METHOD & Style

DMN as a Decision Modeling Language

Bruce Silver

Principal, Bruce Silver Associates

bruce@brsilver.com

methodandstyle.com

RuleML/DecisionCamp 2016

OMG: Success = Wide Adoption

- DMN attempts to replicate the success of BPMN
  - Wide business user adoption
    - Model defined by diagrams and tables
  - Executable semantics
    - Formal metamodel
    - XML serialization
  - ***Tool-independent format and semantics***
    - Spec managed by Object Management Group (OMG)

Date: May 2016

OBJECT MANAGEMENT GROUP®

Decision Model and Notation (DMN)

V1.1

OMG Document Number: formal/2016-06-01

Standard document URL: http://www.omg.org/spec/DMN/1.1

Normative Machine Consumable File(s):

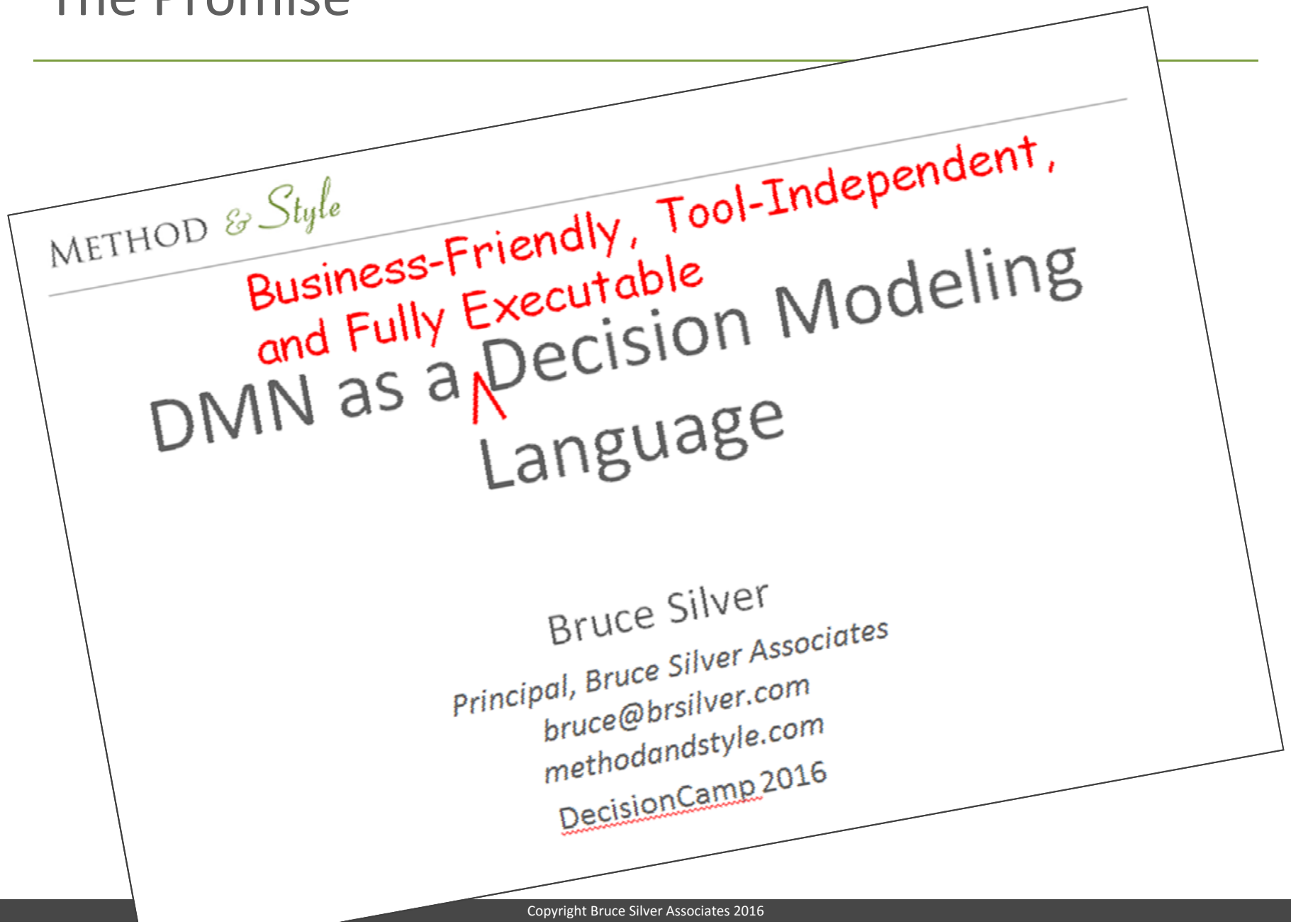
http://www.omg.org/spec/DMN/20151101/dmn.xmi

http://www.omg.org/spec/DMN/20151101/dmn.xsd

Informative Machine Consumable File(s):

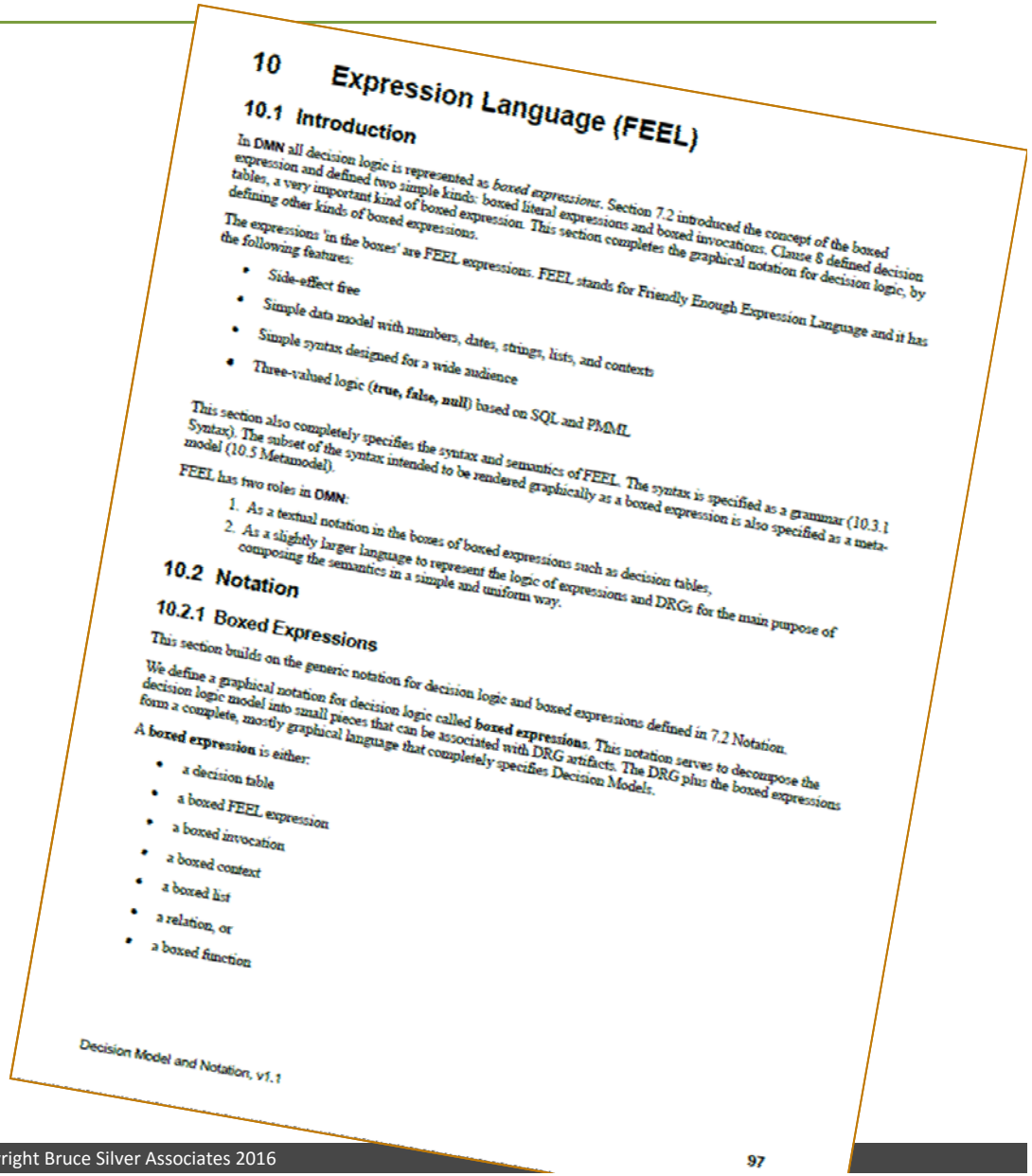
http://www.omg.org/spec/DMN/20151101/ch11example.xml

The Promise



Friendly-Enough Expression Language (FEEL)

- Execution requires a formal expression language
- Tool-independence requires standardization of the language
- It should be “business-friendly”…
- … but also rich enough for real-world decision logic



10 Expression Language (FEEL)

10.1 Introduction

In DMN all decision logic is represented as *boxed expressions*. Section 7.2 introduced the concept of the boxed expression and defined two simple kinds: boxed literal expressions and boxed invocations. Clause 8 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 (*10.3.1 Syntax*). The subset of the syntax intended to be rendered graphically as a boxed expression is also specified as a meta-model (*10.5 Metamodel*).

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.

10.2 Notation

10.2.1 Boxed Expressions

This section builds on the generic notation for decision logic and boxed expressions defined in 7.2 Notation.

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

Decision Model and Notation, v1.1

Outline

1. The Promise
2. How the Language Works
3. Meeting the Implementation Challenge

1. The Promise

What Is DMN's Objective?

