Proposal for:

Decision Model and Notation (DMN) Specification 1.0

(Revised Submission)
August 23, 2013

In response to: Decision Model and Notation (DMN) RFP (OMG Document bmi/2011-03-04)

Submission Team

OMG Submitters
Decision Management Solutions
Escape Velocity
FICO
International Business Machines
Oracle

Co-Authors
KU Leuven
Knowledge Partners International
Model Systems
TIBCO
Copyrights
Copyright © 2013, Decision Management Solutions
Copyright © 2013, Escape Velocity LLC
Copyright © 2013, Fair Isaac Corporation
Copyright © 2013, International Business Machines Corporation
Copyright © 2013, Knowledge Partners International
Copyright © 2013, KU Leuven
Copyright © 2013, Model Systems Limited
Copyright © 2013, Oracle Incorporated
Copyright © 2013, TIBCO Software Inc.

Permission to copy, display, perform, modify and distribute the Case Management 1.0 Specification (the “Specification”, which includes WSDL and schema documents), and to authorize others to do the foregoing, in any medium without fee or royalty is hereby granted for the purpose of developing and evaluating the Specification.

Decision Management Solutions, Escape Velocity LLC, Fair Isaac Corporation, International Business Machines Corporation, Knowledge Partners International, KU Leuven, Model Systems Limited, Oracle Incorporated and TIBCO Software Inc (collectively, the “Authors”) each will grant a license to third parties, under royalty-free and otherwise reasonable, non-discriminatory terms and conditions, to their respective essential patent claims that they deem necessary to implement the Specification.

DISCLAIMERS:

THE SPECIFICATION IS PROVIDED "AS IS," AND THE AUTHORS MAKE NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT, OR TITLE; THAT THE CONTENTS OF THE SPECIFICATION ARE SUITABLE FOR ANY PURPOSE; NOR THAT THE IMPLEMENTATION OF SUCH CONTENTS WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.

THE AUTHORS WILL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF ANY USE OF THE SPECIFICATION OR THE PERFORMANCE OR IMPLEMENTATION OF THE CONTENTS THEREOF.

You may remove these disclaimers from your modified versions of the Specification provided that you effectively disclaim all warranties and liabilities on behalf of all copyright holders in the copies of any such modified versions you distribute.

The name and trademarks of the Authors may NOT be used in any manner, including advertising or publicity pertaining to the Specification or its contents without specific, written prior permission. Title to copyright in the Specification will at all times remain with the Authors.

No other rights are granted by implication, estoppel or otherwise.
# Table of Contents

Decision Model and Notation (DMN) Specification 1.0 ............................................. 1  
Table of Contents ........................................................................................................... 3  
Table of Figures ............................................................................................................. 8  
Table of Tables .............................................................................................................. 11  
Introduction .................................................................................................................. 13  
  Submitting organizations ............................................................................................. 13  
  Co-Authors .................................................................................................................. 13  
  Submission Contacts .................................................................................................... 13  
  Acknowledgements ...................................................................................................... 13  
  Status of the Document ............................................................................................... 14  
  IPR and Patents ........................................................................................................... 14  
  Typographical Conventions ......................................................................................... 14  
  Responses to RFP Requirements ................................................................................ 14  
  Guide to the Submission ............................................................................................. 14  

1 Scope .......................................................................................................................... 16  

2 Conformance .............................................................................................................. 16  
  2.1 General conformance requirements ....................................................................... 17  
    2.1.1 Visual appearance ............................................................................................ 17  
    2.1.2 Decision semantics ......................................................................................... 17  
    2.1.3 Attributes and model associations .................................................................. 17  

3 Normative References .............................................................................................. 18  
  3.1 Normative ............................................................................................................ 18  
  3.2 Non-normative ...................................................................................................... 18  

4 Introduction to DMN ............................................................................................... 19  
  4.1 Context ................................................................................................................ 19  
  4.2 Basic concepts ...................................................................................................... 23  
    4.2.1 Decision requirements level .......................................................................... 23  
    4.2.2 Decision logic level ....................................................................................... 24  
  4.3 Scope and uses of DMN ....................................................................................... 27  
    4.3.1 Modeling human decision-making ................................................................. 27  
    4.3.2 Modeling requirements for automated decision-making ............................... 28
5 Decision Requirements (DRG and DRD) ................................................................. 30
  5.1 Introduction ........................................................................................................... 30
  5.2 Notation .................................................................................................................. 30
    5.2.1 DRD Elements ................................................................................................. 31
    5.2.2 DRD Requirements ........................................................................................ 33
    5.2.3 Connection rules .............................................................................................. 35
    5.2.4 Partial views and hidden information .............................................................. 35
  5.3 Metamodel ............................................................................................................ 37
    5.3.1 DMN Element metamodel .............................................................................. 37
    5.3.2 Definitions metamodel ................................................................................... 38
    5.3.3 Import metamodel .......................................................................................... 40
    5.3.4 Element Collection metamodel ..................................................................... 40
    5.3.5 DRG Element metamodel .............................................................................. 41
    5.3.6 Decision metamodel ...................................................................................... 42
    5.3.7 Business Context Element metamodel ........................................................ 45
    5.3.8 Business Knowledge Model metamodel ........................................................ 47
    5.3.9 Input Data metamodel .................................................................................... 49
    5.3.10 Knowledge Source metamodel .................................................................... 49
    5.3.11 Information Requirement metamodel ........................................................ 50
    5.3.12 Knowledge Requirement metamodel .......................................................... 51
    5.3.13 Authority Requirement metamodel .............................................................. 52
  5.4 Examples .............................................................................................................. 53
  6 Relating Decision Logic to Decision Requirements ............................................. 53
    6.1 Introduction ......................................................................................................... 53
    6.2 Notation ............................................................................................................... 56
      6.2.1 Boxed Expressions ....................................................................................... 56
      6.2.2 Boxed literal expression ............................................................................... 57
      6.2.3 Boxed invocation ......................................................................................... 58
    6.3 Metamodel .......................................................................................................... 58
      6.3.1 Expression metamodel ................................................................................. 60
      6.3.2 ItemDefinition metamodel .......................................................................... 62
      6.3.3 InformationItem metamodel ....................................................................... 64
7 Decision Table ........................................................................................................68
  7.1 Introduction ........................................................................................................68
  7.2 Notation ...............................................................................................................69
      7.2.1 Line style and color ..................................................................................70
      7.2.2 Table orientation .....................................................................................70
      7.2.3 Input expressions ......................................................................................71
      7.2.4 Input values ...............................................................................................71
      7.2.5 Table name and output name ....................................................................71
      7.2.6 Output values ............................................................................................71
      7.2.7 Multiple outputs .......................................................................................72
      7.2.8 Input entries ..............................................................................................73
      7.2.9 Merged input entry cells ..........................................................................73
      7.2.10 Output entry .............................................................................................74
      7.2.11 Hit policy ..................................................................................................74
      7.2.12 Completeness indicator ..........................................................................76
      7.2.13 Aggregation .............................................................................................76
  7.3 Metamodel ...........................................................................................................77
      7.3.1 Decision Table metamodel .......................................................................77
      7.3.2 Decision Table Clause metamodel .........................................................80
      7.3.3 Decision Rule metamodel ........................................................................81
  7.4 Examples .............................................................................................................82

8 Simple Expression Language (S-FEEL) ...............................................................85
  8.1 S-FEEL syntax ...................................................................................................86
  8.2 S-FEEL data types ............................................................................................87
  8.3 S-FEEL semantics .............................................................................................88
  8.4 Use of S-FEEL expressions ..............................................................................89
      8.4.1 Item definitions .........................................................................................89
      8.4.2 Invocations ...............................................................................................89
      8.4.3 Decision tables .........................................................................................89

9 Expression Language (FEEL) ................................................................................90
9.1 Introduction ........................................................................................................................................... 90
9.2 Notation .................................................................................................................................................. 90
  9.2.1 Boxed Expressions ............................................................................................................................. 90
  9.2.2 FEEL .................................................................................................................................................. 98
9.3 Full FEEL syntax and semantics ........................................................................................................... 100
  9.3.1 Syntax ............................................................................................................................................... 100
  9.3.2 Semantics ........................................................................................................................................ 105
  9.3.3 XML Data ....................................................................................................................................... 123
  9.3.4 Built-in functions .............................................................................................................................. 126
9.4 Relationship of FEEL to DRG and Boxed Expressions ...................................................................... 132
9.5 Metamodel .......................................................................................................................................... 133
  9.5.1 Context metamodel .......................................................................................................................... 134
  9.5.3 FunctionDefinition metamodel ....................................................................................................... 135
  9.5.4 List metamodel ................................................................................................................................ 135
  9.5.5 Relation metamodel ........................................................................................................................ 136
9.6 Examples .............................................................................................................................................. 136
  9.6.1 Context ........................................................................................................................................... 136
  9.6.2 Calculation ..................................................................................................................................... 137
  9.6.3 If, In .................................................................................................................................................. 137
  9.6.4 Sum entries of a list .......................................................................................................................... 137
  9.6.5 Invocation of user-defined PMT function ......................................................................................... 137
  9.6.6 Sum weights of recent credit history .............................................................................................. 137
  9.6.7 Determine if credit history contain a bankruptcy event ................................................................. 138
10 DMN Example ...................................................................................................................................... 138
  10.1 The business process model .......................................................................................................... 138
  10.2 The decision requirements level ....................................................................................................... 139
  10.3 The decision logic level .................................................................................................................... 142
11 Exchange formats ................................................................................................................................. 152
  11.1 Interchanging Incomplete Models ................................................................................................... 152
  11.2 Machine Readable Files .................................................................................................................. 153
  11.3 XSD .................................................................................................................................................... 153
    11.3.1 Document Structure .................................................................................................................. 153
    11.3.2 References within the DMN XSD ............................................................................................. 153
Annex A. Relation to BPMN ..................................................................................................................... 155
# Table of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1</td>
<td>Aspects of modeling</td>
<td>20</td>
</tr>
<tr>
<td>Figure 2</td>
<td>DMN Constructs</td>
<td>22</td>
</tr>
<tr>
<td>Figure 3</td>
<td>Basic elements of a decision model</td>
<td>23</td>
</tr>
<tr>
<td>Figure 4</td>
<td>Knowledge sources</td>
<td>23</td>
</tr>
<tr>
<td>Figure 5</td>
<td>A simple Decision Requirements Diagram (DRD)</td>
<td>24</td>
</tr>
<tr>
<td>Figure 6</td>
<td>Combining business knowledge models</td>
<td>24</td>
</tr>
<tr>
<td>Figure 7</td>
<td>Decision and corresponding value expression</td>
<td>25</td>
</tr>
<tr>
<td>Figure 8</td>
<td>Business knowledge model and corresponding value expression</td>
<td>26</td>
</tr>
<tr>
<td>Figure 9</td>
<td>Business knowledge model and corresponding decision table</td>
<td>26</td>
</tr>
<tr>
<td>Figure 10</td>
<td>Decision with Listed Input Data option</td>
<td>32</td>
</tr>
<tr>
<td>Figure 11</td>
<td>The Listed Input Data option</td>
<td>33</td>
</tr>
<tr>
<td>Figure 12</td>
<td>Knowledge Sources representing authorities</td>
<td>34</td>
</tr>
<tr>
<td>Figure 13</td>
<td>Knowledge source representing predictive analytics</td>
<td>35</td>
</tr>
<tr>
<td>Figure 14</td>
<td>DRDs as partial views of a DRG</td>
<td>36</td>
</tr>
<tr>
<td>Figure 15</td>
<td>DMNElements Class Diagram</td>
<td>37</td>
</tr>
<tr>
<td>Figure 16</td>
<td>Definitions Class Diagram</td>
<td>38</td>
</tr>
<tr>
<td>Figure 17</td>
<td>Decision Class Diagram</td>
<td>42</td>
</tr>
<tr>
<td>Figure 18</td>
<td>BusinessContextElement class diagram</td>
<td>45</td>
</tr>
<tr>
<td>Figure 19</td>
<td>BusinessKnowledgeModel class diagram</td>
<td>47</td>
</tr>
<tr>
<td>Figure 20</td>
<td>InputData class diagram</td>
<td>49</td>
</tr>
<tr>
<td>Figure 21</td>
<td>Boxed expression</td>
<td>56</td>
</tr>
<tr>
<td>Figure 22</td>
<td>Boxed expression with separated name and expression boxes</td>
<td>57</td>
</tr>
<tr>
<td>Figure 23</td>
<td>Decision table with italicized literals</td>
<td>57</td>
</tr>
<tr>
<td>Figure 24</td>
<td>Decision table with string literals</td>
<td>57</td>
</tr>
<tr>
<td>Figure 25</td>
<td>Boxed invocation</td>
<td>58</td>
</tr>
<tr>
<td>Figure 26</td>
<td>Expression class diagram</td>
<td>60</td>
</tr>
<tr>
<td>Figure 27</td>
<td>Rules as rows</td>
<td>70</td>
</tr>
<tr>
<td>Figure 28</td>
<td>Rules as columns</td>
<td>70</td>
</tr>
<tr>
<td>Figure 29</td>
<td>Rules as crosstab (optional input and output values not shown)</td>
<td>71</td>
</tr>
<tr>
<td>Figure 30</td>
<td>Horizontal table with compound output</td>
<td>72</td>
</tr>
</tbody>
</table>
Figure 31: Vertical table with compound output ................................................................. 72
Figure 32: Crosstab with compound output ........................................................................ 72
Figure 33: Merged rule input cells allowed ....................................................................... 73
Figure 34: Merged rule input cells not allowed ................................................................. 73
Figure 35: Shorthand notation for vertical tables (rules as columns) ................................. 74
Figure 36: Full notation for vertical tables (rules as columns) ........................................... 74
Figure 37: DecisionTable class diagram .......................................................................... 77
Figure 38: Boxed expression .............................................................................................. 91
Figure 39: Boxed expression with separated name and expression boxes ....................... 91
Figure 40: Boxed FEEL expression ................................................................................... 92
Figure 41: Boxed invocation .............................................................................................. 92
Figure 42: Vertical context ................................................................................................. 93
Figure 43: Horizontal context ............................................................................................ 93
Figure 44: Use of context entries ....................................................................................... 94
Figure 45: Use of final result box ...................................................................................... 94
Figure 46: Vertical context with decision table entry ......................................................... 95
Figure 47: Use of boxed expressions with a decision table ............................................... 96
Figure 48: Vertical list ........................................................................................................ 97
Figure 49: Horizontal list ................................................................................................... 97
Figure 50: Relation ............................................................................................................ 97
Figure 51: Boxed function ................................................................................................. 98
Figure 52: Expression class diagram ................................................................................ 133
Figure 53: Example context ............................................................................................... 137
Figure 54: Example business process ................................................................................ 139
Figure 55: DRD of all automated decision-making ............................................................ 140
Figure 56: DRD for Bureau Strategy decision point .......................................................... 141
Figure 57: DRD for Application Routing decision point .................................................... 141
Figure 58: Strategy decision logic ..................................................................................... 144
Figure 59: Strategy table decision logic ............................................................................ 145
Figure 60: Bureau Call Type decision logic ..................................................................... 145
Figure 61: Bureau call type table decision logic ............................................................... 145
Figure 62: Eligibility decision logic .................................................................................. 145
Figure 63: Eligibility rules decision logic ................................................................. 146
Figure 64: Pre-Bureau Risk Category decision logic .................................................. 146
Figure 65: Pre-bureau risk category table decision logic .......................................... 146
Figure 66: Application Risk Score decision logic ...................................................... 147
Figure 67: Application risk score model decision logic ............................................. 147
Figure 68: Routing decision logic ............................................................................. 148
Figure 69: Routing rules decision logic ................................................................... 148
Figure 70: Post-Bureau Risk Category decision logic ............................................... 148
Figure 71: Post-bureau risk category table decision logic ....................................... 149
Figure 72: Pre-Bureau Affordability decision logic .................................................. 149
Figure 73: Post-Bureau Affordability decision logic ................................................ 150
Figure 74: Affordability calculation decision logic .................................................... 150
Figure 75: Credit contingency factor table decision logic ....................................... 151
Figure 76: Required Monthly Installment decision logic ........................................ 151
Figure 77: Installment calculation decision logic ...................................................... 151
Figure 78: Applicant Data input data sample ............................................................ 152
Figure 79: Requested Product input data sample ..................................................... 152
Figure 80: Bureau Data input data sample ............................................................... 152
Figure 81: Decision-making in BPMN .................................................................... 156
Table of Tables

Table 1: DRD components ............................................................................................................. 31
Table 2: Requirements connection rules ......................................................................................... 35
Table 3: DMNElement attributes and model associations ............................................................... 37
Table 4: Definitions attributes and model associations ................................................................. 39
Table 5: Import attributes and model associations ........................................................................ 40
Table 6: ElementCollection attributes and model associations .................................................... 41
Table 7: Decision attributes and model associations ...................................................................... 43
Table 8: BusinessContextElement attributes and model associations .......................................... 46
Table 9: PerformanceIndicator attributes and model associations ............................................... 46
Table 10: OrganisationalUnit attributes and model associations ................................................ 46
Table 11: BusinessKnowledgeModel attributes and model associations ..................................... 48
Table 12: InputData attributes and model associations .................................................................. 49
Table 13: KnowledgeSource attributes and model associations ................................................... 50
Table 14: InformationRequirement attributes and model associations ........................................ 51
Table 15: KnowledgeRequirement attributes and model associations .......................................... 52
Table 16: AuthorityRequirement attributes and model associations ............................................. 52
Table 17: Expression attributes and model associations ............................................................... 61
Table 18: ItemDefinition attributes and model associations ......................................................... 63
Table 19: InformationItem attributes and model associations ...................................................... 65
Table 20: LiteralExpression attributes and model associations ................................................... 66
Table 21: Invocation attributes and model associations .................................................................. 67
Table 22: Binding attributes and model associations ...................................................................... 68
Table 23: DecisionTable attributes and model associations ......................................................... 79
Table 24: Clause attributes and model associations ...................................................................... 81
Table 25: DecisionRule attributes and model associations ........................................................... 82
Table 26: Examples of decision tables ............................................................................................ 83
Table 27: FEEL range comparisons ............................................................................................... 98
Table 28: FEEL numbers and calculations .................................................................................... 99
Table 29: Context attributes and model association ..................................................................... 134
Table 30: **ContextEntry attributes and model associations** .......................................................... 134
Table 31: **FunctionDefinition attributes and model associations** ........................................ 135
Table 32: **List attributes and model associations** ................................................................. 135
Table 33: **Relation attributes and model associations** ......................................................... 136
Table 34: **BPMN tasks relevant to DMN** ............................................................................. 157
Table 35: **Mandatory requirements** .................................................................................... 161
Table 36: **Optional requirements** ....................................................................................... 164
Table 37: **Issues to be discussed** ....................................................................................... 165
Introduction

This section presents information regarding the RFP response.

- Submitting organizations
- Co-authors
- Submission contacts
- Acknowledgements
- Status of the document
- IPR and patents
- Typographical conventions
- Responses to RFP requirements
- Guide to the submission

Submitting organizations

The following companies are formal submitting members of OMG:

- Decision Management Solutions
- Escape Velocity
- FICO
- International Business Machines
- Oracle

Co-Authors

The following organizations have contributed to the development of this specification but are not formal submitters:

- KU Leuven
- Knowledge Partners International
- Model Systems
- TIBCO

Submission Contacts

Martin Chapman, Oracle, martin.chapman@oracle.com
Dave Ings, IBM, ings@ca.ibm.com

Acknowledgements

The following persons were members of the core team that contributed to the content specification:

In addition, the following persons contributed valuable ideas and feedback that improved the content and the quality of this specification: Bas Janssen, Robert Lario, Pete Rivett.
Status of the Document
This document is an initial specification for review and comment by OMG members.

IPR and Patents
The submitters intend to contribute this work to OMG on a RF on RAND basis.

Typographical Conventions
The type styles shown below are used in this document to distinguish programming statements from ordinary English. However, these conventions are not used in section headings where no distinction is necessary.

Times/Times New Roman - 12 pt.: Standard body text
Courier - 12 pt.: Name of modeling element (class or association)
Arial – 12pt.: syntax element.
Arial – 10 pt.: Examples and non-normative remarks
Note: *Italic* text also represents the name of a document, specification, or other publication.

Responses to RFP Requirements
See Annex C.

Guide to the Submission
Section 1 summarizes the goals of the specification.
Section 2 defines three levels of conformance with the specification: Conformance Level 1, Conformance Level 2 and Conformance Level 3.
Section 3 lists normative references.
Section 4 discusses the scope and uses of DMN and introduces the principal concepts, including the two levels of DMN: the decision requirements level and the decision logic level.
Section 5 defines the decision requirements level of DMN: the Decision Requirements Graph (DRG) and its notation as a Decision Requirements Diagram (DRD).
Section 6 introduces the principles by which decision logic may be associated with elements in a DRG: i.e. how the decision requirements level and decision logic level are related to each other.
Sections 7, 8 and 9 then define the decision logic level of DMN:
- Section 7 defines the notation and syntax of Decision Tables in DMN
- Section 8 defines S-FEEL: a subset of FEEL to support decision tables
- Section 9 defines the full syntax and semantics of FEEL: the default expression language used for the Decision Logic level of DMN.
Section 10 provides an example of DMN used to model automated decision-making in a simple business process.

Section 11 addresses exchange formats and provides references to machine-readable files (XSD and XMI).

The Annexes provide non-normative background information:

- Annex A discusses the relationship between DMN and BPMN
- Annex B suggests principles for encapsulating decision models as decision services
- Annex C provides cross-references between the requirements in the RFP and responses provided by this submission
- Annex D provides a glossary of terms.
1 Scope

The primary goal of DMN is to provide a common notation that is readily understandable by all business users, from the business analysts needing to create initial decision requirements and then more detailed decision models, to the technical developers responsible for automating the decisions in processes, and finally, to the business people who will manage and monitor those decisions. DMN creates a standardized bridge for the gap between the business decision design and decision implementation. DMN notation is designed to be useable alongside the standard BPMN business process notation.

Another goal is to ensure that decision models are interchangeable across organizations via an XML representation.

The submission team has brought forth expertise and experience from the existing decision modeling community and has sought to consolidate the common ideas from these divergent notations into a single standard notation.

2 Conformance

Software may claim compliance or conformance with DMN 1.0 if and only if the software fully matches the applicable compliance points as stated in the specification. Software developed only partially matching the applicable compliance points may claim that the software was based on this specification, but may not claim compliance or conformance with this specification.

The specification defines three levels of conformance, namely Conformance Level 1, Conformance Level 2 and Conformance Level 3.

An implementation claiming conformance to Conformance Level 1 is not required to support Conformance Level 2 or Conformance Level 3. An implementation claiming conformance to Conformance Level 2 is not required to support Conformance Level 3.

An implementation claiming conformance to Conformance Level 1 shall comply with all of the specifications set forth in sections 5 (Decision Requirements), 6 (Decision Logic) and 7 (Decision Table) of this document. An implementation claiming conformance to Conformance Level 1 is never required to interpret expressions (modeled as an Expression elements) in decision models. However, to the extent that an implementation claiming conformance to Conformance Level 1 provides an interpretation to an expression, that interpretation MUST be consistent with the semantics of expressions as specified in section 6.

An implementation claiming conformance to Conformance Level 2 shall comply with all of the specifications set forth in sections 5 (Decision Requirements), 6 (Decision Logic) and 7 (Decision Table) of this document. In addition it is required to interpret expressions in the simple expression language (S-FEEL) specified in section 8.

An implementation claiming conformance to Conformance Level 3 shall comply with all of the specifications set forth in sections 5 (Decision Requirements), 6 (Decision Logic), 7 (Decision Table) and 9 (Expression language) of this document. Notice that the simple expression language that is specified in section 8 is a subset of FEEL, and that, therefore, an implementation claiming conformance
to Conformance Level 3 can also claim conformance to Conformance Level 2 (and to Conformance Level 1).

In addition, an implementation claiming conformance to any of the three DMN 1.0 conformance levels shall comply with all of the requirements set forth in Section 2.1.

2.1 General conformance requirements

2.1.1 Visual appearance

A key element of DMN is the choice of shapes and icons used for the graphical elements identified in this specification. The intent is to create a standard visual language that all decision modelers will recognize and understand. An implementation that creates and displays decision model diagrams shall use the graphical elements, shapes, and markers illustrated in this specification.

There is flexibility in the size, color, line style, and text positions of the defined graphical elements, except where otherwise specified.

The following extensions to a DMN Diagram are permitted:

- New markers or indicators MAY be added to the specified graphical elements. These markers or indicators could be used to highlight a specific attribute of a DMN element or to represent a new subtype of the corresponding concept.

- A new shape representing a new kind of artifact may be added to a Diagram, but the new shape SHALL NOT conflict with the shape specified for any other DMN element or marker.

- Graphical elements MAY be colored, and the coloring may have specified semantics that extend the information conveyed by the element as specified in this standard.

- The line style of a graphical element MAY be changed, but that change SHALL NOT conflict with any other line style required by this specification.

An extension SHALL NOT change the specified shape of a defined graphical element or marker (e.g., changing a dashed line into a plain line, changing a square into a triangle, or changing rounded corners into squared corners).

2.1.2 Decision semantics

This specification defines many semantic concepts used in defining decisions and associates them with graphical elements, markers, and connections.

To the extent that an implementation provides an interpretation of some DMN diagram element as a semantic specification of the associated concept, the interpretation shall be consistent with the semantic interpretation herein specified.

2.1.3 Attributes and model associations

This specification defines a number of attributes and properties of the semantic elements represented by the graphical elements, markers, and connections. Some attributes are specified as mandatory, but have no representation or only optional representation. And some attributes are specified as optional.
For every attribute or property that is specified as mandatory, a conforming implementation SHALL provide some mechanism by which values of that attribute or property can be created and displayed. This mechanism SHALL permit the user to create or view these values for each DMN element specified to have that attribute or property.

Where a graphical representation for that attribute or property is specified as required, that graphical representation SHALL be used. Where a graphical representation for that attribute or property is specified as optional, the implementation MAY use either a graphical representation or some other mechanism.

If a graphical representation is used, it SHALL be the representation specified. Where no graphical representation for that attribute or property is specified, the implementation MAY use either a graphical representation or some other mechanism. If a graphical representation is used, it SHALL NOT conflict with the specified graphical representation of any other DMN element.

3 Normative References

3.1 Normative

BPMN 2.0
  [http://www.omg.org/spec/BPMN/2.0](http://www.omg.org/spec/BPMN/2.0)

RIF
- RIF Core dialect, H. Boley et al. (Eds.), W3C Recommendation, 22 June 2010.
  [http://www.w3.org/TR/rif-core/](http://www.w3.org/TR/rif-core/)
- RIF data types and built-ins 1.0, A. Polleres et al. (Eds.), W3C Recommendation, 22 June 2010.
  [http://www.w3.org/TR/rif-dtb/XML 1.0 (Second Edition)](http://www.w3.org/TR/rif-dtb/XML 1.0 (Second Edition))
- RIF production rule dialect, Ch. de Sainte Marie et al. (Eds.), W3C Recommendation, 22 June 2010.
  [http://www.w3.org/TR/rif-prd/](http://www.w3.org/TR/rif-prd/)

RFC-2119
- Key words for use in RFCs to Indicate Requirement Levels, S. Bradner, IETF RFC 2119, March 1997.

3.2 Non-normative

None
4 Introduction to DMN

4.1 Context

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, and (optionally) automated.

Decision-making is addressed from two different perspectives by existing modeling standards:

- Business process models (e.g. BPMN) can describe the coordination of decision-making within business processes by defining specific tasks or activities within which the decision-making takes place.
- Decision logic (e.g. PRR, PMML) can define the specific logic used to make individual decisions, for example as business rules, decision tables, or executable analytic models.

However, a number of authors (including members of the submission team) have observed that decision-making has an internal structure which is not conveniently captured in either of these modeling perspectives. Our intention is that DMN will provide a third perspective – the Decision Requirements Diagram – forming a bridge between business process models and decision logic models:

- Business process models will define tasks within business processes where decision-making is required to occur
- Decision Requirements Diagrams will define the decisions to be made in those tasks, their interrelationships, and their requirements for decision logic
- Decision logic will define the required decisions in sufficient detail to allow validation and/or automation.

Taken together, Decision Requirements Diagrams and decision logic can provide a complete decision model which complements a business process model by specifying in detail the decision-making carried out in process tasks. The relationships between these three aspects of modeling are shown in Figure 1.
The resulting connected set of models will allow detailed modeling of the role of business rules and analytic models in business processes, cross-validation of models, top-down process design and automation, and automatic execution of decision-making (e.g. by a business process management system calling a decision service deployed from a business rules management system).

Although Figure 1 shows a linkage between a business process model and a decision model for the purposes of explaining the relationship between DMN and other standards, it must be stressed that DMN is not dependent on BPMN, and its two levels—decision requirements and decision logic—may be used independently or in conjunction to model a domain of decision-making without any reference to business processes (see section 4.3).

DMN will provide constructs spanning both decision requirements and decision logic modeling. For decision requirements modeling, it defines the concept of a Decision Requirements Graph (DRG) comprising a set of elements and their connection rules, and a corresponding notation: the Decision

---

**Figure 1: Aspects of modeling**
 Requirements Diagram (DRD). For decision logic modeling it provides a language called FEEL for defining and assembling decision tables, calculations, if/then/else logic, simple data structures, and externally defined logic from Java and PMML into executable expressions with formally defined semantics. It also provides a notation for decision logic (“boxed expressions”) allowing components of the decision logic level to be drawn graphically and associated with elements of a Decision Requirements Diagram. The relationship between these constructs is shown in Figure 2.

Again, although Figure 2 depicts these decision modeling constructs as interlinked, it is possible to use them independently or in any combination. For example, it is possible to use DMN only to draw DRDs, or only to define decision tables, or only to write FEEL expressions.
Figure 2: DMN Constructs
4.2 Basic concepts

4.2.1 Decision requirements level

The word ‘decision’ has two definitions in common use: it may denote the act of choosing among multiple possible options; or it may denote the option that is chosen. In this specification, we adopt the former usage: a decision is the act of determining an output value (the chosen option), from a number of input values, using logic defining how the output is determined from the inputs. This decision logic may include one or more business knowledge models which encapsulate business know-how in the form of business rules, analytic models, or other formalisms. This basic structure, from which all decision models are built, is shown in Figure 3.

![Figure 3: Basic elements of a decision model](image)

Authorities may be defined for decisions or business knowledge models, which might be (for example) domain experts responsible for defining or maintaining them, or source documents from which business knowledge models are derived, or sets of test cases with which the decisions must be consistent. These are called knowledge sources (see Figure 4).

![Figure 4: Knowledge sources](image)

A decision is said to “require” its inputs in order to determine its output. The inputs may be input data, or the outputs of other decisions. (In either case they may be data structures, rather than just simple data items.) If the inputs of a decision Decision1 include the output of another decision Decision2, Decision1 “requires” Decision2. Decisions may therefore be connected in a network called a Decision Requirements Graph (DRG), which may be drawn as a Decision Requirements Diagram (DRD).
A DRD shows how a set of decisions depend on each other, on input data, and on business knowledge models. A simple example of a DRD with only two decisions is shown in Figure 5.

![Figure 5: A simple Decision Requirements Diagram (DRD)](image)

A decision may require multiple business knowledge models, and a business knowledge model may require multiple other business knowledge models, as shown in Figure 6. This will allow (for example) the modeling of complex decision logic by combining diverse areas of business knowledge, and the provision of alternative versions of decision logic for use in different situations.

![Figure 6: Combining business knowledge models](image)

DRGs and their notation as DRDs are specified in detail in section 5.

### 4.2.2 Decision logic level

The components of the decision requirements level of a decision model may be described, as they are above, using only business concepts. This level of description is often sufficient for business analysis of a domain of decision-making, to identify the business decisions involved, their interrelationships, the areas of business knowledge and data required by them, and the sources of the business knowledge. Using decision logic, the same components may be specified in greater detail, to capture a complete set of business rules and calculations, and (if desired) to allow the decision-making to be fully automated.

Decision logic may also provide additional information about how to display elements in the decision model. For example, the decision logic element for a decision table may specify whether to show the rules as rows or as columns. The decision logic element for a calculation may specify whether to line up terms vertically or horizontally.
The correspondence between concepts at the decision requirements level and the decision logic level is described below. Please note that in the figures below, as in Figure 1 and Figure 2, the grey ellipses and dotted lines are drawn only to indicate correspondences between concepts in different levels for the purposes of this introduction. They do not form part of the notation of DMN, which is formally defined in sections 5.2, 7.2 and 9.2. It is envisaged that implementations will provide facilities for moving between levels of modeling, such as “opening”, “drilling down” or “zooming in”, but DMN does not specify how this should be done.

At the decision logic level, every decision in a DRG is defined using a value expression which specifies how the decision’s output is determined from its inputs. At that level, the decision is considered to be the evaluation of the expression. The value expression may be notated using a boxed expression, as shown in Figure 7.

![Decision and corresponding value expression](image)

**Figure 7: Decision and corresponding value expression**

In the same way, at the decision logic level, a business knowledge model is defined using a value expression that specifies how an output is determined from a set of inputs. Value expressions may be encapsulated as functions, which may be invoked from decisions’ value expressions; business knowledge models are examples of such functions (but decision logic may also include functions which do not correspond to business knowledge models). The interpretation of business knowledge models as functions in DMN means that the combination of business knowledge models as in Figure 6 has the clear semantics of functional composition. The value expression of a business knowledge model may be notated using a boxed function, as shown in Figure 8.
A business knowledge model may contain any decision logic which is capable of being represented as a function. This will allow the import of many existing decision logic modeling standards (e.g. for business rules and analytic models) into DMN. An important format of business knowledge, specifically supported in DMN, is the Decision Table. Such a business knowledge model may be notated using a Decision Table, as shown in Figure 9.

In most cases, the logic of a decision is encapsulated into business knowledge models, and the value expression associated with the decision specifies how the business knowledge models are invoked, and how the results of their invocations are combined to compute the output of the decision. The decision’s value expression may also specify how the output is determined from its input entirely within itself, without invoking a business knowledge model: in that case, no business knowledge model is associated with the decision (neither at the decision requirements level nor at the decision logic level).

An expression language for defining decision logic in DMN, covering all the above concepts, is specified fully in section 9. This is FEEL: the Friendly Enough Expression Language. The notation for Decision Tables is specified in detail in section 7.
4.3 Scope and uses of DMN

Decision modeling is carried out by business analysts in order to understand and define the decisions used in a business or organization. Such decisions are typically operational decisions made in day-to-day business processes, rather than the strategic decision-making for which fewer rules and representations exist.

Three uses of DMN can be discerned in this context:

1. For modeling human decision-making
2. For modeling the requirements for automated decision-making
3. For implementing automated decision-making.

4.3.1 Modeling human decision-making

DMN may be used to model the decisions made by personnel within an organization. Human decision-making can be broken down into a network of interdependent constituent decisions, and modeled using a DRD. The decisions in the DRD would probably be described at quite a high level, using natural language rather than decision logic.

Knowledge sources may be defined to model governance of decision-making by people (e.g. a manager), regulatory bodies (e.g. an ombudsman), documents (e.g. a policy booklet) or bodies of legislation (e.g. a government statute). These knowledge sources may be linked together, for example to show that a decision is governed (a) by a set of regulations defined by a regulatory body, and (b) by a company policy document maintained by a manager.

Business knowledge models may be used to represent specific areas of business knowledge drawn upon when making decisions. This will allow DMN to be used as a tool for formal definition of requirements for knowledge management. Business knowledge models may be linked together to show the interdependencies between areas of knowledge (in a manner similar to that used in the existing technique of Knowledge Structure Mapping). Knowledge sources may be linked to the business knowledge models to indicate how the business knowledge is governed or maintained, for example to show that a set of business policies (the business knowledge model) is defined in a company policy document (the knowledge source).

In some cases it may be possible to define specific rules or algorithms for the decision-making. These may be modeled using decision logic (e.g. business rules or decision tables) to specify business knowledge models in the DRD, either descriptively (to record how decisions are currently made, or how they were made during a particular period of observation) or prescriptively (to define how decisions should be made, or will be made in the future).

