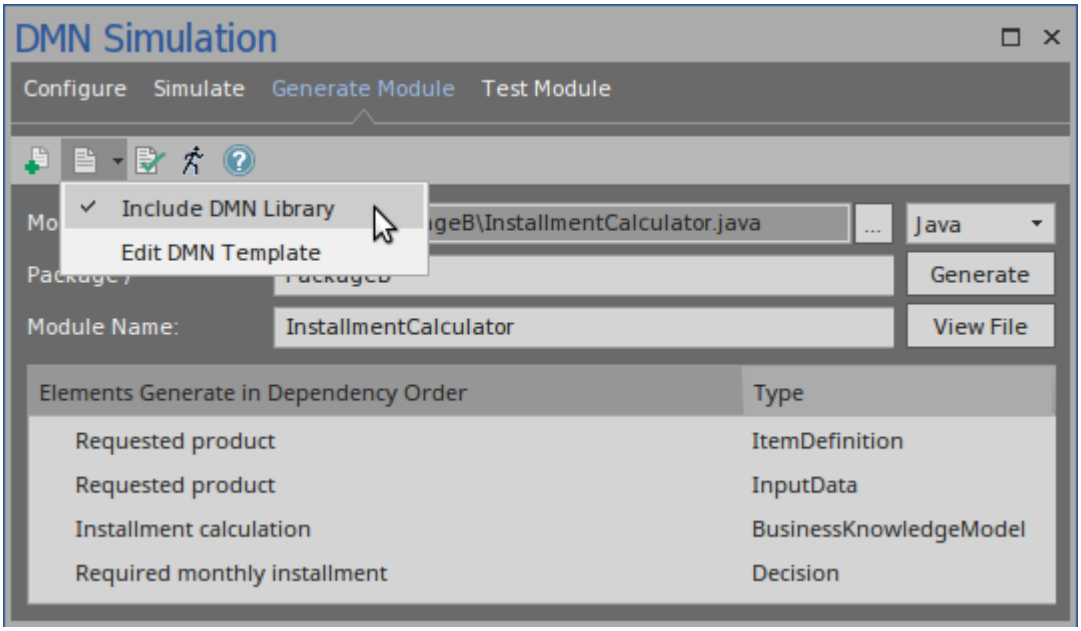


Click on the Generate button, and check that you can see this string in the System Output window, 'DMN' page:

DMN Module is successfully compiled.



Note: Since this model uses a built-in function PMT, the DMN Library has to be included:

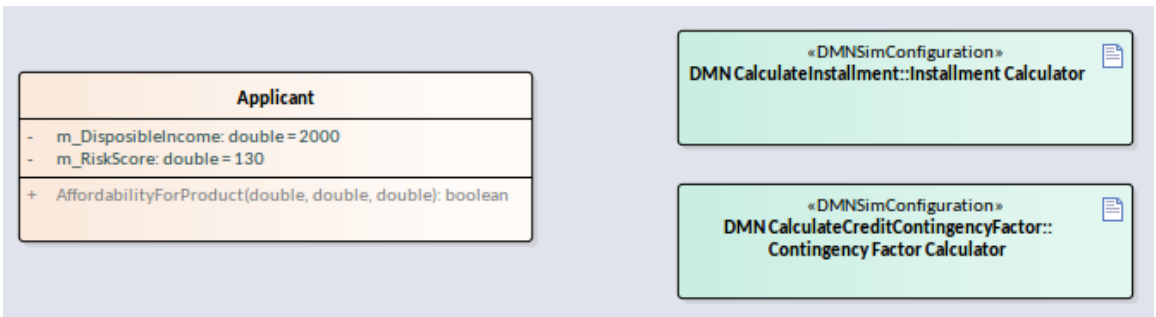


Click on the Generate button, and check that you can see this string in the System Output window, 'DMN' page:

DMN Module is successfully compiled.