1. Business-friendly modeling, tool-independent, fully executable, rich enough to handle all decision logic?

2. Or… simply model-based decision "requirements" (not executable)?

3. Or… business-friendly, tool-independent, fully executable… but limited to simple decision logic?

DMN Conformance Levels

- Level 3: full FEEL support
  - Business-friendly, tool-independent, executable, supports all decision logic

DMN Conformance Levels

- Level 3: full FEEL support
  - Business-friendly, tool-independent, executable, supports all decision logic
- Level 2: Simple FEEL (S-FEEL) only, focus on decision tables
  - Business-friendly, tool-independent, executable, but limited to simple logic

DMN Conformance Levels

- Level 3: full FEEL support
  - Business-friendly, tool-independent, executable, supports all decision logic
- Level 2: Simple FEEL (S-FEEL) only, focus on decision tables
  - Business-friendly, tool-independent, executable, but limited to simple logic
- Level 1: Decision requirements only, not executable
  - Diagrams only, tool-independent, no formal expression language

DMN Conformance Levels

- Level 3: full FEEL support
  - Business-friendly, tool-independent, executable, supports all decision logic
- Level 2: Simple FEEL (S-FEEL) only, focus on decision tables
  - Business-friendly, tool-independent, executable, but limited to simple logic
- Level 1: Decision requirements only, not executable
  - Diagrams only, tool-independent, no formal expression language

- Level 2a: Some other expression language
  - Business-friendly, executable, supports most decision logic, but not tool-independent

An Unlikely Promise

- Three months ago…
  - Only Oracle (inventor of FEEL) seemed to have interest in Level 3
- "Too hard to implement"
- "Model-driven requirements much better than current practice"
- "Leave execution to the programmers"

The Turnaround

- bpmNEXT 2016
  - Oracle
  - Trisotech
  - Sapiens
  - WfMC

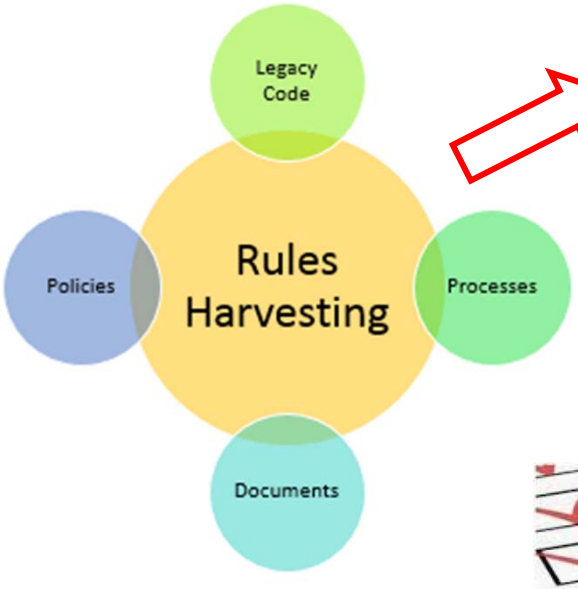
What Is a (DMN) Decision Model?

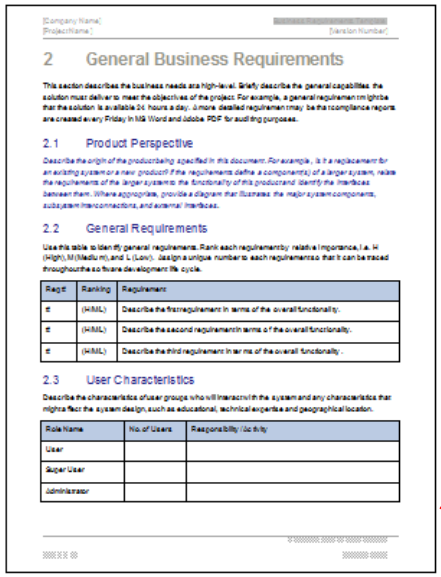
- Rules-based logic that determines an output data value for any allowed set of input data values

- Determination of a conclusion fact value from a given set of condition fact values

Business Rule Management… the Old Way

- Business “harvests” all rules
- …translates them into business requirements documents
- …that programmers translate into executable rule language
- It takes a few cycles to get it right





2 General Business Requirements

This section describes the business needs at a high-level. Briefly describe the general capabilities the solution must deliver to meet the objectives of the project. For example, a general requirement might be that the solution is available 24 hours a day. More detailed requirements may be that compliance reports are created every Friday in MS Word and Adobe PDF for auditing purposes.

2.1 Product Perspective

Describe the origin of the product being specified in this document. For example, is it a replacement for an existing system or a new product? If the requirements define a component(s) of a larger system, relate the requirements of the larger system to the functionality of this product and identify the interfaces between them. Where appropriate, provide a diagram that illustrates the major system components, subsystem interconnections, and external interfaces.

2.2 General Requirements

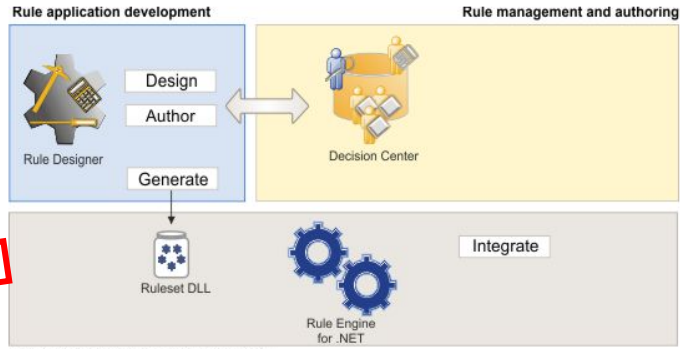
Use this table to identify general requirements. Rank each requirement by relative importance, i.e. H (High), M (Medium), and L (Low). Assign a unique number to each requirement so that it can be traced throughout the software development life cycle.

Req’t	Ranking	Requirement
1	(H/M/L)	Describe the first requirement in terms of the overall functionality.
2	(H/M/L)	Describe the second requirement in terms of the overall functionality.
3	(H/M/L)	Describe the third requirement in terms of the overall functionality.

2.3 User Characteristics

Describe the characteristics of user groups who will interact with the system and any characteristics that might affect the system design, such as educational, technical expertise and geographical location.

Role Name	No. of Users	Responsibility/Activity
User		
Super User		
Administrator		



Problems with the Old Way

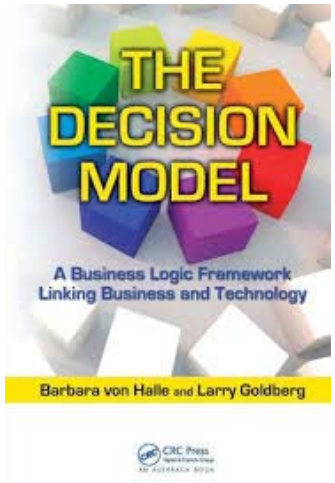
- Inefficiency
  - Most harvested rules are unrelated to the decision of interest
- Incomplete and inconsistent
  - Text-based “decision requirements” not easily validated for completeness, consistency
- Vendor lock-in
  - Proprietary rule languages = higher cost
- Lack of business agility
  - Long and repeated cycles of development and user acceptance testing

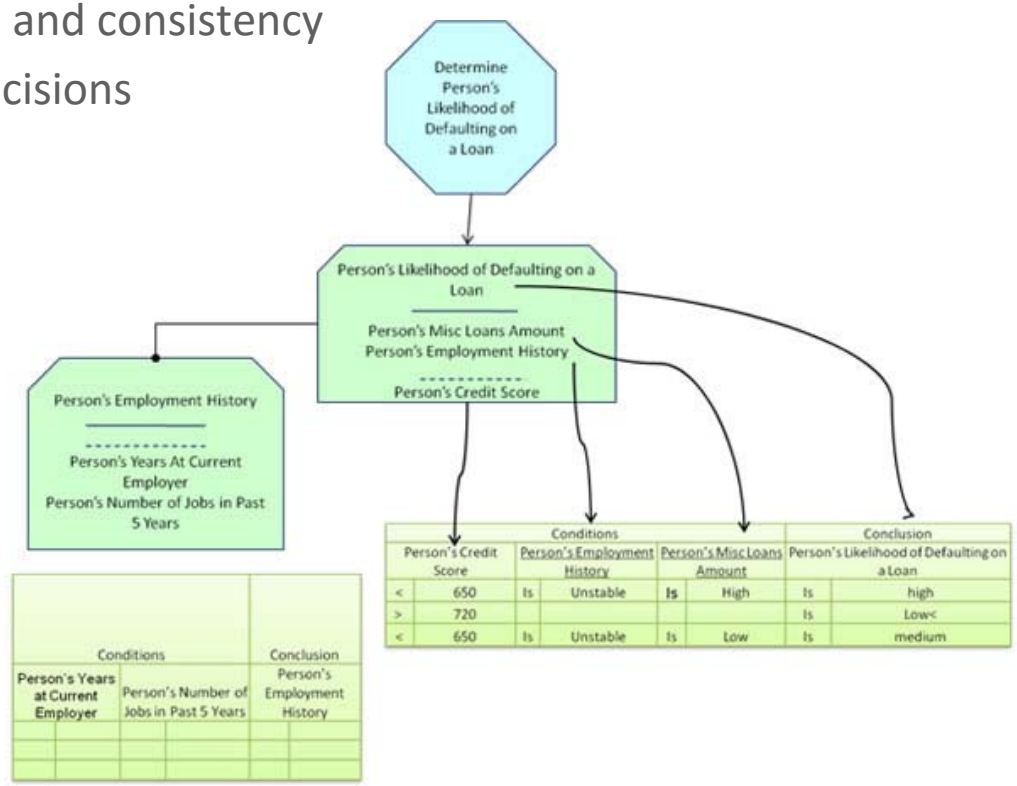
"Decisions First!"