Decision-making modeled in DMN may be mapped to tasks or activities within a business process modeled using BPMN. At a high level, a collaborative decision-making task may be mapped to a subset of decisions in a DRD representing the overall decision-making behavior of a group or department. At a more detailed level, it is possible to model the interdependencies between decisions made by a number of individuals or groups using BPMN collaborations: each participant in the decision-making is represented by a separate pool in the collaboration and a separate DRD in the decision model. Decisions in those DRDs are then mapped to tasks in the pools, and input data in the DRDs are mapped to the content of messages passing between the pools.
The combined use of BPMN and DMN thus provides a graphical language for describing multiple levels of human decision-making within an organization, from activities in business processes down to a detailed definition of decision logic. Within this context DMN models will describe collaborative organizational decisions, their governance, and the business knowledge required for them.

### 4.3.2 Modeling requirements for automated decision-making

The use of DMN for modeling the requirements for automated decision-making is similar to its use in modeling human decision-making, except that it is entirely prescriptive, rather than descriptive, and there is more emphasis on the detailed decision logic.

For full automation of decisions, the decision logic must be complete, i.e. capable of providing a decision result for any possible set of values of the input data.

However, partial automation is more common, where some decision-making remains the preserve of personnel. Interactions between human and automated decision-making may be modeled using collaborations as above, with separate pools for human and automated decision-makers, or more simply by allocating the decision-making to separate tasks in the business process model, with user tasks for human decision-making and business rule tasks for automated decision-making. So, for example, an automated business rules task might decide to refer some cases to a human reviewer; the decision logic for the automated task needs to be specified in full but the reviewer’s decision-making could be left unspecified.

Once decisions in a DRD are mapped to tasks in a BPMN business process flow, it is possible to validate across the two levels of models. For example, it is possible to verify that all input data in the DRDs are provided by previous tasks in the business process, and that the business process uses the results of decisions only in subsequent tasks or gateways. DMN models the relationships between Decisions and Business Processes so that the Decisions that must be made for a Business Process to complete can be identified and so that the specific decision-making tasks that perform or execute a Decision can be specified. In DMN 1.0 no formal mapping of DMN ItemDefinitions or DMN InputData to BPMN DataObject is proposed but an implementation could include such a check in a situation where such a mapping could be determined.

Together, BPMN and DMN therefore allow specification of the requirements for automated decision-making and its interaction with human decision making within business processes. These requirements may be specified at any level of detail, or at all levels. The three-tier mapping between business process models, DRDs and decision logic will allow the definition of these requirements to be supported by model-based computer-aided design tools.

### 4.3.3 Implementing automated decision-making

If all decisions and business knowledge models are fully specified using decision logic, it becomes possible to execute decision models.

One possible scenario is the use of “decision services” deployed from a Business Rules Management System (BRMS) and called by a Business Process Management System (BPMS). A decision service encapsulates the decision logic supporting a DRD, providing interfaces that correspond to subsets of input data and decisions within the DRD. When called with a set of input data, the decision service will evaluate the specified decisions and return their results. The constraint in DMN that all decision logic
is free of side-effects means that decision services will comply with SOA principles, simplifying system design.

The structure of a decision model, as visualized in the DRD, may be used as a basis for planning an implementation project. Specific project tasks may be included to cover the definition of decision logic (e.g. rule discovery using human experts, or creation of analytic models), and the implementation of components of the decision model.

Some decision logic representing the business knowledge encapsulated in decision services needs to be maintained over time by personnel responsible for the decisions, using special “knowledge maintenance interfaces”. DMN supports the effective design and implementation of knowledge maintenance interfaces: any business knowledge requiring maintenance should be modeled as business knowledge models in the DRD, and the responsible personnel as knowledge sources. DRDs then provide a specification of the required knowledge maintenance interfaces and their users, and the decision logic specifies the initial configuration of the business knowledge to be maintained.

Other decision logic needs to be refreshed by regular analytic modeling. The representation of business knowledge models as functions in DMN makes the use of analytic models in decision services very simple: any analytic model capable of representation as a function may be directly called by or imported into a decision service.

4.3.4 Combining applications of modeling

The three contexts described above are not mutually exclusive alternatives; a large process automation project might use DMN in all three ways.

First, the decision-making within the existing process might be modeled, to identify the full extent of current decision making and the areas of business knowledge involved. This “as-is” analysis provides the baseline for process improvement.

Next, the process might be redesigned to make the most effective use of both automated and human decision-making, often using collaboration between the two (e.g. using automated referrals to human decision-makers, or decision support systems which advise or constrain the user). Such a redesign involves modeling the requirements for the decision-making to occur in each process task and the roles and responsibilities of individuals or groups in the organization. This model provides a “to-be” specification of the required process and the decision-making it coordinates.

Comparison of the “as-is” and “to-be” models will indicate requirements not just for automation technology, but for change management: changes in the roles and responsibilities of personnel, and training to support new or modified business knowledge.

Finally, the “to-be” model will be implemented as executable system software. Provided the decision logic is fully specified in FEEL and/or other external logic (e.g. externally defined Java methods or PMML models), components of the decision model may be implemented directly as software components.

DMN does not prescribe any particular methodology for carrying out the above activities; it only supports the models used for them.
5 Decision Requirements (DRG and DRD)

5.1 Introduction
The decision requirements level of a decision model in DMN consists of a Decision Requirements Graph (DRG) depicted in one or more Decision Requirements Diagrams (DRDs).

A DRG models a domain of decision-making, showing the most important elements involved in it and the dependencies between them. The elements modeled are decisions, areas of business knowledge, sources of business knowledge, and input data:

- A **Decision** element denotes the act of determining an output from a number of inputs, using decision logic which may reference one or more Business Knowledge Models.
- A **Business Knowledge Model** element denotes a function encapsulating business knowledge, e.g. as business rules, a decision table, or an analytic model.
- An **Input Data** element denotes information used as an input by one or more Decisions.
- A **Knowledge Source** element denotes an authority for a Business Knowledge Model or Decision.

The dependencies between these elements express three kinds of requirements: information, knowledge and authority:

- An **Information Requirement** denotes Input Data or Decision output being used as input to a Decision.
- A **Knowledge Requirement** denotes the invocation of a Business Knowledge Model by the decision logic of a Decision.
- An **Authority Requirement** denotes the dependence of a DRG element on a Knowledge Source, or the dependence of a Knowledge Source on a DRG element.

These components are summarized in Table 1 and described in more detail in section 5.2.

A DRG is a graph composed of elements connected by requirements, and is self-contained in the sense that all the modeled requirements for any Decision in the DRG (its immediate sources of information, knowledge and authority) are present in the same DRG. It is important to distinguish this complete definition of the DRG from a DRD presenting any particular view of it, which may be a partial or filtered display: see section 5.2.4.

5.2 Notation
The notation for all components of a DRD is summarized in Table 1 and described in more detail below.
### Table 1: DRD components

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
<th>Notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elements</td>
<td><strong>Decision</strong>&lt;br&gt;A decision denotes the act of determining an output from a number of inputs, using decision logic which may reference one or more business knowledge models.</td>
<td><img src="image" alt="Decision" /></td>
</tr>
<tr>
<td><strong>Business Knowledge Model</strong></td>
<td><strong>A business knowledge model denotes a function encapsulating business knowledge, e.g. as business rules, a decision table, or an analytic model.</strong></td>
<td><img src="image" alt="Business knowledge" /></td>
</tr>
<tr>
<td>Input Data</td>
<td><strong>An input data element denotes information used as an input by one or more decisions. When enclosed within a knowledge model, it denotes the parameters to the knowledge model.</strong></td>
<td><img src="image" alt="Input data" /></td>
</tr>
<tr>
<td>Knowledge Source</td>
<td><strong>A knowledge source denotes an authority for a business knowledge model or decision.</strong></td>
<td><img src="image" alt="Knowledge source" /></td>
</tr>
<tr>
<td>Requirements</td>
<td><strong>Information Requirement</strong>&lt;br&gt;An information requirement denotes input data or a decision output being used as one of the inputs of a decision</td>
<td><img src="image" alt="Information Requirement" /></td>
</tr>
<tr>
<td></td>
<td><strong>Knowledge Requirement</strong>&lt;br&gt;A knowledge requirement denotes the invocation of a business knowledge model</td>
<td><img src="image" alt="Knowledge Requirement" /></td>
</tr>
<tr>
<td></td>
<td><strong>Authority Requirement</strong>&lt;br&gt;An authority requirement denotes the dependence of a DRD element on a knowledge source, or the dependence of a knowledge source on input data</td>
<td><img src="image" alt="Authority Requirement" /></td>
</tr>
</tbody>
</table>

### 5.2.1 DRD Elements

#### 5.2.1.1 Decision notation

A Decision is represented in a DRD as a rectangle, normally drawn with solid lines, as shown in Table 1. Implementations MUST be able to label each Decision by displaying its Name, and MAY be able to label it by displaying other properties such as its Question or Description. If displayed, the label MUST be clearly inside the shape of the DRD element.

If the Listed Input Data option is exercised (see 5.2.1.3), all the Decision’s requirements for Input Data MUST be listed beneath the Decision’s label and separated from it by a horizontal line, as shown in Figure 10. The listed Input Data names MUST be clearly inside the shape of the DRD element.
The properties of a Decision are listed and described in 5.3.6.

**5.2.1.2 Business Knowledge Model notation**

A Business Knowledge Model is represented in a DRD as a rectangle with two clipped corners, normally drawn with solid lines, as shown in Table 1. Implementations MUST be able to label each Business Knowledge Model by displaying its Name, and MAY be able to label it by displaying other properties such as its Description. If displayed, the label MUST be clearly inside the shape of the DRD element.

The properties of a Business Knowledge Model are listed and described in 5.3.7.

**5.2.1.3 Input Data notation**

An Input Data element is represented in a DRD as a shape with two parallel straight sides and two semi-circular ends, normally drawn with solid lines, as shown in Table 1. Implementations MUST be able to label each Input Data element by displaying its Name, and MAY be able to label it by displaying other properties such as its Description. If displayed, the label MUST be clearly inside the shape of the DRD element.

An alternative compliant way to display requirements for Input Data, especially useful when DRDs are large or complex, is that Input Data are not drawn as separate notational elements in the DRD, but are instead listed on those Decision elements which require them. For convenience in this specification this is called the “Listed Input Data” option. Implementations MAY offer this option. Figure 11 shows two equivalent DRDs, one drawing Input Data elements, the other exercising the Listed Input Data option. Note that if an Input Data element is not displayed it MUST be listed on all Decisions which require it (unless it is deliberately hidden as discussed in 5.2.4).
The properties of an Input Data element are listed and described in 5.3.9.

5.2.1.4 Knowledge Source notation

A Knowledge Source is represented in a DRD as a shape with three straight sides and one wavy one, normally drawn with solid lines, as shown in Table 1. Implementations MUST be able to label each Knowledge Source element by displaying its Name, and MAY be able to label it by displaying other properties such as its Description. If displayed, the label MUST be clearly inside the shape of the DRD element.

The properties of a Knowledge Source element are listed and described in 5.3.10.

5.2.2 DRD Requirements

5.2.2.1 Information Requirement notation

Information Requirements may be drawn from Input Data elements to Decisions, and from Decisions to other Decisions. They represent the dependency of a Decision on information from input data or the results of other Decisions. They may also be interpreted as data flow: a DRD displaying only Decisions, Input Data and Information Requirements is equivalent to a dataflow diagram showing the communication of information between those elements at evaluation time. The Information Requirements of a valid DRG form a directed acyclic graph.

An Information Requirement is represented in a DRD as an arrow drawn with a solid line and a solid arrowhead, as shown in Table 1. The arrow is drawn in the direction of information flow, i.e. towards the Decision that requires the information.
5.2.2.2 Knowledge Requirement notation

Knowledge Requirements may be drawn from Business Knowledge Models to Decisions, and from Business Knowledge Models to other Business Knowledge Models. They represent the invocation of business knowledge when making a decision. They may also be interpreted as function calls: a DRD displaying only Decisions, Business Knowledge Models and Knowledge Requirements is equivalent to a function hierarchy showing the function calls involved in evaluating the Decisions. The Knowledge Requirements of a valid DRG form a directed acyclic graph.

A Knowledge Requirement is represented in a DRD as an arrow drawn with a dashed line and an open arrowhead, as shown in Table 1. The arrows are drawn in the direction of the information flow of the result of evaluating the function, i.e. toward the element that requires the business knowledge.

5.2.2.3 Authority Requirement notation

Authority Requirements may be used in two ways:

a) They may be drawn from Knowledge Sources to Decisions, Business Knowledge Models and other Knowledge Sources, where they represent the dependence of the DRD element on the knowledge source. This would typically be used to record the fact that a set of business rules must be derived from or consistent with a published document, e.g. a piece of legislation or a statement of business policy. An example of this use of Knowledge Sources is shown in Figure 12.

b) They may be drawn from Input Data and Decisions to Knowledge Sources, where, in conjunction with use (a), they represent the derivation of Business Knowledge Models from instances of Input Data and Decision results, using analytics. The Knowledge Source typically represents the analytic model (or modeling process); the Business Knowledge Model represents the executable logic generated from or dependent on the model. An example of this use of a Knowledge Source is shown in Figure 13.
An Authority Requirement is represented in a DRD as an arrow drawn with a dashed line and a filled circular head, as shown in Table 1. The arrows are drawn from the source of authority to the element governed by it.

5.2.3 Connection rules

The rules governing the permissible ways of connecting elements with requirements in a DRD are described in Section 5.2.2 above and summarized in Table 2.

Note that no requirements may be drawn terminating in Input Data.

Table 2: Requirements connection rules

<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decision</td>
<td>Decision</td>
</tr>
<tr>
<td>Business knowledge model</td>
<td>Business Knowledge Model</td>
</tr>
<tr>
<td>Knowledge Source</td>
<td>Knowledge Source</td>
</tr>
<tr>
<td>Input Data</td>
<td>Input Data</td>
</tr>
</tbody>
</table>

5.2.4 Partial views and hidden information

The metamodel (see section 5.3) provides properties for each of the DRG elements which would not normally be displayed on the DRD, but provide additional information about their nature or function. For example, for a Decision these include properties specifying which BPMN processes and tasks
make use of the Decision. Implementations MUST provide facilities for specifying and displaying such properties.

For any significant domain of decision-making a DRD representing the complete DRG may be a large and complex diagram. Implementations MAY provide facilities for displaying DRDs which are partial or filtered views of the DRG, e.g. by hiding categories of elements, or hiding or collapsing areas of the network. DMN does not specify how such views should be notated, but whenever information is hidden implementations SHOULD provide a clear visual indication that this is the case.

Two examples of DRDs providing partial views of a DRG are shown in Figure 14: DRD 1 shows only the immediate requirements of a single decision; DRD 2 shows only Information Requirements and the elements they connect. In this example, for the purposes of illustration only, the approach taken is to use a fine dashed outline for any element with some hidden requirements.

![Figure 14: DRDs as partial views of a DRG](image)

In DMN v1.0, DRDs are not represented in the metamodel and may therefore not be interchanged; a set of definitions comprising a DRG may be interchanged, and the recipient may generate any desired DRD from them which is supported by the receiving implementation.
5.3 Metamodel

5.3.1 DMN Element metamodel

DMNElement is the abstract super class for the decision requirement model elements. It provides the mandatory attribute id and the optional attributes name and description, which all are Strings, and which other elements will inherit. The id of a DMNElement element MUST be unique within the containing element.

DMNElement has three abstract specializations: Expression, BusinessContextElement and DRGEElement and four concrete specializations: Definitions, ItemDefinition, InformationItem and ElementCollection.

Table 3 presents the attributes and model associations of the DMNElement element.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name: String</td>
<td>The name of this element.</td>
</tr>
<tr>
<td>id: String</td>
<td>The string that identifies this DMNElement uniquely within its containing Definitions element.</td>
</tr>
<tr>
<td>description:</td>
<td>A description of this element.</td>
</tr>
</tbody>
</table>
5.3.2 Definitions metamodel

The Definitions class is the outermost containing object for all elements of a DMN decision model. It defines the scope of visibility and the namespace for all contained elements. Elements that are contained in an instance of Definitions have their own defined life-cycle and are not deleted with the deletion of other elements. The interchange of DMN files will always be through one or more Definitions.

Definitions is a kind of DMNElement, from which an instance of Definitions inherits the id and optional name and description attributes, which are Strings.

An instance of Definitions has a namespace, which is a String. The namespace identifies the default target namespace for the elements in the Definitions and follows the convention established by XML Schema.

An instance of Definitions may specify an expressionLanguage, which is a String that identifies the default expression language used in elements within the scope of this Definitions. This value may be overridden on each individual LiteralExpression. The language MUST be specified in a URI format. The Default is FEEL (section 9).

An instance of Definitions may specify a typeLanguage, which is a String that identifies the default type language used in elements within the scope of this Definitions. For example, a typeLanguage value of “http://www.w3.org/2001/XMLSchema” indicates that the data structures defined within that Definitions are, by default, in the form of XML Schema types. If unspecified, the default is FEEL. This value may be overridden on each individual ItemDefinition. The typeLanguage MUST be specified in a URI format.

An instance of Definitions is composed of zero or more drgElements, which are instances of DRGElement, zero or more collections, which are instances of ElementCollection, zero or more itemDefinition, which are instances of ItemDefinition and of zero or more businessContextElement, which are instances of BusinessContextElement.
It may contain any number of associated import, which are instances of Import. Imports are used to import elements defined outside of this Definitions, e.g. in other Definitions elements, and to make them available for use by elements in this Definitions.

Definitions inherits all the attributes and model associations from DMNElement. Table 4 presents the additional attributes and model associations of the Definitions element.

Table 4: Definitions attributes and model associations

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>namespace: String</td>
<td>This attribute identifies the namespace associated with this Definitions and follows the convention established by XML Schema.</td>
</tr>
<tr>
<td>expressionLanguage: String [0..1]</td>
<td>This attribute identifies the expression language used in LiteralExpressions within the scope of this Definitions. The Default is FEEL (section 9). This value MAY be overridden on each individual LiteralExpression. The language MUST be specified in a URI format.</td>
</tr>
<tr>
<td>typeLanguage: String [0..1]</td>
<td>This attribute identifies the type language used in LiteralExpressions within the scope of this Definitions. The Default is FEEL (section 9). This value MAY be overridden on each individual ItemDefinition. The language MUST be specified in a URI format.</td>
</tr>
<tr>
<td>itemDefinition: ItemDefinition [*]</td>
<td>This attribute lists the instances of ItemDefinition that are contained in this Definitions.</td>
</tr>
<tr>
<td>drgElement: DRGEelement [*]</td>
<td>This attribute lists the instances of DRGEElement that are contained in this Definitions.</td>
</tr>
<tr>
<td>businessContextElement: BusinessContextElement [*]</td>
<td>This attribute lists the instances of BusinessContextElement that are contained in this Definitions.</td>
</tr>
<tr>
<td>collection ElementCollection [*]</td>
<td>This attribute lists the instances of ElementCollection that are contained in this Definitions.</td>
</tr>
<tr>
<td>import: Import [*]</td>
<td>This attribute is used to import externally defined elements and make them available for use by elements in this Definitions.</td>
</tr>
</tbody>
</table>
5.3.3 Import metamodel

The Import class is used when referencing external elements, either DMN DRGElement instances contained in other Definitions elements, or non-DMN elements, such as an XML Schema or a PMML file. Imports must be explicitly defined.

An instance of Import has an importType, which is a String that specifies the type of import associated with the element. For example, a value of “http://www.w3.org/2001/XMLSchema” indicates that the imported element is an XML schema. A value of <DMN namespace> indicates that the imported element is a DMN Definitions element.

The location of the imported element may be specified by associating an optional locationURI with an instance of Import. The locationURI is a String that MUST be in URI format.

An instance of Import has a namespace, which is a String that identifies the namespace of the imported element.

Table 5 presents the attributes and model associations of the Import element.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>importType: String</td>
<td>Specifies the style of import associated with this Import.</td>
</tr>
<tr>
<td>locationURI: String [0..1]</td>
<td>Identifies the location of the imported element. MUST be in URI format.</td>
</tr>
<tr>
<td>namespace: String</td>
<td>Identifies the namespace of the imported element.</td>
</tr>
</tbody>
</table>

5.3.4 Element Collection metamodel

The ElementCollection class is used to define named groups of DRGElement instances.

ElementCollections may be used for any purpose relevant to an implementation, for example:

- To identify the requirements subgraph of a set one or more decisions (i.e. all the elements in the closure of the requirements of the set)
- To identify the elements to be depicted on a DRD.

ElementCollection is a kind of DMNElement, from which an instance of ElementCollection inherits the id and optional name and description attributes, which are Strings. The id of a ElementCollection element MUST be unique within the containing instance of Definitions.

An ElementCollection element has any number of associated drgElements, which are the instances of DRGElement that this ElementCollection defines together as a group. Notice that an ElementCollection element must reference the instances of DRGElement that it collects, not contain them: instances of DRGElement can only be contained in Definitions elements.
ElementCollection inherits all the attributes and model associations from DMNElement. Table 6 presents the additional attributes and model associations of the ElementCollection element.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>drgElement: DRGElement [*]</td>
<td>This attribute lists the instances of DRGElement that this ElementCollection groups.</td>
</tr>
</tbody>
</table>

### 5.3.5 DRG Element metamodel

DRGElement is the abstract super class for all DMN elements that are contained within Definitions and that have a graphical representation in a DRD. All the elements of a DMN decision model that are not contained directly in a Definitions element (specifically: all three kinds of requirement, bindings, clause and decision rules, import, and objective) MUST be contained in an instance of DRGElement, or in a model element that is contained in an instance of DRGElement, recursively.

The concrete specializations of DRGElement are Decision, InputData, BusinessKnowledgeModel and KnowledgeSource.

DRGElement is a specialization of DMNElement, from which it inherits the id and optional name and description attributes. The id of a DRGElement element MUST be unique within the containing instance of Definitions.

A Decision Requirements Diagram (DRD) is the diagrammatic representation of one or more instances of DRGElement and their information, knowledge and authority requirement relations. The instances of DRGElement are represented as the vertices in the diagram; the edges represent instances of InformationRequirement, KnowledgeRequirement or AuthorityRequirement (see sections 5.3.11, 5.3.12 and 5.3.13). The connection rules are specified in section 5.2.3).

DRGElement inherits all the attributes and model associations of DMNElement. It does not define additional attributes and model associations of the DRGElement element.
5.3.6 Decision metamodel

In DMN 1.0, the class Decision is used to model a decision. Decision is a concrete specialization of DRGElemet and it inherits the mandatory id and optional name and description from DMNElement.

In addition, it may have a question and allowedAnswers, which are all Strings. The optional description attribute is meant to contain a brief description of the decision-making embodied in the Decision. The optional question attribute is meant to contain a natural language question that characterizes the Decision such that the output of the Decision is an answer to the question. The optional allowedAnswers attribute is meant to contain a natural language description of the answers allowed for the question such as Yes/No, a list of allowed values, a range of numeric values etc.

In a DRD, an instance of Decision is represented by a decision diagram element.
A Decision element is composed of an optional decisionLogic, which is an instance of Expression, and of zero or more informationRequirement, knowledgeRequirement and authorityRequirement elements, which are instances of InformationRequirement, KnowledgeRequirement and AuthorityRequirement, respectively.

The requirement subgraph of a Decision element is the directed graph composed of the Decision element itself, its informationRequirements, its knowledgeRequirements, and the union of the requirement subgraphs of each requiredDecision or requiredKnowledge element: that is, the requirement subgraph of a Decision element is the closure of the informationRequirement, requiredInput, requiredDecision, knowledgeRequirement and requiredKnowledge associations starting from that Decision element.

An instance of Decision – that is, the model of a decision – is said to be well-formed if and only if all of its informationRequirement and knowledgeRequirement elements are well-formed. That condition entails, in particular, that the requirement subgraph of a Decision element MUST be acyclic, that is, that a Decision element MUST not require itself, directly or indirectly.

Besides its logical components: information requirements, decision logic etc, the model of a decision may also document a business context for the decision (see section 5.3.7 and Figure 18).

In DMN 1.0, the business context for an instance of Decision is defined by its association with any number of supportedObjectives, which are instances of Objective as defined in OMG BMM, any number of impactedPerformanceIndicators, which are instances of PerformanceIndicator, any number of decisionMaker and any number of decisionOwner, which are instances of OrganisationalUnit.

Decision inherits all the attributes and model associations from DRGElement. Table 7 presents the additional attributes and model associations of the Decision class.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>question: String [0..1]</td>
<td>A natural language question that characterizes the Decision such that the output of the Decision is an answer to the question.</td>
</tr>
<tr>
<td>allowedAnswers: String [0..1]</td>
<td>A natural language description of the answers allowed for the question such as Yes/No, a list of allowed values, a range of numeric values etc.</td>
</tr>
<tr>
<td>decisionLogic: Expression [0..1]</td>
<td>The instance of Expression that represents the decision logic for this Decision.</td>
</tr>
<tr>
<td>Attribute</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>informationRequirement:</td>
<td>InformationRequirement [*] This attribute lists the instances of InformationRequirement that compose this Decision.</td>
</tr>
<tr>
<td>knowledgeRequirement:</td>
<td>KnowledgeRequirement [*] This attribute lists the instances of KnowledgeRequirement that compose this Decision.</td>
</tr>
<tr>
<td>authorityRequirement:</td>
<td>AuthorityRequirement [*] This attribute lists the instances of AuthorityRequirement that compose this Decision.</td>
</tr>
<tr>
<td>supportedObjective:</td>
<td>BMM::Objective [*] This attribute lists the instances of BMM::Objective that are supported by this Decision.</td>
</tr>
<tr>
<td>impactedPerformanceIndicator:</td>
<td>PerformanceIndicator [*] This attribute lists the instances of PerformanceIndicator that are impacted by this Decision.</td>
</tr>
<tr>
<td>decisionMaker:</td>
<td>OrganisationalUnit [*] The instances of OrganisationalUnit that make this Decision.</td>
</tr>
<tr>
<td>decisionOwner:</td>
<td>OrganisationalUnit [*] The instances of OrganisationalUnit that own this Decision.</td>
</tr>
<tr>
<td>usingProcesses:</td>
<td>BPMN::process [*] This attribute lists the instances of BPMN::process that require this Decision to be made.</td>
</tr>
<tr>
<td>usingTasks:</td>
<td>BPMN::task [*] This attribute lists the instances of BPMN::task that make this Decision.</td>
</tr>
</tbody>
</table>
5.3.7 Business Context Element metamodel

The abstract class `BusinessContextElement`, and its concrete specializations `PerformanceIndicator` and `OrganizationUnit` are placeholders, anticipating a definition to be adopted from other OMG meta-models, such as OMG OSM when it is further developed.

In DMN 1.0, `BusinessContextElement` is a specialization of `DMNElement`, from which it inherits the `id` and optional `name`, a description attributes.

In addition, instances of `BusinessContextElements` may have an URI, which is a Strings that must be in URI format, and

- an instance of `PerformanceIndicator` references any number of `implyingDecision`, which are the Decision elements that impact it;
• an instance of OrganisationalUnit references any number of decisionMade and of decisionOwned, which are the Decision elements that model the decisions that the organization unit makes or owns.

BusinessContextElement inherits all the attributes and model associations from DMNElement. Table 8 presents the additional attributes and model associations of the BusinessContextElement class.

Table 8: BusinessContextElement attributes and model associations

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>URI: String [0..1]</td>
<td>The URI of this BusinessContextElement.</td>
</tr>
</tbody>
</table>

PerformanceIndicator inherits all the attributes and model associations from BusinessContextElement. Table 9 presents the additional attributes and model associations of the PerformanceIndicator class.

Table 9: PerformanceIndicator attributes and model associations

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>impactingDecision: Decision[*]</td>
<td>This attribute lists the instances of Decision that impact this PerformanceIndicator.</td>
</tr>
</tbody>
</table>

OrganisationalUnit inherits all the attributes and model associations from BusinessContextElement. Table 10 presents the additional attributes and model associations of the OrganisationalUnit class.

Table 10: OrganisationalUnit attributes and model associations

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>decisionMade: Decision[*]</td>
<td>This attribute lists the instances of Decision that are made by this OrganisationalUnit.</td>
</tr>
<tr>
<td>decisionOwned: Decision[*]</td>
<td>This attribute lists the instances of Decision that are owned by this OrganisationalUnit.</td>
</tr>
</tbody>
</table>
5.3.8 Business Knowledge Model metamodel

![BusinessKnowledgeModel class diagram](image)

The business knowledge models that are associated with a decision are reusable modular expressions of all or part of their decision logic.

In DMN 1.0, the class `BusinessKnowledgeModel` is used to model a business knowledge model. `BusinessKnowledgeModel` is a concrete specialization of `DRGElement` and it inherits the mandatory `id` and optional `name` and `description` attributes from `DMNElement`.

In a DRD, an instance of `BusinessKnowledgeModel` is represented by a **business knowledge model** diagram element.

A `BusinessKnowledgeModel` element may have zero or more `knowledgeRequirement`, which are instances of `KnowledgeRequirement`, and zero or more `authorityRequirement`, which are instances of `AuthorityRequirement`.

The **requirement subgraph** of a `BusinessKnowledgeModel` element is the directed graph composed of the `BusinessKnowledgeModel` element itself, its `knowledgeRequirement` elements, and the union of the requirement subgraphs of all the `requiredKnowledge` elements that are referenced by its `knowledgeRequirements`.

An instance of `BusinessKnowledgeModel` is said to be **well-formed** if and only if, either it does not have any `knowledgeRequirement`, or all of its `knowledgeRequirement` elements are well-formed. That condition entails, in particular, that the requirement subgraph of a
BusinessKnowledgeModel element MUST be acyclic, that is, that a BusinessKnowledgeModel element MUST not require itself, directly or indirectly.

At the decision logic level, a BusinessKnowledgeModel element defines a function. It may be composed of an associated body, which is an instance of Expression and of zero or more parameter, which are instances of InformationItem. The body that is associated with a BusinessKnowledgeModel element is the reusable module of decision logic that is represented by this BusinessKnowledgeModel element. The parameters in a BusinessKnowledgeModel element are the inputVariables that are referenced by its body.

BusinessKnowledgeModel inherits all the attributes and model associations from DRGElement. Table 11 presents the additional attributes and model associations of the BusinessKnowledgeModel class.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>body: Expression [0..1]</td>
<td>The instance of Expression that describes the logic represented by this BusinessKnowledgeModel, that is, the body of the function that it defines.</td>
</tr>
<tr>
<td>parameter: InformationItem [*]</td>
<td>This attribute lists the instances of InformationItem that model the parameters of the function that this BusinessKnowledgeModel defines.</td>
</tr>
<tr>
<td>knowledgeRequirement: KnowledgeRequirement [*]</td>
<td>This attribute lists the instances of KnowledgeRequirement that compose this BusinessKnowledgeModel.</td>
</tr>
<tr>
<td>authorityRequirement: AuthorityRequirement [*]</td>
<td>This attribute lists the instances of AuthorityRequirement that compose this BusinessKnowledgeModel.</td>
</tr>
</tbody>
</table>
5.3.9 Input Data metamodel

DMN 1.0 uses the class `InputData` to model the inputs of a decision whose values are defined outside of the decision model.

`InputData` is a concrete specialization of `DRGElement` and it inherits the mandatory `id` and optional `name` and `description` from `DMNElement`.

Instances of `InputData` may reference an `itemDefinition`, which is an `ItemDefinition` element that specifies the type of data that is this `InputData` represents.

In a DRD, an instance of `InputData` is represented by an `input data` diagram element. An `InputData` element does not have a `requirement subgraph`, and it is always well-formed.

`InputData` inherits all the attributes and model associations from `DRGElement`. Table 12 presents the additional attributes and model associations of the `InputData` class.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>itemDefinition</code></td>
<td>ItemDefinition [0..1] The instance of ItemDefinition that describes the data type expected for this InputData.</td>
</tr>
</tbody>
</table>

5.3.10 Knowledge Source metamodel

In DMN 1.0, the class `KnowledgeSource` is used to model authoritative knowledge sources in a decision model.
In a DRD, an instance of KnowledgeSource is represented by a knowledge source diagram element.

KnowledgeSource is a concrete specialization of DRGElement, and thus of DMNElement, from which it inherits the mandatory id and optional name and description from DMNElement and the mandatory Id from DRGElement. In addition, a KnowledgeSource has a locationURI, which is a String that MUST be specified in a URI format. It has a type, which is a String, and an owner, which is <an instance of OrganisationalUnit?>.

A KnowledgeSource element is also composed of zero or more authorityRequirement element, which are instances of AuthorityRequirement.

KnowledgeSource inherits all the attributes and model associations from DRGElement. Table 13 presents the attributes and model associations of the KnowledgeSource class.

Table 13: KnowledgeSource attributes and model associations

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>locationURI: String [0..1]</td>
<td>The URI where this KnowledgeSource is located. The locationURI MUST be specified in a URI format.</td>
</tr>
<tr>
<td>type: String [0..1]</td>
<td>The type of this KnowledgeSource.</td>
</tr>
<tr>
<td>owner: OrganisationalUnit [0..1]</td>
<td>The owner of this KnowledgeSource.</td>
</tr>
<tr>
<td>authorityRequirement: AuthorityRequirement [*]</td>
<td>This attribute lists the instances of AuthorityRequirement that contribute to this KnowledgeSource.</td>
</tr>
</tbody>
</table>

5.3.11 Information Requirement metamodel

The class InformationRequirement is used to model an information requirement, as represented by a plain arrow in a DRD.

An InformationRequirement element is a component of a Decision element, and it associates that requiring Decision element with a requiredDecision element, which is an instance of Decision, or a requiredInput element, which is an instance of InputData.

An InformationRequirement element is composed of a variable, which is an instance of InformationItem, and that represents the InformationRequirement element at the decision logic level.

Notice that an InformationRequirement element must reference the instance of Decision or InputData that it associates with the requiring Decision element, not contain it: instances of Decision or InputData can only be contained in Definitions elements.
An instance of InformationRequirement is said to be well-formed if and only if all of the following are true:

- it references a requiredDecision or a requiredInput element, but not both,
- the referenced requiredDecision or requiredInput element is well-formed,
- and the Decision element that contains the instance of InformationRequirement is not in the requirement subgraph of the referenced requiredDecision element, if this InformationRequirement element references one.

Table 14 presents the attributes and model associations of the InformationRequirement element.

Table 14: InformationRequirement attributes and model associations

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>requiredDecision: Decision [0..1]</td>
<td>The instance of Decision that this InformationRequirement associates with its containing Decision element.</td>
</tr>
<tr>
<td>requiredInput: InputData [0..1]</td>
<td>The instance of InputData that this InformationRequirement associates with its containing Decision element.</td>
</tr>
<tr>
<td>variable: InformationItem</td>
<td>The instance of InformationItem that represents this InformationRequirement in the logic of the requiring Decision.</td>
</tr>
</tbody>
</table>

5.3.12 Knowledge Requirement metamodel

The class KnowledgeRequirement is used to model a knowledge requirement, as represented by a dashed arrow in a DRD.

A KnowledgeRequirement element is a component of a Decision element or of a BusinessKnowledgeModel element, and it associates that requiring Decision or BusinessKnowledgeModel element with a requiredKnowledge element, which is an instance of BusinessKnowledgeModel.

Notice that a KnowledgeRequirement element must reference the instance of BusinessKnowledgeModel that it associates with the requiring Decision or BusinessKnowledgeModel element, not contain it: instances of BusinessKnowledgeModel can only be contained in Definitions elements.

An instance of KnowledgeRequirement is said to be well-formed if and only if all of the following are true:
- it references a requiredKnowledge element,
• the referenced requiredKnowledge element is well-formed,
• and, if the InformationRequirement element is contained in an instance of BusinessKnowledgeModel, that BusinessKnowledgeModel element is not in the requirement subgraph of the referenced requiredKnowledge element.