DMN Binding to Class & Intelli-sense

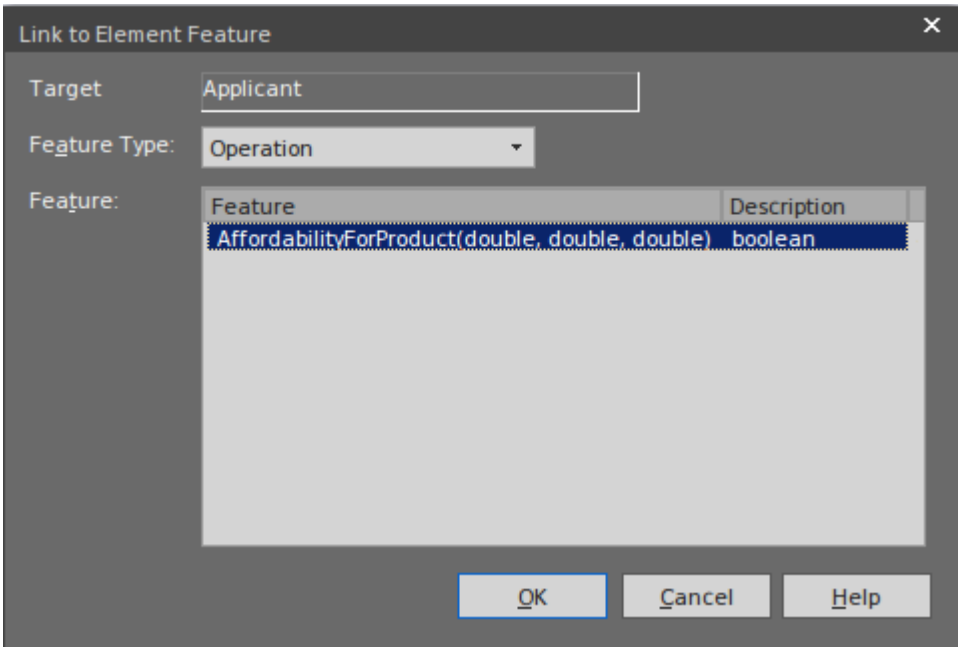
Put the two DMNSimConfiguration Artifacts on the Class diagram.



Use the Quick Linker to create a Dependency connector

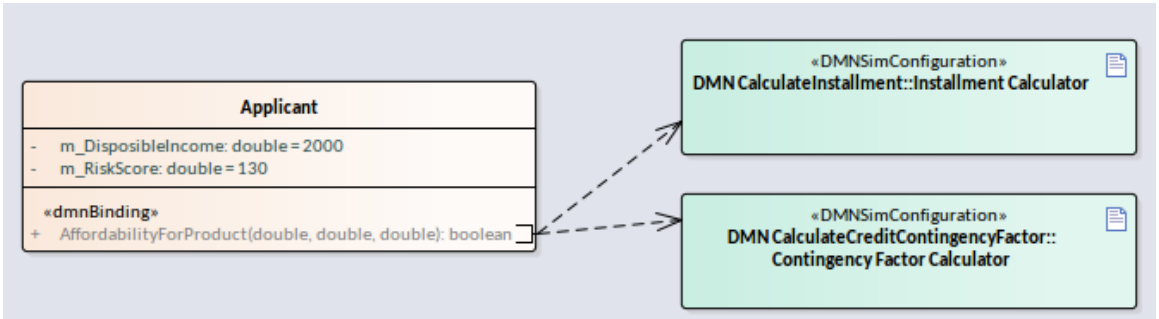
from the Class *Applicant* to the DMN Artifact.

On creation of the connector, the dialog will prompt you to choose the operation to be bound to the DMN module.