- Start with the needed decisions, not the rules
  - Decisions determine the rules you need

Model-Based Requirements

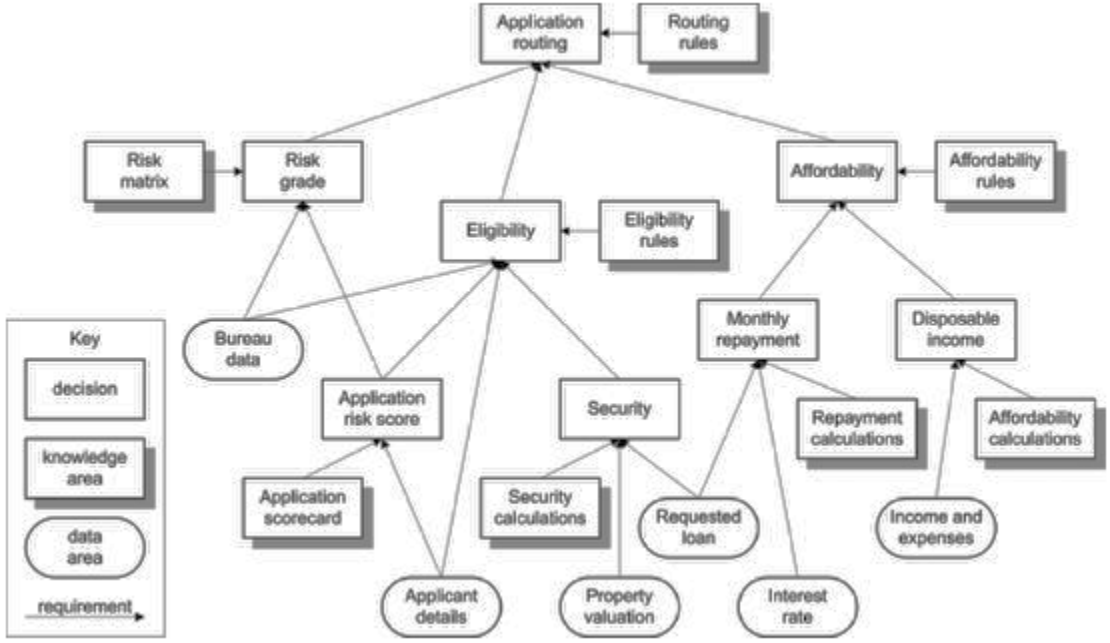
- Created by business in diagrams and tables
- … following a systematic methodology
- Models verifiable for completeness and consistency
- Decisions depend on supporting decisions and input data

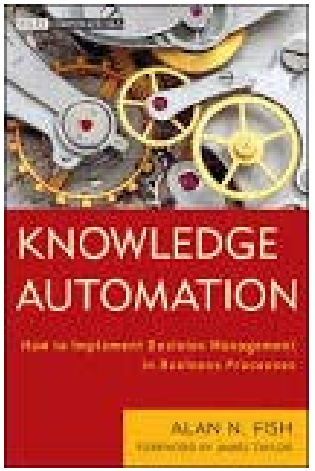




Decision Requirements "End-to-End"

- May be executed in multiple steps
- May include external decisions, human decisions
- May invoke reusable bits of decision logic ("knowledge areas")





DMN: Decision Model *and* Notation

- Metamodel provides formal definition of all model elements
  - Also defines the XML Schema
- XML and graphical models are ***equivalent***
- XML serialization can be interpreted or compiled and ***executed***

Date: 3 December 2015

OMG

OBJECT MANAGEMENT GROUP

Decision Model and Notation

Version 1.1 RTF

OMG Document Number: dtc/15-11-51

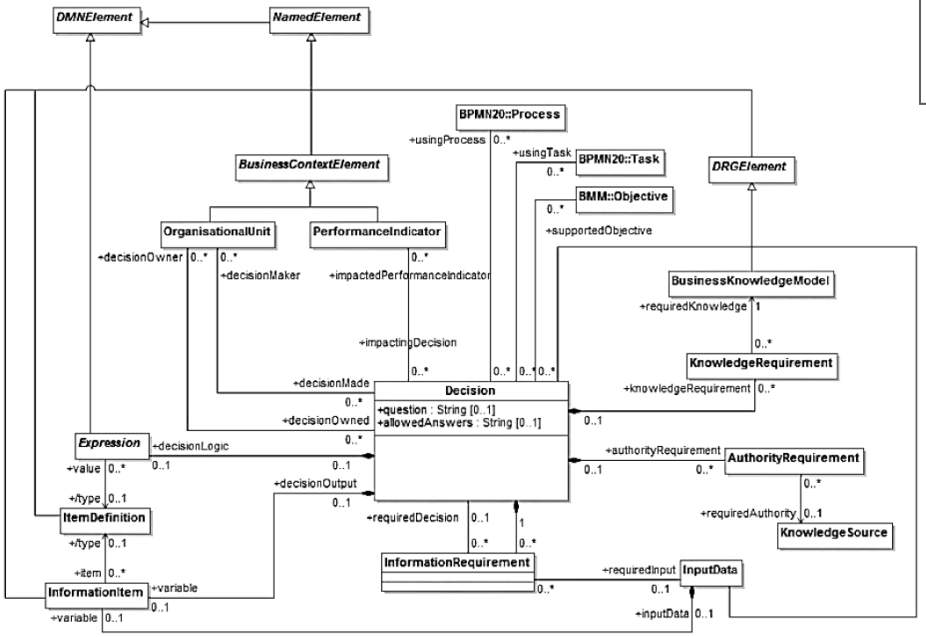
Standard document URL: http://www.omg.org/spec/DMN/1.1

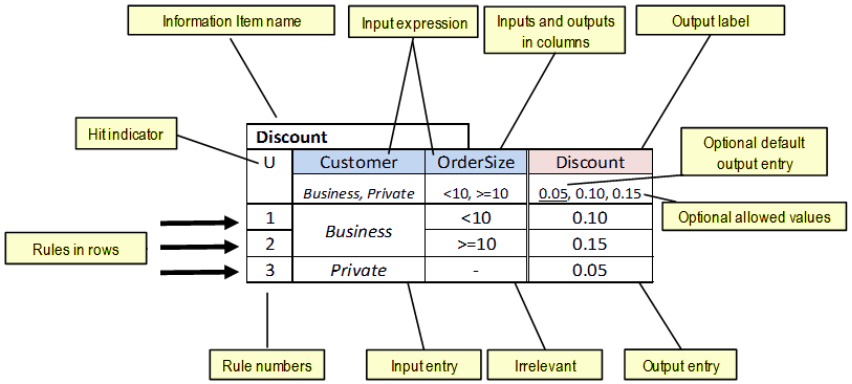
Machine Consumable File(s):

http://www.omg.org/spec/DMN/20151101/dmn.xmi

http://www.omg.org/spec/DMN/20151101/dmn.xsd

http://www.omg.org/spec/DMN/20151101/ch11example.xml





Business-Friendly… but Executable

- Decision logic defined by business users
- Decision logic is model-based
  - No more “requirements documents”
  - Verifiable for consistency, completeness
- Language syntax and formats are tool-independent
- Decision logic is “directly executable”
  - Provides end-to-end testable outcomes
  - May be exported to BRE rule language for performance, integration with physical data
- A common language shared by business and IT
- The language has 2 ***equivalent*** formats
  - Visual (DRD + boxed expressions)
  - Computable (XML)

FEEL and Boxed Expressions

- Executability requires a formal expression language
  - DMN wanted one that is business-friendly and tool-independent
    - Javascript, UEL, etc rejected as “developer languages”
    - Some advocated Excel formula language
    - Some advocated no formal expression language
  - In the end, DMN invented its own expression language: FEEL
  - FEEL is just an expression language (determines a value)
- Boxed expressions
  - Tabular definition of decision logic statements
- In combination, DRD + boxed expressions + FEEL constitute an executable language

AutoEligibilityScore							
onHighTheftList	if car.Make-model in HighTheftList then true else false;						
PotentialTheftCategory	U	onHighTheftList	car.Price	car.isConvertible			
	1	TRUE	-	-	*High*		
	2	FALSE	>$45000	-	*High*		
	3	FALSE	<=$45000	TRUE	*High*		
	4	FALSE	$[20000..45000]	FALSE	*Moderate*		
	5	FALSE	<$20000	FALSE	*Low*		
PotentialOccupant InjuryCategory	P	car.hasDriverAirbags	car.hasPassengerAirbags	car.hasSideAirbags	car.hasRollbar	car.isConvertible	PotentialOccupant InjuryCategory *(ExtremelyHigh, High, Moderate, Low)*
	1	FALSE	FALSE	FALSE	-	-	*ExtremelyHigh*
	2	-	-	-	FALSE	TRUE	*ExtremelyHigh*
	3	TRUE	FALSE	FALSE	-	-	*High*
	4	TRUE	TRUE	FALSE	-	-	*Moderate*
	5	TRUE	TRUE	TRUE	-	-	*Low*
AutoEligibility	U	PotentialOccupant InjuryCategory	PotentialTheft Category				
	1	*ExtremelyHigh*	-	*Not eligible*			
	2	*High*	-	*Provisional*			
	3	*Moderate, Low*	*High*	*Provisional*			
	4	*Moderate, Low*	*Moderate, Low*	*Eligible*			
if AutoEligibility="Not eligible" then 100 else if AutoEligibility="Provisional" then 50 else 0							

DMN's 5 Essential Elements

1. Decision Requirements Diagram
2. Decision Table formats
3. Standard expression language FEEL
4. Standard tabular format for complex expressions ("boxed expressions")
5. XML serialization of the model

Conformance Level 1

1. Decision Requirements Diagram
2. ~~Decision Table formats~~
3. ~~Standard expression language FEEL~~
4. ~~Standard tabular format for complex expressions (“boxed expressions”)~~
5. ~~XML serialization of the model~~

Conformance Level 2

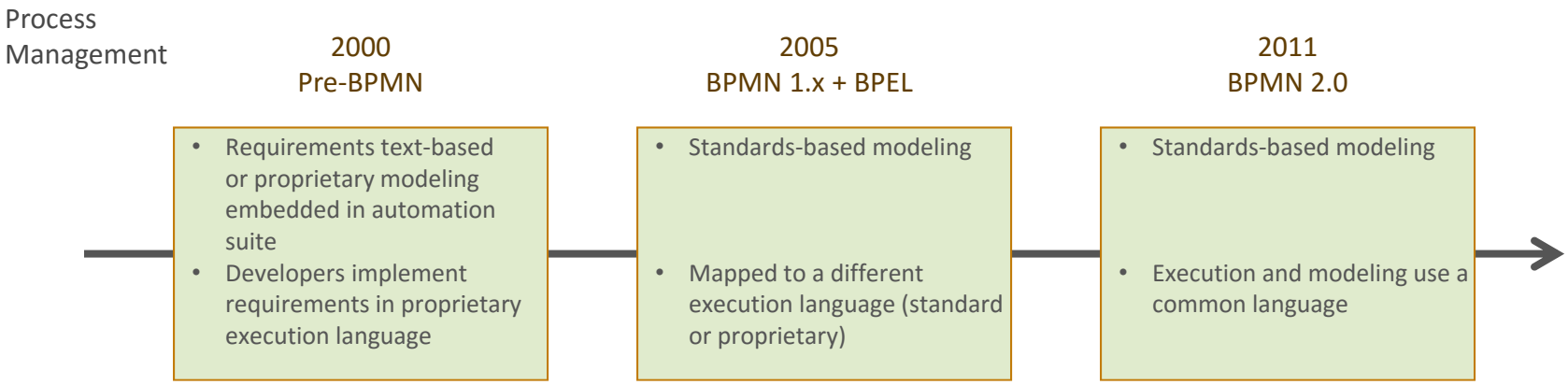
1. Decision Requirements Diagram
2. Decision Table formats (S-FEEL only)
3. ~~Standard expression language FEEL~~
4. ~~Standard tabular format for complex expressions (“boxed expressions”)~~
5. ~~XML serialization of the model~~