Table 15 presents the attributes and model associations of the KnowledgeRequirement element.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>requiredKnowledge:</td>
<td>The instance of BusinessKnowledgeModel that this KnowledgeRequirement associates with its containing</td>
</tr>
<tr>
<td>BusinessKnowledgeModel</td>
<td>Decision or BusinessKnowledgeModel element.</td>
</tr>
</tbody>
</table>

5.3.13 Authority Requirement metamodel

The class AuthorityRequirement is used to model an authority requirement, as represented by an arrow drawn with a dashed line and a filled circular head in a DRD.

An AuthorityRequirement element is a component of a Decision, BusinessKnowledgeModel or KnowledgeSource element, and it associates that requiring Decision, BusinessKnowledgeModel or KnowledgeSource element with a requiredAuthority element, which is an instance of KnowledgeSource, a requiredDecision element, which is an instance of Decision, or a requiredInput element, which is an instance of InputData.

Notice that an AuthorityRequirement element must reference the instance of KnowledgeSource, Decision or InputData that it associates with the requiring element, not contain it: instances of KnowledgeSource, Decision or InputData can only be contained in Definitions elements.

Table 16 presents the attributes and model associations of the AuthorityRequirement element.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>requiredAuthority:</td>
<td>The instance of KnowledgeSource that this AuthorityRequirement associates with its containing KnowledgeSource, Decision or BusinessKnowledgeModel element.</td>
</tr>
<tr>
<td>KnowledgeSource [0..1]</td>
<td></td>
</tr>
</tbody>
</table>
5.4 Examples
Examples of DRDs are provided in section 10.2.

6 Relating Decision Logic to Decision Requirements

6.1 Introduction
Section 5 described how the decision requirements level of a decision model – a DRG represented in one or more DRDs – may be used to model the structure of an area of decision making. However, the details of how each decision's outcome is derived from its inputs must be modeled at the decision logic level. This section introduces the principles by which decision logic may be associated with elements in the DRG. Specific representations of decision logic (decision tables and FEEL expressions) are then defined in sections 7, 8 and 9.

The decision logic level of a decision model in DMN consists in one or more value expressions. In DMN 1.0, the elements of decision logic modeled as value expressions are literal expressions, decision tables and invocations:

- a **literal expression** represents decision logic as text that describes how an output value is derived from its input values. The expression language may, but need not, be formal or executable: examples of literal expressions include a plain English description of the logic of a decision, a first order logic theory, a Java computer program and a PMML document. DMN 1.0 specifies an expression language: **FEEL** (see section 9), and a basic subset of FEEL (see section 8) that is the default language for literal expressions in DMN decision tables (section 7).

- A **decision table** is a tabular representation of decision logic, based on a discretization of the possible values of the inputs of a decision, and organized into rules that map discretized input values onto discrete output values (see section 7).

- An **invocation** may be used as an alternative to FEEL, to model how a decision invokes decision logic that is represented by a Business Knowledge Model.

Decision logic is added to a decision model by including a value expression component in some of the decision model elements in the DRG:

- From a decision logic viewpoint, a decision is a piece of logic that defines how a given question is answered, based on the input data. As a consequence, each **decision** element in a decision model may include a value expression that describes how a decision outcome is derived from its required input, possibly invoking a business knowledge model;
From a decision logic viewpoint, a business knowledge model is a piece of decision logic that is defined as a function in order to be re-used in multiple decisions. As a consequence, each business knowledge model element may include a value expression, which is the body of that function.

Another key component of the decision logic level is the variable: Variables are used to represent input values in value expressions: input values are assigned to variables, and value expressions reference variables. Variables link information requirements in the DRG to the value expressions at the decision logic level:

- From a decision logic viewpoint, an information requirement is a requirement for an externally provided value to be assigned to a free variable in the decision logic, so that a decision can be evaluated. As a consequence, each information requirement in a decision model includes a variable that represents the associated data input in the decision’s expression.

- The variables that are used in the body of the function defined by a business knowledge model element in the DRG must be bound to the information sources each of the requiring decision. As a consequence, each business knowledge model includes zero or more variables that are the parameters of the function.

The third key element of the decision logic level are the item definitions, that describe the types and structures of data items in a decision model: input data elements in the DRG, and variables and value expressions, at the decision logic level, may reference an associated item definition, that describes the type and structure of the data expected as input, assigned to the variable or resulting from the evaluation of the expression.

Notice that knowledge sources are not represented at the decision logic level: knowledge sources are part of the documentation of the decision logic, not of the decision logic itself.

The dependencies between decisions, required information sources and business knowledge models, as represented by the information and knowledge requirements in a DRG, constrain how the value expressions associated with these elements relate to each other.

As explained above, every information requirement at the DRG level is associated with a (variable, expression) pair at the decision logic level. Each input variable that is referenced by a decision’s expression must be the variable in one of the decision’s information requirements, and each variable in a decision’s information requirement must be an input variable of the decision’s expression. The expression that is associated with a variable in an information requirement specifies the value to be assigned to the variable when evaluating the decision’s expression:

- If a decision requires another decision, the expression in the pair that is associated with the information requirement is the required decision’s expression, thus assigning the value of the required decision to an input variable of the requiring decision. This is the generic mechanism in DMN for composing decisions at the decision logic level;

- if a decision requires an input data, the expression in the pair that is associated with the information requirement is outside of the decision model and is, therefore, not represented as an explicit expression: the variable is assigned the value of the data source attached to the input data at execution time. This is the generic mechanism in DMN for instantiating the data requirements for a decision. Notice that, for required input data, FEEL allows test data to be included in the place where the external value expression cannot be explicitly represented.
The input variables of a decision's decision logic must not be used outside that value expression or its component value expressions: the decision element defines the lexical scope of the input variables for its decision logic. To avoid name collisions and ambiguity, the name of a variable must be unique within its scope. When DRG elements are mapped to FEEL, the name of a variable is the same as the (possibly qualified) name of its associated input data or decision, which guarantees its uniqueness.

When DRG elements are mapped to FEEL, all the decisions and input data in a DRG define a context, which is the literal expression that represents the logic associated with the decision element and that represents that scope (see 9.3.2.8). The information requirement elements in a decision are context entries in the associated context, where the key is the name of the variable that the information requirement defines, and where the expression is the context that is associated with the required decision or input data element that the information requirement references. The value expression that is associated with the decision as its decision logic is the expression in the context entry that specifies what is the result of the context.

In the same way, a business knowledge model element defines the lexical scope of its parameters, that is, of the input variable for its body.

In FEEL, the literal expression and scoping construct that represents the logic associated with a business knowledge model element is a function definition (see 8.3), where the formal parameters are the names of the parameters in the business knowledge model element, and the expression is the value expression that is the body of the business knowledge model element.

If a business knowledge model element requires one or more other business knowledge models, it must have an explicit value expression that describes how the required business knowledge models are invoked and their results combined or otherwise elaborated.

At the decision logic level, a decision invokes a required business knowledge model by evaluating the business knowledge model's value expression with the parameters bound to its own input value. How this may be achieved depends on how the decision logic is partitioned between the decision and business knowledge models:

- If a decision element requires more than one business knowledge element, its value expression must be a literal expression that specifies how the business knowledge model elements are invoked and how their results are combined into the decision's outcome.
- If a decision does not require any business knowledge models, its value expression must be a literal expression or decision table that specifies the entire decision logic for deriving the output from the inputs.
- Similarly, if a decision element requires only one business knowledge model element, but the logic of the decision elaborates on the logic of its required business knowledge model, the decision element must have a literal expression that specifies how the business knowledge model's value expression is invoked, and how its result is elaborated to provide the decision's outcome.
- In all other cases (i.e. when a decision requires exactly one business knowledge model and does not elaborate the logic), the value expression of a decision element may be a value expression of type invocation. In a value expression of type invocation, only the bindings of the business knowledge model parameters to the decisions input data need be specified: the outcome of the
decision is the result returned by the business knowledge model's value expression for the values passed to its parameters.

The binding of a business knowledge model's parameter is a value expression that specifies how the value passed to that parameter is derived from the values of the input variables of the invoking decision.

Using a value expression of type invocation is never required, even when possible: FEEL specifies its own invocation mechanism for more complex usages, and a FEEL literal expression can always be used instead of a value expression of type invocation.

6.2 Notation

6.2.1 Boxed Expressions

We define a graphical notation for decision logic called boxed expressions. This notation serves to decompose the decision logic model into small pieces that can be associated with DRG artifacts. The DRD plus the boxed expressions form a complete, mostly graphical language that completely specifies Decision Models.

In addition to the generic notion of boxed expression, this section specifies two kinds of boxed expressions:

- boxed literal expression,
- boxed invocation.

The boxed expression for a decision table is defined in section 7. Further types of boxed expressions are defined for FEEL, in section 9.

Boxed expressions are defined recursively, i.e. boxed expressions can contain other boxed expressions.

The top-level boxed expression corresponds to the decision logic of a single DRG artifact. This boxed expression MUST have a name box that contains the name of the DRG artifact. The name box may be attached in a single box on top, as shown in Figure 21:

![Boxed expression](image)

Figure 21: Boxed expression

Alternatively, the name box and expression box can be separated by white space and connected on the left side with a dotted line, as shown in Figure 22:
Name is the only visual link defined between DRD elements and boxed expressions. Graphical tools are expected to support appropriate graphical links, for example, clicking on a decision shape opens a decision table. How the boxed expression is visually associated with the DRD element is left to the implementation.

### 6.2.2 Boxed literal expression

In a boxed expression, a literal expression is represented by its text. However, two notational conventions are provided to improve the readability of boxed literal expressions: typographical string literals and typographical date and time literals.

#### 6.2.2.1 Typographical string literals

A string literal such as "DECLINED" can be represented alternatively as the italicized literal *DECLINED*. For example, Figure 23 is equivalent to Figure 24:

**Figure 23: Decision table with italicized literals**

<table>
<thead>
<tr>
<th>UC</th>
<th>Risk Category</th>
<th>Credit Contingency Factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><em>HIGH</em>, <em>DECLINE</em></td>
<td>0.6</td>
</tr>
<tr>
<td>2</td>
<td><em>MEDIUM</em></td>
<td>0.7</td>
</tr>
<tr>
<td>3</td>
<td><em>LOW</em>, <em>VERY LOW</em></td>
<td>0.8</td>
</tr>
</tbody>
</table>

**Figure 24: Decision table with string literals**

<table>
<thead>
<tr>
<th>UC</th>
<th>Risk Category</th>
<th>Credit Contingency Factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>“HIGH”, “DECLINE”</td>
<td>0.6</td>
</tr>
<tr>
<td>2</td>
<td>“MEDIUM”</td>
<td>0.7</td>
</tr>
<tr>
<td>3</td>
<td>“LOW”, “VERY LOW”</td>
<td>0.8</td>
</tr>
</tbody>
</table>
To avoid having to discerning whether \textit{HIGH, DECLINE} is "HIGH", "DECLINE" or "HIGH, DECLINE", typographical string literals should be free of "," characters. FEEL typographical string literals must conform to grammar rule 27 (name).

\textbf{6.2.2.2 Typographical date and time literals}

A date, time, date and time, or duration expression such as date("2013-08-09") can be represented alternatively as the bold italicized literal 2013-08-09. The literal must obey the syntax specified in sections 9.3.2.2.4, 9.3.2.2.5 and 9.3.2.2.6.

\textbf{6.2.3 Boxed 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.

The representation of an invocation is the name of the business knowledge model with the parameters’ bindings explicitly listed.

As a boxed expression, an invocation is represented by a box containing the name of the business knowledge model to be invoked, and boxes for a list of bindings, where each binding is represented by two boxed expressions on a row: the box on the left contains the name of a parameter, and the box on the right contains the binding expression, that is the expression whose value is assigned to the parameter for the purpose of evaluating the invoked business knowledge model (see Figure 25).

<table>
<thead>
<tr>
<th>Name</th>
<th>Binding expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>invoked business model</td>
<td></td>
</tr>
<tr>
<td>parameter 1</td>
<td>Binding expression 1</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>parameter 2</td>
<td>Binding expression 2</td>
</tr>
<tr>
<td>parameter n</td>
<td>Binding expression n</td>
</tr>
</tbody>
</table>

\textbf{Figure 25: Boxed invocation}

The invoked business knowledge model is represented by the name of the business knowledge model. Any other visual linkage is left to the implementation.

\textbf{6.3 Metamodel}

An important characteristic of decisions and business knowledge models, in DMN, is that they may contain an expression that describes the logic by which a modeled decision shall be made, or pieces of that logic.
In DMN 1.0, the class `Expression` is the abstract super class for all expressions that are used to describe complete or parts of decision logic in DMN models and that return a single value when interpreted (section 6.3.1).

DMN 1.0 defines three concrete kinds of `Expression`; `LiteralExpression`, `DecisionTable` (see section 7) and `Invocation`.

An expression may reference variables, such that the value of the expression, when interpreted, depends on the values assigned to the referenced variables. In DMN 1.0, the class `InformationItem` is used to model variables in expressions.

The value of an expression, like the value assigned to a variable, may have a structure and a range of allowable values. In DMN 1.0, the class `ItemDefinition` is used to model data structures and ranges.
6.3.1 Expression metamodel

An important characteristic of decisions and business knowledge models, in DMN, is that they may contain an expression that describes the logic by which a modeled decision shall be made, or pieces of that logic.

In DMN 1.0, the class Expression is the abstract super class for all expressions that are used to describe complete or parts of decision logic in DMN models and that return a single value when interpreted.
Expression is an abstract specialization of DMNElement, from which it inherits the id, and optional name and description attributes.

An instance of Expression is a component of a Decision element, of a BusinessKnowledgeModel element, or of an ItemDefinition element, or it is a component of another instance of Expression, directly or indirectly. The id of an Expression element MUST be unique within the containing instance of Decision, BusinessKnowledgeModel or ItemDefinition.

An instance of Expression references zero or more inputVariables, which are instances of InformationItem. The inputVariables are lexically scoped, in instances of Expression, and the scope is defined by the instance of Decision that contains them as part of an informationRequirement element, or by the instance of BusinessKnowledgeModel that contains them as parameters. An Expression element that is contained in an instance of ItemDefinition MUST NOT reference any inputVariable.

An instance of Expression references an optional itemDefinition, which is an instance of ItemDefinition that specifies its range of possible values.

An instance of Expression can be interpreted to derive a single value from the values assigned to its inputVariables. How the value of an Expression element is derived from the values assigned to its inputVariables depends on the concrete kind of the Expression.

Expression inherits from the attributes and model associations of DMNElement. Table 17 presents the additional attributes and model associations of the Expression element.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inputVariable:</td>
<td>This attributes lists the instances of InformationItem that are free in this</td>
</tr>
<tr>
<td>InformationItem [*]</td>
<td>Expression.</td>
</tr>
<tr>
<td>itemDefinition:</td>
<td>The instance of ItemDefinition to which the value of this Expression must conform.</td>
</tr>
<tr>
<td>ItemDefinition [0..1]</td>
<td></td>
</tr>
<tr>
<td>inputClause:</td>
<td>The containing instance of Clause, if this Expression is an inputEntry element in an instance of DecisionTable.</td>
</tr>
<tr>
<td>Clause [0..1]</td>
<td></td>
</tr>
<tr>
<td>outputClause:</td>
<td>The containing instance of Clause, if this Expression is an outputEntry element in an instance of DecisionTable.</td>
</tr>
<tr>
<td>Clause [0..1]</td>
<td></td>
</tr>
</tbody>
</table>
6.3.2 ItemDefinition metamodel

In DMN, the inputs and output of decisions are data items whose value, at the decision logic level, is assigned to variables or represented by value expressions.

An important characteristic of data items in decision models is their structure. DMN does not require a particular format for this data structure, but it does designate a subset of FEEL as its default.

In DMN 1.0, the class ItemDefinition is used to model the structure and the range of values of the input and the outcome of decisions.

As a concrete specialization of DMNElement, an instance of ItemDefinition has an id and an optional name and description. The id of an ItemDefinition element MUST be unique within the containing instance of Definitions.

The default type language for all elements can be specified in the Definitions element using the typeLanguage attribute. For example, a typeLanguage value of http://www.w3.org/2001/XMLSchema” indicates that the data structures using by elements within that Definitions are in the form of XML Schema types. If unspecified, the default is FEEL.

Notice that the data types that are built in the typeLanguage that is associated with an instance of Definitions need not be redefined by ItemDefinition elements contained in that Definitions element: they are considered imported and can be referenced in DMN elements within the Definitions element.

The type language can be overridden locally using the typeLanguage attribute in the ItemDefinition element.

Notice, also, that the data types and structures that are defined at the top level in a data model that is imported using an Import element that is associated with an instance of Definitions need not be redefined by ItemDefinition elements contained in that Definitions element: they are considered imported and can be referenced in DMN elements within the Definitions element.

An ItemDefinition element may have a typeDefinition, which is a String that defines the data structure using the typeLanguage, or a typeRef, which is a String that references a builtin data type in the associated typeLanguage or a type or data structure defined at the top level in an external document using the typeLanguage: in the latter case, the external document MUST be imported in the Definitions element that contains the instance of ItemDefinition, using an Import element. For example, in the case of data structures contributed by an XML schema, an Import would be used to specify the file location of that schema, and the typeRef attribute would reference the type or element definition in the imported schema.

By default, the name of an ItemDefinition is the name of the type that is defined in its typeDefinition or referenced in its typeRef. An ItemDefinition element MUST NOT have both a typeDefinition and a typeRef.

If the type language is FEEL the builtin types are the FEEL built-in data types: number, string, boolean, duration, time and date and time.

An ItemDefinition element may restrict the values that are allowed from the typeDefinition or typeRef, using the allowedValue attribute: each allowedValue is an
instance of Expression that specifies a single allowed value or a range of allowed values from the typeDefinition or typeRef. The itemDefinition of the allowedValues MUST be the containing ItemDefinition element itself and MAY be omitted. If an ItemDefinition element contains one or more allowedValues, the list of the allowedValues specifies the complete range of values that this ItemDefinition represents. If an ItemDefinition element does not contain an allowedValue, its range of allowed values is the full range of the referenced typeRef or defined typeDefinition.

In cases where the values that an ItemDefinition element represents are collections of values in the allowed range, the multiplicity can be projected into the attribute isCollection. The default value for this attribute is false.

An alternative way to define an instance of ItemDefinition is as a composition of other ItemDefinition elements. An instance of ItemDefinition may reference zero or more itemComponentRef, which are ItemDefinition elements: each value in the range of an ItemDefinition element that references at least one itemComponentRef is made of one value in the range each of the referenced itemComponentRef elements.

An ItemDefinition element must be defined using only one of the alternative ways:

- inline definition of a data type or structure using a typeDefinition, possibly restricted with allowedValues;
- reference to a built-in or imported typeRef, possibly restricted with allowedValues;
- composition of other ItemDefinition elements, referencing itemComponentRef.

That is, an ItemDefinition element that references an itemComponentRef element MUST NOT have a typeDefinition, a typeRef or allowedValues. Reciprocally, an ItemDefinition element that has a typeDefinition or a typeRef attribute MUST NOT reference any itemComponentRef. As already mentioned above, an ItemDefinition element MUST NOT have both a typeDefinition and a typeRef.

The ItemDefinition element specializes DMNElement and it inherits its attributes and model associations. Table 18 presents the additional attributes and model associations of the ItemDefinition element.

**Table 18: ItemDefinition attributes and model associations**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>typeDefinition: String [0..1]</td>
<td>This attribute is used to define in line the base data structure for this ItemDefinition</td>
</tr>
<tr>
<td>typeRef: String [0..1]</td>
<td>This attribute is used to identifies the base type of this ItemDefinition</td>
</tr>
<tr>
<td>Attribute</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>typeLanguage:</td>
<td>This attribute identifies the type language used to specify the base type of this ItemDefinition. This value overrides the type language specified in the Definitions element. The language MUST be specified in a URI format.</td>
</tr>
<tr>
<td>allowedValue:</td>
<td>This attribute lists the Expression elements that define the values or range of values in the base type that are allowed in this ItemDefinition.</td>
</tr>
<tr>
<td>itemComponentRef:</td>
<td>This attribute lists the ItemDefinition elements that compose this ItemDefinition.</td>
</tr>
<tr>
<td>IsCollection:</td>
<td>Setting this flag to true indicates that the actual values defined by this ItemDefinition are collections of allowed values. The default is false.</td>
</tr>
</tbody>
</table>

### 6.3.3 InformationItem metamodel

In DMN 1.0, the class InformationItem is used to model variables at the decision logic level in decision models.

InformationItem is a concrete subclass of DMNElement, from which it inherits the id, and optional name and description attributes, except that an InformationItem element MUST have a name attribute, which is the name that is used to represent it in other Expression elements. The name of an InformationItem element MUST be unique within its scope.

In DMN, variables represent the values that are input to a decision, in the description of the decision’s logic, or the values that are passed to a module of decision logic that is defined as a function (and that is represented by a business knowledge model element). In the first case, a variable is the realization, at the decision logic level, of one of the information requirements (at the decision requirements level) of a decision; in the second case, a variable is one of the parameters of the function that is the realization, at the decision logic level, of a business knowledge model element.

As a consequence, an InformationItem element MUST be either a variable in an instance of InformationRequirement or a parameter in an instance of BusinessKnowledgeModel; it MUST NOT be both. The scope of an InformationItem element is the Decision that contains the containing InformationRequirement element, or the containing BusinessKnowledgeModel element.

A variable in an instance of InformationRequirement MUST be an inputVariable in the decisionLogic in the Decision element that contains the InformationRequirement element. A parameter in an instance of BusinessKnowledgeModel MUST be an inputVariable in the valueExpression in that BusinessKnowledgeModel element.
As a concrete specialization of Expression, an InformationItem element can be interpreted and assigned a value. Specifically:

- An InformationItem element is assigned the value of the requiredDecision that is referenced by its containing instance of InformationRequirement, if it references one.
- An InformationItem element that is a parameter in a BusinessKnowledgeModel element can only be assigned a value using a Binding element as part of an instance of Invocation.
- Otherwise, an InformationItem element is assigned a value by the external data source that is attached at runtime to the requiredInput element that its containing instance of InformationRequirement references. How a data source is attached to an instance of InputData at runtime, and how it assigns a value to an InformationItem element is out of the scope of DMN 1.0.

In any case, the valueDefinition element that is associated with an instance of InformationItem must be compatible with the valueDefinition that is associated with the DMN model element from which it takes its value.

InformationItem inherits of all the attributes and model associations of DMNElement. Table 19 presents the additional attributes and model associations of the InformationItem element.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/valueExpression:</td>
<td>The Expression whose value is assigned to this InformationItem. This is a derived attribute</td>
</tr>
<tr>
<td>[0..1]</td>
<td></td>
</tr>
<tr>
<td>informationRequirement:</td>
<td>The instance of InformationRequirement in which this InformationItem is a part, if any.</td>
</tr>
<tr>
<td>InformationRequirement [0..1]</td>
<td></td>
</tr>
<tr>
<td>itemDefinition:</td>
<td>The instance of ItemDefinition to which the value of this InformationItem must conform.</td>
</tr>
<tr>
<td>ItemDefinition [0..1]</td>
<td></td>
</tr>
</tbody>
</table>

6.3.4 Literal expression metamodel

In DMN 1.0, the class LiteralExpression is used to model a value expression whose value is specified by text in some specified expression language.

LiteralExpression is a concrete subclass of Expression, from which it inherits the id, inputVariable and valueDefinition attributes.

An instance of LiteralExpression has an optional text, which is a String, and an optional expressionLanguage, which is a String that identifies the expression language of the text. If no
expressionLanguage is specified, the expression language of the text is the expressionLanguage that is associated with the containing instance of Definitions. The expressionLanguage MUST be specified in a URI format. The default expression language is FEEL.

As a subclass of Expression, each instance of LiteralExpression has a value. The text in an instance of LiteralExpression determines its value, according to the semantics of the LiteralExpression’s expressionLanguage. The semantics of DMN 1.0 decision models as described in this specification applies only if the text of all the instances of LiteralExpression in the model are valid expressions in their associated expression language.

An instance of LiteralExpression may include an import, which is an instance of Import that identifies where the text of the LiteralExpression is located. An instance of LiteralExpression MUST NOT have both a text and an import. The importType of the import MUST be the same as the expressionLanguage of the LiteralExpression element.

LiteralExpression inherits of all the attributes and model associations of Expression. Table 20 presents the additional attributes and model associations of the LiteralExpression element.

Table 20: LiteralExpression attributes and model associations

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>text: String [0..1]</td>
<td>The text of this LiteralExpression. It MUST be a valid expression in the expressionLanguage.</td>
</tr>
<tr>
<td>expressionLanguage: String [0..1]</td>
<td>This attribute identifies the expression language used in this LiteralExpression. This value overrides the expression language specified for the containing instance of DecisionRequirementDiagram. The language MUST be specified in a URI format.</td>
</tr>
<tr>
<td>import: Import [0..1]</td>
<td>The instance of Import that specifies where the text of this LiteralExpression is located.</td>
</tr>
</tbody>
</table>

6.3.5 Invocation metamodel

Invocation is a mechanism that permits the evaluation of one value expression – the invoked expression – inside another value expression – the invoking expression – by binding locally the input variables of the invoked expression to values inside the invoking expression. In an invocation, the input variables of the invoked expression are usually called: parameters. Invocation permits the same value expression to be re-used in multiple expressions, without having to duplicate it as a sub-expression in all the using expressions.
In DMN, the class `Invocation` is used to model invocations as a kind of `Expression`. `Invocation` is a concrete specialization of `Expression`, from which it inherits the `id`, `inputVariable` and `valueDefinition` attributes.

An instance of `Invocation` is made of zero or more binding, which are instances of `Binding`, and model how the parameters of the invoked expression are bound to the `inputVariables` of the invoking instance of `Expression`.

An instance of `Invocation` references a `calledFunction`, which is the instance of `Expression` to be invoked.

The value of an instance of `Invocation` is the value of the associated `calledFunction`, with its `inputVariables` assigned values at runtime per the bindings in the `Invocation`.

`Invocation` MAY be used to model invocations in decision models, when a `Decision` element has exactly one `knowledgeRequirement` element, and when the `decisionLogic` in the `Decision` element consists only in invoking the `BusinessKnowledgeModel` element that is referenced by that `requiredKnowledge` and a more complex value expression is not required.

Using `Invocation` instances as the `decisionLogic` in `Decision` elements permits the re-use of the body of an instance of `BusinessKnowledgeModel` as the logic for any instance of `Decision` that requires that `BusinessKnowledgeModel`, where each requiring `Decision` element specifies its own bindings for the `BusinessKnowledgeModel` element’s parameters.

The `calledFunction` that is associated with the `Invocation` element MUST BE the body of the `BusinessKnowledgeModel` element that is required by the `Decision` element that contains the `Invocation`; that is, the `calledFunction` is a derived attribute. The `Invocation` element MUST have exactly one binding for each parameter in the `BusinessKnowledgeModel` element.

`Invocation` inherits of all the attributes and model associations of `Expression`. Table 21 presents the additional attributes and model associations of the `Invocation` element.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>/calledFunction</code>:</td>
<td>The <code>Expression</code> that is invoked by this <code>Invocation</code>. It MUST BE the body</td>
</tr>
<tr>
<td></td>
<td>of the <code>BusinessKnowledgeModel</code> element that is required by the <code>Decision</code></td>
</tr>
<tr>
<td></td>
<td>element that contains this <code>Invocation</code>. This is a derived attribute</td>
</tr>
<tr>
<td><code>binding</code>: Binding</td>
<td>This attribute lists the instances of <code>Binding</code> used to bind the <code>inputVariables</code> of the <code>calledFunction</code> in this <code>Invocation</code>.</td>
</tr>
</tbody>
</table>
6.3.6 Binding metamodel

In DMN 1.0, the class Binding is used to model, in an Invocation element, the binding of the free variables – or parameters – in the invoked expression to the input variables of the invoking expression.

An instance of Binding is made of one bindingFormula, which is an instance of Expression, and of one reference to a parameter, which is an instance of InformationItem.

The inputVariables of the bindingFormula in a Binding element MUST be a subset of the inputVariables in the owning instance of Invocation.

The parameter referenced by a Binding element MUST be one of the parameters of the BusinessKnowledgeModel element that contains the calledFunction element that is invoked by the containing instance of Invocation.

When the Invocation element is executed, each InformationItem element that is referenced as a parameter by a binding in the Invocation element is assigned, at runtime, the value of the bindingFormula.

Table 22 presents the attributes and model associations of the Binding element.

Table 22: Binding attributes and model associations

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>parameter:</td>
<td>The InformationItem on which the calledFunction of the owning instance of Invocation depends that is bound by this Binding.</td>
</tr>
<tr>
<td>bindingFormula:</td>
<td>The instance of Expression to which the parameter in this Binding is bound when the owning instance of Invocation is evaluated.</td>
</tr>
</tbody>
</table>

7 Decision Table

7.1 Introduction

One of the ways to express the decision logic corresponding to the DRD decision artifact is a decision table. 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. The decision table contains all (and only) the inputs required to determine the output. Moreover, a complete table contains all possible combinations of input values (all the rules).
Decision tables and decision table hierarchies have a proven track record in decision logic representation. It is one of the purposes of DMN to standardize different forms and types of decision tables.

A decision table consists of:

- a name
- a set of inputs, each input optionally associated with a type and list of input values
- a set of outputs, each output optionally associated with a type and list of output values
- a list of rules in rows or columns of the table (depending on orientation), where each rule is composed of the specific input entries and output entries of the table row (or column)

The decision table shows the rules in a shorthand notation by arranging the entries in table cells:

<table>
<thead>
<tr>
<th>input expression 1</th>
<th>input expression 2</th>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>y</td>
<td>z</td>
</tr>
</tbody>
</table>

The three highlighted cells in the decision table fragment above represent the following rule:

IF input expression 1 matches x AND input expression 2 matches y THEN a result (a "hit") is z.

This shorthand notation shows all inputs in the same order in every rule and therefore has a number of readability and verification advantages.

The list of rules expresses the logic of the decision. If rules are allowed to contain overlapping input combinations, the table hit policy indicates how overlapping rules have to be interpreted, in order to avoid inconsistency.

The list of rules may contain all possible combinations of input values, in which case the table is called “complete”.

### 7.2 Notation

This section builds on the generic notation for decision logic and boxed expressions defined in section 6.2.

A decision table representation standardizes:

- the orientation (rules as rows, columns or crosstab), as shown by the table
- placement of inputs, outputs and (optional) list of values in standard locations on a grid of cells
- line style and optional use of color
- the contents of specific rule input and output entry cells
- the hit policy, indicating how to interpret overlapping input combinations
- the aggregation, indicating how multiple hits are aggregated
- placement of table name, hit policy (H), completeness indicator (C), aggregation (A) and rule numbers
7.2.1 Line style and color
Line style is normative. There is a double line between the inputs section and the outputs section, and there is a double line between inputs/outputs and the rule entry cells. Other cells are separated by a single line.

Color is suggested, but does not influence the meaning. It is considered good practice to use different colors for inputs, outputs and rule values.

7.2.2 Table orientation
Depending on size, a decision table can be presented horizontally (rules as rows), vertically (rules as columns), or crosstab (rules composed from two input dimensions). Crosstab tables can only have the default hit policy (see later).

The table must be arranged in one of the following ways (see Figure 27, Figure 28, Figure 29). Cells labeled in italics are optional.

<table>
<thead>
<tr>
<th>table name</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>HC</strong></td>
</tr>
<tr>
<td><strong>input expression 1</strong></td>
</tr>
<tr>
<td><strong>value 1a, value 1b</strong></td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
</tbody>
</table>

Figure 27: Rules as rows

<table>
<thead>
<tr>
<th>table name</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>input expression 1</strong></td>
</tr>
<tr>
<td><strong>input expression 2</strong></td>
</tr>
<tr>
<td><strong>Output name</strong></td>
</tr>
<tr>
<td><strong>HC</strong></td>
</tr>
</tbody>
</table>

Figure 28: Rules as columns
Crosstab tables with more than two inputs are possible (but not shown here).

7.2.3 Input expressions

Input expressions are usually simple, for example, a name (e.g. Customer Status) or a test (e.g. Age<25). The expression can be any text (e.g., natural language text) but SHOULD NOT conflict with FEEL syntax.

7.2.4 Input values

Input expressions may be expected to result in a limited number or a limited range of values. It is important to model these expected input values, because a decision table may be considered complete if its rules cover all combinations of expected input values.

Regardless of how the expected input values are modeled, input values should be exclusive and complete. Exclusive means that input values do not overlap. Complete means that all relevant input values from the domain are covered.

For example, the following two input value ranges overlap: <5, <10. The following two ranges are incomplete: <5, >5.

The list of input values is optional. If provided, it is a list of unary tests restricting the corresponding inputs to values that test true. The list can be any text (e.g., natural language text) but SHOULD NOT conflict with FEEL syntax.

7.2.5 Table name and output name

The table name or the output name MUST be specified.

If the table name is specified, the output name MUST be the same as the table name, or left unspecified (empty box for single-output tables, or omitted box for multiple-output tables). If the table name is not specified, the output name MUST be specified.

7.2.6 Output values

The output entries of a decision table are often drawn from a list of output values. The ordering of the list of output values can be used to specify the priority when multiple rules match but only one hit should be returned. The ordering is also used when the hit policy is output order.
The list of output values is optional. If provided, it is a list restricting the value of the output entries to the given list of values. The list can be any text (e.g., natural language text) but SHOULD NOT conflict with FEEL syntax.

### 7.2.7 Multiple outputs

The decision table can show a compound output (see Figure 30, Figure 31, Figure 32).

<table>
<thead>
<tr>
<th>table name</th>
<th>input expression 1</th>
<th>input expression 2</th>
<th>output name</th>
<th>output 1</th>
<th>output 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>input value 1a,</td>
<td>input value 2a,</td>
<td></td>
<td>output value 1a,</td>
<td>output value 2a,</td>
</tr>
<tr>
<td></td>
<td>input value 1b</td>
<td>input value 2b</td>
<td></td>
<td>output value 1b</td>
<td>output value 2b</td>
</tr>
<tr>
<td>1</td>
<td>input entry 1a</td>
<td>input entry 2a</td>
<td>output entry 1a</td>
<td>output entry 2a</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>input entry 2b</td>
<td></td>
<td>output entry 1b</td>
<td>output entry 2a</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>input entry 1b</td>
<td>-</td>
<td>output entry 1a</td>
<td>output entry 2b</td>
<td></td>
</tr>
</tbody>
</table>

Figure 30: Horizontal table with compound output

<table>
<thead>
<tr>
<th>table name</th>
<th>input expression 1</th>
<th>input expression 2</th>
<th>output name</th>
<th>output 1</th>
<th>output 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>input value 1a,</td>
<td>input value 2a,</td>
<td></td>
<td>output value 1a,</td>
<td>output value 2a,</td>
</tr>
<tr>
<td></td>
<td>input value 1b</td>
<td>input value 2b</td>
<td></td>
<td>output value 1b</td>
<td>output value 2b</td>
</tr>
<tr>
<td>output name</td>
<td>input entry 1a</td>
<td>input entry 2a</td>
<td>output entry 1a</td>
<td>output entry 1b</td>
<td></td>
</tr>
<tr>
<td></td>
<td>input entry 2b</td>
<td></td>
<td>output entry 2a</td>
<td>output entry 2a</td>
<td></td>
</tr>
<tr>
<td>HC</td>
<td></td>
<td></td>
<td>output entry 1a</td>
<td>output entry 2b</td>
<td></td>
</tr>
</tbody>
</table>

Figure 31: Vertical table with compound output

<table>
<thead>
<tr>
<th>table name</th>
<th>output name</th>
<th>input expression 1</th>
<th>input expression 2</th>
<th>input expression 2</th>
<th>output expression 1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>output 1,</td>
<td>input entry 1a</td>
<td>input entry 1a</td>
<td>output entry 1a,</td>
<td>output entry 1a,</td>
</tr>
<tr>
<td></td>
<td>output 2</td>
<td>input entry 1b</td>
<td>output entry 2a,</td>
<td>output entry 1a,</td>
<td>output entry 2a,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>output entry 1b,</td>
<td>output entry 2a,</td>
<td>output entry 1a,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>output entry 2b,</td>
<td>output entry 2b,</td>
<td>output entry 1a,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>output entry 2b,</td>
<td>output entry 2b,</td>
<td>output entry 1a,</td>
</tr>
</tbody>
</table>

Figure 32: Crosstab with compound output
7.2.8 Input entries

Rule input entries are expressions. The expression can be any text (e.g., natural language text) but SHOULD NOT conflict with FEEL syntax. A dash symbol (‘-‘) can be used to mean any input value, i.e., the input is irrelevant for the containing rule.