Here is what has happened after the DMN Binding:

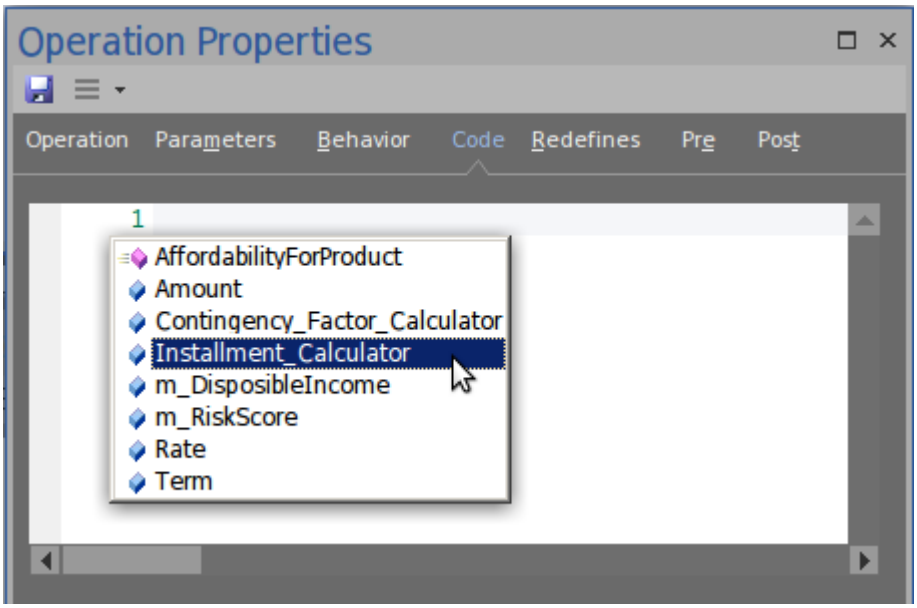
- The operation takes a stereotype <<dmnBinding>>
- The Dependency connector is linked to the operation
- Multiple DMN Artifacts can be bound to the same operation



After DMN Bindings, Intelli-sense for the operation's code editor will support DMN Modules. To trigger the

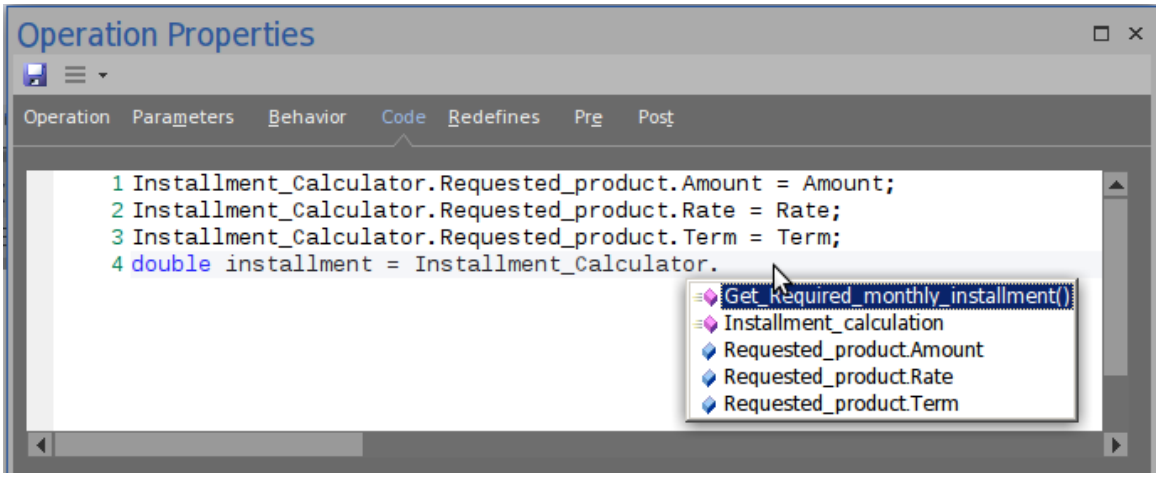
Intelli-sense, use these key combinations:

- Ctrl+Space - in most of the cases
- Ctrl+Shift+Space - when Ctrl+Space does not work after a parenthesis '('; for example, a function's arguments, or inside an 'If' condition's parentheses



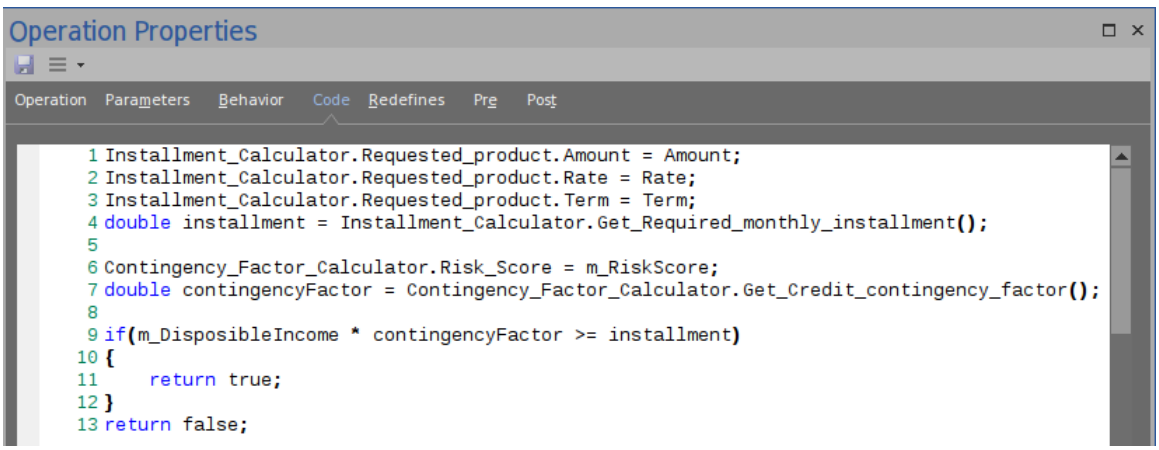
- Class attributes will be listed - m_RiskScore, m_DisposibleIncome
- Operation parameters will be listed - Amount, Rate, Term
- Operations will be listed - AffordabilityForProduct
- All bound DMN Modules will be listed - Contingency_Factor_Calculator, Installment_Calculator

It is quite easy to compose the code with Intelli-sense support. On accessing the DMN Module, all the Input Datas, Decisions and Business Knowledge Models will be listed for selection.



This illustration shows that we are selecting Get_Required_monthly_installment() from the Installment_Calculator.

This is the final implementation for the operation.



Code Generation for Class (With DMN Integration)

'Generate Code on Class Applicant' produces this code:

```
public class Applicant {

    private double m_DisposibleIncome = 2000;
    private double m_RiskScore = 130;

    PackageA.ContingencyFactorCalculator Contingency_Factor_Calculator = new PackageA.ContingencyFactorCalculator();
    PackageB.InstallmentCalculator Installment_Calculator = new PackageB.InstallmentCalculator();

    public boolean AffordabilityForProduct(double Amount, double Rate, double Term){
        //WARNING: Code in this function will be overwritten when generate from EA because this operation has a flush type of stereotype
        Installment_Calculator.Requested_product.Amount = Amount;
        Installment_Calculator.Requested_product.Rate = Rate;
        Installment_Calculator.Requested_product.Term = Term;
        double installment = Installment_Calculator.Get_Required_monthly_installment();

        Contingency_Factor_Calculator.Risk_Score = m_RiskScore;
        double contingencyFactor = Contingency_Factor_Calculator.Get_Credit_contingency_factor();

        if(m_DisposibleIncome * contingencyFactor >= installment) {
            return true;
        }
        return false;
    }
}//end Applicant
```

- The DMN Module(s) are generated as attributes of the Class
- The dmnBinding operation's code is updated

Note: Regardless of whether the generation option is 'Overwrite' or' Synchronize', the operation's code will be updated if it has the stereotype 'dmnBinding'.