Conformance Level 3

1. Decision Requirements Diagram
2. Decision Table formats
3. Standard expression language FEEL
4. Standard tabular format for complex expressions (“boxed expressions”)
5. XML serialization of the model

The Lessons of BPMN

- Mapping models to a different language for execution is a short-term expedient
  - … that doesn't succeed in the long run
- Business users want this: "What You Model Is What You Execute"



What's NOT in DMN1.1

- UI for data modeling
- Business glossary
- Policy/constraint-oriented business rules (except as "knowledge sources")
  - Link to Structured Business Vocabulary and Rules (SBVR)
- Anything to do with methodology
- Anything to do with governance
- Anything to do with logic testing
  - Consistency/completeness checking
  - Test case generation and simulation
- Anything to do with execution
  - Actions to obtain input data
  - Access to physical data or systems
  - Actions triggered by decision result
  - Fault handling
  - Performance optimization

2. How the Language Works

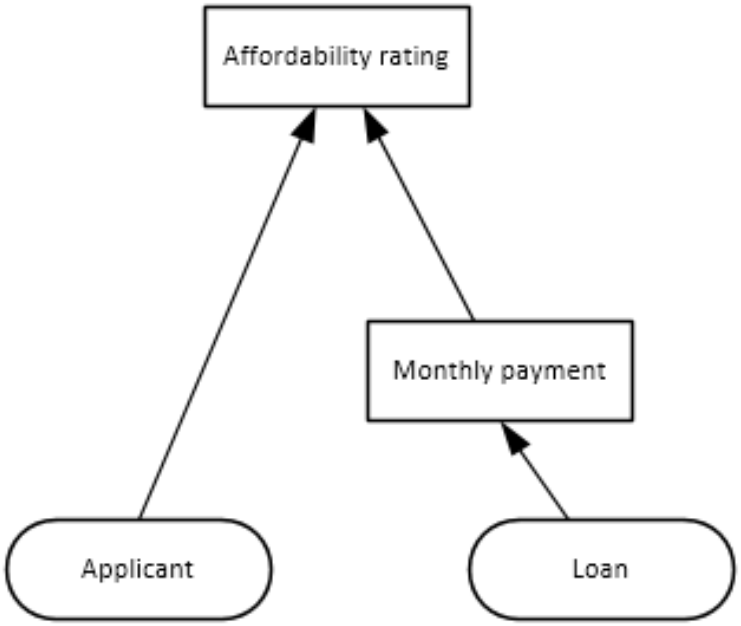
What is a Decision?

- Determination of output fact value resulting from specified input fact values
  - A “fact” is a variable with specified allowed values
  - DMN uses the term “variable” instead of “fact”
- Each decision defines corresponding variable with same name to hold its value
  - Value determined by the decision’s “value expression”
  - … typically either a decision table or literal expression
- Decision table value expressions based on “decision rules”
  - If condition1=true and condition2=true and… then outputExpression
  - … where condition1, condition2, etc are boolean expressions of inputs
  - Decision rules are NOT constraint rules
    - DMN decisions do not specify objective to optimize

Decision Requirements

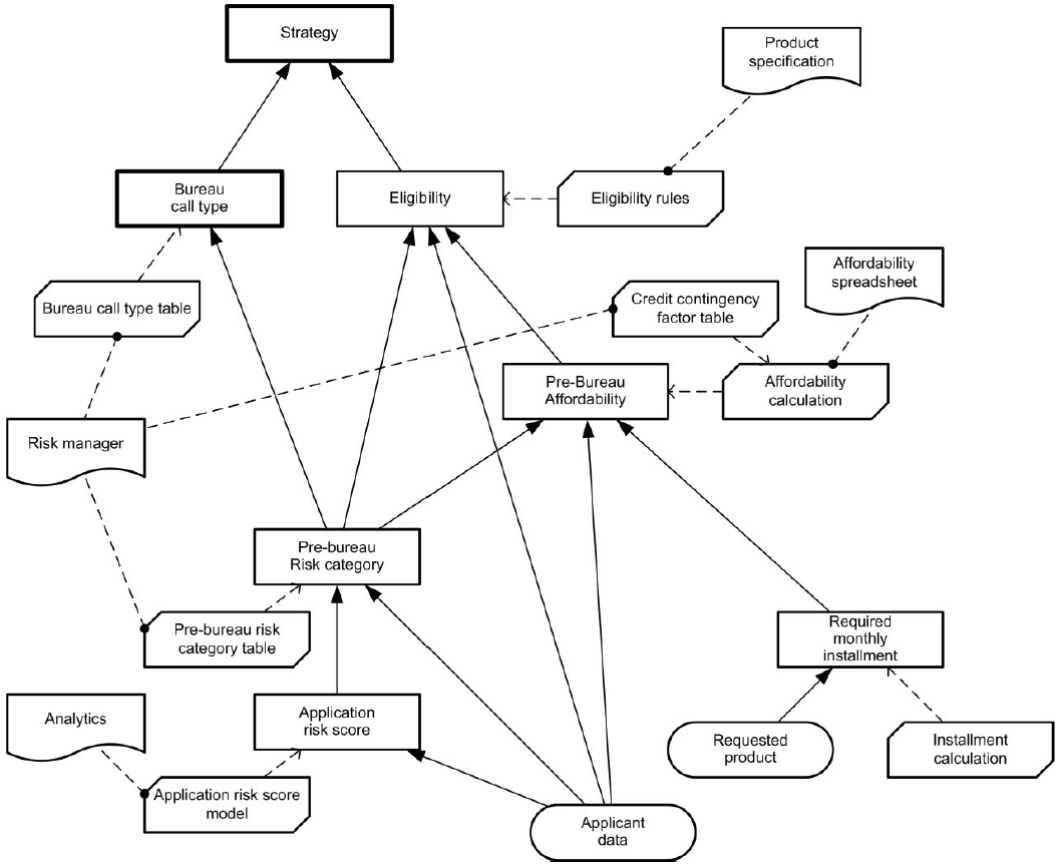
- Decisions require source data
- Decisions require other decisions

- Described in the model as a Decision Requirements Graph (DRG)
  - *Information requirements* link a decision (rectangle) to its required input data (ovals) and other supporting decisions
- Decision Requirements Diagram (DRD) is graphical depiction of DRG
  - May be a *filtered or truncated view*
  - Information requirements drawn as solid arrows
  - Decision's value expression may reference *only* its information requirements
- DRD able to describe structure of "end-to-end" decision logic
  - … even when executed in multiple steps



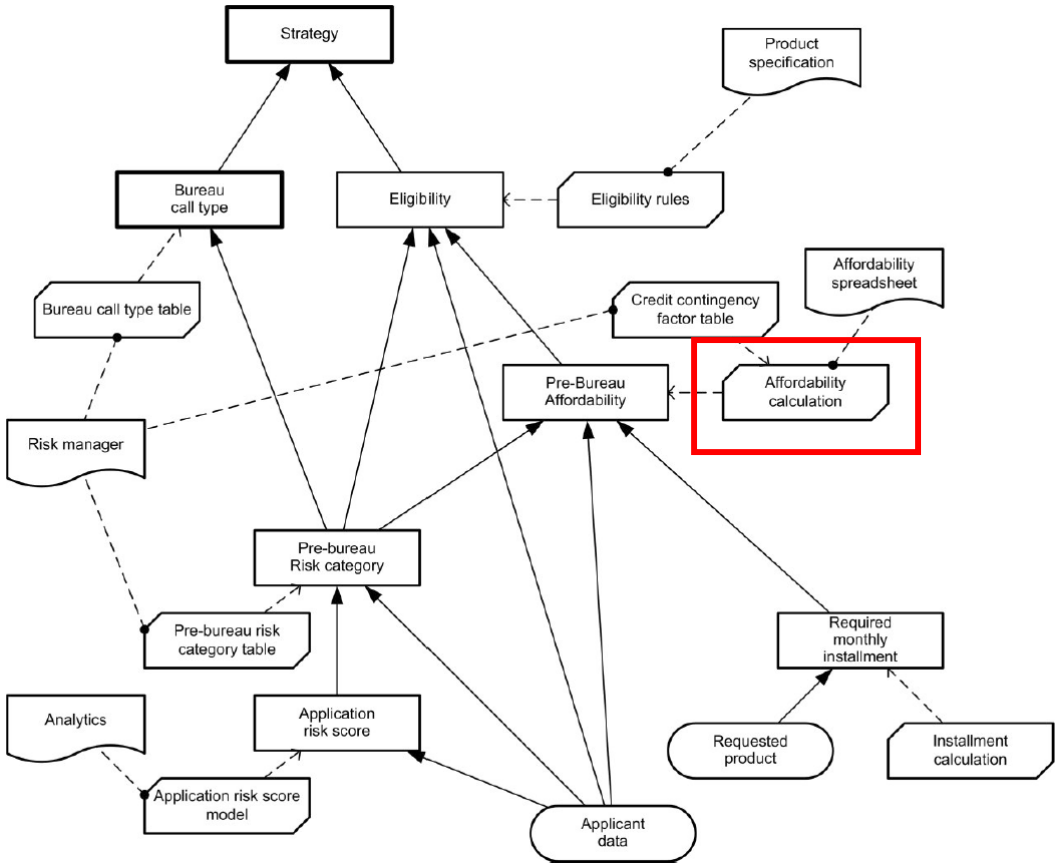
Decision Requirements Diagram

- Network of Decisions, Input Data, and Business Knowledge Models
  - High-level view of the end-to-end decision logic
  - Can describe an "extended" business decision, executed in multiple steps separated in time
    - … including human decisions, external decisions
  - Together with value expressions of the decision/BKM nodes, provides a declarative statement of the decision logic
- Requirements (connectors)
  - *Information requirements* link supporting decisions, input data
  - *Knowledge requirements* represent calls to reusable decision functions (BKMs)
  - *Authority requirements* link to associated policies, analytic models, experts…