The values in a unary test should be ‘-‘ or a subset of the input values specified. For example, if the input values for input 'Age' are specified as [0..120], then an input entry of <0 SHOULD be reported as invalid.

Tables containing a ‘-‘ are called contracted tables. The others are called expanded.

Tables where every input entry is true, false, or ‘-‘ are historically called limited-entry tables, but there is no need to maintain this restriction.

Evaluation of the expressions in a decision table does not produce side-effects. The order of input entries is not related to any execution order in implementation.

7.2.9 Merged input entry cells

Adjacent input entry cells from different rules with the same content and same (or no) prior cells can be merged, as shown in Figure 33 and Figure 34. Rule output cells cannot be merged (except in crosstabs).

<table>
<thead>
<tr>
<th>table name</th>
</tr>
</thead>
<tbody>
<tr>
<td>HC</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
</tbody>
</table>

Figure 33: Merged rule input cells allowed

<table>
<thead>
<tr>
<th>table name</th>
</tr>
</thead>
<tbody>
<tr>
<td>HC</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
</tbody>
</table>

Figure 34: Merged rule input cells not allowed
7.2.10 Output entry

A rule output entry is an expression. The expression can be any text (e.g., natural language text) but SHOULD NOT conflict with FEEL syntax.

In vertical (rules as columns) tables with a single output name which is identical to the table name, a shorthand notation may be used to indicate: output applies (X) or does not apply (-), as is common practice in decision tables. The table in Figure 35 is shorthand notation for the table in Figure 36.

<table>
<thead>
<tr>
<th>table name</th>
<th>input expression 1</th>
<th>input entry 1a</th>
<th>input entry 1b</th>
<th>input entry 1c</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>input expression 2</td>
<td>input entry 2a</td>
<td>input entry 2b</td>
<td>-</td>
</tr>
<tr>
<td>output entry a</td>
<td>X</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>output entry b</td>
<td>-</td>
<td>X</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td>output entry c</td>
<td>-</td>
<td>-</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td>HC</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

Figure 35: Shorthand notation for vertical tables (rules as columns)

<table>
<thead>
<tr>
<th>table name</th>
<th>value 1a, value 1b, value 1c</th>
<th>input expression 1</th>
<th>input entry 1a</th>
<th>input entry 1b</th>
<th>input entry 1c</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>value 2a, value 2b</td>
<td>input expression 2</td>
<td>input entry 2a</td>
<td>input entry 2b</td>
<td>-</td>
</tr>
<tr>
<td>Output name</td>
<td>value 1a, value 1b, value 1c</td>
<td>output name</td>
<td>output entry 1a</td>
<td>output entry 1b</td>
<td>output entry 1c</td>
</tr>
<tr>
<td>HC</td>
<td></td>
<td></td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

Figure 36: Full notation for vertical tables (rules as columns)

7.2.11 Hit policy

A decision table may have several rules, and in general more than one rule may be matched for a given set of inputs. The hit policy specifies what the result of the decision table is in such cases, and also contains additional information that can be used to check correctness at design-time. For clarity, the hit policy is summarized using a single character in a particular decision table cell. Tools may support only a subset of hit policies, but the table type must be clear and therefore the hit policy indication is mandatory.

The hit policy MUST default to Unique. Decision tables with the Unique hit policy do not contain rules with overlapping input entries.
If rules are allowed to contain overlapping input entries, the hit policy indicates how these overlapping rules have to be interpreted. A single hit table returns the output of one rule only; a multiple hit table may return the output of multiple rules (or a function of the outputs, e.g. sum of values).

A single hit table returns the output of one rule only. It may or may not contain overlapping rules. In case of overlapping rules, the hit policy has to indicate which of the matching rules to select.

**Single hit policies** for single output decision tables are:

1. **Unique**: no overlap is possible and all rules are exclusive. Only a single rule can be matched. This is the default.
2. **Any**: there may be overlap, but all of the matching rules show the same output, so any match can be used.
3. **Priority**: multiple rules can match, with different outputs. This policy returns the matching rule with the highest output priority. Output priorities are specified in an ordered list of values, for example, the list of expected output values.
4. **First**: multiple (overlapping) rules can match, with different output entries. The first hit by rule order is returned (and evaluation can halt). This is a common usage, because it resolves inconsistencies by forcing the first hit. It is important to distinguish this type of table from others because the meaning depends on the order of the rules. The last rule is often the catch-remainder. Because of this order, the table is hard to validate manually and therefore has to be used with care.

A multiple hit table may return output entries from multiple rules. The result will be a list of rule outputs or a function of the outputs.

**Multiple hit policies** for single output decision tables can be:

5. **No order**: returns all hits in a unique list in arbitrary order.
6. **Output order**: returns all hits in decreasing priority order. Output priorities are specified in an ordered list of values.
7. **Rule order**: returns all hits in rule order. Note: the meaning may depend on the order of the rules.

Other policies, such as more complex manipulations on the outputs, can be performed by post-processing the output list (outside the decision table).

The single letter for hit policy also identifies if a table is single or multiple hit.

To reduce complexity, decision tables with compound outputs support only the following hit policies: Unique, Any, First, No order, and Rule order.

**Note 1**
Crosstab tables are unique and complete by definition and therefore do not need a hit policy indication.

**Note 2**
The order of the rules in a decision table does not influence the meaning, except in **First** tables (single hit) and **Rule** order tables (multiple hit). These tables should be used with care.
7.2.12 Completeness indicator

Table completeness is an optional attribute. By default, tables are complete, producing a result for every possible case. If not the indicator should read I(ncomplete). Incomplete tables may specify a default output.

7.2.13 Aggregation

Multiple hits must be aggregated into a single result. DMN 1.0 specifies six aggregation indicators, namely: collect, sum, min, max, count, average. Optionally, the aggregation indicator may be included in the table. The default is collect.

Aggregation indicators have no incidence on decision tables with single hit policies.

In decision tables with multiple hit policies, the semantics of the aggregation indicators are:

1. collect. The result of the decision table is the list of all the outputs, ordered or unordered per the hit policy.
2. sum. The result of the decision table is the sum of all the outputs.
3. min. The results of the decision table is the smallest value of all the outputs.
4. max. The results of the decision table is the largest value of all the outputs.
5. count. The results of the decision table is the number of outputs.
6. average. The results of the decision table is the average value of all the outputs, defined as the sum divided by the count, where the semantics of sum and count are as specified above.

Other policies, such as more complex manipulations on the outputs, can be performed by post-processing the output list (outside the decision table).
7.3 Metamodel

In DMN 1.0, the class DecisionTable is used to model a decision table. DecisionTable is a concrete specialization of Expression.

An instance of DecisionTable contains a set of rules, which are instances of DecisionRule, and a set of clauses, which are instances of Clause.

It has a preferredOrientation, which must be one of the enumerated DecisionTableOrientation: Rule-as-Row, Rule-as-Column or CrossTable. An instance of DecisionTable SHOULD BE represented as specified by its preferredOrientation, as defined in section 7.2.2.

An instance of DecisionTable has an associated hitPolicy, which must be one of the enumerated HitPolicy: UNIQUE, FIRST, PRIORITY, ANY, UNORDERED, RULE ORDER,
OUTPUT ORDER. The default value for the hitPolicy attribute is: UNIQUE. In the diagrammatic representation of an instance of DecisionTable, the hitPolicy is represented as specified in section 7.2.11.

The semantics that is associated with an instance of DecisionTable depends on its associated hitPolicy, as specified below and in section 7.2.11. The hitPolicy attribute of an instance of DecisionTable is represented as specified in section 7.2.11.

If the hitPolicy associated with an instance of DecisionTable is FIRST or RULE ORDER, the rules that are associated with the DecisionTable MUST be ordered. The ordering is represented by the explicit numbering of the rules in the diagrammatic representation of the DecisionTable.

If the hitPolicy associated with an instance of DecisionTable is PRIORITY or OUTPUT ORDER, the outputEntry of one of the clauses in the DecisionTable MUST be ordered, and these outputEntries MUST be associated as conclusions to the rules in the DecisionTable. In the diagrammatic representation of the DecisionTable, the ordering is represented as specified in section 7.2.11.

An instance of DecisionTable has an associated aggregation, which is one of the enumerated BuiltinAggregator (see section 7.2.13). The default value for aggregation is: COLLECT.

As a kind of Expression, an instance of DecisionTable has a value, which depends on the conclusions of the associated rules, the associated hitPolicy and the associated aggregation, if any. The value of an instance of DecisionTable is determined according to the following specification:

- if the associated hitPolicy is UNIQUE, the value of an instance of DecisionTable is the value of the conclusion of the only applicable rule (see section 7.3.3, Decision Rule, for the definition of rule applicability);
- if the associated hitPolicy is FIRST, the value of an instance of DecisionTable is the value of the conclusion of the first applicable rule, according to the rule ordering;
- if the associated hitPolicy is PRIORITY, the value of an instance of DecisionTable is the value of the conclusion of the first applicable rule, according to the ordering of the outputEntry in the clause of its conclusion (see section 7.3.2, Decision Table Clause);
- if the associated hitPolicy is ANY, the value of an instance of DecisionTable is the value of any of the applicable rules;
- if the associated hitPolicy is UNORDERED, the value of an instance of DecisionTable is the result of applying the aggregation function specified by the aggregation attribute of the DecisionTable to the unordered set of the values of the conclusions of all the applicable rules;
- if the associated hitPolicy is RULE ORDER, the value of an instance of DecisionTable is the result of applying the aggregation function specified by the aggregation attribute of
the DecisionTable to the set of the values of the conclusions of all the applicable rules, ordered according to the rule ordering;

- if the associated hitPolicy is OUTPUT ORDER, the value of an instance of DecisionTable is the result of applying the aggregation function specified by the aggregation attribute of the DecisionTable to the set of the values of the conclusions of all the applicable rules, ordered according to the ordering of the outputEntry in the clause of its conclusion (see section 7.3.2, Decision Table Clause).

DecisionTable has an optional Boolean attribute isComplete. If an instance of DecisionTable has an isComplete attribute, the value of the attribute MUST be false is the DecisionTable is not complete. An instance of DecisionTable is said to be complete if and only if, for any valid binding of the DecisionTable inputVariables, at least one of the DecisionTable rules is applicable (see also section 7.2.12).

DecisionTable has an optional Boolean attribute isConsistent. If an instance of DecisionTable has an isConsistent attribute, the value of the attribute MUST be false unless the DecisionTable is consistent. An instance of DecisionTable is said to be consistent if and only if, for any valid binding of the DecisionTable inputVariables, all the applicable rules have the same value.

DecisionTable inherits all the attributes and model associations from Expression. Table 23 presents the additional attributes and model associations of the DecisionTable element.

**Table 23: DecisionTable attributes and model associations**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>clause: Clause [*]</td>
<td>This attributes lists the instances of Clause that compose this DecisionTable.</td>
</tr>
<tr>
<td>rule: DecisionRule [*]</td>
<td>This attributes lists the instances of DecisionRule that compose this DecisionTable.</td>
</tr>
<tr>
<td>hitPolicy: HitPolicy</td>
<td>The hit policy that determines the semantics of this DecisionTable. Default is: UNIQUE.</td>
</tr>
<tr>
<td>aggregation: BuiltinAggregator</td>
<td>The aggregation function to be applied to the values of the applicable rules when there are more than one, to determine the value of this DecisionTable. Default is: COLLECT.</td>
</tr>
<tr>
<td>isComplete: Boolean [0..1]</td>
<td>If present, this attribute MUST be false unless this DecisionTable is complete. Default is: false.</td>
</tr>
</tbody>
</table>
### 7.3.2 Decision Table Clause metamodel

In a decision table, a clause specifies a *subject*, which is defined by an *input expression* or an *output domain*, and the finite set of the sub-domains of the subject’s domain that are relevant for the piece of decision logic that is described by the decision table.

In DMN 1.0, the class `Clause` is used to model a decision table clause.

An instance of `Clause` is made of an optional *input expression* and of a set of *inputEntry*, or of an optional *name* and a set of *outputEntry*, which are instances of *Expression*. A `Clause` element MUST have a set of *inputEntry* if it has an *inputExpression*, it MUST have a set of *outputEntry* if it does not have an *inputExpression*. A `Clause` element MUST NOT have both *inputEntry* and *outputEntry*.

An instance of `Clause` may have a *name*, which is a String, and it may reference an *outputDefinition*, which is an `ItemDefinition` element. An instance of `Clause` that does not have an *inputExpression* MUST reference an *outputDefinition*. An instance of `Clause` that contains an *inputExpression* MUST NOT reference an *outputDefinition*. If a `Clause` element that references an *outputDefinition* does not have a name, its default name is the name of the referenced `ItemDefinition` element.

The *valueDefinition* of an *inputEntry* element MUST be Boolean and it MAY be omitted. The *inputEntry* elements MUST test the value of its containing clause’s *inputExpression*, possibly implicitly. The *valueDefinition* of an *outputEntry* MUST be the *outputDefinition* or a specialization of the *outputDefinition* of the containing clause, and it MAY be omitted: if the *valueDefinition* of an *outputEntry* is omitted, it defaults to the *outputDefinition* of the containing clause.

In a tabular representation of the containing instance of `DecisionTable`, the representation of an instance of `Clause` depends on the orientation of the decision table. For instance, if the decision table is represented horizontally (rules as row, see section 7.2.2), instances of `Clause` are represented as columns, with the *inputExpression* or the name of the `Clause` element represented in the top cell, its domain of value optionally listed in the cell below, and each of the cells below representing one of the *inputEntry* or *outputEntry* in the `Clause`. All the instances of `Clause` made of a set of *inputEntry*, MUST be represented on the right of any instance of `Clause` made of a set of *outputEntry*.

Table 24 presents the attributes and model associations of the `Clause` element.

<table>
<thead>
<tr>
<th><strong>isConsistent</strong>: Boolean [0..1]</th>
<th>If present, this attribute MUST be <em>false</em> unless this <code>DecisionTable</code> is <em>consistent</em>. Default is: <em>false</em>.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>preferredOrientation</strong>:</td>
<td>The preferred orientation for the diagrammatic representation of this <code>DecisionTable</code>. This <code>DecisionTable</code> SHOULD BE represented as specified by this attribute.</td>
</tr>
<tr>
<td><code>DecisionTableOrientation</code> [0..1]</td>
<td></td>
</tr>
<tr>
<td>Attribute</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>inputExpression:</td>
<td>Expression [0.1] The subject of this input Clause.</td>
</tr>
<tr>
<td>outputDefinition:</td>
<td>ItemDefinition [0.1] The range of this output Clause.</td>
</tr>
<tr>
<td>name:</td>
<td>String [0.1] The name of this output Clause.</td>
</tr>
<tr>
<td>inputEntry:</td>
<td>Expression [*] This attribute lists the instances of Expression that compose</td>
</tr>
<tr>
<td>outputEntry:</td>
<td>Expression [*] This attribute lists the instances of Expression that compose</td>
</tr>
</tbody>
</table>
inputEntry referenced as a condition by the DecisionRule element, and each conjunct is a disjunction of all the rule’s conditions that are contained in the same Clause element.

Table 25 presents the attributes and model associations of the DecisionRule element.

Table 25: DecisionRule attributes and model associations

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>condition: Expression [*]</td>
<td>This attribute lists the instances of Expression that compose the condition of this DecisionRule.</td>
</tr>
<tr>
<td>conclusion: Expression [1..*]</td>
<td>This attribute lists the instances of Expression that compose the conclusion of this DecisionRule.</td>
</tr>
</tbody>
</table>

7.4 Examples

Table 26 provides examples for the various types of decision table discussed in this section. Further examples may be found in 10.3, in the context of a complete example of a DMN decision model.
Table 26: Examples of decision tables

<table>
<thead>
<tr>
<th>Single Hit</th>
<th>Unique</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Applicant Risk Rating</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Applicant Age</td>
</tr>
<tr>
<td>U</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>&gt; 60</td>
</tr>
<tr>
<td>2</td>
<td>bad</td>
</tr>
<tr>
<td>3</td>
<td>[25..60]</td>
</tr>
<tr>
<td>4</td>
<td>&lt; 25</td>
</tr>
<tr>
<td>5</td>
<td>bad</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Applicant Risk Rating</th>
<th>Applicant Age</th>
<th>&lt; 25</th>
<th>[25..60]</th>
<th>&gt; 60</th>
</tr>
</thead>
<tbody>
<tr>
<td>Medical History</td>
<td>good</td>
<td>bad</td>
<td>-</td>
<td>good</td>
</tr>
<tr>
<td>Applicant Risk Rating</td>
<td>Low</td>
<td>Medium</td>
<td>Medium</td>
<td>Medium</td>
</tr>
<tr>
<td>U</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Applicant Risk Rating</th>
<th>Applicant Age</th>
<th>&lt; 25</th>
<th>[25..60]</th>
<th>&gt; 60</th>
</tr>
</thead>
<tbody>
<tr>
<td>Medical History</td>
<td>good</td>
<td>bad</td>
<td>-</td>
<td>good</td>
</tr>
<tr>
<td>Low</td>
<td>X</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Medium</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>High</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>U</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Single Hit</th>
<th>Any</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Person Loan Compliance</strong></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>Persons Credit Rating from Bureau</td>
</tr>
<tr>
<td>-------</td>
<td>-----------------------------------</td>
</tr>
<tr>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>2</td>
<td>Not A</td>
</tr>
<tr>
<td>3</td>
<td>-</td>
</tr>
<tr>
<td>4</td>
<td>-</td>
</tr>
</tbody>
</table>

*Example case:* not A, >= $10K, >= 50K -> Not Compliant (rules 2,3,4)
### Single Hit Priority

<table>
<thead>
<tr>
<th>Applicant Risk Rating</th>
<th>Applicant Age</th>
<th>Medical History</th>
<th>Applicant Risk Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>High, Medium, Low</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **1**: 
  - Applicant Age: \( \geq 25 \)
  - Medical History: good
  - Risk Rating: Medium

- **2**: 
  - Applicant Age: > 60
  - Medical History: bad
  - Risk Rating: High

- **3**: 
  - Applicant Age: 
  - Medical History: bad
  - Risk Rating: Medium

- **4**: 
  - Applicant Age: < 25
  - Medical History: good
  - Risk Rating: Low

### Single Hit First

<table>
<thead>
<tr>
<th>Special Discount</th>
<th>Type of Order</th>
<th>Customer Location</th>
<th>Type of Customer</th>
<th>Special Discount %</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **1**: 
  - Type of Order: Web
  - Customer Location: US
  - Type of Customer: Wholesaler
  - Special Discount %: 10

- **2**: 
  - Type of Order: Phone
  - Customer Location: -
  - Type of Customer: -
  - Special Discount %: 0

- **3**: 
  - Type of Order: -
  - Customer Location: Non-US
  - Type of Customer: -
  - Special Discount %: 0

- **4**: 
  - Type of Order: -
  - Customer Location: Retailer
  - Type of Customer: -
  - Special Discount %: 5

**Example case**: Web, non-US, Retailer \( \rightarrow \) 0 (rule 3)

### Multiple Hit

<table>
<thead>
<tr>
<th>Holidays</th>
<th>Age</th>
<th>Years of Service</th>
<th>Holidays</th>
<th>Aggregation=\text{sum}</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-</td>
<td>-</td>
<td>22</td>
<td>1 2 3 4 5 6 7</td>
</tr>
</tbody>
</table>

**Example case**: Age=58, Service=31 \( \rightarrow \) Result=\text{sum}(22, 5, 3)=30
Multiple Hit Output order

<p>| Holidays |</p>
<table>
<thead>
<tr>
<th>Age</th>
<th>Years of Service</th>
<th>Holidays</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>22, 5, 3, 2</td>
</tr>
<tr>
<td>1</td>
<td>-</td>
<td>22</td>
</tr>
<tr>
<td>2</td>
<td>&gt;= 60</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>-</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>&lt; 18</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>&gt;= 60</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>-</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>[18..60)</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>[45..60)</td>
<td>2</td>
</tr>
</tbody>
</table>

Example case: Age=58, Service=31 -> Result=(22, 5, 3)

Multiple Hit Rule order

<table>
<thead>
<tr>
<th>Student Financial Package Eligibility</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
</tr>
<tr>
<td>---</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
</tbody>
</table>

Example case: For GPA=3.6, EC Activities=4, NHS Membership -> result = (20% scholarship, 30% loan)

8 Simple Expression Language (S-FEEL)

DMN 1.0 defines the friendly enough expression language (FEEL) for the purpose of giving standard executable semantics to many kinds of expressions in decision model (see section 9).

This section defines a simple subset of FEEL, S-FEEL, for the purpose of giving standard executable semantics to decision models that use only simple expressions: in particular, decision models where the decision logic is modeled mostly or only using decision tables.
8.1 S-FEEL syntax

The syntax for the S-FEEL expressions used in this section is specified in the EBNF below: it is a subset of the FEEL syntax and the production numbering is from the FEEL EBNF, section 9.3.1.2.

Grammar rules:

4 arithmetic expression =
4.a addition | subtraction |
4.b multiplication | division |
4.c exponentiation |
4.d arithmetic negation ;

5 simple expression = arithmetic expression | simple value ;

6 simple expressions = simple expression , { "", simple expression } ;

7 simple positive unary test =
7.a [ "<" | "<=" | ">" | ">=" ] , endpoint |
7.b interval ;

8 interval = ( open interval start | closed interval start ) , endpoint , "." , endpoint , ( open interval end | closed interval end ) ;

9 open interval start = "(" | "[")" ;

10 closed interval start = "["

11 open interval end = ")" | "[" ;

12 closed interval end = "]" ;

13 simple positive unary tests = simple positive unary test , { ",", simple positive unary test } ;

14 simple unary tests =
14.a simple positive unary tests |
14.b "not", "(" simple positive unary tests, ")" |
14.c "".|."

18 endpoint = simple value ;

19 simple value = qualified name | simple literal ;

20 qualified name = name , { ".", name } ;

21 addition = expression , "+" , expression ;

22 subtraction = expression , "-" , expression ;

23 multiplication = expression , "*" , expression ;

24 division = expression , "/" , expression ;

25 exponentiation = expression , "**" , expression ;
8.2 S-FEEL data types

S-FEEL supports all FEEL data types: number, string, boolean, day time duration, year month duration, time and date, although with a simplified definition for some of them.

S-FEEL number has the same literal and values spaces as the XML Schema decimal datatype. Implementations are allowed to limit precision to 34 decimal digits and to round toward the nearest neighbor with ties favoring the even neighbor. Notice that “precision is not reflected in this value space: the number 2.0 is not distinct from the number 2.00” [XSD]. Notice, also, that this value space is totally ordered. The definition of S-FEEL number is a simplification over the definition of FEEL number.

S-FEEL supports FEEL string and FEEL Boolean: FEEL string has the same literal and values spaces as the Java String and XML Schema string datatypes. FEEL boolean has the same literal and values spaces as the Java Boolean and XML Schema Boolean datatypes.

S-FEEL supports the FEEL time data type. The lexical and value spaces of FEEL time are the literal and value spaces of the XML Schema time datatype. Notice that, “since the lexical representation allows an optional time zone indicator, time values are partially ordered because it may not be able to determine the order of two values one of which has a time zone and the other does not. Pairs of time values with or without time zone indicators are totally ordered” [XSD].

S-FEEL does not support FEEL date and time. However, it supports the date type, which is like FEEL date and time with hour, minute, and second required to be absent. The lexical and value spaces of FEEL date are the literal and value spaces of the XML Schema date datatype.

S-FEEL supports the FEEL day time duration and year month duration datatypes. FEEL day time duration and year month duration have the same literal and value spaces as the XPath Data Model.
dayTimeDuration and yearMonthDuration datatypes, respectively. That is, FEEL day time duration is derived from the XML Schema duration datatype by restricting its lexical representation to contain only the days, hours, minutes and seconds components, and FEEL year month duration is derived from the XML Schema duration datatype by restricting its lexical representation to contain only the year and month components.

The FEEL data types are specified in details in section 9.3.2.2.

8.3 S-FEEL semantics

S-FEEL contains only a limited set of basic features that are common to most expression and programming languages, and on the semantics of which most expression and programming languages agree.

For the sake of simplicity, the semantics of S-FEEL expressions are defined, in this section, with respect to the semantics of the XML Schema datatypes and of the corresponding XQuery and XPath functions and operators. A complete stand-alone specification of the semantics is to be found in section 9.3.2, as part of the definition of FEEL. Within the scope of S-FEEL, the two definitions are equivalent and equally normative.

Arithmetic addition and subtraction (grammar rule 4a) have the same semantics as:

- op:numeric-add and op:numeric-subtract, when its two operands are numbers;
- op:add-yearMonthDurations and op:subtract-yearMonthDurations, when the two operands are year month durations;
- op:add-dayTimeDuration and subtract:dayTimeDurations, when the two operands are day time durations;
- op:add-yearMonthDuration-to-date and op:subtract-yearMonthDuration-from-date, when the first operand is a year month duration and the second operand is a date;
- op:add-dayTimeDuration-to-date and op:subtract-dayTimeDuration-from-date, when the first operand is a day time duration and the second operand is a date;
- op:add-dayTimeDuration-to-time and op:subtract-dayTimeDuration-from-time, when the first operand is a day time duration and the second operand is a time.

In addition, arithmetic subtraction has the semantics of op:subtract-dates or op:subtract-times, when the two operands are dates or times, respectively.

Arithmetic addition and subtraction are not defined in other cases.

Arithmetic multiplication and division (grammar rule 4b) have the same semantics as defined for op:numeric-multiply and op:numeric-divide, respectively, when the two operands are numbers. They are not defined otherwise. Arithmetic exponentiation (grammar rule 4c) is defined as the multiplication of the first operand by itself as many time as indicated by the second operand. It is defined only when the first operand is a number and the second operand is an integer (that is, a number with no decimal or, equivalently whose decimals are all zeros).

Arithmetic negation (grammar rule 4d) is defined only when its operand is a number: in that case, its semantics is according to the specification of op:numeric-unary-minus.
Comparison operator (grammar rule 7.a) between numbers are defined according to the specification of op:numeric-equal, op:numeric-less-than and op:numeric-greater-than, comparison between dates are defined according to the specification of op:date-equal, op:date-less-than and op:date-greater-than; comparison between times are defined according to the specification of op:time-equal, op:time-less-than and op:time-greater-than; comparison between year month durations are defined according to the specification of op:duration-equal, op:yearMonthDuration-less-than and op:yearMonthDuration-greater-than; comparison between day time durations are defined according to the specification of op:duration-equal, op:dayTimeDuration-less-than and op:dayTimeDuration-greater-than.

String and Booleans can only be compared for equality: the semantics of strings and Booleans equality is as defined in the specification of fn:codepoint-equal and op:Boolean-equal, respectively.

Comparison operators are defined only when the two operands have the same type, except for year month duration and day time duration, which can be compared for equality. Notice, however, that “with the exception of the zero-length duration, no instance of xs:dayTimeDuration can ever be equal to an instance of xs:yearMonthDuration.” [XFO].

Given an expression o to be tested and two endpoint e₁ and e₂:

- is in the interval (e₁..e₂), also noted ]e₁..e₂[, if and only if o > e₁ and o < e₁
- is in the interval (e₁..e₂] also noted ]e₁..e₂], if and only if o > e₁ and o ≤ e₁
- is in the interval [e₁..e₂) also noted [e₁..e₂[, if and only if o ≥ e₁ and o ≤ e₁
- is in the interval [e₁..e₂] also noted [e₁..e₂[, if and only if o ≥ e₁ and o < e₁.

An expression to be tested satisfies an instance of simple unitary tests (grammar rule 9) if and only if, either the expression is a list and the expression satisfies at least one simple unitary test in the list, or the simple unitary tests is “-“.

8.4 Use of S-FEEL expressions

This section summarizes which kinds of S-FEEL expressions are allowed in which role, when the expression language is S-FEEL.

8.4.1 Item definitions

The expression that defines an allowed value must be a simple unary test (grammar rule 7), where only the values in the defined or referenced type that satisfy the test are allowed values.

8.4.2 Invocations

In the bindings of an invocation, the binding formula must be a simple expression (grammar rule 5).

8.4.3 Decision tables

Each input expression must be a simple expression (grammar rule 5).
Each input value must be a simple unary test (grammar rule 7), where the value that is tested is the value of the input expression of the containing clause. A list of input values must be an instance of simple unary tests (grammar rule 9).

Each list of output values must be an instance of simple expressions (grammar rule 6).

Each input entry must be a simple unary test (grammar rule 7), where the value that is tested is the value of the input expression of the containing clause. If a rule has multiple disjunctive conditions applying to the same input expression, the expression in the corresponding decision table cell must be an instance of simple unary tests (grammar rule 9).

Each output entry must be a simple expression (grammar rule 5).

9 Expression Language (FEEL)

9.1 Introduction

In DMN, all decision logic is represented as boxed expressions. Section 6.2 introduced the concept of the boxed expression and defined two simple kinds: boxed literal expressions and boxed invocations. Section 7 defined decision tables, a very important kind of boxed expression. This section completes the graphical notation for decision logic, by defining other kinds of boxed expressions.

The expressions 'in the boxes' are FEEL expressions. FEEL stands for Friendly Enough Expression Language and it has the following features:

- Side-effect free
- Simple data model with numbers, dates, strings, lists, and contexts
- Simple syntax designed for a wide audience
- Three-valued logic (true, false, null) based on SQL and PMML

This section also completely specifies the syntax and semantics of FEEL. The syntax is specified as a grammar (9.3.1). The subset of the syntax intended to be rendered graphically as a boxed expression is also specified as a meta-model (9.5).

FEEL has two roles in DMN:

1. As a textual notation in the boxes of boxed expressions such as decision tables,
2. As a slightly larger language to represent the logic of expressions and DRGs for the main purpose of composing the semantics in a simple and uniform way

9.2 Notation

9.2.1 Boxed Expressions

This section builds on the generic notation for decision logic and boxed expressions defined in section 6.2.

We define a graphical notation for decision logic called boxed expressions. This notation serves to decompose the decision logic model into small pieces that can be associated with DRG artifacts. The
DRG plus the boxed expressions form a complete, mostly graphical language that completely specifies Decision Models.

A **boxed expression** is either

- a decision table,
- a boxed FEEL expression,
- a boxed invocation,
- a boxed context,
- a boxed list,
- a relation, or
- a boxed function.

Boxed expressions are defined recursively, *i.e.* boxed expressions can contain other boxed expressions. The top-level boxed expression corresponds to the decision logic of a single DRG artifact. This boxed expression MUST have a name box that contains the name of the DRG artifact. The name box may be attached in a single box on top, as shown in Figure 38:

![Figure 38: Boxed expression](image)

Alternatively, the name box and expression box can be separated by white space and connected on the left side with a dotted line, as shown in Figure 39:

![Figure 39: Boxed expression with separated name and expression boxes](image)

Name is the only link defined between DRG elements and decision logic elements. Graphical tools are expected to support appropriate graphical links, for example, clicking on a decision shape opens a decision table.

### 9.2.1.1 Decision Tables

The executable decision tables defined here use the same notation as the decision tables defined in Section 7. Their execution semantics is defined in section 9.3.2.7.
9.2.1.2 Boxed FEEL expression

A **boxed FEEL expression** is any FEEL expression $e$, as defined by the FEEL grammar (section 9.3.1), in a table cell, as shown in Figure 40:

$$e$$

**Figure 40: Boxed FEEL expression**

The meaning of a boxed expression containing $e$ is $\text{FEEL}(e, s)$, where $s$ is the scope. The scope includes the context derived from the containing DRD as described in 9.4, and any boxed contexts containing $e$.

It is usually good practice to make $e$ relatively simple, and compose small boxed expressions into larger boxed expressions.

9.2.1.3 Boxed Invocation

The syntax for boxed invocation is described in section 6.2.3. This syntax may be used to invoke any function (e.g. business knowledge model, FEEL builtin function, boxed function definition).

The box labeled 'invoked business knowledge model' can be any boxed expression whose value is a function, as shown in Figure 41:

<table>
<thead>
<tr>
<th>Name</th>
<th>function-valued expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>parameter 1</td>
<td>binding expression 1</td>
</tr>
<tr>
<td>parameter 2</td>
<td>binding expression 2</td>
</tr>
<tr>
<td>…</td>
<td></td>
</tr>
<tr>
<td>parameter $n$</td>
<td>binding expression $n$</td>
</tr>
</tbody>
</table>

**Figure 41: Boxed invocation**

The boxed syntax maps to the textual syntax defined by grammar rules 40, 41, 42, 43. Boxed invocation uses named parameters. Positional invocation can be achieved using a boxed expression containing a textual positional invocation.

The boxed syntax requires at least one parameter. A parameterless function must be invoked using the textual syntax, e.g.

```text
function-valued expression()
```
Formally, the meaning of a boxed invocation is given by the semantics of the equivalent textual invocation, e.g., `function-valued expression(param1: binding expression1, param2: binding expression2, ...)`.

### 9.2.1.4 Boxed Context

A **boxed context** is a collection of \( n \) (name, value) pairs with an optional result value. Each pair is called a context entry. Context entries may be separated by whitespace and connected with a dotted line on the left (top). The intent is that all the entries of a context should be easily identified by looking down the left edge of a vertical context or across the top edge of a horizontal context. Cells must be arranged in one of the following ways (see Figure 42, Figure 43):

<table>
<thead>
<tr>
<th>Name 1</th>
<th>Value 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name 2</td>
<td>Value 2</td>
</tr>
<tr>
<td>Name ( n )</td>
<td>Value ( n )</td>
</tr>
<tr>
<td>Result</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 42: Vertical context**

<table>
<thead>
<tr>
<th>Name 1</th>
<th>Name 2</th>
<th>Name ( n )</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value 1</td>
<td>Value 2</td>
<td>Value ( n )</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 43: Horizontal context**

The context entries in a context are often used to decompose a complex expression into simpler expressions, each with a name. These context entries may be thought of as intermediate results. For example, contexts without a final Result box are useful for representing case data (see Figure 44).
**Applicant Data**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Age</td>
<td>51</td>
</tr>
<tr>
<td>MaritalStatus</td>
<td>&quot;M&quot;</td>
</tr>
<tr>
<td>EmploymentStatus</td>
<td>&quot;EMPLOYED&quot;</td>
</tr>
<tr>
<td>ExistingCustomer</td>
<td>false</td>
</tr>
<tr>
<td>Monthly Income</td>
<td>10000.00</td>
</tr>
<tr>
<td>Repayments</td>
<td>2500.00</td>
</tr>
<tr>
<td>Expenses</td>
<td>3000.00</td>
</tr>
</tbody>
</table>

**Figure 44: Use of context entries**

Contexts with a final result box are useful for representing calculations (see Figure 45).

**Eligibility**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Age</td>
<td>Applicant. Age</td>
</tr>
<tr>
<td>Monthly Income</td>
<td>Applicant. Monthly. Income</td>
</tr>
<tr>
<td>Pre-Bureau Risk Category</td>
<td>Affordability. Pre-Bureau Risk Category</td>
</tr>
<tr>
<td>Installment Affordable</td>
<td>Affordability. Installment Affordable</td>
</tr>
</tbody>
</table>

if Pre-Bureau Risk Category = "DECLINE" or
   Installment Affordable = false or
   Age < 18 or
   Monthly Income < 100
then "INELIGIBLE"
else "ELIGIBLE"

**Figure 45: Use of final result box**
When decision tables are (non-result) context entries, the output cell can be used to name the entry, thus saving space. Any format decision table can be used in a vertical context. A jagged right edge is allowed. Whitespace between context entries may be helpful.