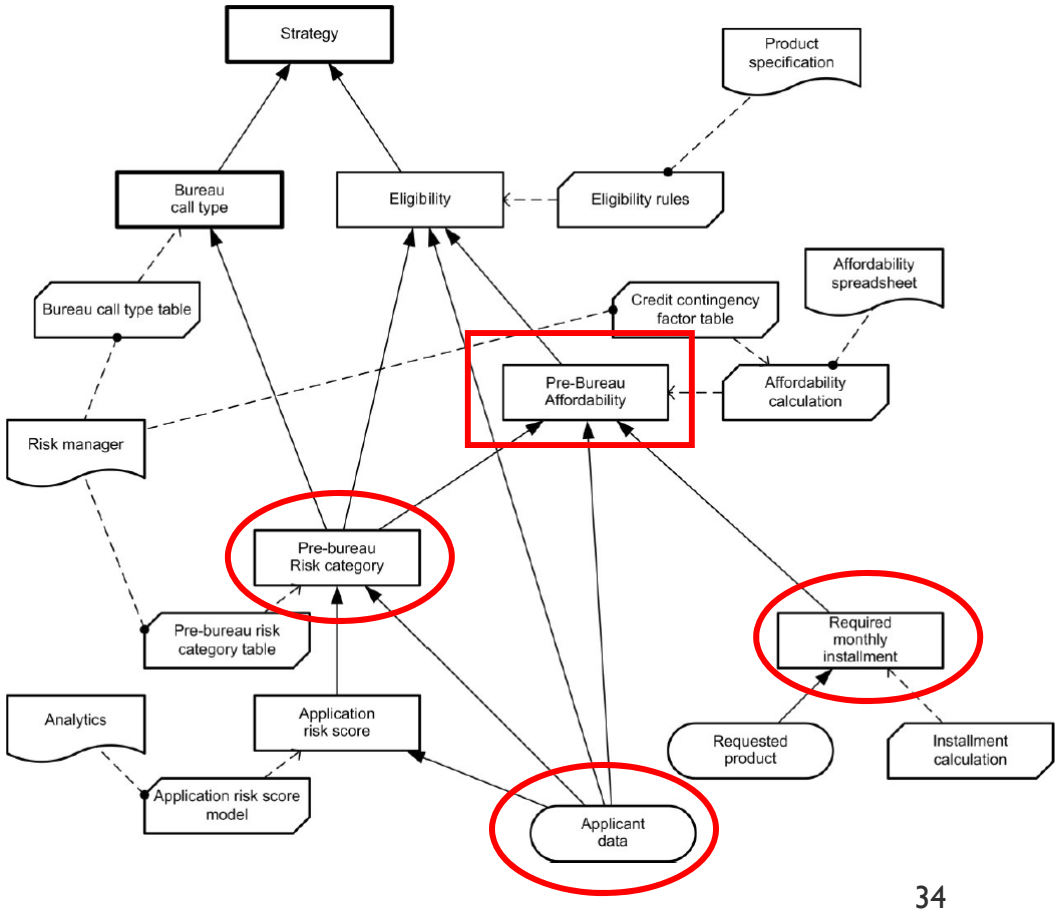
Business Knowledge Models (BKMs)

- Represent reusable decision logic
  - Affordability calculation invoked by *Pre-Bureau Affordability* and *Post-Bureau Affordability* (not shown in this DRD)
- "Invoked" by decisions…
  - … after mapping its inputs to BKM parameters
  - … and can invoke other BKMs
- Are controversial
  - Spec example uses them even when no logic reuse is apparent
    - Requires unnecessary parameter mapping
    - Clutters the graphics with both calling and called node depicted in diagram
    - Business users confused by parameterized logic



DRD Logic

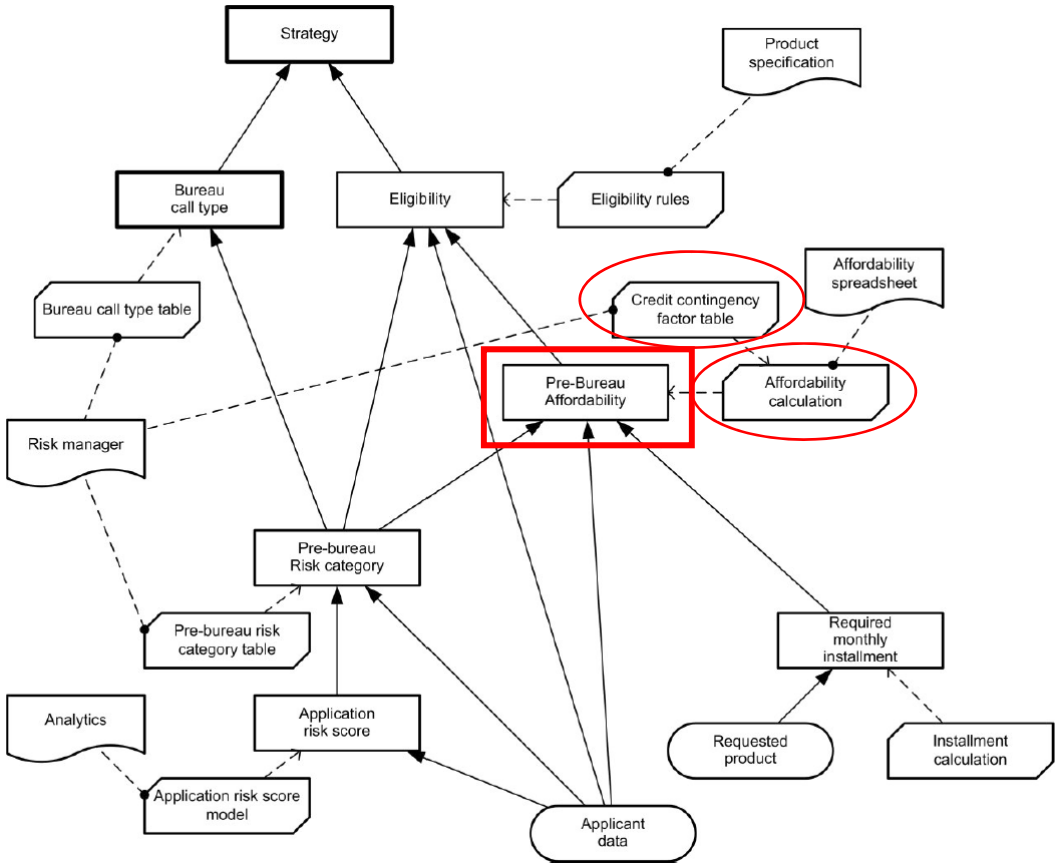
- Example: *Pre-Bureau Affordability*
  - Defines a variable with same name
  - Has 3 inputs: *Pre-bureau Risk category, Applicant data,* and *Required monthly installment*
    - Visualized as *information requirement*
    - Decision's value expression (not shown in DRD) depends *only* on *these inputs*



DRD Logic

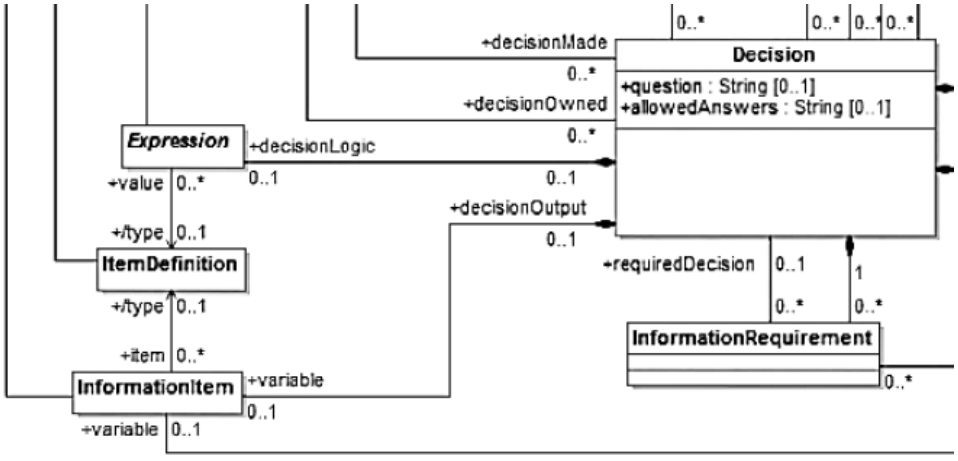
- Decision invokes a BKM: *Affordability calculation*
  - Visualized as *knowledge requirement*
  - The BKM is based on the knowledge source *Affordability spreadsheet*
    - Visualized as *authority requirement*
    - Effectively just a diagram annotation
  - BKM invokes another BKM *Credit contingency factor table*

- Executing the decision does this:
  1. *Pre-Bureau Affordability* inputs are mapped to the parameters of *Affordabiity calculation*
  2. *Affordability calculation* value expression is evaluated (after mapping/calling *Credit contingency factor table*)
  3. *Affordability calculation* output value saved to variable *Pre-Bureau Affordability*



Decision Metamodel and Schema

- Key attributes of a decision are
  - Name
  - Variable (InformationItem)
  - Datatype (itemDefinition)
  - InformationRequirements, KnowledgeRequirements
  - Value expression, either…
    - LiteralExpression
    - DecisionTable
    - Invocation (plus a couple more)

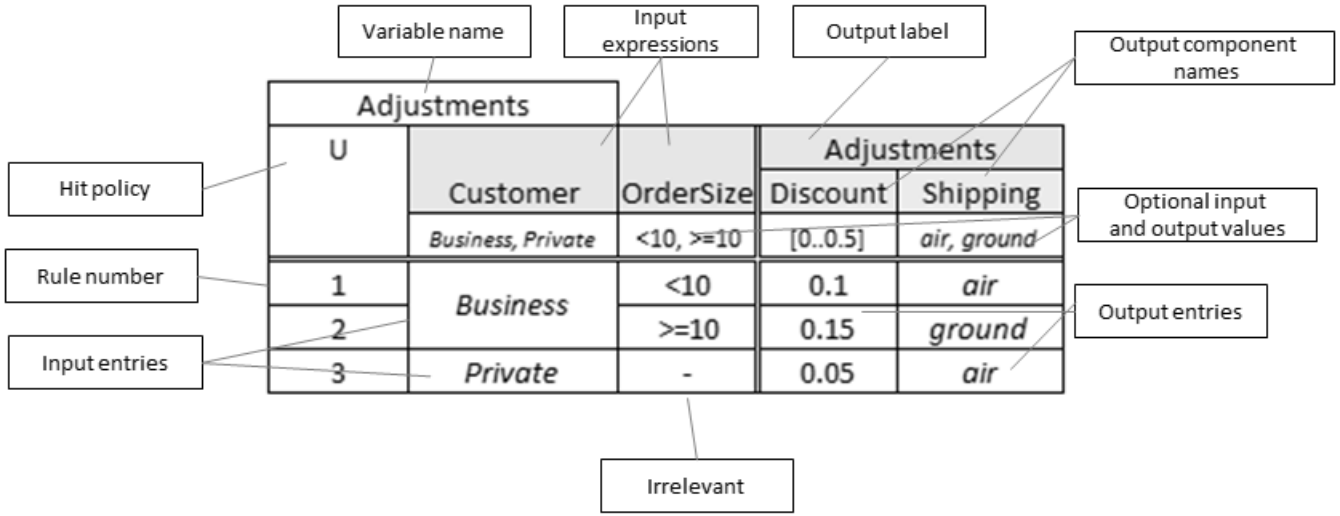


XML serialization:

```
226  <decision name="Pre-bureauRiskCategory" id="d_Pre-bureauRiskCategory">
227    <variable name="Pre-bureauRiskCategory" typeRef="tns:tRiskCategory"/>
228    <informationRequirement>
229      <requiredDecision href="#d_ApplicationRiskScore"/>
230    </informationRequirement>
231    <informationRequirement>
232      <requiredInput href="#i_ApplicantData"/>
233    </informationRequirement>
234    <knowledgeRequirement>
235      <requiredKnowledge href="#b_Pre-bureauRiskCategoryTable"/>
236    </knowledgeRequirement>
237    <invocation>
238      <literalExpression>
239        <text>Pre-bureauRiskCategoryTable</text>
240      </literalExpression>
241      <binding>
242        <parameter name="ExistingCustomer"/>
243        <literalExpression>
244          <text>ApplicantData.ExistingCustomer</text>
245        </literalExpression>
246      </binding>
247      <binding>
248        <parameter name="ApplicationRiskScore"/>
249        <literalExpression>
250          <text>ApplicationRiskScore</text>
251        </literalExpression>
252      </binding>
253    </invocation>
254  </decision>
255  <decision name="Post-bureauRiskCategory" id="d_Post-bureauRiskCategory">
256    <variable name="Post-bureauRiskCategory" typeRef="tns:tRiskCategory"/>
257    <informationRequirement>
258      <requiredInput href="#i_ApplicantData"/>
259    </informationRequirement>
260    <informationRequirement>
261      <requiredDecision href="#d_ApplicationRiskScore"/>
```

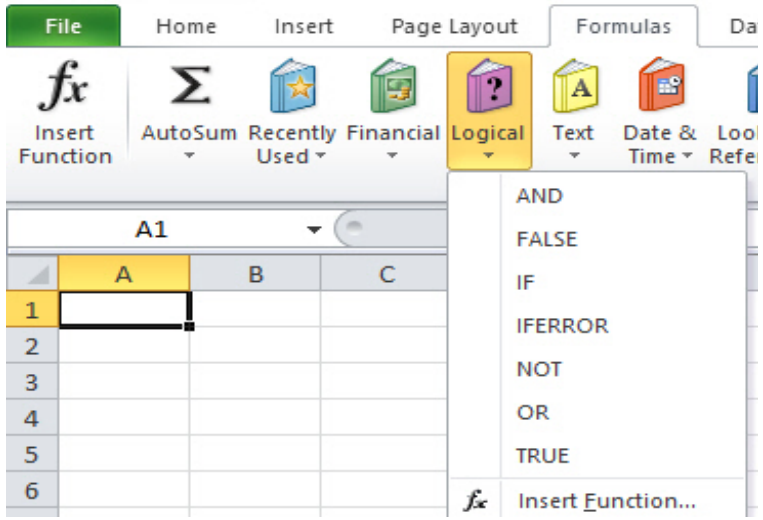
Decision Table

- The most familiar type of decision logic
- Several layouts allowed - most DMN tools use “rules-as-rows”
- In that layout…
  - Columns are inputs and outputs
  - Optional allowed values in header row allow verification of rule gaps and overlaps
  - Each row below header is a decision rule
    - Cells (“input entries”) when combined with input expression define a boolean condition
    - If all cells in a rule are true, rule “matches” and output entry expression is enabled
    - Hit policy specifies output selection when multiple rules match
- Declarative, row/column order does not affect the result



FEEL

- DMN's standard expression language
  - Business-friendly and capable of handling real-world decision logic



- Excel formulas vs FEEL
  - Which is more business-friendly?
  - Consider this rule in natural language:
    If Risk is "Low" or "Medium" and PTI<0.3 then "Approved" else "Declined"
    - Excel formula syntax:
      `=IF(AND(OR(Risk="Medium",Risk="Low"),PTI<0.3),"Approved","Declined")`
    - FEEL syntax:
      `if Risk in ["Low", "Medium"] and PTI <0.3 then "Approved" else "Declined"`
- Not so easy to implement, vendors slow to support it
  - Spaces in names of variables and built-in functions makes parsing a lot more work
  - Rich built-in functions and operators
- … but FEEL adoption is critical to DMN as a standard

FEEL

- Defined as formal grammar in spec Chapter 10
  - Pure expression language, just generates a value
  - Can reference variables but not define or assign
  - Composable: any value may be replaced by FEEL expression
- Strongly typed language
- Rich features
  - *Qualified names* of structured variables:
    - Customer.Age
  - *Built-in functions*
    - Subset of XPATH 2.0 functions
  - *User-defined functions* streamline logic reuse
    - payment (principalAmt, ratePct, termMonths)
  - *Filter expressions* for table query and joins
    - Order.item[price<100]
  - *Sort*
  - Advanced capabilities with *FEEL operators*
    - if [Boolean expression] then [expression1] else [expression2]
    - Iteration: for [item] in [list or relation] return [expression(item)]

10.3.1.2 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 =
   a. textual expression |
   b. 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 | simple positive 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, ")" |

Boxed Expressions

- Standard tabular UI for more complex value expression
  - Allows a single decision to define its value expression in multiple steps
- Decision table is one form of boxed expression
- Otherwise, typically a 2-column table
  - First column (shaded) is a *variable name*
  - Second column is *value expression*
- Invocation
  - Maps decision inputs to parameters of an invoked BKM
- Context
  - Advanced value expression type
  - Each context entry (row) defines a local variable and its value expression
  - Context entries may reference other local variables
  - Final result box (no variable) is value expression for the context as a whole

Required monthly installment	
Installment calculation	
Product Type	Requested product . ProductType
Rate	Requested product . Rate
Term	Requested product . Term
Amount	Requested product . Amount

Invocation parameter mapping as boxed expression

Installment calculation	
(Product Type, Rate, Term, Amount)	
Monthly Fee	if Product Type = "STANDARD LOAN" then 20.00 else if Product Type = "SPECIAL LOAN" then 25.00 else null
Monthly Repayment	PMT(Rate, Term, Amount)
Monthly Repayment + Monthly Fee	

Context as boxed expression

Recap: What Is DMN?

- 5 Key Elements
  1. Decision Requirements Diagram
  2. Decision Table
  3. Metamodel and Schema
  4. FEEL
  5. Boxed Expressions

- Combination of all 5 fulfills DMN's objectives:
  - Business-friendly, communicates logic via diagrams and tables
  - Tool-independent
  - Executable

- A complete, executable, tool-independent decision model can be represented in 2 equivalent ways
  - Graphically, as DRD + boxed expressions… OR
  - Serialized as XML

3. Meeting the Implementation Challenge

What Do We Mean by "Execution"?

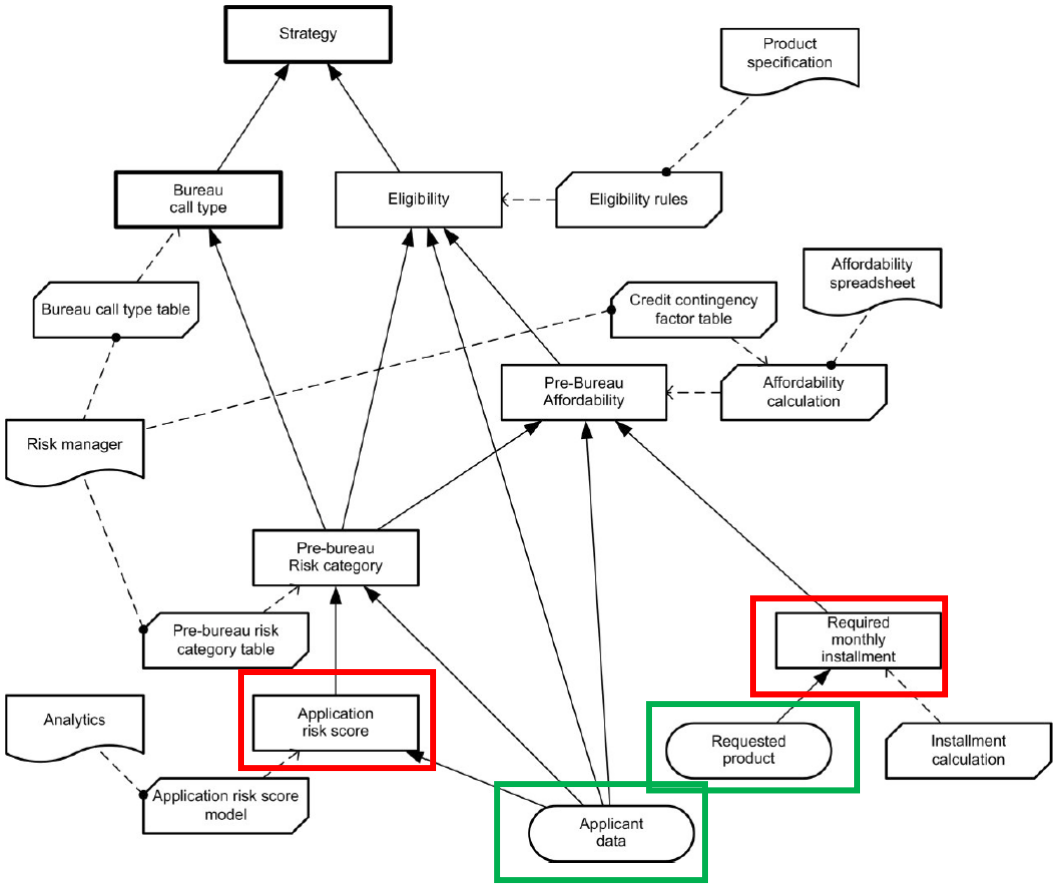
- At design-time, given values of input data elements (and human decisions), determine the value of any decision in the DRD
  - In the modeling tool, not deployed to runtime
  - Purpose is validation/analysis of decision logic, not production execution
    - Based on logical data not physical data
    - Does not require ability to handle massive datasets
    - Does not require production-level performance and reliability