![Figure 46: Vertical context with decision table entry](image)

Color is suggested.

The names must be legal FEEL names. The values and optional result are boxed expressions.

Boxed contexts may have a decision table as the result, and use the named context entries to compute the inputs, and give them names. For example (see Figure 47):
### Post-Bureau Risk Category

<table>
<thead>
<tr>
<th>Existing Customer</th>
<th>Applicant. ExistingCustomer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Credit Score</td>
<td>Report. CreditScore</td>
</tr>
<tr>
<td>Application Risk Score</td>
<td>Affordability Model(Applicant, Product). Application Risk Score</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>UC</th>
<th>Existing Customer</th>
<th>Application Risk Score</th>
<th>Credit Score</th>
<th>Post-Bureau Risk Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>true</td>
<td>&lt;=120</td>
<td>&lt;590</td>
<td>“HIGH”</td>
</tr>
<tr>
<td>2</td>
<td>true</td>
<td>&lt;=120</td>
<td>[590..610]</td>
<td>“MEDIUM”</td>
</tr>
<tr>
<td>3</td>
<td>true</td>
<td>&gt;610</td>
<td>“LOW”</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>false</td>
<td>&gt;120</td>
<td>&lt;600</td>
<td>“HIGH”</td>
</tr>
<tr>
<td>5</td>
<td>false</td>
<td>&gt;120</td>
<td>[600..625]</td>
<td>“MEDIUM”</td>
</tr>
<tr>
<td>6</td>
<td>false</td>
<td>&gt;625</td>
<td>“LOW”</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>false</td>
<td>&lt;=100</td>
<td>&lt;580</td>
<td>“HIGH”</td>
</tr>
<tr>
<td>8</td>
<td>false</td>
<td>&lt;=100</td>
<td>[580..600]</td>
<td>“MEDIUM”</td>
</tr>
<tr>
<td>9</td>
<td>false</td>
<td>&gt;600</td>
<td>“LOW”</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>false</td>
<td>&gt;100</td>
<td>&lt;590</td>
<td>“HIGH”</td>
</tr>
<tr>
<td>11</td>
<td>false</td>
<td>&gt;100</td>
<td>[590..615]</td>
<td>“MEDIUM”</td>
</tr>
<tr>
<td>12</td>
<td>false</td>
<td>&gt;615</td>
<td>“LOW”</td>
<td></td>
</tr>
</tbody>
</table>

Figure 47: Use of boxed expressions with a decision table

Formally, the meaning of a boxed context is \{ “Name 1”: Value 1, “Name 2”: Value 2, ..., “Name n”: Value n \} if no Result is specified. Otherwise, the meaning is \{ “Name 1”: Value 1, “Name 2”: Value 2, ..., “Name n”: Value n, “result”: Result \}.result. Recall that the bold face indicates
elements in the FEEL Semantic Domain. The scope includes the context derived from the containing DRG as described in 9.4.

### 9.2.1.5 Boxed List

A **boxed list** is a list of $n$ items. Cells must be arranged in one of the following ways (see Figure 48, Figure 49):

<table>
<thead>
<tr>
<th>Item 1</th>
<th>Item 2</th>
<th>Item $n$</th>
</tr>
</thead>
</table>

**Figure 48: Vertical list**

| Item 1, Item 2, Item $n$ |

**Figure 49: Horizontal list**

Line styles are normative. The items are boxed expressions. Formally, the meaning of a boxed list is just the meaning of the list, i.e. [ Item 1, Item 2, ..., Item $n$ ]. The scope includes the context derived from the containing DRG as described in 9.4.

### 9.2.1.6 Relation

A vertical list of homogeneous horizontal contexts (with no result cells) can be displayed with the names appearing just once at the top of the list, like a relational table, as shown in Figure 50:

<table>
<thead>
<tr>
<th>Name 1</th>
<th>Name 2</th>
<th>Name $n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value 1a</td>
<td>Value 2a</td>
<td>Value $na$</td>
</tr>
<tr>
<td>Value 1b</td>
<td>Value 2b</td>
<td>Value $nb$</td>
</tr>
<tr>
<td>Value $1m$</td>
<td>Value $2m$</td>
<td>Value $nm$</td>
</tr>
</tbody>
</table>

**Figure 50: Relation**

### 9.2.1.7 Boxed Function

The boxed expression associated with a Business Knowledge Model must be a boxed function. A boxed function has 3 cells:
1. Kind, containing the initial letter of one of the following:
   - FEEL
   - PMML
   - Java

   The Kind box can be omitted for Feel functions, including decision tables.

2. Parameters: 0 or more comma-separated names, in parentheses

3. Body: a boxed expression

The 3 cells must be arranged as shown in Figure 51:

<table>
<thead>
<tr>
<th>K</th>
<th>(Parameter1, Parameter2, …)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Body</td>
</tr>
</tbody>
</table>

Figure 51: Boxed function

For FEEL functions, the Body is an expression that references the parameters. For externally defined functions, the Body is a context as described in 9.3.2.10.2.

Formally, the meaning of a boxed function is just the meaning of the function, *i.e.*, FEEL(function(Parameter1, Parameter2, …) Body) if the Kind is FEEL, and FEEL(function(Parameter1, Parameter2, …) external Body) otherwise. The scope includes the context derived from the containing DRG as described in 9.4.

### 9.2.2 FEEL

A subset of FEEL, defined in the next section, serves as the notation "in the boxes" of boxed expressions. FEEL extends JSON objects. A JSON object is a number, a string, a context (JSON calls them maps) or a list of JSON objects. FEEL adds date, time, and duration objects, functions, friendlier syntax for literal values, and does not require the context keys to be quoted.

Here we give a "feel" for the language by starting with some simple examples.

#### 9.2.2.1 Comparison of ranges

Note that ranges and lists of ranges appear in decision table rule test cells. In the examples in Table 27, this portion of the syntax is shown in **bold**.

<table>
<thead>
<tr>
<th>FEEL Expression</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 in &lt;=5</td>
<td>true</td>
</tr>
</tbody>
</table>
Strings can also be compared. Dates, times, and durations may be compared, but only if they have been given names (by making them context entries). For example, assuming the following context entries:

- *Christmas 2012*: `date("2012-12-25")`
- *New Year’s Eve 2012*: `date("2012-12-31")`
- *Valentine’s Day 2013*: `date("2013-02-14")`

the following expression is true:

`New Year’s Eve 2012 in (Christmas 2012..Valentine's Day 2013)`.

### 9.2.2.2 Numbers

FEEL numbers and calculations are exemplified in Table 28.

<table>
<thead>
<tr>
<th>FEEL Expression</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>decimal(1, 2)</code></td>
<td>1.00</td>
</tr>
<tr>
<td><code>.25 + .2</code></td>
<td>0.45</td>
</tr>
<tr>
<td><code>.10 * 30.00</code></td>
<td>3.0000</td>
</tr>
<tr>
<td><code>1 + 3/2*2 - 2**3</code></td>
<td>-4.0</td>
</tr>
<tr>
<td><code>1/3</code></td>
<td>0.3333333333333333333333333333333333333333333</td>
</tr>
<tr>
<td><code>decimal(1/3, 2)</code></td>
<td>0.33</td>
</tr>
<tr>
<td><code>1 = 1.000</code></td>
<td>true</td>
</tr>
<tr>
<td><code>1.01/2</code></td>
<td>0.505</td>
</tr>
<tr>
<td><code>decimal(0.505, 2)</code></td>
<td>0.50</td>
</tr>
<tr>
<td><code>decimal(0.515, 2)</code></td>
<td>0.52</td>
</tr>
</tbody>
</table>
9.3 Full FEEL syntax and semantics

Section 8 introduced a subset of FEEL sufficient to support decision tables for Conformance Level 2 (see section 2). The full DMN friendly-enough expression language (FEEL) required for Conformance Level 3 is specified here. FEEL is a simple language with inspiration drawn from Java, Javascript, Xpath, SQL, PMML, Lisp, and many others.

The syntax is defined using grammar rules that show how complex expressions are composed of simpler expressions. Likewise, the semantic rules show how the meaning of a complex expression is composed from the meaning of constituent simpler expressions.

DMN completely defines the meaning of FEEL expressions that do not invoke externally-defined functions. There are no implementation-defined semantics. FEEL expressions (that do not invoke externally-defined functions) have no side-effects and have the same interpretation in every conformant implementation. Externally-defined functions SHOULD be deterministic and side-effect free.

9.3.1 Syntax

FEEL syntax is defined as grammar here and equivalently as a UML Class diagram in the meta-model (9.5).

9.3.1.1 Grammar notation

The grammar rules use the ISO EBNF notation. Each rule defines a non-terminal symbol S in terms of some other symbols S₁, S₂, ... Symbols may contain spaces. The following table summarizes the EBNF notation.

<table>
<thead>
<tr>
<th>Example</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>S = S₁ ;</td>
<td>Symbol S is defined in terms of symbol S₁</td>
</tr>
<tr>
<td>S₁</td>
<td>S₂</td>
</tr>
<tr>
<td>S₁, S₂</td>
<td>S₁ followed by S₂</td>
</tr>
<tr>
<td>[S₁]</td>
<td>S₁ occurring 0 or 1 time</td>
</tr>
<tr>
<td>{S₁}</td>
<td>S₁ repeated 0 or more times</td>
</tr>
<tr>
<td>k * S₁</td>
<td>S₁ repeated k times</td>
</tr>
<tr>
<td>&quot;and&quot;</td>
<td>literal terminal symbol</td>
</tr>
</tbody>
</table>

We extend the ISO notation with character ranges for brevity, as follows:
A character range has the following EBNF syntax:

- character range = "[", low character, ",", high character, "]" ;
- low character = unicode character ;
- high character = unicode character ;
- unicode character = simple character | code point ;
- code point = "\\u", hexadecimal digit, 4 * [hexadecimal digit] ;
- hexadecimal digit = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9" |
  "a" | "A" | "b" | "B" | "c" | "C" | "d" | "D" | "e" | "E" | "f" | "F" ;

A simple character is a single Unicode character, e.g. a, 1, $, etc. Alternatively, a character may be specified by its hexadecimal code point value, prefixed with "\u".

Every Unicode character has a numeric code point value. The low character in a range must have numeric value less than the numeric value of the high character.

For example, hexadecimal digit can be described more succinctly using character ranges as follows:

- hexadecimal digit = [0-9] [a-f] [A-F] ;

Note that the character range that includes all Unicode characters is \u0-\u10FFFF.

### 9.3.1.2 Grammar rules

The complete FEEL grammar is specified below. Grammar rules are numbered, and in some cases alternatives are lettered, for later reference. Boxed expression syntax (rule 55) is used to give execution semantics to boxed expressions.

1. expression = textual expression | boxed expression ;
2. textual expression =
   a. function definition | for expression | if expression | quantified expression |
   b. disjunction |
   c. conjunction |
   d. comparison |
   e. arithmetic expression |
   f. instance of |
   g. path expression |
   h. filter expression | function invocation |
   i. literal | unary test | name | "(", textual expression, ")" ;
3. textual expressions = textual expression , { ",", textual expression } ;
4. arithmetic expression =
   a. addition | subtraction |
b. multiplication | division |
c. exponentiation |
d. arithmetic negation |

5. simple expression = arithmetic expression | simple value |
6. simple expressions = simple expression , { "," , simple expression } |
7. simple positive unary test =
   a. [ "<" | "<=" | ">" | ">=" ] , endpoint |
   b. interval |
8. interval = ( open interval start | closed interval start ) , endpoint , "..." , endpoint , ( open interval end | closed interval end ) |
9. open interval start = "(" | "[")" |
10. closed interval start = "[" |
11. open interval end = ")" | ""] |
12. closed interval end = "]" |
13. simple positive unary tests = simple positive unary test , { "," , simple positive unary test } |
14. simple unary tests =
   a. simple positive unary tests |
   b. "not", "(" , simple positive unary tests , ")" |
   c. "" - ";
15. positive unary test = simple positive unary test | "null" |
16. positive unary tests = positive unary test , { "," , positive unary test } |
17. unary tests =
   a. positive unary tests |
   b. "not", "(" , positive unary tests , ")" |
   c. "" - ";
18. endpoint = simple value |
19. simple value = qualified name | simple literal |
20. qualified name = name , { "." , name } |
21. addition = expression , "+" , expression |
22. subtraction = expression , "-" , expression |
23. multiplication = expression , "*" , expression |
24. division = expression , "/" , expression |
25. exponentiation = expression , "+" , expression |
26. arithmetic negation = "-" , expression ;
27. name = name start , { name part | additional name symbols } ;
28. name start = name start char , { name part char } ;
29. name part = name part char , { name part char } ;
31. name part char = name start char | digit | \uB7 | [\u0300-\u036F] | [\u203F-\u2040] ;
32. additional name symbols = "." | "/" | "," | ":" | "+" | ":" ;
33. literal = simple literal | "null" ;
34. simple literal = numeric literal | string literal | Boolean literal ;
35. string literal = "" , { character – ("" | vertical space ) } , "" ;
36. Boolean literal = "true" | "false" ;
37. numeric literal = digits , [ ",", digits ] | ",", digits ;
38. digit = [0-9] ;
39. digits = digit , { digit } ;
40. function invocation = expression , parameters ;
41. parameters = "(", ( named parameters | positional parameters ) , ")" ;
42. named parameters = parameter name , ":" , expression ,
   { ",", parameter name , ":" , expression } ;
43. parameter name = name ;
44. positional parameters = [ expression , ",", expression ] ;
45. path expression = expression , ",", name ;
46. for expression = "for" , name , "in" , expression { ",", name , "in" , expression } , "return" ,
   expression ;
47. if expression = "if" , expression , "then" , expression , "else" expression ;
48. quantified expression = ("some" | "every") , name , "in" , expression ,
   { name , "in" , expression } , "satisfies" , expression ;
49. disjunction = expression , "or" , expression ;
50. conjunction = expression , "and" , expression ;
51. comparison =
   a. expression , ( "=" | ">=" | "<=" | "<" | "<=" | "">" | ">=" ) , expression |
   b. expression , "between" , expression , "and" , expression |
c. expression, "in", (positive unary test | positive unary tests);

52. filter expression = expression, "[", expression, "]";

53. instance of = expression, "instance", "of", type;

54. type = qualified name;

55. boxed expression = list | function definition | context;

56. list = "[", [expression, {"", expression}], "]";

57. function definition = "function", "("[, [formal parameter, ",", formal parameter], ")", ["external"], expression;

58. formal parameter = parameter name;

59. context = "{", [context entry, ",", context entry], "}";

60. context entry = key, ":", expression;

61. key = name | string literal;

Additional syntax rules:

- Operator precedence is given by the order of the alternatives in grammar rules 2 and 4. E.g., multiplication has higher precedence than addition, and addition has higher precedence than comparison. Addition and subtraction have equal precedence, and like all FEEL infix binary operators, are left associative.
- A name may contain spaces but may not contain a sequence of 2 or more spaces.
- Java-style comments can be used, i.e. // to end of line and /* … */.

9.3.1.3 Literals, data types, built-in functions

FEEL supports literal syntax for numbers, strings, booleans, and null. (See grammar rules, section 9.3.1.2). Literals can be mapped directly to values in the FEEL semantic domain (section 9.3.2.1).

FEEL supports the following datatypes:

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

Duration and date/time datatypes have no literal syntax. They must be constructed from a string representation using a built-in function (9.3.4.1).
9.3.1.4 Contexts, Lists, Qualified Names, and Context Lists

A context is a map of key-value pairs called context entries, and is written using curly braces to delimit the context, commas to separate the entries, and a colon to separate key and value (grammar rule 59). The key can be a string or a name. The value is an expression.

A list is written using square brackets to delimit the list, and commas to separate the list items (grammar rule 56).

Contexts and lists can reference other contexts and lists, giving rise to a directed acyclic graph. Naming is path based. The qualified name (QN) of a context entry is of the form $N_1.N_2...N_n$ where $N_1$ is the name of an in-scope context.

Nested lists encountered in the interpretation of $N_1.N_2...N_n$ are preserved. E.g.,

\[
\{[a: \{b: 1\}, \{b: \{2.1, 2.2\}\}], \{a: \{b: 3\}, \{b: 4\}, \{b: 5\}\}\}\].a.b = \\
\{[[b: 1], \{b: \{2.1, 2.2\}\}], [[b: 3], \{b: 4\}, \{b: 5\}\]}].b = \\
\{[1, [2.1, 2.2]], [3, 4, 5]\}
\]

Nested lists can be flattened using the flatten() built-in function (9.3.4).

9.3.1.5 Ambiguity

Names of context entries and function parameters can contain commonly used characters such as space and apostrophe (but cannot contain a colon or comma (':' or ',')). This naming freedom makes FEEL’s syntax ambiguous. Ambiguity is resolved using the scope. Names are matched from left to right against the names in-scope, and the longest match is preferred. In the case where the longest match is not desired, parenthesis or other punctuation (that is not allowed in a name) can be used to disambiguate a FEEL expression. For example, to subtract b from a if ‘a-b’ is the name of an in-scope context entry, one could write

\[(a) - (b)\]

9.3.2 Semantics

FEEL semantics is specified by mapping syntax fragments to values in the FEEL semantic domain. Literals (9.3.1.3) can be mapped directly. Expressions composed of literals are mapped to values in the semantic domain using simple logical and arithmetic operations on the mapped literal values. In general, the semantics of any FEEL expression are composed from the semantics of its sub-expressions. Every FEEL expression $e$ in scope $s$ can be mapped to an element $e$ in the FEEL semantic domain. This mapping defines the meaning of $e$ in $s$. The mapping may be written $e = \text{FEEL}(e,s)$. Two FEEL expressions $e_1$ and $e_2$ are equivalent in scope $s$ if and only if $\text{FEEL}(e_1,s) = \text{FEEL}(e_2,s)$. When $s$ is understood from context (or not important), we may abbreviate the equivalence as $e_1 = e_2$.

9.3.2.1 Semantic Domain

The FEEL semantic domain $D$ consists of an infinite number of values of the following kinds: functions, lists, contexts, ranges, datatypes, and the distinguished value null. Each kind is disjoint (e.g. a value cannot be both a number and a list).
A function is a lambda expression with lexical closure or is externally defined by Java or PMML. A list is an ordered collection of domain elements, and a context is a partially ordered collection of (string, value) pairs called context entries.

We use *italics* to denote syntactic elements and *boldface* to denote semantic elements. For example, FEEL(\([1+1, \ 2+2]\)) = \([2, \ 4]\)

Note that we use bold [] to denote a list in the FEEL semantic domain, and bold numbers 2, 4 to denote those decimal values in the FEEL semantic domain.

### 9.3.2.2 Semantics of literals and datatypes

FEEL datatypes are listed in 9.3.2.2. The meaning of the datatypes includes

1. a mapping from a literal form (which in some cases is a string) to a value in the semantic domain
2. a precise definition of the set of semantic domain values belonging to the datatype, and the operations on them.

Each datatype describes a (possibly infinite) set of values. The sets for the datatypes defined below are disjoint.

We use *italics* to indicate a literal and *boldface* to indicate a value in the semantic domain.

#### 9.3.2.2.1 number

FEEL Numbers are based on IEEE 754-2008 Decimal128 format, with 34 decimal digits of precision and rounding toward the nearest neighbor with ties favoring the even neighbor. Numbers are a restriction of the XML Schema type precisionDecimal, and are equivalent to Java BigDecimal with MathContext DECIMAL128.

Grammar rule 37 defines literal numbers. Literals consist of base 10 digits and an optional decimal point. –INF, +INF, and NaN literals are not supported. There is no distinction between -0 and 0. The `number(from, grouping separator, decimal separator)` built-in function supports a richer literal format. E.g. FEEL(number("1.000.000,01", ",", ",",)) = 1000000.01.

FEEL does not support a literal scientific notation. E.g., 1.2e3 is not valid FEEL syntax. Use 1.2*10**3 instead.

A FEEL number is represented in the semantic domain as a pair of integers \((p, s)\) such that \(p\) is a signed 34 digit integer carrying the precision information, and \(s\) is the scale, in the range \([-6111..6176]\). Each such pair represents the number \(p/10^s\). To indicate the numeric value, we write \(\text{value}(p, s)\). E.g. \(\text{value}(100,2)=1\). If precision is not of concern, we may write the value as simply 1. Note that many different pairs have the same value. For example, \(\text{value}(1,0) = \text{value}(10,1) = \text{value}(100,2)\).

There is no value for notANumber, positiveInfinity, or negativeInfinity. Use null instead.

#### 9.3.2.2.2 string

Grammar rule 35 defines literal strings as a double-quoted sequence of characters, e.g. "abc".

The literal string "abc" is mapped to the semantic domain as a sequence of three Unicode characters a, b, and c, written "abc".
9.3.2.2.3  boolean

The Boolean literals are given by grammar rule 36. The values in the semantic domain are \textit{true} and \textit{false}.

9.3.2.2.4  time

FEEL does not have time literals, although we use boldface time literals to represent values in the semantic domain. Times can be expressed using a string literal and the \textit{time()} built-in function. The literal format of the characters within the quotes of the string literal is defined by the lexical space of the XML Schema time datatype. A time in the semantic domain is a sequence of numbers for the hour, minute, second, and optional timezone offset.

A time \textit{t} can also be represented as the number of seconds since midnight. We write this as \textit{value}_{t}(t). E.g., \textit{value}_{t}(01:01:01)=3661. The \textit{value}_{t} function is one-to-one and so it has an inverse function \textit{value}_{t}^{-1}. E.g., \textit{value}_{t}^{-1}(3661)=01:01:01.

9.3.2.2.5  date and date-time

FEEL does not have date-time literals, although we use boldface date-time literals to represent values in the semantic domain. Dates can be expressed using a string literal and the \textit{date()} built-in function; date-times can be expressed using a string literal and the \textit{date and time()} built-in function. The literal format of the characters within the quotes of the string literal is defined by the lexical space of the XML Schema date and dateTime datatypes. A date and time in the semantic domain is a sequence of numbers for the year, month, day, and optional hour, minute, second, and timezone offset. I.e., the value for a date and time consists of a sequence of 3 numbers for the date, an optional 3 additional numbers for the time, and an optional timezone offset. The year must be in the range [-999,999,999..999,999,999].

A date and time \textit{d} can be represented as a number of seconds since a reference date and time (called the epoch). We write \textit{value}_{dt}(d) to represent the number of seconds between \textit{d} and the epoch. The \textit{value}_{dt} function is one-to-one and so it has an inverse function \textit{value}_{dt}^{-1}. E.g., \textit{value}_{dt}^{-1}(value_{dt}(d)) = d. \textit{value}_{dt}^{-1} returns \textit{null} rather than a date with a year outside the legal range.

9.3.2.2.6  days and time duration

FEEL does not have duration literals although we use boldface duration literals to represent values in the semantic domain. Durations can be expressed using a string literal and the \textit{duration()} built-in function. The literal format of the characters within the quotes of the string literal is defined by the lexical space of the XPath Data Model dayTimeDuration datatype. A days and time duration in the semantic domain is a sequence of numbers for the days, hours, minutes, and seconds of duration, normalized such that the sum of these numbers is minimized. For example, \textit{FEEL(duration("P0DT25H"))} = P1DT1H.

The value of a days and time duration can be expressed as a number of seconds. E.g., \textit{value}_{dtd}(P1DT1H) = 90000. The \textit{value}_{dtd} function is one-to-one and so it has an inverse function \textit{value}_{dtd}^{-1}. E.g., \textit{value}_{dtd}^{-1}(90000) = P1DT1H.
9.3.2.2.7 years and months duration

FEEL does not have duration literals, although we use boldface duration literals to represent values in the semantic domain. Durations can be expressed using a string literal and the `duration()` built-in function. The literal format of the characters within the quotes of the string literal is defined by the lexical space of the [XPath Data Model](http://www.w3.org/TR/xpath-data-model) `yearMonthDuration` datatype. A years and months duration in the semantic domain is a pair of numbers for the years and months of duration, normalized such that the sum of these numbers is minimized. For example, `FEEL(duration('P0Y13M')) = P1Y1M`. The value of a years and months duration can be expressed as a number of months. E.g., `value(ymd(P1Y1M)) = 13`. The `value(ymd)` function is one-to-one and so it has an inverse function `value(ymd⁻¹)`. E.g., `value(ymd⁻¹)(13) = P1Y1M`.

9.3.2.3 Ternary logic

FEEL, like SQL and PMML, uses of ternary logic for truth values. This makes `and` and `or` complete functions from `D × D → D`. Ternary logic is used in [Predictive Modeling Markup Language](http://www.dmg.org/PredictiveModelingMarkupLanguage) to model missing data values.

9.3.2.4 Lists and filters

Lists are immutable and may be nested. The first element of a list `L` can be accessed using `L[1]` and the last element can be accessed using `L[-1]`.

If `FEEL(L) = L` is a list in the FEEL semantic domain, the first element is `FEEL(L[1]) = L[1]`. If `L` is empty then `L[1] = null`.

`L` can be filtered with a boolean expression in square brackets. The expression in square brackets can reference a list element using the name `item`, unless the list element is a context that contains the key "item". If the list element is a context, then its context entries may be referenced within the filter expression without the 'item.' prefix. For example:

```
{1, 2, 3, 4}[item > 2] = {3, 4}
{ {x:1, y:2}, {x:2, y:3} }[x=1] = { {x:1, y:2} }
{ {x:1, y:2}, {x:2, y:3} }[[x=1] = {}
```

The filter expression is evaluated for each item in list, and a list containing only items where the filter expression is true is returned.

For convenience, any function or operator that expects a list as input but instead receives a non-list semantic domain element `e` behaves as if it had received `[e]` as input.

Also for convenience, a selection using the "." operator with a list of contexts on its left hand side returns a list of selections, i.e. `FEEL(e.f, c) = [ FEEL(f, c'), FEEL(f, c''), ... ]` where `FEEL(e) = [ e', e'', ... ]` and `c'` is `c` augmented with the context entries of `e'`, `c''` is `c` augmented with the context entries of `e''`, etc.

9.3.2.5 Context

A FEEL context is a partially ordered collection of (key, expression) pairs called context entries. In the syntax, keys can be either names or strings. Keys are mapped to strings in the semantic domain. These
strings are distinct within a context. A context in the domain is denoted using bold FEEL syntax with string keys, e.g. \{ "key_1" : expr_1, "key_2" : expr_2, ... \}.

The syntax for selecting the value of the entry named \textit{key}_1 from context-valued expression \textit{m} is \textit{m.key}_1.

If \textit{key}_1 is not a legal name or for whatever reason one wishes to treat the key as a string, the following syntax is allowed: \textit{get value}(\textit{m}, "key_1"). Selecting a value by key from context \textit{m} in the semantic domain is denoted as \textit{m.key}_1 or \textit{get value}(\textit{m}, "key_1")

To retrieve a list of key,value pairs from a context \textit{m}, the following built-in function may be used: \textit{get entries}(\textit{m}).

For example, the following is true:
\[
get entries({\text{key}_1 : "value_1"})[\text{key="key}_1"]\.value = "value_1"
\]

An expression in a context entry may not reference the key of the same context entry, but may reference keys (as QNs) from other context entries in the same context. These references must be acyclic and form a partial order. The expressions in a context must be evaluated consistent with this partial order.

\subsection{Ranges}

FEEL supports a compact syntax for a range of values, useful in decision table test cells and elsewhere. A range maps to the semantic domain as a single comparable value (number, date/time/duration, or string) or a pair of comparable endpoint values and an endpoint inclusivity code that indicates whether one or both endpoints are included in the range.

The range syntax supports literal and symbolic endpoints, but not arbitrary expressions. Because date/time/duration values have no literal syntax, symbolic endpoints must be used for ranges of these types. \textit{E.g.}, the following context defines a range of duration named \textit{soon} from one to two minutes, inclusive.

\[
\{
\begin{align*}
\text{one min: duration("PT1M")}, \\
\text{two min: duration("PT2M")}, \\
\text{soon: [one min..two min]}
\end{align*}
\}
\]

\subsection{Decision Table}

A decision table as described in section 6 has 10 kinds of cells:

\begin{enumerate}
\item Hit policy and Completeness Code (HC)
\item Rule number (1,2,3,...)
\item Input (I1, I2, ...)
\item Optional input values (IV1a, IV1b, IV2a, IV2b, ...)
\item Output O
\item Optional multiple outputs (O1, O2, ...)
\item Output values for each output (OV1a, OV1b)
\item Rule tests (RT1a, R21b, RT2a, RT2b, etc.)
\item Rule output (RO1a, RO1b, RO2a, RO2b ...)
\end{enumerate}
10. Aggregation (A)

E.g., the 'rules as rows' layout looks like:

<table>
<thead>
<tr>
<th>HC</th>
<th>I1</th>
<th>I2</th>
<th>O</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>IV1a, IV1b</td>
<td>IV2a, IV2b</td>
<td>O1, O2</td>
</tr>
<tr>
<td>1</td>
<td>RT1a</td>
<td>RT2a</td>
<td>RO1a, RO2a</td>
</tr>
<tr>
<td>2</td>
<td>RT2b</td>
<td>RO1b</td>
<td>RO2b</td>
</tr>
<tr>
<td>3</td>
<td>RT1b, ANY</td>
<td>RO1a</td>
<td>RO2a</td>
</tr>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The execution semantics of a decision table as described schematically above is:

**FEEL(**

decision table(
  inputs: [I1, I2, ...],
  input values: [[IV1a, IV1b], [IV2a, IV2b], ...],
  outputs: [O1, O2, ...],
  output values: [[OV1a, OV1b], [OV2a, OV2b]],
  rules:
    [RT1a, RT2a,..., RO1a, RO2a],
    [RT1a, RT2b,..., RO1b, RO2b],
    [RT1b, ANY,..., RO1a, RO2a],
  hit policy: H, completeness: C, aggregation: A ) )

That is, the meaning is defined by the meaning of the FEEL built-in decision table function (see 9.3.4.5).

9.3.2.7.1 Name

Note that we do not include the Name cell here because only the top-level boxed expression has a Name cell, and the decision table may not be the top-level BE. When a decision table is the top-level boxed expression, it must have a name and the output cell must be blank or the same as the name. When a decision table is an entry in a boxed context, the name cell of the context entry may be omitted. In this case, the output cell must contain the context entry name and cannot be blank unless it is the last entry. When a decision table is not top-level and not a context entry, the decision table has no name and the output cell is uninterpreted.

9.3.2.7.2 Hit policy

The hit policy is described in the specification of the decision table built-in (9.3.4.5). The default hit policy is "U", for Unique. Crosstab decision tables always have unique hits.

Customized hit policies such as returning the maximum value, can be added easily using a boxed expression to post-process the output list.
9.3.2.7.3 Complete
Completeness is described in the specification of the decision table built-in (9.3.4.5). The default value is "C", for Complete.

9.3.2.7.4 Input
Each input cell is a boxed expression. The expression must be a textual expression (grammar rule 2).

9.3.2.7.5 Input values
Input values are optional. If provided, it is a comma-separated list of FEEL expressions. Each input value must be a unary test (grammar rule 10). A list of input values must be an instance of unary tests (grammar rule 11).

The list restricts the corresponding input to the given list of values, mapping others to null. The rule test cell can test for null as a proxy "otherwise", that is, to test for values not in the list of input values.

9.3.2.7.6 Output
When a decision table is an entry in a boxed context, the output cell may be used to contain the name of the entry, instead of a separate entry name cell, thus saving space.

9.3.2.7.7 Output Values
If provided, the list of output values must be an instance of textual expressions (grammar rule 3).

9.3.2.7.8 Rule tests
Rule test cells, also called input entries, contain FEEL expressions restricted to contain only a (list of) unary test (grammar rule 10) or a list of unary tests (grammar rule 11).

The meaning of rule matching is composed from these rule test expressions. The rule tests are combined with corresponding inputs to form in expressions (FEEL grammar rule 51.c) that are anded (FEEL grammar rule 50) to determine whether a rule is matched.

9.3.2.7.9 Rule output
A rule output cell is a boxed expression. The expression must be a textual expression (grammar rule 2).

9.3.2.7.10 Aggregation
The aggregation is described in the specification of the decision table built-in (9.3.4.5). The default value is “collect”.

9.3.2.8 Scope and context stack
A FEEL expression e is always evaluated in a well-defined set of name bindings that are used to resolve QNs in e. This set of name bindings is called the scope of e. Scope is modeled as a list of contexts. A scope s contains the contexts with entries that are in scope for e. The last context in s is the built-in context. Next to last in s is the global context. The first context in s is the context immediately containing e (if any). Next are enclosing contexts of e (if any).
The QN of \( e \) is the QN of the first context in \( s \) appended with .N, where N is the name of entry in the first context of \( s \) containing \( e \). QNs in \( e \) are resolved by looking through the contexts in \( s \) from first to last.

9.3.2.8.1 Local context

If \( e \) denotes the value of a context entry of context \( m \), then \( m \) is the local context for \( e \), and \( m \) is the first element of \( s \). Otherwise, \( e \) has no local context and the first element of \( s \) is the global context, or in some cases explained later, the first element of \( s \) is a special context.

All of the entries of \( m \) are in-scope for \( e \), but the \textit{depends on} graph must be acyclic. This provides a simple solution to the problem of the confusing definition above: if \( m \) is the result of evaluating the context expression \( m \) that contains \( e \), how can we know it in order to evaluate \( e \)? Simply evaluate the context entries in \textit{depends on} order.

9.3.2.8.2 Global context

The global context is a context provided for convenience and 'pre-compilation'. Any number of expressions can be named and represented in a FEEL context \( m \). The syntactic description \( m \) of this context can be evaluated once, that is, mapped to the FEEL domain as \( m \), and then re-used to evaluate many expressions.

9.3.2.8.3 Built-in context

The built-in context contains all the built-in functions.

9.3.2.8.4 Special context

Some FEEL expressions are interpreted in a \textit{special} context that is pushed on the front of \( s \). For example, a filter expression is repeatedly executed with special first context containing the name 'item' bound to successive list elements. A function is executed with a special first context containing argument name->value mappings.

Qualified names (QNs) in FEEL expressions are interpreted relative to \( s \). The meaning of a FEEL expression \( e \) in scope \( s \) is denoted as \texttt{FEEL}(\( e \), \( s \)). We can also say that \( e \) evaluates to \( e \) in scope \( s \), or \( e = \texttt{FEEL}(e, s) \). Note that \( e \) and \( s \) are elements of the FEEL domain. \( s \) is a list of contexts.

9.3.2.9 Mapping between FEEL and other domains

A FEEL expression \( e \) denotes a value \( e \) in the semantic domain. Some kinds of values can be passed between FEEL and external Java methods, between FEEL and external PMML models, and between FEEL and XML, as summarized in the following table:

<table>
<thead>
<tr>
<th>FEEL value</th>
<th>Java</th>
<th>XML</th>
<th>PMML</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>java.math.BigDecimal</td>
<td>decimal</td>
<td>decimal, PROB-NUMBER, PERCENTAGE-NUMBER</td>
</tr>
<tr>
<td></td>
<td></td>
<td>integer</td>
<td>integer, INT-NUMBER</td>
</tr>
<tr>
<td></td>
<td></td>
<td>double</td>
<td>double, REAL-NUMBER</td>
</tr>
<tr>
<td>string</td>
<td>java.lang.String</td>
<td>string</td>
<td>string, FIELD-NAME</td>
</tr>
<tr>
<td>date and time,</td>
<td>javax.xml.datatype.</td>
<td>date, dateTime</td>
<td>date, dateTime, time</td>
</tr>
</tbody>
</table>
Sometimes we do not want to evaluate a FEEL expression e, we just want to know the type of e. Note that if e has QNs, then a context may be needed for type inference. We write \texttt{type}(e) as the type of the domain element \texttt{FEEL}(e, c).