DMN XML as an Executable Program

- You can take a DMN model in XML and execute it
  - Given values for the input data (and human decisions, external decisions), generate values for all decision nodes
- DRD logic is inferential, “partially ordered”
  - If decision A requires value of decision B, then B must be evaluated before A
- Decision table logic is purely declarative
- Tools are free to develop their own algorithms for walking the DRD

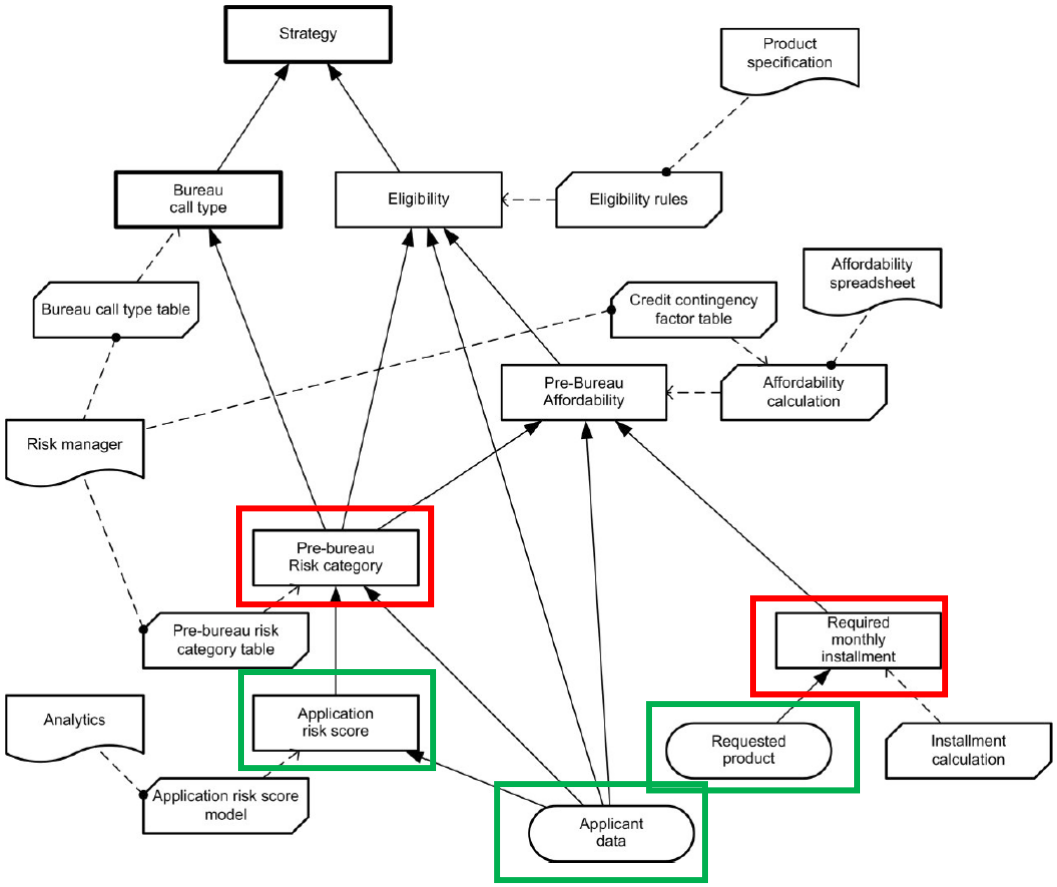
Walking the DRD

- Forward chaining example
  1. List input data and decisions with known values
  2. List ready decisions (all inputs on known values list)
  3. Evaluate first ready decision
  4. Iterate 1-3 until done



Walking the DRD

- Forward chaining example
  1. List input data and decisions with known values
  2. List ready decisions (all inputs on known values list)
  3. Evaluate first ready decision
  4. Iterate 1-3 until done



The Hard Part: FEEL

- Parsing
  - Names may contain spaces
    - … and arithmetic operators (+, -, /, *)
    - … and logical operators (and, or)
    - … and path operators (.)
    - … and XML-unfriendly characters (')
    - Need to consult “names in scope” (information requirements, imports) in order to parse
  - Built-in functions contain spaces, also
    - E.g., starts with() not starts-with()
  - Context-dependent syntax
    - A + B means addition if A, B are numbers, concatenation if A, B are strings, or possibly a name

```
Abc.com.customer list[customer’s first + last name = "Joe Smith"].monthly income
```

Namespace Variable name Component name Component name

- So far all efforts to simplify implementation rejected as “business-unfriendly”

The Hard Part: FEEL

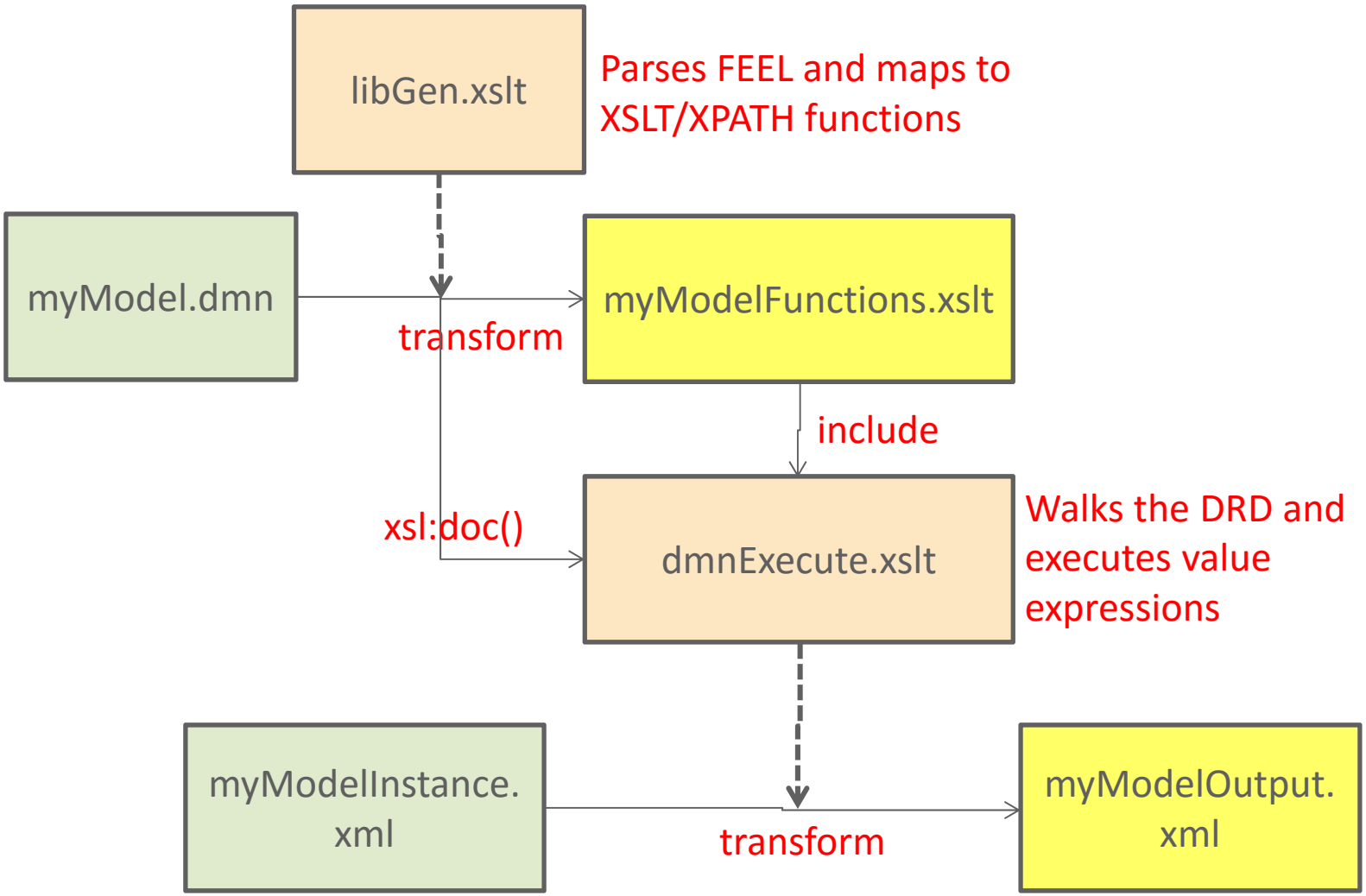
- Built-in functions, including…
  - String functions like: replace(inputString, regexPattern, replacementString, flags)
  - List functions like: insert before(list, position, newItem), index of(list, match), distinct values(list)
  - Date/time functions like: years and months duration(startDateTime, endDateTime)
  - Sort(list)
- Filter expressions
  - Order.item[price >100]
- User-defined functions
  - Monthly payment(loan.principal*(1 + loan.points), loan.rate, loan.term)
- Operators
  - Iteration: for item in order return price(item)
  - Some/every: some item in order satisfies price(item)>100
  - If..then: if order.item[1].price >100 then “high” else “low”

State of Play Q1 2016

- Tool vendors ignoring FEEL and boxed expressions
  - "Not business-friendly"
  - Too hard to parse
  - Built-in functions and operators too hard to implement
  - "Our customers are not asking for it" (!)