### 9.3.2.10 Function Semantics

FEEL functions can be

- built-in, \textit{e.g.}, \textit{decision table} (see section 9.3.4.5), or
- user-defined, \textit{e.g.}, \textit{function(age) age < 21}, or
- externally defined, \textit{e.g.},
  \textit{function(angle) external }
  \texttt{
  \{
  java: \\
  \quad \texttt{class: "java.lang.Math",} \\
  \quad \texttt{method signature: "cos(double)"}
  \}}

FEEL built-ins are specified in 9.3.4.

#### 9.3.2.10.1 User-defined functions

User-defined functions have the form

\[
\textit{function}(X_1, \ldots, X_n) \ e
\]

The terms \(X_1, \ldots, X_n\) are parameter names. The function body is \(e\). The meaning of \texttt{FEEL(function}(X_1, \ldots, X_n) \ e, s) is an element in the FEEL semantic domain that we denote as \texttt{function(argument list: [X_1, \ldots, X_n], body: e, scope: s)} (shortened to \texttt{f} below). FEEL functions are lexical closures, \textit{i.e.}, the body is an expression that references the formal parameters and any other names in scope \(s\).

The invocation of a FEEL user-defined function \(f\) is denoted as \(f(Y_1, \ldots, Y_n)\). The meaning \texttt{FEEL(f}(Y_1, \ldots, Y_n), S), where \(f\) has already been interpreted, is computed as follows:
1. the parameter names \(X_1, \ldots, X_n\) and corresponding values \(Y_1, \ldots, Y_n\) are paired in a context \(c = \{X_1 : Y_1, \ldots, X_n : Y_n\}\). \(Y_i = \text{FEEL}(Y_i, S)\).
2. \(e\) is interpreted in \(s'\), where \(s' = \text{insert before}(s, 1, c)\) (see 9.3.4.4)

### 9.3.2.10.2 Externally-defined functions

FEEL externally-defined functions have the following form

\[
\text{function}(X_1, \ldots, X_n) \text{ external mapping-information}
\]

Mapping-information is a context that must have one of the following forms:

\[
\{
  \text{java: \{class: class-name, method signature: method-signature\}}
\}
\]

or

\[
\{
  \text{pmml: \{document: IRI, model: model-name\}}
\}
\]

The meaning of an externally defined function is an element in the semantic domain that we denote as \(\text{function(\text{argument list: } [X_1, \ldots, X_n], \text{external: mapping-information})}\). In the mapping information, \textit{class-name} is a string name of a Java class on the classpath. Classpath configuration is implementation-defined. \textit{Method-signature} is a string consisting of the name of a public static method in the named class, and an argument list containing only Java argument type names. The argument type information is used to resolve overloaded methods and may be used to detect out-of-domain errors before runtime. \textit{IRI} is the resource identifier for a PMML document, and \textit{model-name} is an optional name of a model in the document. If no model-name is specified, the first model in the document is used.

When an externally-defined function is invoked, actual argument values and result value are converted when possible using the type mapping table for Java or PMML [ref to this table in Types and Inference]. When a conversion is not possible, \texttt{null} is substituted. If a result cannot be obtained, \textit{e.g.} an exception is thrown, the result of the invocation is \texttt{null}.

Passing parameter values to the external method or model requires knowing the expected parameter types. For Java, this information is obtained using reflection. For PMML, this information is obtained from the mining schema and data dictionary elements associated with independent variables of the selected model.

Note that DMN does not completely define the semantics of a Decision Model that uses externally-defined functions. Externally-defined functions should have no side-effects and be deterministic.

### 9.3.2.10.3 Function name

To name a function, define it as a context entry. \textit{E.g.}

\[
\{
  \text{isPositive : function(x) } x > 0,
  \text{isNotNegative : function(x) isPositive(x+1)},
\}
\]
result: isNotNegative(0)

9.3.2.10.4 Positional and named parameters
An invocation of any FEEL function (built-in, user-defined, or externally-defined) can use positional parameters or named parameters. If positional, all parameters must be supplied. If named, unsupplied parameters are bound to `null`.

9.3.2.11 Semantic mappings
The meaning of each substantive grammar rule is given below by mapping the syntax to a value in the semantic domain. The value may depend on certain input values, themselves having been mapped to the semantic domain. The input values may have to obey additional constraints. The input domain(s) may be a subset of the semantic domain. Inputs outside of their domain result in a `null` value.

<table>
<thead>
<tr>
<th>Grammar Rule</th>
<th>FEEL Syntax</th>
<th>Mapped to Domain</th>
</tr>
</thead>
<tbody>
<tr>
<td>57</td>
<td><code>function(n_1, ..., n_N) e</code></td>
<td><code>function(argument list: [n_1, ..., n_N], body: e, scope: s)</code></td>
</tr>
<tr>
<td>57</td>
<td><code>function(n_1, ..., n_N) external e</code></td>
<td><code>function(argument list: [n_1, ..., n_N], external: e)</code></td>
</tr>
</tbody>
</table>

See 9.3.2.6.

<table>
<thead>
<tr>
<th>Grammar Rule</th>
<th>FEEL Syntax</th>
<th>Mapped to Domain</th>
</tr>
</thead>
<tbody>
<tr>
<td>46</td>
<td><code>for n_1 in e_1, n_2 in e_2, ... return e</code></td>
<td><code>[ FEEL(e, s'), FEEL(e, s''), ... ]</code></td>
</tr>
<tr>
<td>47</td>
<td><code>if e_1 then e_2 else e_3</code></td>
<td><code>if FEEL(e_1) then FEEL(e_2) else FEEL(e_3)</code></td>
</tr>
<tr>
<td>48</td>
<td><code>some n_1 in e_1, n_2 in e_2, ... satisfies e</code></td>
<td><code>FEEL(e, s') or FEEL(e, s'') or ...</code></td>
</tr>
<tr>
<td>48</td>
<td><code>every n_1 in e_1, n_2 in e_2, ... satisfies e</code></td>
<td><code>FEEL(e, s') and FEEL(e, s'') and ...</code></td>
</tr>
<tr>
<td>49</td>
<td><code>e_1 or e_2 or ...</code></td>
<td><code>FEEL(e_1) or FEEL(e_2) or ...</code></td>
</tr>
<tr>
<td>50</td>
<td><code>e_1 and e_2 and ...</code></td>
<td><code>FEEL(e_1) and FEEL(e_2) and ...</code></td>
</tr>
<tr>
<td>51.a</td>
<td><code>e = null</code></td>
<td><code>FEEL(e) = null</code></td>
</tr>
<tr>
<td>51.a</td>
<td><code>e != null</code></td>
<td><code>FEEL(e) != null</code></td>
</tr>
</tbody>
</table>
Notice that we use bold syntax to denote contexts, lists, conjunctions, disjunctions, conditional expressions, true, false, and null in the FEEL domain.

The meaning of the conjunction \( a \) and \( b \) and the disjunction \( a \) or \( b \) is defined by ternary logic. Because these are total functions, the input can be true, false, or otherwise (meaning any element of \( D \) other than true or false).

<table>
<thead>
<tr>
<th>( a )</th>
<th>( b )</th>
<th>( a ) and ( b )</th>
<th>( a ) or ( b )</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>true</td>
<td>otherwise</td>
<td>null</td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>otherwise</td>
<td>false</td>
<td>null</td>
</tr>
<tr>
<td>otherwise</td>
<td>true</td>
<td>null</td>
<td>true</td>
</tr>
<tr>
<td>otherwise</td>
<td>false</td>
<td>false</td>
<td>null</td>
</tr>
<tr>
<td>otherwise</td>
<td>otherwise</td>
<td>null</td>
<td>null</td>
</tr>
</tbody>
</table>

Negation is accomplished using the built-in function \textbf{not}. The ternary logic is

<table>
<thead>
<tr>
<th>( a )</th>
<th>\textbf{not}(( a ))</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>otherwise</td>
<td>null</td>
</tr>
</tbody>
</table>

A conditional \textbf{if} \( a \) \textbf{then} \( b \) \textbf{else} \( c \) is equal to \( b \) if \( a \) is true, and equal to \( c \) otherwise.

\( s' \) is the scope \( s \) with a special first context containing keys \( n_1, n_2, \) etc. bound to the first element of the Cartesian product of \( \text{FEEL}(e_1) \) x \( \text{FEEL}(e_2) \) x ..., \( s'' \) is \( s \) with a special first context containing keys bound to the second element of the Cartesian product, etc.

<table>
<thead>
<tr>
<th>Grammar Rule</th>
<th>FEEL Syntax</th>
<th>Input Domain</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>51.a</td>
<td>( e_1 &lt; e_2 )</td>
<td>( e_1 ) and ( e_2 ) must both be of the same kind/datatype – both numbers, both strings, \textit{etc.}</td>
<td>\textit{See below}</td>
</tr>
</tbody>
</table>

Equality and inequality map to several kind- and datatype-specific tests:
<table>
<thead>
<tr>
<th>kind/datatype</th>
<th>expression</th>
</tr>
</thead>
</table>
| list          | $e_1 = e_2$  
  lists must be same length N and $e_1[i] = e_2[i]$ for $1 \leq i \leq N$. |
| context       | contexts must have same set of keys K and $e_1.k = e_2.k$ for every k in K |
| range         | the ranges must specify the same endpoints and the same endpoint inclusivity code. |
| function      | internal functions must have the same parameters, body, and scope. Externally defined functions must have the same parameters and external mapping information. |
| number        | value($e_1$) = value($e_2$). Value is defined in 9.3.2.2.1. Precision is not considered. |
| string        | $e_1$ is the same sequence of characters as $e_2$ |
| date and time | all 7 components (9.3.2.2.5), treating unspecified optional components as null, must be equal |
| time          | all 4 components (9.3.2.2.4), treating unspecified optional components as null, must be equal |
| days and time duration | value($e_1$) = value($e_2$). Value is defined in 9.3.2.2.6. |
| years and months duration | value($e_1$) = value($e_2$). Value is defined in 9.3.2.2.7. |
| boolean       | $e_1$ and $e_2$ must both be true or both be false |

By definition, $\text{FEEL}(e_1 \neq e_2) = \text{FEEL}(\text{not}(e_1=e_2))$.

The other comparison operators are defined only for the datatypes as shown in the following table. ‘>’ is similar to ‘<’ and is omitted for brevity. $e_1 < e_2$ is defined as $e_1 \leq e_2$ or $e_1 = e_2$.

<table>
<thead>
<tr>
<th>datatype</th>
<th>expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>value($e_1$) &lt; value($e_2$). Value is defined in 9.3.2.2.1. Precision is not considered.</td>
</tr>
<tr>
<td>string</td>
<td>sequence of characters $e_1$ is lexicographically less than the sequence of characters $e_2$. I.e., the sequences are padded to the same length if needed with \u0 characters, stripped of common prefix characters, and then the first character in each sequence is compared.</td>
</tr>
<tr>
<td>date and time</td>
<td>value$_{dt}$(e$<em>1$) &lt; value$</em>{dt}$(e$<em>2$). value$</em>{dt}$ is defined in 9.3.2.2.5. If one input has a null timezone offset, that input uses the timezone offset of the other input.</td>
</tr>
</tbody>
</table>
time

\( \text{value}(e_1) < \text{value}(e_2) \). \text{value} is defined in 9.3.2.2.4. If one input has a null timezone offset, that input uses the timezone offset of the other input.

days and time duration

\( \text{value}_{\text{dtd}}(e_1) < \text{value}_{\text{dtd}}(e_2) \). \text{value}_{\text{dtd}} is defined in 9.3.2.2.6.

years and months duration

\( \text{value}_{\text{ymd}}(e_1) = \text{value}_{\text{ymd}}(e_2) \). \text{value}_{\text{ymd}} is defined in 9.3.2.2.7.

FEEL supports additional syntactic sugar for comparison. Note that Grammar Rules (section 9.3.1.2) are used in decision table condition cells.

<table>
<thead>
<tr>
<th>Grammar Rule</th>
<th>FEEL Syntax</th>
<th>Equivalent FEEL Syntax</th>
<th>applicability</th>
</tr>
</thead>
<tbody>
<tr>
<td>51.b</td>
<td>( e_1 \text{ between } e_2 \text{ and } e_3 )</td>
<td>( e_1 \geq e_2 \text{ and } e_1 \leq e_3 )</td>
<td></td>
</tr>
<tr>
<td>51.c</td>
<td>( e_1 \text{ in } [e_2,e_3,...] )</td>
<td>( e_1 = e_2 \text{ or } e_1 = e_3 \text{ or...} )</td>
<td>( e_2 \text{ and } e_3 \text{ are endpoints} )</td>
</tr>
<tr>
<td>51.c</td>
<td>( e_1 \text{ in } [e_2,e_3,...] )</td>
<td>( e_1 \in e_2 \text{ or } e_1 \in e_3 \text{ or...} )</td>
<td>( e_2 \text{ and } e_3 \text{ are ranges} )</td>
</tr>
<tr>
<td>51.c</td>
<td>( e_1 \in \leq e_2 )</td>
<td>( e_1 \leq e_2 )</td>
<td></td>
</tr>
<tr>
<td>51.c</td>
<td>( e_1 \in &lt;e_2 )</td>
<td>( e_1 &lt; e_2 )</td>
<td></td>
</tr>
<tr>
<td>51.c</td>
<td>( e_1 \in \geq e_2 )</td>
<td>( e_1 \geq e_2 )</td>
<td></td>
</tr>
<tr>
<td>51.c</td>
<td>( e_1 \in &lt;e_2 )</td>
<td>( e_1 &lt; e_2 )</td>
<td></td>
</tr>
<tr>
<td>51.c</td>
<td>( e_1 \in (e_2..e_3) )</td>
<td>( e_1 &gt; e_2 \text{ and } e_1 &lt; e_3 )</td>
<td></td>
</tr>
<tr>
<td>51.c</td>
<td>( e_1 \in (e_2..e_3] )</td>
<td>( e_1 &gt; e_2 \text{ and } e_1 \leq e_3 )</td>
<td></td>
</tr>
<tr>
<td>51.c</td>
<td>( e_1 \in [e_2..e_3) )</td>
<td>( e_1 &gt; e_2 \text{ and } e_1 &lt; e_3 )</td>
<td></td>
</tr>
<tr>
<td>51.c</td>
<td>( e_1 \in [e_2..e_3] )</td>
<td>( e_1 &gt; e_2 \text{ and } e_1 \leq e_3 )</td>
<td></td>
</tr>
</tbody>
</table>

In Grammar Rule 51c, the qualified name must evaluate to a comparable constant value at modeling time, i.e. the endpoint must be a literal or a named constant.

<table>
<thead>
<tr>
<th>Grammar Rule</th>
<th>FEEL</th>
<th>Input Domain and Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>21</td>
<td>( e_1 + e_2 )</td>
<td>\text{See below}</td>
</tr>
<tr>
<td>22</td>
<td>( e_1 - e_2 )</td>
<td>\text{See below}</td>
</tr>
</tbody>
</table>

Addition and subtraction are defined in the following table. Note that if input values are not of the listed types, the result is null.

<table>
<thead>
<tr>
<th>type( (e_1) )</th>
<th>type( (e_2) )</th>
<th>( e_1 + e_2, \ e_1 - e_2 )</th>
<th>result type</th>
</tr>
</thead>
</table>

Let $e_1 = (p_1, s_1)$ and $e_2 = (p_2, s_2)$ as defined in 9.3.2.2.1. If \( \text{value}(p_1, s_1) +/- \text{value}(p_2, s_2) \) requires a scale outside the range of valid scales, the result is \textbf{null}. Else the result is \((p, s)\) such that

- \( s \leq \max(s_1, s_2) \)
- \( s \) is maximized subject to the limitation that \( p \) has 34 digits or less
- \( \varepsilon \) is a possible rounding error.

| number | number | Let $e_1 = (p_1, s_1)$ and $e_2 = (p_2, s_2)$ as defined in 9.3.2.2.1. If \( \text{value}(p_1, s_1) +/- \text{value}(p_2, s_2) \) requires a scale outside the range of valid scales, the result is \textbf{null}. Else the result is \((p, s)\) such that
- \( s \leq \max(s_1, s_2) \)
- \( s \) is maximized subject to the limitation that \( p \) has 34 digits or less
- \( \varepsilon \) is a possible rounding error. |
| date and time | date and time | Addition is undefined. Subtraction is defined as \( \text{value}_{\text{dtd}}^{-1}(\text{value}_{\text{dt}}(e_1) - \text{value}_{\text{dt}}(e_2)) \), where \( \text{value}_{\text{dt}} \) is defined in 9.3.2.2.5 and \( \text{value}_{\text{dtd}}^{-1} \) is defined in 9.3.2.2.6. |
| time | time | Addition is undefined. Subtraction is defined as \( \text{value}_{\text{dtd}}^{-1}(\text{value}_{\text{dt}}(e_1) - \text{value}_{\text{dt}}(e_2)) \), where \( \text{value}_{\text{dt}} \) is defined in 9.3.2.2.4 and \( \text{value}_{\text{dtd}}^{-1} \) is defined in 9.3.2.2.6. |
| years and months duration | years and months duration | \( \text{value}_{\text{ymd}}^{-1}(\text{value}_{\text{ymd}}(e_1) +/- \text{value}_{\text{ymd}}(e_2)) \) where \( \text{value}_{\text{ymd}} \) and \( \text{value}_{\text{ymd}}^{-1} \) is defined in 9.3.2.2.7. |
| days and time duration | days and time duration | \( \text{value}_{\text{dtd}}^{-1}(\text{value}_{\text{dtd}}(e_1) +/- \text{value}_{\text{dtd}}(e_2)) \) where \( \text{value}_{\text{dtd}} \) and \( \text{value}_{\text{dtd}}^{-1} \) is defined in 9.3.2.2.6 |
| date and time | years and months duration | \( \text{value}_{\text{dt}}^{-1}(\text{value}_{\text{dt}}(e_1) +/- \text{value}_{\text{ymd}}(e_2)) \) where \( \text{value}_{\text{dt}} \) and \( \text{value}_{\text{dt}}^{-1} \) is defined in 9.3.2.2.5 and \( \text{value}_{\text{ymd}} \) is defined in 9.3.2.2.7. |
| years and months duration | date and time | Subtraction is undefined. Addition is commutative and is defined by the previous rule. |
| date and time | days and time duration | \( \text{value}_{\text{dt}}^{-1}(\text{value}_{\text{dt}}(e_1) +/- \text{value}_{\text{dtd}}(e_2)) \) where \( \text{value}_{\text{dt}} \) and \( \text{value}_{\text{dt}}^{-1} \) is defined in 9.3.2.2.5 and \( \text{value}_{\text{dtd}} \) is defined in 9.3.2.2.6. |
| days and time duration | date and time | Subtraction is undefined. Addition is commutative and is defined by the previous rule. |
| time | days and time | \( \text{value}_{\text{t}}^{-1}(\text{value}_{\text{t}}(e_1) +/- \text{value}_{\text{dtd}}(e_2)) \) where \( \text{value}_{\text{t}} \) and \( \text{value}_{\text{t}}^{-1} \) is defined in 9.3.2.2.4 and \( \text{value}_{\text{dtd}} \) is |
time duration defined in 9.3.2.6.

Subtraction is undefined. Addition is commutative and is defined by the previous rule.

time
days and time duration

<table>
<thead>
<tr>
<th>Grammar</th>
<th>FEEL</th>
<th>Input Domain and Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rule</td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>(e_1 \ast e_2)</td>
<td>See below</td>
</tr>
<tr>
<td>24</td>
<td>(e_1 / e_2)</td>
<td>See below</td>
</tr>
</tbody>
</table>

Multiplication and division are defined in the following table. Note that if input values are not of the listed types, the result is null.

<table>
<thead>
<tr>
<th>type(e_1)</th>
<th>type(e_2)</th>
<th>(e_1 \ast e_2)</th>
<th>(e_1 / e_2)</th>
<th>result type</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>number</td>
<td>If (\text{value}(p_1,s_1) \ast \text{value}(p_2,s_2)) requires a scale outside the range of valid scales, the result is null. Else the result is ((p,s)) such that</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(\text{value}(p,s) = \text{value}(p_1,s_1) \ast \text{value}(p_2,s_2) + \varepsilon)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(s \leq s_1 + s_2)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(s) is maximized subject to the limitation that (p) has 34 digits or less</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(\varepsilon) is a possible rounding error</td>
<td></td>
<td></td>
</tr>
<tr>
<td>number</td>
<td>number</td>
<td>If (\text{value}(p_2,s_2)=0) or (\text{value}(p_1,s_1) / \text{value}(p_2,s_2)) requires a scale outside the range of valid scales, the result is null. Else the result is ((p,s)) such that</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(\text{value}(p,s) = \text{value}(p_1,s_1) / \text{value}(p_2,s_2) + \varepsilon)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(s \leq s_1 - s_2)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(s) is maximized subject to the limitation that (p) has 34 digits or less</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(\varepsilon) is a possible rounding error</td>
<td></td>
<td></td>
</tr>
<tr>
<td>years and months duration</td>
<td>number</td>
<td>(\text{value}<em>{\text{ymd}}^{-1}(\text{value}</em>{\text{ymd}}(e_1) \ast \text{value}<em>{\text{ymd}}(e_2))) where (\text{value}</em>{\text{ymd}}) and (\text{value}_{\text{ymd}}^{-1}) are defined in 9.3.2.2.7.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>number</td>
<td>years and months duration</td>
<td>If (\text{value}(e_2)=0), the result is null. Else the result is (\text{value}<em>{\text{ymd}}^{-1}(\text{value}</em>{\text{ymd}}(e_1) / \text{value}(e_2))) where (\text{value}<em>{\text{ymd}}) and (\text{value}</em>{\text{ymd}}^{-1}) are defined in 9.3.2.2.7.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[\text{see above, reversing } e_1 \text{ and } e_2\]
<table>
<thead>
<tr>
<th>Grammar Rule</th>
<th>FEEL Syntax</th>
<th>Input Domain</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>$e_1 \times \times e_2$</td>
<td>$\text{type}(e_1)$ is number. $\text{value}(e_2)$ is an integer in the range ([-999,999,999..999,999,999]).</td>
<td>If $\text{value}(e_2)=0$, the result is null. Else the result is $\text{value}<em>{\text{dd}}^{-1}(\text{value}</em>{\text{dd}}(e_1) \times \text{value}(e_2))$ where $\text{value}<em>{\text{dd}}$ and $\text{value}</em>{\text{dd}}^{-1}$ are defined in 9.3.2.2.6.</td>
</tr>
</tbody>
</table>

Note that $\text{type}$ is not mapped to the domain, and $\text{null}$ is not the name of a type, and $\text{null}$ is not an instance of any type.

<table>
<thead>
<tr>
<th>Grammar Rule</th>
<th>FEEL Syntax</th>
<th>Mapped to Domain</th>
</tr>
</thead>
<tbody>
<tr>
<td>53</td>
<td>$e \text{ instance of type}$</td>
<td>true iff type$(e)$ is type</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Grammar Rule</th>
<th>FEEL Syntax</th>
<th>Equivalent FEEL Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>26</td>
<td>-$e$</td>
<td>0-$e$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Grammar Rule</th>
<th>FEEL Syntax</th>
<th>Mapped to Domain</th>
<th>Applicability</th>
</tr>
</thead>
<tbody>
<tr>
<td>40, 41, 44</td>
<td>$e(e_1,..)$</td>
<td>$e(e_1,..)$</td>
<td>$e$ is a function with matching arity</td>
</tr>
</tbody>
</table>
An invocation can use positional arguments or named arguments. If positional, all arguments must be supplied. If named, unsupplied arguments are bound to **null**. Note that **e** can be a user-defined function, a user-defined external function, or a built-in function.

<table>
<thead>
<tr>
<th>Grammar Rule</th>
<th>FEEL</th>
<th>Mapped to Domain</th>
<th>Applicability</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td><strong>e.name</strong></td>
<td><strong>e.&quot;name&quot;</strong></td>
<td><strong>type(e)</strong> is a context</td>
</tr>
<tr>
<td>20</td>
<td><strong>e.name</strong></td>
<td>see below</td>
<td><strong>type(e)</strong> is a date/time/duration</td>
</tr>
</tbody>
</table>

If **type(e)** is date and time, time, or a duration, and **name** is a property name as shown in the following table, then the meaning is given by the following table:

<table>
<thead>
<tr>
<th><strong>type(e)</strong></th>
<th><strong>e . name</strong></th>
<th><strong>name</strong> =</th>
</tr>
</thead>
<tbody>
<tr>
<td>date and time</td>
<td>result is the <strong>named</strong> component of the date and time object <strong>e</strong>. Valid names are shown to the right. hour, minute, second, and timezone values may be <strong>null</strong>.</td>
<td>year, month, day, hour, minute, second, timezone</td>
</tr>
<tr>
<td>time</td>
<td>result is the <strong>named</strong> component of the time object <strong>e</strong>. Valid names are shown to the right. timezone values may be <strong>null</strong>.</td>
<td>hour, minute, second, timezone</td>
</tr>
<tr>
<td>years and months duration</td>
<td>result is the <strong>named</strong> component of the years and months duration object <strong>e</strong>. Valid names are shown to the right.</td>
<td>years, months</td>
</tr>
<tr>
<td>days and time duration</td>
<td>result is the <strong>named</strong> component of the days and time duration object <strong>e</strong>. Valid names are shown to the right.</td>
<td>days, hours, minutes, seconds</td>
</tr>
</tbody>
</table>

For example, **FEEL**(date and time("03-07-2012Z").year) = **2012**.
value(e_2) = 1

| 56 | e_1[e_2] | list of items e such that i is in e iff i is in e_1 and FEEL(e_2, s') = true, where s' is the scope s with a special first context containing the context entry ("item", i) and if i is a context, the special context also contains all the context entries of i. | e_1 is a list and type(FEEL(e_2, s')) is boolean |

| 56 | e_1[e_2] | [e_1] if FEEL(e_2, s') = true, where s' is the scope s with a special first context containing the context entry ("item", e_1) and if e_1 is a context, the special context also contains all the context entries of e_1. Else []. | e_1 is not a list and not null and type(FEEL(e_2, s')) is boolean |

<table>
<thead>
<tr>
<th>Grammar Rule</th>
<th>FEEL Syntax</th>
<th>Mapped to Domain (scope s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>59</td>
<td>{ n_1 : e_1, n_2 : e_2, ...}</td>
<td>{ &quot;n_1&quot;: FEEL(e_1, s_1), &quot;n_2&quot;: FEEL(e_2, s_2), ...} such that the s_i are all s with a special first context c_i containing a subset of the entries of this result context. If c_i contains the entry for n_j, then c_j does not contain the entry for n_i.</td>
</tr>
<tr>
<td>56</td>
<td>[e_1, e_2, ...]</td>
<td>[ FEEL(e_1), FEEL(e_2), ...]</td>
</tr>
</tbody>
</table>

9.3.2.12 Error Handling
When a built-in function encounters input that is outside its defined domain, the function should report or log diagnostic information if appropriate, and must return null.

9.3.3 XML Data
FEEL supports XML Data in the FEEL context by mapping XML Data into the FEEL Semantic Domain. Let XE(e, p) be a function mapping an XML element e and a parent FEEL context p to a FEEL context, as defined in the following tables. XE makes use of another mapping function, XV(v), that maps an XML value v to the FEEL semantic domain.
XML namespace semantics are not supported by the mappings. For example, given the namespace prefix declarations `xmlns:p1="http://example.org/foobar"` and `xmlns:p2="http://example.org/foobar"`, the tags `p1:myElement` and `p2:myElement` are the same element using XML namespace semantics but are different using XML without namespace semantics.

### 9.3.3.1 Semantic mapping for XML elements (XE)

Herein, `e` is the name of an XML element, `a` is the name of one of its attributes, `c` is a child element, and `v` is a value. The parent context `p` is initially empty.

<table>
<thead>
<tr>
<th>XML</th>
<th>context entry in p</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;e /&gt;</code></td>
<td>&quot;e&quot; : null</td>
<td>empty element → null-valued entry in <code>p</code></td>
</tr>
<tr>
<td><code>&lt;q:e /&gt;</code></td>
<td>&quot;q$e&quot; : null</td>
<td>namespaces are ignored. Colonized names are changed to legal identifiers.</td>
</tr>
<tr>
<td><code>&lt;e&gt;v&lt;/e&gt;</code></td>
<td>&quot;e&quot;:<code> XV(v)</code></td>
<td>unrepeated element without attributes</td>
</tr>
<tr>
<td><code>&lt;e&gt;v1&lt;/e&gt; &lt;e&gt;v2&lt;/e&gt;</code></td>
<td>&quot;e&quot;: [ <code>XV(v1) , XV(v2)</code> ]</td>
<td>repeating element without attributes</td>
</tr>
<tr>
<td><code>&lt;e a=&quot;v&quot;/&gt; &lt;c1&gt;v1&lt;/c1&gt; &lt;c2&gt;v2&lt;/c2&gt;vcn</code></td>
<td>&quot;e&quot;: { &quot;@a&quot;: <code> XV(v)</code>, &quot;c1&quot;: <code> XV(v1)</code>, &quot;cn&quot;: [ <code>XV(v2) , XV(v3)</code> ] }</td>
<td>attribute names are prefixed with @. An element containing attributes or child elements → context</td>
</tr>
<tr>
<td><code>&lt;e a=&quot;v&quot;/&gt; v2</code></td>
<td>&quot;e&quot;: { &quot;@a&quot;: <code> XV(v1)</code>, &quot;$content&quot;: <code>XV(v2)</code> }</td>
<td><code>v2</code> is contained in a generated $content entry</td>
</tr>
</tbody>
</table>

An entry in the FEEL Domain mapping column such as "e" : null indicates a context entry with string key "e" and value null. The context entries are contained by context `p` that corresponds to the containing XML element, or to the XML document itself.

The mapping does not replace namespace prefixes with the namespace IRIs. FEEL requires only that keys within a context be distinct, and the namespace prefixes are sufficient.

### 9.3.3.2 Semantic mapping for XML values (XV)

If an XML document was parsed with a schema, then some atomic values may have a datatype other than string. The following table defines how a typed XML value `v` is mapped to FEEL.

<table>
<thead>
<tr>
<th>Type of <code>v</code></th>
<th>FEEL Semantic Domain</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td><code>FEEL(v)</code></td>
</tr>
<tr>
<td>string</td>
<td><code>FEEL(&quot;v&quot;)</code></td>
</tr>
<tr>
<td>date</td>
<td>&quot;@a&quot;: FEEL(date(&quot;v&quot;))</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------------------</td>
</tr>
<tr>
<td>dateTime</td>
<td>&quot;@a&quot;: FEEL(date and time(&quot;v&quot;))</td>
</tr>
<tr>
<td>time</td>
<td>&quot;@a&quot;: FEEL(time(&quot;v&quot;))</td>
</tr>
<tr>
<td>duration</td>
<td>&quot;@a&quot;: FEEL(duration(&quot;v&quot;))</td>
</tr>
<tr>
<td>list, e.g. &quot;v₁ v₂&quot;</td>
<td>[ XV(v₁), XV(v₂) ]</td>
</tr>
<tr>
<td>element</td>
<td>XE(v)</td>
</tr>
</tbody>
</table>

### 9.3.3.3 XML example

The following schema and instance are equivalent to the following FEEL:

#### 9.3.3.3.1 schema

```xml
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="http://www.example.org"
targetNamespace="http://www.example.org"
  elementFormDefault="qualified">
  <xsd:element name="Context">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="Employee">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="salary" type="xsd:decimal"/>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
        <xsd:element name="Customer" maxOccurs="unbounded">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="loyalty_level" type="xsd:string"/>
              <xsd:element name="credit_limit" type="xsd:decimal"/>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```
9.3.3.3.2 instance

```xml
<Context xmlns:tns="http://www.example.org"
          xmlns="http://www.example.org">
  <tns:Employee>
    <tns:salary>13000</tns:salary>
  </tns:Employee>
  <Customer>
    <loyalty_level>gold</loyalty_level>
    <credit_limit>10000</credit_limit>
  </Customer>
  <Customer>
    <loyalty_level>gold</loyalty_level>
    <credit_limit>20000</credit_limit>
  </Customer>
  <Customer>
    <loyalty_level>silver</loyalty_level>
    <credit_limit>5000</credit_limit>
  </Customer>
</Context>
```

9.3.3.3 equivalent FEEL context

```feel
{  
  Context: {  
    tns$Employee: { tns$salary: 13,000 },  
    Customer: [  
      { loyalty_level: "gold", credit_limit: 10000 },  
      { loyalty_level: "gold", credit_limit: 20000 },  
      { loyalty_level: "silver", credit_limit: 5000 } ]  
  }  
}
```

9.3.4 Built-in functions

To promote interoperability, FEEL includes a library of built-in functions. The syntax and semantics of the built-ins are required for a conformant FEEL implementation.

9.3.4.1 Conversion functions

FEEL supports many conversions between values of different types. Of particular importance is the conversion from strings to dates, times, and durations. There is no literal representation for date, time, or duration. Also, formatted numbers such as \(1,000.00\) must be converted from a string by specifying the grouping separator and the decimal separator.
Built-ins are summarized in the following table. The first column shows the name and parameters. A question mark (?) denotes an optional parameter. The second column specifies the domain for the parameters. The parameter domain is specified as one of

- a type, e.g. number, string
- any – any element from the semantic domain, including null
- not null – any element from the semantic domain, excluding null.

A superscript means additional constraints on the domain are described later. Whenever a parameter is outside its domain, the result of the built-in is null.

<table>
<thead>
<tr>
<th>Name(parameters)</th>
<th>Parameter Domain</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>date(from)</td>
<td>string&lt;sup&gt;1&lt;/sup&gt;</td>
<td>convert from to a date</td>
<td>date(&quot;2012-12-25&quot;) – date(&quot;2012-12-24&quot;) = duration(&quot;P1D&quot;)</td>
</tr>
<tr>
<td>date(from)</td>
<td>date and time</td>
<td>convert from to a date (set time components to null)</td>
<td>date( date and time(&quot;2012-12-25T11:00:00Z&quot;)) = date(&quot;2012-12-25&quot;)</td>
</tr>
<tr>
<td>date and time(from)</td>
<td>string&lt;sup&gt;2&lt;/sup&gt;</td>
<td>convert from to a date and time</td>
<td>date and time(&quot;2012-12-24T23:59:00&quot;) + duration(&quot;PT1M&quot;) = date and time(&quot;2012-12-25T00:00:00&quot;)</td>
</tr>
<tr>
<td>time(from)</td>
<td>string&lt;sup&gt;3&lt;/sup&gt;</td>
<td>convert from to time</td>
<td>time(&quot;23:59:00&quot;) + duration(&quot;PT2M&quot;) = time(&quot;00:01:00&quot;)</td>
</tr>
<tr>
<td>time(from)</td>
<td>time, date and time</td>
<td>convert from to time (ignoring date components)</td>
<td>time( date and time(&quot;2012-12-25T11:00:00Z&quot;)) = time(&quot;11:00:00Z&quot;)</td>
</tr>
<tr>
<td>decimal(from, scale)</td>
<td>number, number&lt;sup&gt;4&lt;/sup&gt;</td>
<td>set the scale of from</td>
<td>number(1/3, 2) = .33</td>
</tr>
<tr>
<td>number(from, grouping separator, decimal separator)</td>
<td>all 3 are string&lt;sup&gt;5&lt;/sup&gt;</td>
<td>convert from to a number</td>
<td>number(&quot;1 000,0&quot;, &quot;&quot;,&quot;&quot;,&quot;&quot;) = number(&quot;1,000.0&quot;, &quot;,&quot;,&quot;&quot;)</td>
</tr>
<tr>
<td>string(from)</td>
<td>non-null</td>
<td>convert from to a string</td>
<td>string(1.1) = &quot;1.1&quot;</td>
</tr>
<tr>
<td>duration(from)</td>
<td>string&lt;sup&gt;6&lt;/sup&gt;</td>
<td>convert from to a days and time or years and months duration</td>
<td>date and time(&quot;2012-12-24T23:59:00&quot;) - date and time(&quot;2012-12-22T03:45:00&quot;) = duration(&quot;P2DT20H14M&quot;)</td>
</tr>
<tr>
<td>years and months duration(from, to)</td>
<td>both are date and time</td>
<td>return duration between from and to</td>
<td>years and months duration( date(&quot;2011-12-22&quot;), date(&quot;2013-08-24&quot;) ) = duration(&quot;P1Y9M&quot;)</td>
</tr>
</tbody>
</table>

Additional constraints on input domains:

DMN Revised Submission bmi/2013-08-01
1. *from* must be in the lexical space of \texttt{xs:date} [ref xml schema]
2. *from* must be in the lexical space of \texttt{xs:dateTime} [ref xml schema]
3. *from* must be in the lexical space of \texttt{xs:time} [ref xml schema]
4. scale is in the range \([-6111..6176]\]
5. grouping separator is space (\'\'), comma (\',\'), period (\'.\'), or null. Decimal separator is period, comma, or null, but not the same as the grouping separator (unless null), *from* must have syntax specified by grammar rule \texttt{37}, after removing grouping separators (if not null) and after changing the decimal separator (if not null) to a period.
6. *from* must be in the lexical space of the \texttt{xs:dayTimeDuration} or \texttt{xs:yearMonthDuration} [ref xpath]

**Issues:**

1. Extended time zone, e.g. 23:59:00@America/Los_Angeles

### 9.3.4.2 Boolean function

| \texttt{not(negand)} | boolean | logical negation | \texttt{not(true)} = \texttt{false}  
| \texttt{not(null)} = \texttt{null} |

### 9.3.4.3 String functions

| \texttt{substring(string, start position, length?)} | string, number\(^1\) | return length (or all) characters in \texttt{string}, starting at \texttt{start position}.  
\(1^{st}\) position is 1, last position is -1 | \texttt{substring("foobar",3)} = \texttt{"obar"}  
\texttt{substring("foobar",3,3)} = \texttt{"oba"}  
\texttt{substring("foobar", -2, 1)} = \texttt{"a"} |
| \texttt{string length(string)} | string | return length of \texttt{string} | \texttt{string length("foo")} = 3 |
| \texttt{upper case(string)} | string | return uppercased \texttt{string} | \texttt{upper case("aBc4")} = \texttt{"ABC4"} |
| \texttt{lower case(string)} | string | return lowercased \texttt{string} | \texttt{lower case("aBc4")} = \texttt{"abc4"} |
| \texttt{substring before (string, match)} | string, string | return substring of \texttt{string} before the \texttt{match} in \texttt{string} | \texttt{substring before("foobar", \"bar\")} = \texttt{"foo"}  
\texttt{substring before("foobar", \"xyz\")} = \texttt{\"\"} |
| \texttt{substring after (string, match)} | string, string | return substring of \texttt{string} after the \texttt{match} in \texttt{string} | \texttt{substring after("foobar", \"ob\")} = \texttt{"ar"}  
\texttt{substring after("\"", \"a\")} = \texttt{\"\"} |
<p>| \texttt{replace(input, pattern, replacement, flags?)} | string(^2) | regular expression pattern matching and replacement | \texttt{replace(&quot;abcd&quot;, &quot;(ab){(a)&quot;, &quot;[1=$1][2=$2]&quot;)} = &quot;[1=ab][2=]cd&quot; |
| \texttt{contains(string, match)} | string | does the \texttt{string} contain the \texttt{match}? | \texttt{contains(&quot;foobar&quot;, &quot;of&quot;) = false} |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>starts with</td>
<td>string, match</td>
<td>string</td>
<td>does the string start with the</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>match?</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>starts with(&quot;foobar&quot;, &quot;fo&quot;) = true</td>
</tr>
<tr>
<td>ends with</td>
<td>string, match</td>
<td>string</td>
<td>does the string end with the</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>match?</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ends with(&quot;foobar&quot;, &quot;r&quot;) = true</td>
</tr>
<tr>
<td>matches</td>
<td>input, pattern,</td>
<td>string²</td>
<td>does the input match</td>
</tr>
<tr>
<td></td>
<td>flags?</td>
<td></td>
<td>the regexp pattern?</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>matches(&quot;foobar&quot;, &quot;fo*b&quot;) = true</td>
</tr>
</tbody>
</table>

**Additional constraints on input domains:**

1. *start position* must be a non-zero integer (0 scale number) in the range \([-L..L]\), where \(L\) is the length of the string. *length* must be in the range \([1..E]\), where \(E\) is \(L – start position\) if *start position* is positive,
   and \(-start position\) otherwise.

2. *flags* must be ...

### 9.3.4.4 List functions

<table>
<thead>
<tr>
<th>list contains(list, element)</th>
<th>list, any element of the semantic domain including null</th>
<th>does the list contain the element?</th>
<th>list contains([1,2,3], 2) = true</th>
</tr>
</thead>
<tbody>
<tr>
<td>count(list)</td>
<td>list</td>
<td>return size of list</td>
<td>count([1,2,3]) = 3</td>
</tr>
<tr>
<td>min(list)</td>
<td>list of comparable items</td>
<td>return minimum(maximum) item</td>
<td>min([1,2,3]) = 1 max([1,2,3]) = 3</td>
</tr>
<tr>
<td>max(list)</td>
<td>list of comparable items</td>
<td>return sum of list items</td>
<td>sum([1,2,3]) = 6</td>
</tr>
<tr>
<td>mean(list)</td>
<td>list of numbers</td>
<td>return arithmetic mean (average)</td>
<td>mean([1,2,3]) = 2</td>
</tr>
<tr>
<td>and(list)</td>
<td>list</td>
<td>return conjunction of list items</td>
<td>and([true,null,true]) = false</td>
</tr>
<tr>
<td>or(list)</td>
<td>list</td>
<td>return disjunction of list items</td>
<td>or([false,null,true]) = true</td>
</tr>
<tr>
<td>sublist(list, start position, length?)</td>
<td>list, number¹, number²</td>
<td>return list of length (or all)</td>
<td>sublist([1,2,3], 1, 2) = [2]</td>
</tr>
<tr>
<td>append(list, item...)</td>
<td>list, any element including null</td>
<td>return new list with items</td>
<td>append([1], 2, 3) = [1,2,3]</td>
</tr>
<tr>
<td>concatenate(list...)</td>
<td>list</td>
<td>return new list that is a</td>
<td>concatenate([1,2],[3]) = [1,2,3]</td>
</tr>
</tbody>
</table>
concatenation of the arguments

<table>
<thead>
<tr>
<th>Function</th>
<th>Domain</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>insert before(list, position, newItem)</td>
<td>list, number(^\d), any element including null</td>
<td>return new list with newItem inserted at position</td>
<td>insert before([1,3],1,2) = [1,2,3]</td>
</tr>
<tr>
<td>remove(list, position)</td>
<td>list, number(^\d)</td>
<td>list with item at position removed</td>
<td>remove([1,2,3], 2) = [1,3]</td>
</tr>
<tr>
<td>reverse(list)</td>
<td>list</td>
<td>reverse the list</td>
<td>reverse([1,2,3]) = [3,2,1]</td>
</tr>
<tr>
<td>index of(list, match)</td>
<td>list, any element including null</td>
<td>return ascending list of list positions containing match</td>
<td>index of([1,2,3,2],2) = [2,4]</td>
</tr>
<tr>
<td>union(list...)</td>
<td>list</td>
<td>concatenate with duplicate removal</td>
<td>union([1,2],[2,3]) = [1,2,3]</td>
</tr>
<tr>
<td>distinct values(list)</td>
<td>list</td>
<td>duplicate removal</td>
<td>distinct values([1,2,3,2,1] = [1,2,3]</td>
</tr>
<tr>
<td>flatten(list)</td>
<td>list</td>
<td>flatten nested lists</td>
<td>flatten(([1,2],[[3], 4]) = [1,2,3,4]</td>
</tr>
</tbody>
</table>

**Additional constraints on input domains:**

1. *position* must be a non-zero integer (0 scale number) in the range \([-L..L]\), where \(L\) is the length of the list.
2. *length* must be in the range \([1..E]\), where \(E = L – start\ position\) if *start position* is positive, and –*start position* otherwise.

### 9.3.4.5 Decision Table

The normative notation for decision tables is specified in section 7. A textual representation as a builtin function is provided here in order to link the semantics with the syntax in the same way as is done for the rest of FEEL. Note that the parameters correspond to classes and associations in the metamodel in section 7.3.

**Parameters** (* means optional):

<table>
<thead>
<tr>
<th>Name</th>
<th>Domain</th>
</tr>
</thead>
<tbody>
<tr>
<td>input expressions</td>
<td>a list of (N&gt;0) expressions, each legal as the LHS of an <em>in</em> expression</td>
</tr>
<tr>
<td>input values*</td>
<td>list of (N) unary test ((15)) lists. A unary test restricts the associated input to a particular value or range of values.</td>
</tr>
<tr>
<td>outputs*</td>
<td>a name ((27)) or list of (M&gt;0) names</td>
</tr>
<tr>
<td>output values*</td>
<td>if outputs is a list, a list of lists of values, one list per output; else a list of values for the one output</td>
</tr>
<tr>
<td>rules</td>
<td>a list of 0 or more rules. A rule is a list of (N) input entries followed by (M) output entries. An input entry is a list or a unary test ((15)). The list contains 0 or more unary test.</td>
</tr>
<tr>
<td>hit policy*</td>
<td>&quot;Unique&quot;, &quot;Any&quot;, &quot;Priority&quot;, &quot;First&quot;, &quot;NoOrder&quot;, &quot;RuleOrder&quot;, &quot;OutputOrder&quot; (default is &quot;Unique&quot;)</td>
</tr>
<tr>
<td>completeness*</td>
<td>&quot;C&quot;, &quot;I&quot; (default is &quot;C&quot;)</td>
</tr>
</tbody>
</table>
aggregation* | “collect”, "sum", "min", "max", "mean", or "count"

An input entry containing the empty list, corresponding to a “-“ in the notation, matches any input.

The decision table function evaluates a decision table that is specified using N input expressions, M outputs, and R rules. Each rule has N+M entries.

The decision table 'input expressions' parameter is a list of N expressions. The decision table 'rules' parameter is a list of R rules.

Each rule is a list containing N input entries, one per input, and M output entries.

A rule is matched if all its FEEL equivalent tests are true. Input entry i in rule j is equivalent to one of the following FEEL expressions:

1. If rules[j][i] is positive unary tests (grammar rule 16), the FEEL equivalent test is inputs[i] in rules[j][i].
2. If rules[j][i] is unary tests of the form not(x), where x is positive unary tests, the FEEL equivalent test is not(inputs[i] in x).
3. If rules[j][i] is unary tests of the form ' - ', the FEEL equivalent test is true.

A rule is hit if it is matched and the hit policy indicates that the matched rule's output value should be included in the decision table result. Each hit results in one output value (multiple outputs are collected into a single context value). Therefore, multiple hits result in an output list.

The hit policy is specified as one of the following:

- **Unique** – only a single rule can be matched.
- **Any** – multiple rules can match, but they all have the same output,
- **Priority** – multiple rules can match, with different outputs. The output that comes first in the supplied output values list is returned,
- **First** – return the first match in rule order,
- **No Order** – return a list of outputs arbitrary order,
- **Rule Order** – return a list of outputs in rule order,
- **Output Order** – return a list of outputs in the order of the output values list.

To reduce complexity, decision tables with compound outputs support only the following hit policies: Unique, Any, First, No order, and Rule order.

Completeness is specified using the initial letter of one of the following:

- **Complete** – user asserts all input combinations (rule test values) are accounted for. Tools can challenge this assertion. If the assertion is wrong at runtime, the decision table function will return null.
- **Incomplete** – table is incomplete. Return first element of output values list (or null if output values is not supplied) if no rules match.

Completeness can be checked as follows. The input entries must cover all combinations of the input values. If an input expression has no corresponding input values, then the input values are inferred from the input entries as follows: equality tests add the test value to the input values. Inequality tests require
that the input entries must cover all non-null values of that input, by using range expressions such as \(<0, [0..100]\), etc.

A decision table may have no rule hit for a set of input values. In this case, the result is **null**.

The value of the Completeness flag is ignored for execution, but should be used for design-time validation.

The semantics of a decision table are as follows:

1. Rules are tested in arbitrary order, except for first-hit tables, where rules are tested in rule order.
2. If no rules are matched, **null** is returned
3. Let \( m \) be the set of rules matched
4. If No Order, return the list of outputs, one for each \( m \)
5. if Unique or Any return the output for \( m[1] \)
6. if Rule Order, order \( m \) by rule index and return the list of outputs, one for each rule in \( m \)
7. else if Output Order, order \( m \) by output values and return the list of outputs, one for each rule in \( m \).
   If multiple outputs, order each from left to right or top to bottom.

### 9.3.4.6 Sort

Sort a list using an ordering function.

**Parameters** (* means optional):

<table>
<thead>
<tr>
<th>Name</th>
<th>Domain</th>
</tr>
</thead>
<tbody>
<tr>
<td>list</td>
<td>list of any element, be careful with nulls</td>
</tr>
<tr>
<td>precedes</td>
<td>boolean function of 2 arguments defined on every pair of list elements</td>
</tr>
</tbody>
</table>

For example,

```
sort(list: [3,1,4,5,2], precedes: function(x,y) x < y) = [1,2,3,4,5]
```

### 9.4 Relationship of FEEL to DRG and Boxed Expressions

FEEL is used to give execution semantics to Decision Models, that is, to a DRG and its associated Boxed Expressions. Many DRG Elements may have an associated expression, in the form of a boxed expression as described in section 9.2.1. Usage of expressions by DRG Elements is summarized in the following table.

<table>
<thead>
<tr>
<th>DRG Element</th>
<th>Associated Boxed Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>Business Knowledge Model</td>
<td>function (parameterized decision logic)</td>
</tr>
<tr>
<td>Decision</td>
<td>invocation or any other decision logic</td>
</tr>
<tr>
<td>Input Data</td>
<td>sample data</td>
</tr>
</tbody>
</table>
Boxed expressions contain other boxed expressions or FEEL expressions described in 9.2.2. FEEL expressions are given meaning by the FEEL Semantics also described in 9.2.2. Boxed expressions are given meaning by mapping them to FEEL expressions. DRGs can also be given meaning (execution semantics) by mapping them to FEEL expressions.

Let \( D \) be a DRG with elements \( d_1, d_2, \ldots \). Each element has a name \( n \) and a decision logic expression \( e \). The FEEL expression for \( D \) is the context \( \{ n_1:e_1, n_2:e_2, \ldots \} \), such that:

- the context entries are partially ordered by requirements (e.g. the entry for an Input Data element comes before a Decision that uses it)
- the context entries are partially ordered by name reference (referent first)

In other words, the boxed expressions of some DRG elements refer to other DRG elements by name, and each of those relationships must be reflected as a requirement.

### 9.5 Metamodel

![Expression class diagram](image)

The class Expression is extended to support the four new kinds of boxed expressions introduced by FEEL, namely: Context, FunctionDefinition, Relation and List.
Boxed expressions are Expressions that have a standard diagrammatic representation in DMN (see sections 6.2.1 and 9.2.1). FEEL contexts, function definitions, relations and lists SHOULD be modeled as Context, FunctionDefinition, Relation and List elements, respectively, and represented as a boxed expression whenever possible; that is, when they are top-level expressions, since an instance of LiteralExpression cannot contain another Expression element.

9.5.1 Context metamodel

A Context is composed of any number of contextEntries, which are instances of ContextEntry.

A Context element is represented diagrammatically as a boxed context (section 9.2.1.4). A FEEL context (grammar rule 59 and section 9.3.2.5) SHOULD be modeled as a Context element whenever possible.

Context inherits all the attributes and model associations from Expression. Table 29 presents the additional attributes and model associations of the Context element.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>contextEntry</td>
<td>This attribute lists the instances of ContextEntry that compose this Context.</td>
</tr>
</tbody>
</table>

9.5.2 ContextEntry metamodel

The class ContextEntry is used to model FEEL context entries when a context is modeled as a Context element.

An instance of ContextEntry is composed of an optional variable, which is an InformationItem element whose name is the key in the context entry, and of a value, which is the instance of Expression that models the expression in the context entry.

Table 30 presents the attributes and model associations of the ContextEntry element.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>variable</td>
<td>The instance of InformationItem that is contained in this ContextEntry, and whose name is the key in the modeled context entry</td>
</tr>
</tbody>
</table>
### 9.5.3 FunctionDefinition metamodel

A FunctionDefinition has formalParameters and a body. A FunctionDefinition element is represented diagrammatically as a boxed function, as described in section 0. A FEEL function definition (grammar rule 57 and section 9.3.2.11) SHOULD be modeled as a FunctionDefinition element whenever possible.

FunctionDefinition inherits all the attributes and model associations from Expression. Table 31 presents the additional attributes and model associations of the FunctionDefinition element.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FormalParameter: InformationItem [*]</td>
<td>This attributes lists the instances of InformationItem that are the parameters of this Context.</td>
</tr>
<tr>
<td>body: Expression [0..1]</td>
<td>The instance of Expression that is the body in this FunctionDefinition</td>
</tr>
</tbody>
</table>

### 9.5.4 List metamodel

A List is simply a list of element, which are instances of Expressions. A List element is represented diagrammatically as a boxed list, as described in section 9.2.1.5. A FEEL list (grammar rule 56 and section 9.3.2.11) SHOULD be modeled as a List element whenever possible.

List inherits all the attributes and model associations from Expression. Table 32 presents the additional attributes and model associations of the List element.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>element: Expression [*]</td>
<td>This attributes lists the instances of Expression that are the elements in this List.</td>
</tr>
</tbody>
</table>
9.5.5 Relation metamodel

A Relation is convenient shorthand for a list of similar contexts. A Relation has a column instead of repeated ContextEntries, and a List is used for every row, with one of the List’s expression for each column value.

Relation inherits all the attributes and model associations from Expression. Table 33 presents the additional attributes and model associations of the Relation element.

Table 33: Relation attributes and model associations

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>row: List[*]</td>
<td>This attributes lists the instances of List that compose the rows of this Relation.</td>
</tr>
<tr>
<td>column: InformationItem[*]</td>
<td>This attributes lists the instances of InformationItem that define the columns in this Relation.</td>
</tr>
</tbody>
</table>

9.6 Examples

A good way to get a quick overview of FEEL is by example.

FEEL expressions may reference other FEEL expressions by name. Named expressions are contained in a context. Expressions are evaluated in a scope, which is a list of contexts in which to resolve names. The result of the evaluation is an element in the FEEL semantic domain.

9.6.1 Context

Figure 53 shows the boxed context used as an instance of input data for the examples.

<table>
<thead>
<tr>
<th>applicant</th>
<th>age</th>
<th>51</th>
</tr>
</thead>
<tbody>
<tr>
<td>maritalStatus</td>
<td>&quot;M&quot;</td>
<td></td>
</tr>
<tr>
<td>existingCustomer</td>
<td>false</td>
<td></td>
</tr>
<tr>
<td>monthly income</td>
<td>10000</td>
<td></td>
</tr>
<tr>
<td>repayments</td>
<td>2500</td>
<td></td>
</tr>
<tr>
<td>expenses</td>
<td>3000</td>
<td></td>
</tr>
<tr>
<td>requested product</td>
<td>product type</td>
<td>&quot;STANDARD LOAN&quot;</td>
</tr>
<tr>
<td>rate</td>
<td>0.25</td>
<td></td>
</tr>
<tr>
<td>term</td>
<td>36</td>
<td></td>
</tr>
<tr>
<td>amount</td>
<td>100000.00</td>
<td></td>
</tr>
</tbody>
</table>
Notice that there are 6 top-level context entries, represented by the six rows of the table. The value of the context entry named 'applicant' is itself a context, and the value of the context entry named 'monthly' is itself a context. The value of the context entry named 'monthly outgoings' is a list, the value of the context entry named 'credit history' is a relation, i.e. a list of two contexts, one context per row. The value of the context entry named 'PMT' is a function with parameters 'rate', 'term', and 'amount'.

The following examples use the above context. Each example has a pair of equivalent FEEL expressions separated by a horizontal line. Both expressions denote the same element in the semantic domain. The second expression, the ‘answer’, is a literal value.

### 9.6.2 Calculation

\[
\text{monthly income} \times 12
\]

\[
120000
\]

### 9.6.3 If, In

\[
\text{if applicant.maritalStatus in ["M","S"] then "valid" else "not valid"}
\]

"valid"

### 9.6.4 Sum entries of a list

\[
\text{sum(monthly outgoings)}
\]

5500

### 9.6.5 Invocation of user-defined PMT function

\[
\text{PMT(requested product . rate,}
\]

\[
\text{requested product . term,}
\]

\[
\text{requested product . amount)}
\]

3975.982590125562

### 9.6.6 Sum weights of recent credit history

\[
\text{sum(credit history[record date > date("2011-01-01")].weight)}
\]

150
9.6.7 Determine if credit history contain a bankruptcy event

some ch in credit history satisfies ch.event = "bankruptcy"

false

10DMN Example

In this section we present an example of the use of DMN to model the decision-making to be automated in decision services called from a business process management system modeled in BPMN.

10.1 The business process model

Figure 54 shows a simple process for loan originations, modeled in BPMN 2.0. The process handles applications for a loan, obtaining data from a credit bureau only if required for the case, and deciding whether the application should be accepted, declined, or referred for human review. It consists of the following components:

- The Collect application data task collects data describing the Requested product and the Applicant (e.g. through an on-line application form).
- The Bureau Strategy task calls a decision service, passing Requested product and Applicant data. The service returns two decisions: Strategy and Bureau call type.
- A gateway uses the value of Strategy to route the case to Decline application, Collect bureau data or Application routing.
- The Collect bureau data task collects data from a credit bureau according to the Bureau call type decision, then the case is passed to Application routing.
- The Application routing task calls a decision service, passing Requested product, Applicant data and Bureau data (if the Collect bureau data task was not performed, the Bureau data are set to null). The service returns a single decision: Routing.
- A gateway uses the value of Routing to route the case to Accept application, Review application or Decline application.
- The Review application task allows a credit officer to review the case and decide whether it should be accepted or declined.
- A gateway uses the credit officer’s decision to route the case to Accept application or Decline application.
- The Accept application task informs the applicant that their application is accepted and initiates the product.
- The Decline application task informs the applicant that their application is declined.

Note that in this example the two decision points (calls to decision services) are represented in BPMN 2.0 as business rule tasks.
10.2 The decision requirements level

Figure 55 shows a DRD of all the decisions to be automated in this business process. There are three data inputs to be provided to the decision services (Requested product, Applicant data and Bureau data), and three decisions to be returned by the decision services (Strategy, Bureau call type and Routing). Between the two are intermediate decisions: evaluations of risk, affordability and eligibility. Note that the two services share some common elements: e.g. the Required Monthly Installment decision and the Affordability calculation business knowledge model are required for both Strategy and Routing. Other notable features of this DRD include:
- Some decisions (e.g. Pre-bureau risk category) and input data (e.g. Applicant data) are required by multiple decisions, i.e. the information requirements network is not a tree
- Business knowledge models (see Affordability calculation) may be invoked by multiple decisions
- Business knowledge models (see Credit contingency factor) may be invoked by other business knowledge models.

Figure 55: DRD of all automated decision-making

It might be considered more convenient to draw separate (but overlapping) DRDs for the two decision points. Figure 56 shows the DRD of the decisions required for the Bureau Strategy decision point (i.e. the requirements subgraph of the Strategy and Bureau Call Type decisions), and Figure 57 shows the DRD for the Application Routing decision point (i.e. the requirements subgraph of the Routing decision). All three DRDs – Figure 55, Figure 56 and Figure 57 – are views of the same DRG.
Figure 56: DRD for Bureau Strategy decision point

Figure 57: DRD for Application Routing decision point
The DRG depicted in these DRDs shows dependencies between the following decisions:

- The **Strategy** decision, requiring the Bureau call type and Pre-bureau eligibility decisions, invokes the Strategy table shown in Figure 59
- The **Bureau call type** decision, requiring the Pre-bureau risk category decision, invokes the Bureau call type table shown in Figure 61
- The **Eligibility** decision, requiring Applicant data and the Pre-bureau risk category and Pre-bureau affordability decisions, invokes the Eligibility rules shown in Figure 63
- The **Pre-bureau affordability** decision, requiring Applicant data and the Pre-bureau risk category and Required monthly installment decisions, invokes the Affordability calculation boxed expression shown in Figure 74, which in turn invokes the Credit contingency factor table shown in Figure 75
- The **Pre-bureau risk category** decision, requiring Applicant data and the Application risk score decision, invokes the Pre-bureau risk category table shown in Figure 65
- The **Application risk score** decision, requiring Applicant data, invokes the score model shown in Figure 67
- The **Routing** decision, requiring Bureau data and the Post-bureau affordability and Post-bureau risk category decisions, invokes the Routing rules shown in Figure 69
- The **Post-bureau affordability** decision, requiring Applicant data and the Post-bureau risk score and Required monthly installment decisions, invokes the Affordability calculation boxed expression shown in Figure 74, which in turn invokes the Credit contingency factor table shown in Figure 75
- The **Post-bureau risk category** decision, requiring Applicant and Bureau data and the Application risk score decision, invokes the Post-bureau risk category table shown in Figure 71.
- The **Required monthly installment** decision, requiring Requested product data, invokes the Installment calculation boxed expression shown in Figure 77.

### 10.3 The decision logic level

The DRG in Figure 55 is defined in more detail in the following specifications of the value expressions associated with decisions and business knowledge models:

- The **Strategy** decision logic (Figure 58) invokes the Strategy table business knowledge model, passing the output of the Bureau call type decision as the Bureau Call Type parameter, and the output of the Eligibility decision as the Eligibility parameter.
- The **Strategy Table** decision logic (Figure 59) defines a complete, unique-hit decision table deriving Strategy from Eligibility and Bureau Call Type.
- The **Bureau Call Type** decision logic (Figure 60) invokes the Bureau call type table, passing the output of the Pre-bureau risk category decision as the Pre-Bureau Risk Category parameter.
• The **Bureau call type table** decision logic (Figure 61) defines a complete, unique-hit decision table deriving Bureau Call Type from Pre-Bureau Risk Category.

• The **Eligibility** decision logic (Figure 62) invokes the Eligibility rules business knowledge model, passing Applicant data, Age as the Age parameter, the output of the Pre-bureau risk category decision as the Pre-Bureau Risk Category parameter, and the output of the Pre-bureau affordability decision as the Pre-Bureau Affordability parameter.

• The **Eligibility Rules** decision logic (Figure 63) defines a complete, priority-ordered single-hit decision table deriving Eligibility from Pre-Bureau Risk Category, Pre-Bureau Affordability and Age.

• The **Pre-Bureau Risk Category** decision logic (Figure 64) invokes the Pre-bureau risk category table business knowledge model, passing Applicant data, ExistingCustomer as the Existing Customer parameter and the output of the Application risk score decision as the Application Risk Score parameter.

• The **Pre-Bureau Risk Category Table** decision logic (Figure 65) defines a complete, unique-hit decision table deriving Pre-Bureau Risk Category from Existing Customer and Application Risk Score.

• The **Application Risk Score** decision logic (Figure 66) invokes the Application risk score model business knowledge model, passing Applicant data, Age as the Age parameter, Applicant data, MaritalStatus as the Marital Status parameter and Applicant data, EmploymentStatus as the Employment Status parameter.

• The **Application Risk Score Model** decision logic (Figure 67) defines a complete, no-order multiple-hit table with aggregation, deriving Application risk score from Age, Marital Status and Employment Status, as the sum of the Partial scores of all matching rows (this is therefore a predictive scorecard represented as a decision table).

• The **Routing** decision logic (Figure 68) invokes the Routing rules business knowledge model, passing Bureau data, Bankrupt as the Bankrupt parameter, Bureau data, CreditScore as the Credit Score parameter, the output of the Post-bureau risk category decision as the Post-Bureau Risk Category parameter, and the output of the Post-bureau affordability decision as the Post-Bureau Affordability parameter. Note that if Bureau data is null (due to the THROUGH strategy bypassing the Collect bureau data task) the Bankrupt and Credit Score parameters will be null.

• The **Routing Rules** decision logic (Figure 69) defines a complete, priority-ordered single-hit decision table deriving Routing from Post-Bureau Risk Category, Post-Bureau Affordability, Bankrupt and Credit Score.

• The **Post-Bureau Risk Category** decision logic (Figure 70) invokes the Post-bureau risk category business knowledge model, passing Applicant data, ExistingCustomer as the Existing Customer parameter, Bureau data, CreditScore as the Credit Score parameter, and the output of the Application risk score decision as the Application Risk Score parameter. Note that if Bureau data is null (due to the THROUGH strategy bypassing the Collect bureau data task) the Credit Score parameter will be null.
- The **Post-bureau risk category table** decision logic (Figure 71) defines a complete, unique-hit decision table deriving Post-Bureau Risk Category from Existing Customer, Application Risk Score and Credit Score.

- The **Pre-bureau Affordability** decision logic (Figure 72) invokes the Affordability calculation business knowledge model, passing Applicant data . Monthly . Income as the Monthly Income parameter, Applicant data . Monthly . Repayments as the Monthly Repayments parameter, Applicant data . Monthly . Expenses as the Monthly Expenses parameter, the output of the Pre-bureau risk category decision as the Risk Category parameter, and the output of the Required monthly installment decision as the Required Monthly Installment parameter.

- The **Post-bureau affordability** decision logic (Figure 73) invokes the Affordability calculation business knowledge model, passing Applicant data . Monthly . Income as the Monthly Income parameter, Applicant data . Monthly . Repayments as the Monthly Repayments parameter, Applicant data . Monthly . Expenses as the Monthly Expenses parameter, the output of the Post-bureau risk category decision as the Risk Category parameter, and the output of the Required monthly installment decision as the Required Monthly Installment parameter.

- The **Affordability calculation** decision logic (Figure 74) defines a boxed context deriving Affordability from Monthly Income, Monthly Repayments, Monthly Expenses and Required Monthly Installment. One step in this calculation derives Credit contingency factor by invoking the Credit contingency factor table business knowledge model, passing the output of the Risk category decision as the Risk Category parameter.

- The **Credit contingency factor table** decision logic (Figure 75) defines a complete, unique-hit decision table deriving Credit contingency factor from Risk Category.

- The **Required monthly installment** decision logic (Figure 76) invokes the Installment calculation business knowledge model, passing Requested product . ProductType as the Product Type parameter, Requested product . Rate as the Rate parameter, Requested product . Term as the Term parameter, and Requested product . Amount as the Amount parameter.

- The **Installment calculation** decision logic (Figure 77) defines a boxed context deriving monthly installment from Product Type, Rate, Term and Amount. One step in this calculation invokes the external function PMT.

<table>
<thead>
<tr>
<th>Strategy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strategy table</td>
</tr>
<tr>
<td>Bureau Call Type</td>
</tr>
<tr>
<td>Eligibility</td>
</tr>
</tbody>
</table>

Figure 58: Strategy decision logic
### Strategy table

<table>
<thead>
<tr>
<th>UC</th>
<th>Eligibility</th>
<th>Bureau Call Type</th>
<th>Strategy</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>INELIGIBLE</td>
<td>-</td>
<td>DECLINE</td>
</tr>
<tr>
<td>2</td>
<td>ELIGIBLE</td>
<td>FULL, MINI</td>
<td>BUREAU</td>
</tr>
<tr>
<td>3</td>
<td>ELIGIBLE</td>
<td>NONE</td>
<td>THROUGH</td>
</tr>
</tbody>
</table>

Figure 59: Strategy table decision logic

### Bureau call type

<table>
<thead>
<tr>
<th>Bureau call type table</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pre-Bureau Risk Category</td>
</tr>
</tbody>
</table>

Figure 60: Bureau Call Type decision logic

### Bureau call type table

<table>
<thead>
<tr>
<th>UC</th>
<th>Pre-Bureau Risk Category</th>
<th>Bureau Call Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>HIGH, MEDIUM</td>
<td>FULL</td>
</tr>
<tr>
<td>2</td>
<td>LOW</td>
<td>MINI</td>
</tr>
<tr>
<td>3</td>
<td>VERY LOW, DECLINE</td>
<td>NONE</td>
</tr>
</tbody>
</table>

Figure 61: Bureau call type table decision logic

### Eligibility

<table>
<thead>
<tr>
<th>Eligibility rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>Age</td>
</tr>
<tr>
<td>Pre-Bureau Risk Category</td>
</tr>
<tr>
<td>Pre-Bureau Affordability</td>
</tr>
</tbody>
</table>

Figure 62: Eligibility decision logic
Eligibility rules

<table>
<thead>
<tr>
<th>PC</th>
<th>Pre-Bureau Risk Category</th>
<th>Pre-Bureau Affordability</th>
<th>Age</th>
<th>Eligibility</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>DECLINE</td>
<td>-</td>
<td>-</td>
<td>INELIGIBLE</td>
</tr>
<tr>
<td>2</td>
<td>-</td>
<td>false</td>
<td>-</td>
<td>INELIGIBLE</td>
</tr>
<tr>
<td>3</td>
<td>-</td>
<td>-</td>
<td>&lt; 18</td>
<td>INELIGIBLE</td>
</tr>
<tr>
<td>4</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>ELIGIBLE</td>
</tr>
</tbody>
</table>

Figure 63: Eligibility rules decision logic

Pre-bureau risk category

Pre-bureau risk category table

<table>
<thead>
<tr>
<th>Existing Customer</th>
<th>Applicant data</th>
<th>ExistingCustomer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application Risk Score</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 64: Pre-Bureau Risk Category decision logic

Pre-bureau risk category table

<table>
<thead>
<tr>
<th>UC</th>
<th>Existing Customer</th>
<th>Application Risk Score</th>
<th>Pre-Bureau Risk Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>true</td>
<td>&lt; 100</td>
<td>HIGH</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>[100..120]</td>
<td>MEDIUM</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>[120..130]</td>
<td>LOW</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>&gt; 130</td>
<td>VERY LOW</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>&lt; 80</td>
<td>DECLINE</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>[80..90]</td>
<td>HIGH</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>[90..110]</td>
<td>MEDIUM</td>
</tr>
<tr>
<td>8</td>
<td>false</td>
<td>&gt; 110</td>
<td>LOW</td>
</tr>
</tbody>
</table>

Figure 65: Pre-bureau risk category table decision logic
**Application risk score**

<table>
<thead>
<tr>
<th>Age</th>
<th>Applicant data . Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>Marital Status</td>
<td>Applicant data . MaritalStatus</td>
</tr>
<tr>
<td>Employment Status</td>
<td>Applicant data . EmploymentStatus</td>
</tr>
</tbody>
</table>

**Figure 66: Application Risk Score decision logic**

### Application risk score model

<table>
<thead>
<tr>
<th>NC</th>
<th>Age</th>
<th>Marital Status</th>
<th>Employment Status</th>
<th>Partial score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>[18..120]</td>
<td>S, M</td>
<td>UNEMPLOYED, EMPLOYED, SELF-EMPLOYED, STUDENT</td>
<td>32</td>
</tr>
<tr>
<td>2</td>
<td>[18..21]</td>
<td>-</td>
<td>-</td>
<td>32</td>
</tr>
<tr>
<td>3</td>
<td>[22..25]</td>
<td>-</td>
<td>-</td>
<td>35</td>
</tr>
<tr>
<td>4</td>
<td>[26..35]</td>
<td>-</td>
<td>-</td>
<td>40</td>
</tr>
<tr>
<td>5</td>
<td>[36..49]</td>
<td>-</td>
<td>-</td>
<td>43</td>
</tr>
<tr>
<td>6</td>
<td>&gt;=50</td>
<td>-</td>
<td>-</td>
<td>48</td>
</tr>
<tr>
<td>7</td>
<td>-</td>
<td>S</td>
<td>-</td>
<td>25</td>
</tr>
<tr>
<td>8</td>
<td>-</td>
<td>M</td>
<td>-</td>
<td>45</td>
</tr>
<tr>
<td>9</td>
<td>UNEMPLOYED</td>
<td>STUDENT</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>EMPLOYED</td>
<td></td>
<td>45</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>SELF-EMPLOYED</td>
<td></td>
<td>36</td>
<td></td>
</tr>
</tbody>
</table>

Aggregation = sum

**Figure 67: Application risk score model decision logic**
Routing
Routing rules

<table>
<thead>
<tr>
<th>Bankrupt</th>
<th>Bureau data . Bankrupt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Credit Score</td>
<td>Bureau data . CreditScore</td>
</tr>
<tr>
<td>Post-Bureau Risk Category</td>
<td>Post-bureau risk category</td>
</tr>
<tr>
<td>Post-Bureau Affordability</td>
<td>Post-bureau affordability</td>
</tr>
</tbody>
</table>

Figure 68: Routing decision logic

Routing rules

<table>
<thead>
<tr>
<th>PC</th>
<th>Post-Bureau Risk Category</th>
<th>Post-Bureau Affordability</th>
<th>Bankrupt</th>
<th>Credit Score</th>
<th>Routing</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-</td>
<td>false</td>
<td>-</td>
<td>-</td>
<td>DECLINE</td>
</tr>
<tr>
<td>2</td>
<td>-</td>
<td>true</td>
<td>-</td>
<td>-</td>
<td>DECLINE</td>
</tr>
<tr>
<td>3</td>
<td>HIGH</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>REFER</td>
</tr>
<tr>
<td>4</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>&lt; 580</td>
<td>REFER</td>
</tr>
<tr>
<td>5</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>ACCEPT</td>
</tr>
</tbody>
</table>

Figure 69: Routing rules decision logic

Post-bureau risk category

Post-bureau risk category table

<table>
<thead>
<tr>
<th>Existing Customer</th>
<th>Applicant data . ExistingCustomer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Credit Score</td>
<td>Bureau data . CreditScore</td>
</tr>
<tr>
<td>Application Risk Score</td>
<td>Application risk score</td>
</tr>
</tbody>
</table>

Figure 70: Post-Bureau Risk Category decision logic
### Post-bureau risk category table

<table>
<thead>
<tr>
<th>UC</th>
<th>Existing Customer</th>
<th>Application Risk Score</th>
<th>Credit Score</th>
<th>Post-Bureau Risk Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>&lt; 120</td>
<td>&lt; 590</td>
<td>HIGH</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>[590..610]</td>
<td>[590..610]</td>
<td>MEDIUM</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>&gt; 610</td>
<td>&gt; 610</td>
<td>LOW</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>&lt; 600</td>
<td>&lt; 600</td>
<td>HIGH</td>
</tr>
<tr>
<td>5</td>
<td>true</td>
<td>[120..130]</td>
<td>[600..625]</td>
<td>MEDIUM</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>&gt; 625</td>
<td>&gt; 625</td>
<td>LOW</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>&gt; 130</td>
<td>-</td>
<td>VERY LOW</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>&lt;= 100</td>
<td>&lt; 580</td>
<td>HIGH</td>
</tr>
<tr>
<td>9</td>
<td></td>
<td>[580..600]</td>
<td>[580..600]</td>
<td>MEDIUM</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td>&gt; 600</td>
<td>&gt; 600</td>
<td>LOW</td>
</tr>
<tr>
<td>11</td>
<td></td>
<td>&lt; 590</td>
<td>&lt; 590</td>
<td>HIGH</td>
</tr>
<tr>
<td>12</td>
<td></td>
<td>[590..615]</td>
<td>[590..615]</td>
<td>MEDIUM</td>
</tr>
<tr>
<td>13</td>
<td>false</td>
<td>&gt; 615</td>
<td>&gt; 615</td>
<td>LOW</td>
</tr>
</tbody>
</table>

**Figure 71:** Post-bureau risk category table decision logic

### Pre-bureau affordability

**Affordability calculation**

<table>
<thead>
<tr>
<th>Monthly Income</th>
<th>Applicant data . Monthly . Income</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monthly Repayments</td>
<td>Applicant data . Monthly . Repayments</td>
</tr>
<tr>
<td>Monthly Expenses</td>
<td>Applicant data . Monthly . Expenses</td>
</tr>
<tr>
<td>Risk Category</td>
<td>Pre-bureau risk category</td>
</tr>
<tr>
<td>Required Monthly Installment</td>
<td>Required monthly installment</td>
</tr>
</tbody>
</table>

**Figure 72:** Pre-Bureau Affordability decision logic
### Post-bureau affordability

**Affordability calculation**

<table>
<thead>
<tr>
<th>Monthly Income</th>
<th>Applicant data . Monthly . Income</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monthly Repayments</td>
<td>Applicant data . Monthly . Repayments</td>
</tr>
<tr>
<td>Monthly Expenses</td>
<td>Applicant data . Monthly . Expenses</td>
</tr>
<tr>
<td>Risk Category</td>
<td>Post-bureau risk category</td>
</tr>
<tr>
<td>Required Monthly Installment</td>
<td>Required monthly installment</td>
</tr>
</tbody>
</table>

**Figure 73: Post-Bureau Affordability decision logic**

### Affordability calculation

<table>
<thead>
<tr>
<th>Disposable Income</th>
<th>Monthly Income – (Monthly Repayments + Monthly Expenses)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Credit Contingency Factor</td>
<td>Credit contingency factor table</td>
</tr>
<tr>
<td></td>
<td>Risk Category</td>
</tr>
<tr>
<td>Affordability</td>
<td>if Disposable Income * Credit Contingency Factor &gt; Required Monthly Installment then true else false</td>
</tr>
</tbody>
</table>

**Figure 74: Affordability calculation decision logic**
<table>
<thead>
<tr>
<th>UC</th>
<th>Risk Category</th>
<th>Credit Contingency Factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>HIGH, DECLINE</td>
<td>0.6</td>
</tr>
<tr>
<td>2</td>
<td>MEDIUM</td>
<td>0.7</td>
</tr>
<tr>
<td>3</td>
<td>LOW, VERY LOW</td>
<td>0.8</td>
</tr>
</tbody>
</table>

Figure 75: Credit contingency factor table decision logic

### Required monthly installment

<table>
<thead>
<tr>
<th>Installment calculation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product Type</td>
</tr>
<tr>
<td>Rate</td>
</tr>
<tr>
<td>Term</td>
</tr>
<tr>
<td>Amount</td>
</tr>
</tbody>
</table>

Figure 76: Required Monthly Installment decision logic

### Installment calculation

<table>
<thead>
<tr>
<th>Monthly Fee</th>
<th>if Product Type = “STANDARD LOAN”</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>then 20.00</td>
</tr>
<tr>
<td></td>
<td>else if Product Type = “SPECIAL LOAN”</td>
</tr>
<tr>
<td></td>
<td>then 25.00</td>
</tr>
<tr>
<td></td>
<td>else null</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Monthly Repayment</th>
<th>PMT(Rate, Term, Amount)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monthly Repayment</td>
<td>Monthly Repayment + Monthly Fee</td>
</tr>
</tbody>
</table>

Figure 77: Installment calculation decision logic

Finally, the input data elements may be modeled with some concrete sample data. This allows the decision model to be thoroughly validated and even executed. Figure 78, Figure 79 and Figure 80 show boxed contexts defining instance values for Applicant data, Requested product and Bureau data.
### Applicant data

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Age</td>
<td>51</td>
</tr>
<tr>
<td>MaritalStatus</td>
<td>M</td>
</tr>
<tr>
<td>EmploymentStatus</td>
<td>EMPLOYED</td>
</tr>
<tr>
<td>ExistingCustomer</td>
<td>false</td>
</tr>
<tr>
<td>Monthly Income</td>
<td>10,000.00</td>
</tr>
<tr>
<td>Repayments</td>
<td>2,500.00</td>
</tr>
<tr>
<td>Expenses</td>
<td>3,000.00</td>
</tr>
</tbody>
</table>

Figure 78: Applicant Data input data sample

### Requested product

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProductType</td>
<td>STANDARD LOAN</td>
</tr>
<tr>
<td>Rate</td>
<td>0.08</td>
</tr>
<tr>
<td>Term</td>
<td>36</td>
</tr>
<tr>
<td>Amount</td>
<td>100,000.00</td>
</tr>
</tbody>
</table>

Figure 79: Requested Product input data sample

### Bureau data

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bankrupt</td>
<td>false</td>
</tr>
<tr>
<td>CreditScore</td>
<td>600</td>
</tr>
</tbody>
</table>

Figure 80: Bureau Data input data sample

### 11 Exchange formats

#### 11.1 Interchanging Incomplete Models

It is common for DMN models to be interchanged before they are complete. This occurs frequently when doing iterative modeling, where one user (such as a knowledge source expert or business user) first defines a high-level model and then passes it on to another person to complete or refine the model. Such "incomplete" models are ones in which not all of the mandatory model attributes have been filled in yet or the cardinality of the lower bound of attributes and associations has not been satisfied.

XMI allows for the interchange of such incomplete models. In DMN, we extend this capability to interchange of XML files based on the DMN XML-Schema. In such XML files, implementers are expected to support this interchange by:
• Disregarding missing attributes that are marked as "required" in the DMN XML-Schema.
• Reducing the lower bound of elements with "minOccurs" greater than 0.

11.2 Machine Readable Files
All DMN 1.0 machine-readable files, including XSD and XMI files, can be found in OMG Document bmi/2013-08-05, which is a flat zip file.

• For the DMN XMI Model, the main file is DMN10.xmi.
• For the DMN XSD Interchange Part 1 (the interchange definition for Conformance Levels 1 and 2), the main file is DMN10.xsd.
• For the DMN XSD Interchange Part 2 (the interchange definition for Conformance Level 3), the main file is DMN10Level3.xsd.

11.3 XSD

11.3.1 Document Structure
A domain-specific set of model elements is interchanged in one or more DMN files. The root element of each file MUST be <dmn:Definitions>. The set of files MUST be self-contained, i.e. all definitions that are used in a file MUST be imported directly or indirectly using the <dmn:Import> element.

Each file MUST declare a “namespace” that MAY differ between multiple files of one model.

DMN files MAY import non-DMN files (such as XSDs and PMMLs) if the contained elements use external definitions.

11.3.2 References within the DMN XSD
All the DMN elements that may need be referenced contain IDs and within the BPMN XSD, references to elements are expressed via these IDs. The XSD IDREF type is the traditional mechanism for referencing by IDs, however it can only reference an element within the same file. The DMN XSD supports referencing by ID, across files, by utilizing QNames.

A QName consists of two parts: an optional namespace prefix and a local part. When used to reference a DMN element, the local part is expected to be the ID of the element, and the namespace is expected to be the namespace of the containing Definitions element.

For example, consider the following Decision:

<Decision name="Pre-Bureau Risk Category" id="prebureauriskDec01">…</Decision>

When this Decision is referenced, e.g. by an InformationRequirement in a Decision from another file, the reference would take the following form:

<requiredDecision>decision_ns:prebureauriskDec01</requiredDecision>
where “decision_ns” is the namespace prefix associated upon import with the namespace in which the “Pre-Bureau Risk Category” Decision is defined, and “prebureauriskDec01” is the value of the id attribute for the Decision.

Notice that the reference can also take the following form:

<requiredDecision> prebureauriskDec01</requiredDecision>

where the prefix is omitted, if the required decision (in this case Pre-Bureau Risk Category) is defined in the default namespace (e.g. because it is defined in the same instance of Definitions – that is, in the same file – as the requiring decision).

The DMN XSD utilizes IDREFs wherever possible and resorts to QName only when references can span files. In both situations however, the reference is still based on IDs.
ANNEXES

All the Annexes are informative.

Annex A and Annex B discuss issues around the application of DMN in combination with BPMN, including the use of DMN to define the functionality of decision services to be called from tasks defined in BPMN. These sections are intended to provide some direction to practitioners but are non-normative.

Annex C provides cross-reference tables relating the requirements as stated in the RFP to the corresponding responses in this submission.

Annex D provides a non-normative glossary to aid comprehension of the specification.

Annex A. Relation to BPMN

(Informative)

1. Goals of BPMN and DMN

The OMG Business Process Model and Notation standard provides a standard notation for describing business processes as orchestrations of tasks. The success of BPMN has provided a major motivation for DMN, and business decisions described using DMN are expected to be commonly deployed in business processes described using BPMN.

All statements pertaining to BPMN below are from the OMG document reference 11-01-03 unless otherwise stated.

BPMN’s goals are stated in the specification and provide easy comparisons to DMN:

- Goal 1: “The primary goal of BPMN is to provide a notation that is readily understandable by all business users, from the business analysts that create the initial drafts of the processes, to the technical developers responsible for implementing the technology that will perform those processes, and finally, to the business people who will manage and monitor those processes. Thus, BPMN creates a standardized bridge for the gap between the business process design and process implementation.”. DMN users will also be business analysts (designing decisions) and then business users (populating decision models such as decision tables). Technical developers may be responsible for mapping business terms to appropriate data technologies. Therefore DMN can also be said to bridge the decision design by a business analyst, and the decision implementation, typically using some decision execution technology.

- Goal 2: “... To ensure that XML languages designed for the execution of business processes, such as WSBPEL (Web Services Business Process Execution Language), can be visualized with a business-oriented notation.” It is not a stated goal of DMN to be able to visualize other XML languages (such as W3C RIF or OMG PRR); indeed it is expected that DMN would provide the MDA specification layer for such languages. It does not preclude however the use of DMN (such as decision tables) to represent executable forms (such as production rules).

- Goal 3: “The intent of BPMN is to standardize a business process model and notation in the face of many different modeling notations and viewpoints. In doing so, BPMN will provide a
simple means of communicating process information to other business users, process implementers, customers, and suppliers.” Similarly, the intent of DMN is to standardize the decision model and notation across the many different implementations of broadly semantically similar models. In so doing, DMN will also facilitate the communication of decision information across business communities and tools.

2. **BPMN Tasks and DMN Decisions**

Most BPMN diagrams contain some tasks which involve decision-making which can be modeled in DMN. These tasks take input data acquired or generated earlier in the process, and produce decision outputs which are used later in the process. Decision outputs may be used in two principal ways:

- They may be consumed in another process task
- They may influence the choice of sequence flows out of a gateway.

In the latter case, decisions are used to determine which subprocesses or tasks are to be executed (in the process sense). As such, DMN complements BPMN as decision modeling complements process modeling (in the sense of defining orchestrations or work tasks).

For example, Figure 81 shows an example\(^1\) of a BPMN-defined process.

![Figure 81: Decision-making in BPMN](image)

Analyzing this we see:

---

\(^1\) Shipment Process in a Hardware Retailer example, Ch5.1, BPMN 2.0 By Example, June 2010, OMG reference 10-06-02
• a task whose title starts with “Decide…” which makes a decision on (whether to use) normal post or special shipment, and which precedes an exclusive gateway using that decision result
• a task whose title starts with “Check…” which makes a decision on whether extra insurance is necessary, which precedes an inclusive gateway for which an additional process path may be executed based on the decision result
• a task whose title starts with “Assign…” which implies a decision to select a carrier based on some selection criteria. The previous task is effectively collecting data for this decision. In an automated system this would probably be a subprocess embedding a decision and some other activities (such as “prepare paperwork”).

From this example we can see that even a simple business process in BPMN may have several decision-making tasks.

3. **Types of BPMN Tasks relevant to DMN**

BPMN defines\(^2\) different types of tasks that can be considered for decision-making roles. The relevant tasks are as shown in Table 34:

<table>
<thead>
<tr>
<th>Table 34: BPMN tasks relevant to DMN</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Task type(s)</strong></td>
</tr>
<tr>
<td>------------------</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
</tbody>
</table>

---

\(^2\) See ch 10.2.3 in the BPMN Specification.
The Business Rule Task was defined in BPMN 2 as a placeholder for (business-rule-driven) decisions, and is the natural placeholder for a decision task. *Note that business rules (as defined in OMG SBVR) can constrain any type of process activity, not just business decisions.*

Decision tasks may today be encoded using business process script languages.

A future version of BPMN may choose to clarify and extend the definitions of task to better match decision modeling requirements and DMN – to wit, to define a BPMN Decision Task as some task used to make a decision modeled with DMN. In the meantime, the Business Rule Task is the most natural way to express this functionality. However, as noted in sections 4.3.2 and 5.3.6, a Decision in DMN can be associated with any Task, allowing for flexibility in implementation.

### 4. Process gateways and Decisions

Process gateways can be considered of 2 types:

1. A gateway that determines a process route or routes based on existing data
2. A gateway that determines a process route or routes based on the outcome of one or more decisions that are determined by some previous task within the process.

In the latter case, a Decision Task (task used to make a decision using DMN) may need an extended notation to clarify the relationship of the decision task to the gateway(s) that use it.

### 5. Linking BPMN and DMN Models

DMN offers two approaches to linking business process models in BPMN with decision models in DMN; one normative and the other non-normative:

a) **Associating Decisions with Tasks and Processes**

As described in section 5.3.6, in DMN 1.0, the process context for an instance of Decision is defined by its association with any number of usingProcesses, which are instances of Process as defined in OMG BPMN2, and any number of usingTasks, which are instances of Task as defined in OMG BPMN2. Each decision may therefore be associated with one or more business processes (to indicate that the decision is taken during those processes), and/or with one or more specific tasks (to indicate that the tasks involve making the decision). An implementation MUST allow these associations to be defined for each decision.

An implementation MAY perform validation over the two (BPMN and DMN) models, to check, for example, that:
- A Decision is not associated with Tasks that are part of Processes not also associated with the Decision
- A Decision is not associated with Tasks that are not part of any Process associated with the Decision

During development it may be appropriate to associate a Decision only with a Process, but inconsistency between Task and Process associations is not allowed.

Note that this approach allows the relationships between business process models and decision models to be defined and validated, but does not of itself permit the decisions modeled in DMN to be executed automatically by processes modeled in BPMN.

b) Decision Services

One approach to decision automation is described non-normatively in Annex B: the encapsulation of DMN Decisions in a “decision service” called from a BPMN Task (e.g. a Service Task or Business Rule Task, as discussed in Annex A.3 above). The usingProcesses and usingTasks properties allow definition and validation of associations between BPMN and DMN; the definition of decision services then provides a detailed specification of the required interface.

Annex B. Decision services

(Informative)

One important use of DMN will be to define decision-making logic to be automated using “decision services”. A decision service encapsulates a number of decisions and exposes them as a service, which might be consumed (for example) by a task in a BPMN process model. When the service is called, with the necessary input data, it returns the outputs of the encapsulated decisions. Any decision service encapsulating a DMN decision model will be stateless and have no side effects. It might be implemented, for example, as a web service. DMN does not specify how such services should be implemented; this section is to give an indication of the principles which might be followed.

We start with the assumption that the client requires a certain set of decisions to be made, and that the service is created to meet that requirement. The sole function of the decision service is to return the results of evaluating that set of decisions (the “minimal output set”). This requires that the service encapsulate not just the minimal output set but also any decisions in the DRG directly or indirectly required by the minimal output set (the “encapsulation set”). The encapsulation set is the transitive closure of the required decision relation on the minimal output set.

The interface to the decision service will consist of:
- Input: a list of contexts, providing instances of all the Input Data required by the encapsulated decisions
- Output: a context, providing (at least) the results of evaluating all the decisions in the minimal output set, using the provided instance data.

When the service is called, providing the input, it returns the output.

In its simplest form a decision service would always evaluate all decisions in the encapsulation set and return all the results.
For computational efficiency various improvements to this basic interpretation can be imagined, e.g.

- An optional input parameter specifying a list of “requested decisions” (a subset of the encapsulation set). Only the results of the requested decisions would be returned in the output context.

- An optional input parameter specifying a list of “known decisions” (a subset of the encapsulation set), with their results. The decision service would not evaluate these decisions, but would use the provided input values directly.

All such implementation details are left to the software provider.

A decision service is “complete” if it contains decision logic for evaluating all the encapsulated decisions on all possible input data values. A request to the service is “valid” if instances are provided for all Input Data required by those decisions which need to be evaluated, i.e. (in the simple case) all the encapsulated decisions, or (assuming the optional parameters above) any requested decisions which are not already known.

**Annex C. Responses to RFP Requirements**

(Informative)

The following tables provide a cross-reference between the requirements as stated in the Request for Proposals and the corresponding responses provided by this submission.

1. **Mandatory requirements**
### Table 35: Mandatory requirements

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Resolution</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.5.1 A Typology of Decision Models</td>
<td>No typology provided.</td>
</tr>
<tr>
<td>As there is no standard definition or typology of decision models, the DMN standard shall provide such a typology. The typology shall explain how different types of models differ in terms of constructs that are defined by the proposed metamodel. The Decision Model Typology shall cover the general concepts of decision models and particular model types (such as Decision Table, predictive models, optimization models etc). The proposed typology shall cover as many decision model types as possible, but it may not be equally detailed with respect to all the covered model types (e.g. covering a set of model types within a single super-type, without introducing the properties by which they differ individually, and/or without differentiating them individually by properties introduced and used to define other types). The typology shall, at least, identify as individual types the model types for which the standard specifies a notation. The typology shall serve to motivate the proposed standard, to explain limitations and/or restrictions, and to help map the road to future extensions, if it covers decision model types beyond those covered by the standard. It would be expected that this typology would be further detailed and/or extended over time as future versions of DMN support new models.</td>
<td>In the final submission, decision tables and predictive models are components of decision models, rather than types of decision models. The submission team felt that this was the appropriate approach for the first version of this standard. This approach leads to a practical standard that can immediately be used by industry practitioners.</td>
</tr>
</tbody>
</table>
6.5.2 Decision Metamodel

DMN shall provide a metamodel to support decision models, and specify the associated semantics. The metamodel shall have the following characteristics:

1. It shall specify the structure and features of decision models, including, at least, decision variables, decision inputs, governing rules and preliminary or prerequisite decisions, and the properties of a decision model that determine its type according to the proposed typology and/or that affect its actionability and automation.

2. It shall reference, and avoid replication of, any existing OMG metamodels. Examples of such metamodel references could include a business rule metamodel for the specification of the business rules that decisions relate to, and a business process metamodel for any processes connecting a decision and its prerequisite decisions and derivations.

   Note that if such metamodel references can be achieved only through a MOF implementation of DMN, then a MOF implementation of DMN would become mandatory instead of optional.

3. It shall support traceability from a decision model to the business rules that relate to it.

4. It shall support the concept of layered, or hierarchies of, decisions and their appropriate scopes, where multiple instances of models may be combined together. For example, a Decision Table may refer to or delegate detailed logic to other, subsidiary Decision Tables.

5. It shall support iterative modeling.

6. It shall support a subset of the Decision Model Typology and be capable of extension for new types of decision model in future.

Included in the submission. In particular see sections 5.3, 6.3, 7.3 and 9.5.

DMN uses the DRD to specify the structure of decision models, and FEEL to specify detailed decision logic.

DMN references BMM and BPMN. The submission team included members skilled on BPMN, CMMN and BMM, and care was taken to avoid overlap and duplication.

Traceability of business rules is provided in DMN by requirements in the DRD (especially knowledge requirements linking business models to decisions), and value expressions providing the decision logic content of DRD elements (especially business knowledge models): see section 6.

Hierarchies of decisions and decision logic may be defined in a DRD. Business knowledge models may require other business knowledge models, and may be defined as decision tables.

DMN provides two levels of modeling (decision requirements and decision logic) and allows incomplete models at either level.
6.5.3 Decision Model Notation

DMN shall provide an intuitive diagrammatic representation of a decision model for the business user. In particular, it must define a business level, computation independent, notation for the business rules that govern decisions, especially complex ones (e.g. as represented in decision tables).

The notation shall support the following:

1. The representation of decision models at various degrees of completion.

2. The layered or hierarchical representation of a decision model, allowing navigation of dependencies and drill-down exploration of prerequisite decisions.

3. The visual verification, where possible, of the decision-specific properties of a decision model, and, at least, the visual differentiation of all the model types that are individually identified in the proposed Decision Model Typology.

4. At least one visual notation for a Decision Table as this is the most common and understood decision notation in use today.

6.5.4 Decision Model Interchange

DMN shall provide, or reference an existing, interchange format that allows the exchange of decision models between tools. This will require a normative concrete syntax for interchange.

Interchange of models can be achieved through XMI. However the specific format for interchange is required only to be an XML format; DMN model interchange could also be achieved through an extension to W3C RIF, for example.

2. Optional requirements
### Table 36: Optional requirements

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Resolution</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.6.1 Notation Requirement Options</td>
<td>See below</td>
</tr>
<tr>
<td>6.6.1.1 Extension to BPMN</td>
<td><strong>Notations that are specific to DMN may be visually compatible with the BPMN graphical notation standard, in order to facilitate easy cross-model development.</strong>&lt;br&gt;&lt;br&gt;In addition, DMN may propose to extend BPMN with a Decision Task to reference a Decision Model in DMN.</td>
</tr>
<tr>
<td>6.6.1.2 Inclusion of Visual Notation in Decision Model interchange</td>
<td><strong>The interchange format provided or referenced by DMN may include specifics about any visual notation user options.</strong></td>
</tr>
<tr>
<td>6.6.1.3 Usage of Diagram Definition standard</td>
<td><strong>Although no specific Diagram Definition standard for describing graphical elements is stipulated for DMN, it may re-use the OMG standards in this area, such as the OMG Diagram Definition standard.</strong></td>
</tr>
<tr>
<td>6.6.2 Vocabulary and Rule Requirement Options</td>
<td>See below</td>
</tr>
<tr>
<td>6.6.2.1 Usage of SBVR terms</td>
<td><strong>DMN may specify SBVR for the decision model vocabulary, and decision model constraints or conditions could be defined as SBVR business rules.</strong></td>
</tr>
<tr>
<td>6.6.2.2 Usage of UML and other information model terms</td>
<td><strong>DMN may utilize UML, IMM or CWM for decision model terms.</strong></td>
</tr>
<tr>
<td>6.6.2.3 Usage of UML OCL constraints to represent decision conditions</td>
<td><strong>DMN may utilize UML OCL for decision model constraints or conditions.</strong></td>
</tr>
</tbody>
</table>
6.6.2.4 Usage of UML ActionLanguage to represent decision conditions and actions
DMN may utilize UML ActionLanguage (from xUML) for decision model constraints, conditions and actions.
Not included.

6.6.2.5 Mapping to PRR
A mapping from some decision model type and the OMG PRR specification (for production rules) may be provided. Though this, DMN could be considered as a graphical notation for a particular usage for PRR and inference rules.
Not included.

6.6.3 Format Requirement Options
6.6.3.1 MOF provision
MOF (EMOF or CMOF) files may be provided for the DMN metamodel as a part of the specification.
Not included. The MagicDraw source files for the metamodel are available on request to the submission team.

3. Issues to be discussed

Table 37: Issues to be discussed

<table>
<thead>
<tr>
<th>Issue to Discussed</th>
<th>Resolution</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMN should include discussions and roadmap options on:</td>
<td>The introduction section of the final submission addresses these topics to the extent they apply to the submission team’s chosen design approach. In due course the OMG may wish to issue an RFP to driver greater alignment and synergy across BPMN, CMMN and DMN. However that is not a topic the submission team felt it should address in the first version of DMN.</td>
</tr>
<tr>
<td>(a) Extensibility to additional decision model notations</td>
<td></td>
</tr>
<tr>
<td>(b) Dependencies on and references to BPMN, and maintaining compatibility across differing versions of BPMN and DMN</td>
<td></td>
</tr>
<tr>
<td>(c) How DMN can provide models for production rules represented in PRR.</td>
<td></td>
</tr>
<tr>
<td>(d) Whether multiple vocabularies should be enabled within the same decision model.</td>
<td></td>
</tr>
</tbody>
</table>
Annex D. Glossary

(Informative)

A

Aggregation
The production of a single result from multiple hits on a decision table. DMN specifies six aggregation indicators, namely: collect, sum, min, max, count, average. The default is collect.

Any
A hit policy for single hit decision tables with overlapping decision rules: under this policy any match may be used.

Authority Requirement
The dependency of a decision or business knowledge model on a knowledge source which provides a source of authority for the decision logic.

B

Binding
In an invocation, the association of the parameters of the invoked expression with the input variables of the invoking expression, using a binding formula.

Boxed Context
A form of boxed expression showing a collection of n (name, value) pairs with an optional result value.

Boxed Expression
A notation serving to decompose decision logic into small pieces which may be associated graphically with elements of a DRD.

Boxed Function
A form of boxed expression showing the kind, parameters and body of a function.

Boxed Invocation
A form of boxed expression showing the parameter bindings that provide the context for the evaluation of the body of a business knowledge model.

Boxed List
A form of boxed expression showing a list of n items.

Boxed Literal Expression
A form of boxed expression showing a literal expression.

Business Context Element
An element representing the business context of a decision: either an organisational unit or a performance indicator.

Business Knowledge Model
Some decision logic (e.g. a decision table) encapsulated as a reusable function, which may be invoked by decisions or by other business knowledge models.

C

Clause
In a decision table, a clause specifies a subject, which is defined by an input expression or an output domain, and the finite set of the sub-domains of the subject’s domain that are
relevant for the piece of **decision logic** that is described by the decision table.

**Completeness Indicator**
Indicates whether a **decision table** produces a result for every possible case. If so (the default) the indicator “C” is optional. If not, the indicator should read “I”.

**Context**
In **FEEL**, a map of key-value pairs called **context entries**.

**Context Entry**
One key-value pair in a **context**.

**Crosstab Table**
An **orientation** for **decision tables** in which two **input expressions** form the two dimensions of the table, and the **output entries** form a two-dimensional grid.

**D**

**Decision**
The act of determining an **output value** from a number of **input values**, using **decision logic** defining how the output is determined from the inputs.

**Decision Logic**
The logic used to make decisions, defined in DMN as the **value expressions** of **decisions** and **business knowledge models** and represented visually as **boxed expressions**.

**Decision Logic Level**
The detailed level of modeling in DMN, consisting of the **value expressions** associated with **decisions** and **business knowledge models**.

**Decision Model**
A formal model of an area of decision-making, expressed in DMN as **decision requirements** and **decision logic**.

**Decision Point**
A point in a business process at which decision-making occurs, modeled in BPMN 2.0 as a business rule task and possibly implemented as a call to a **decision service**.

**Decision Requirements Diagram**
A diagram presenting a (possibly filtered) view of a **DRG**.

**Decision Requirements Graph**
A graph of **DRG elements** (**decisions**, **business knowledge models** and **input data**) connected by **requirements**.

**Decision Requirements Level**
The more abstract level of modeling in DMN, consisting of a **DRG** represented in one or more **DRDs**.

**Decision Rule**
In a **decision table**, a decision rule specifies associates a set of conclusions or results (**output entries**) with a set of conditions (**input entries**).

**Decision Service**
A software component encapsulating a **decision model** and exposing it as a service, which might be consumed (for example) by a task in a BPMN process model.

**Decision Table**
A tabular representation of a set of related input and output expressions, organized into **decision rules** indicating which **output entry** applies to a specific set of **input entries**.
Definitions
A container for all elements of a DMN decision model. The interchange of DMN files will always be through one or more Definitions.

DMN Element
Any element of a DMN decision model: a DRG Element, Business Context Element, Expression, Definitions, Element Collection, Information Item or Item Definition.

DRD
See Decision Requirements Diagram.

DRG
See Decision Requirements Graph.

DRG Element
Any component of a DRG: a decision, business knowledge model, input data or knowledge source.

E
Element Collection
Used to define named groups of DRG elements within a Definitions.

Expression
A literal expression, decision table or invocation used to define part of the decision logic for a decision model in DMN. Returns a single value when interpreted.

F
FEEL
The "Friendly Enough Expression Language" which is the default expression language for DMN.

First
A hit policy for single hit decision tables with overlapping decision rules: under this policy the first match is used, based on the order of the decision rules.

Formal Parameter
A named, typed value used in the invocation of a function to provide an information item for use in the body of the function.

H
Hit
In a decision table, the successful matching of all input expressions of a decision rule, making the conclusion eligible for inclusion in the results.

Hit Policy
Indicates how overlapping decision rules have to be interpreted. A single hit table returns the output of one rule only; a multiple hit table may return the output of multiple rules or an aggregation of the outputs.

Horizontal
An orientation for decision tables in which decision rules are presented as rows; clauses as columns.

I
Information Item
A DMN element used to model either a variable or a parameter at the decision logic level in DMN decision models.
<table>
<thead>
<tr>
<th>Information Requirement</th>
<th>The dependency of a decision on an input data element or another decision to provide a variable used in its decision logic.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input Data</td>
<td>Denotes information used as an input by one or more decisions, whose value is defined outside of the decision model.</td>
</tr>
<tr>
<td>Input Entry</td>
<td>An expression defining a condition cell in a decision table (i.e. the intersection of a decision rule and an input clause).</td>
</tr>
<tr>
<td>Input Expression</td>
<td>An expression defining the item to be compared with the input entries of an input clause in a decision table.</td>
</tr>
<tr>
<td>Input Value</td>
<td>An expression defining a limited range of expected values for an input clause in a decision table.</td>
</tr>
<tr>
<td>Invocation</td>
<td>A mechanism that permits the evaluation of one value expression another, using a number of bindings.</td>
</tr>
<tr>
<td>Item Definition</td>
<td>Used to model the structure and the range of values of input data and the outcome of decisions, using a type language such as FEEL or XML Schema.</td>
</tr>
</tbody>
</table>

**K**

<table>
<thead>
<tr>
<th>Knowledge Requirement</th>
<th>The dependency of a decision or business knowledge model on a business knowledge model which must be invoked in the evaluation of its decision logic.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Knowledge Source</td>
<td>An authority defined for decisions or business knowledge models, e.g. domain experts responsible for defining or maintaining them, or source documents from which business knowledge models are derived, or sets of test cases with which the decisions must be consistent.</td>
</tr>
</tbody>
</table>

**L**

| Literal Expression     | Text that represents decision logic by describing how an output value is derived from its input values, e.g. in plain English or using the default expression language FEEL. |

**M**

| Multiple Hit           | A type of decision table which may return output entries from multiple decision rules. |

**N**

| No Order               | A hit policy for multiple hit decision tables with overlapping decision rules: under this policy all matches will be returned as a list in an arbitrary order. |

**O**

| Organisational Unit    | A business context element representing the unit of an organization which makes or owns a decision. |
Orientation
The style of presentation of a decision table: horizontal (decision rules as rows; clauses as columns), vertical (rules as columns; clauses as rows), or crosstab (rules composed from two input dimensions).

Output Entry
An expression defining a conclusion cell in a decision table (i.e. the intersection of a decision rule and an output clause).

Output Order
A hit policy for multiple hit decision tables with overlapping decision rules: under this policy all matches will be returned as a list in decreasing priority order. Output priorities are specified in an ordered list of values.

Output Value
An expression defining a limited range of domain values for an output clause in a decision table.

P
Performance Indicator
A business context element representing a measure of business performance impacted by a decision.

Priority
A hit policy for single hit decision tables with overlapping decision rules: under this policy the match is used that has the highest output priority. Output priorities are specified in an ordered list of values.

R
Relation
A form of boxed expression showing a vertical list of homogeneous horizontal contexts (with no result cells) with the names appearing just once at the top of the list, like a relational table.

Requirement
The dependency of one DRG element on another: either an information requirement, knowledge requirement or authority requirement.

Requirement Subgraph
The directed graph resulting from the transitive closure of the requirements of a DRG element; i.e. the sub-graph of the DRG representing all the decision-making required by a particular element.

Rule Order
A hit policy for multiple hit decision tables with overlapping decision rules: under this policy all matches will be returned as a list in the order of definition of the decision rules.

S
S-FEEL
A simple subset of FEEL, for decision models that use only simple expressions: in particular, decision models where the decision logic is modeled mostly or only using decision tables.

Single Hit
A type of decision table which may return the output entry of only a single decision rule.
**U**

Unique

A hit policy for single hit decision tables in which no overlap is possible and all decision rules are exclusive. Only a single rule can be matched.

**V**

Variable

Represents a value that is input to a decision, in the description of its decision logic, or a value that is passed as a parameter to a function.

Vertical

An orientation for decision tables in which decision rules are presented as columns; clauses as rows.

**W**

Well-Formed

Used of a DRG element or requirement to indicate that it conforms to constraints on referential integrity, acyclicity etc.