- Instead, satisfied with passing requirements to commercial BRE rule languages

My Own Implementation of DMN/FEEL



Lending-original.xml (DMN)

- Serialization of the Lending decision example in spec Chapter 11

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!-- edited with XMLSpy v2013 rel. 2 sp2 (x64) (http://www.altova.com) by Bruce Silver (private) -->
3  <?altova_sps C:\Users\Bruce\Documents\DMN\DMN1.1 final\execution\dmn2.sps?>
4  <definitions id="def01" name="Lending decision - Original" namespace="methodandstyle.com/lendingOriginal" xmlns="http://www.omg.org/spec/DMN/20151101/dmn.xsd" xmlns:tns
   ="methodandstyle.com/lendingOriginal" xmlns:feel="http://www.omg.org/spec/FEEL/20140401" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="
   http://www.omg.org/spec/DMN/20151101/dmn.xsd ../../dmn.xsd">
5    <itemDefinition name="tEligibility">
6      <typeRef>feel:string</typeRef>
7      <allowedValues>
8        <text>"INELIGIBLE", "ELIGIBLE"</text>
9      </allowedValues>
10   </itemDefinition>
11   <itemDefinition name="tBureauCallType">
12     <typeRef>feel:string</typeRef>
13     <allowedValues>
14       <text>"FULL", "MINI", "NONE"</text>
15     </allowedValues>
16   </itemDefinition>
17   <itemDefinition name="tStrategy">
18     <typeRef>feel:string</typeRef>
19     <allowedValues>
20       <text>"DECLINE", "BUREAU", "THROUGH"</text>
21     </allowedValues>
22   </itemDefinition>
23   <itemDefinition name="tRiskCategory">
24     <typeRef>feel:string</typeRef>
25     <allowedValues>
26       <text>"DECLINE", "HIGH", "MEDIUM", "LOW", "VERY LOW"</text>
27     </allowedValues>
28   </itemDefinition>
29   <itemDefinition name="tRouting">
30     <typeRef>feel:string</typeRef>
```

Text | Grid | Schema | WSDL | XBRL | Authentic | Browser

Lending-originalFunctions.xslt

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <xsl:stylesheet xmlns="http://methodandstyle/FunctionLib" xmlns:fn="http://www.w3.org/2005/xpath-functions" xmlns:n1="http://www.omg.org/spec/DMN/20151101/dmn.xsd"
   xmlns:n2="http://methodandstyle/FunctionLib" xmlns:n3="methodandstyle/FunctionLib" xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:xsl="
   http://www.w3.org/1999/XSL/Transform" version="2.0">
3      <xsl:function name="n3:evalLX">
4          <xsl:param name="xid"/>
5          <xsl:param name="knownVals"/>
6          <xsl:choose>
7              <xsl:when test="$xid='C%InstallmentCalculation.MonthlyRepayment'">
8                  <xsl:sequence select="( $knownVals/*[@name='Amount'] * $knownVals/*[@name='Rate'] div 12 ) div ( 1 - n3:pow( ( 1 + $knownVals/*[@name='Rate'] div 12 ) , -
   $knownVals/*[@name='Term'] ) )"/>
9              </xsl:when>
10             <xsl:when test="$xid='D%Strategy%R%1%IC%1'">
11                 <xsl:sequence select="$knownVals/*[@name= 'Eligibility']='INELIGIBLE'"/>
12             </xsl:when>
13             <xsl:when test="$xid='D%Strategy%R%1%IC%2'">
14                 <xsl:sequence select="true()"/>
15             </xsl:when>
16             <xsl:when test="$xid='D%Strategy%R%2%IC%1'">
17                 <xsl:sequence select="$knownVals/*[@name= 'Eligibility']='ELIGIBLE'"/>
18             </xsl:when>
19             <xsl:when test="$xid='D%Strategy%R%2%IC%2'">
20                 <xsl:sequence select="$knownVals/*[@name= 'BureauCallType']='FULL' or $knownVals/*[@name= 'BureauCallType']='MINI'"/>
21             </xsl:when>
22             <xsl:when test="$xid='D%Strategy%R%3%IC%1'">
23                 <xsl:sequence select="$knownVals/*[@name= 'Eligibility']='ELIGIBLE'"/>
24             </xsl:when>
25             <xsl:when test="$xid='D%Strategy%R%3%IC%2'">
26                 <xsl:sequence select="$knownVals/*[@name= 'BureauCallType']='NONE'"/>
27             </xsl:when>
28             <xsl:when test="$xid='B%CreditContingencyFactorTable%R%1%IC%1'">
29                 <xsl:sequence select="$knownVals/*[@name= 'RiskCategory']='HIGH' or $knownVals/*[@name= 'RiskCategory']='DECLINE'"/>
```

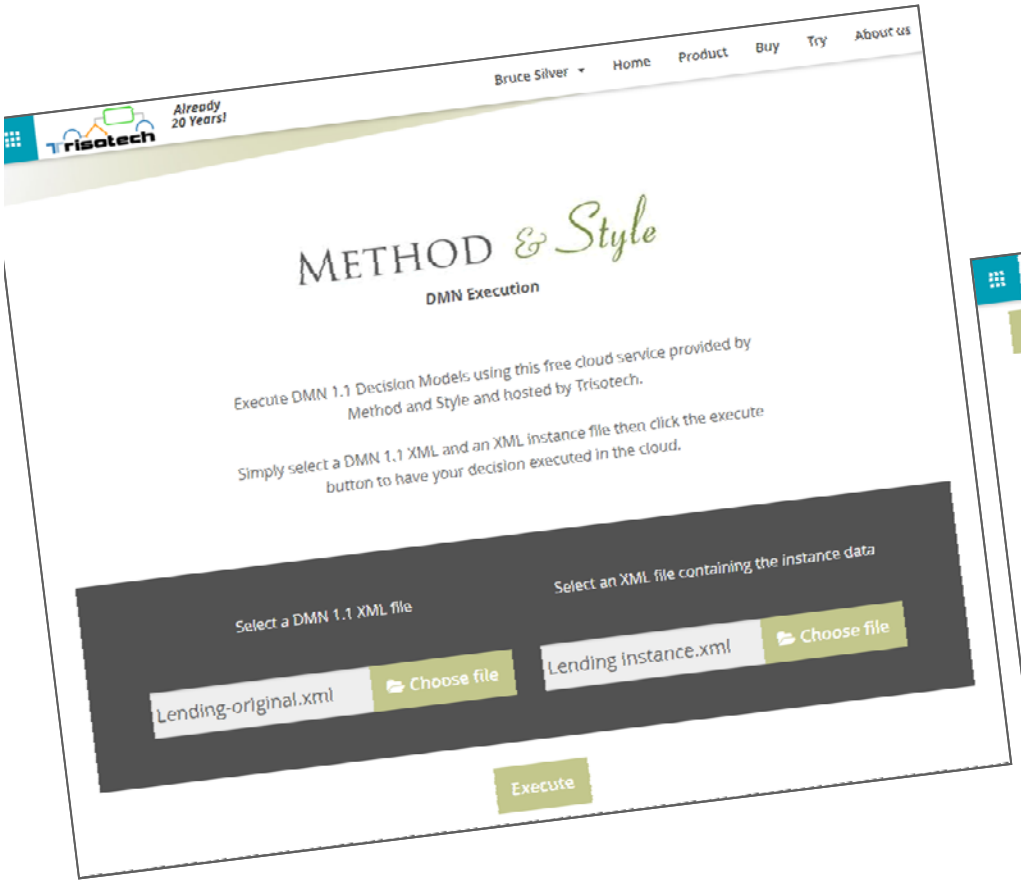
Lending-originalInstance.xml

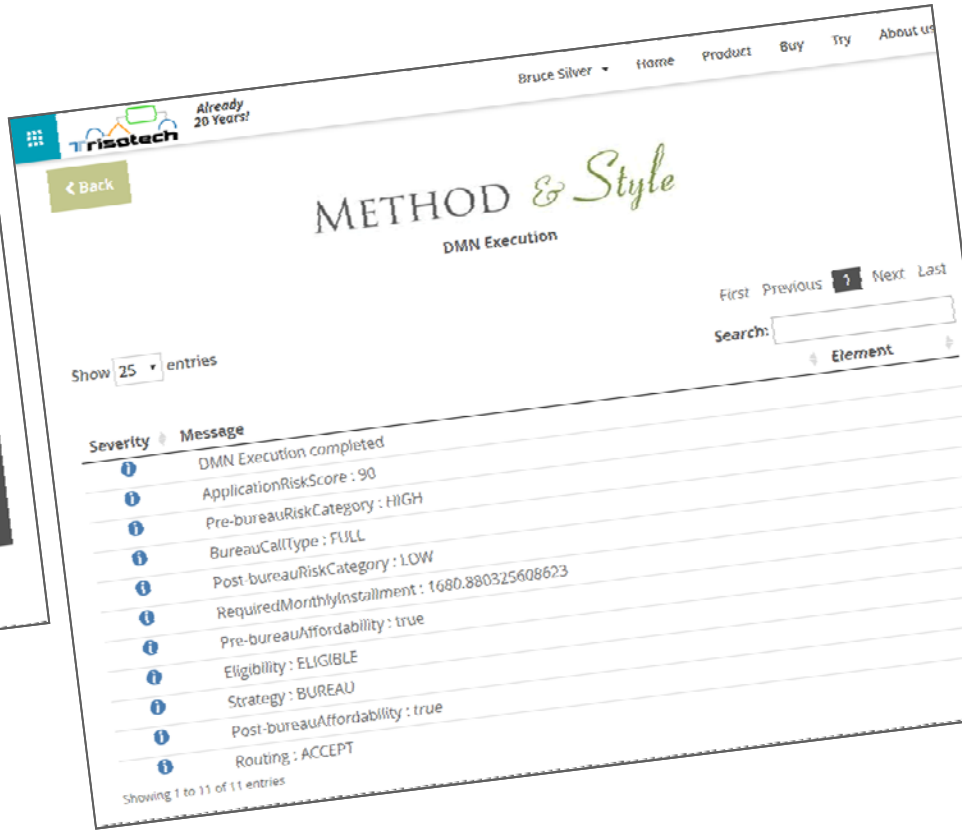
```xml
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="dmnExecute-new.xslt"?>
<decisionModelInstance xmlns="methodandstyle/dmnInstance">
    <decisionModelName>Lending-original.xml</decisionModelName>
    <inputData name="ApplicantData">
        <Monthly>
            <Income>6000</Income>
            <Expenses>2000</Expenses>
            <Repayments>0</Repayments>
        </Monthly>
        <Age>35</Age>
        <ExistingCustomer>true</ExistingCustomer>
        <MaritalStatus>M</MaritalStatus>
        <EmploymentStatus>EMPLOYED</EmploymentStatus>
    </inputData>
    <inputData name="RequestedProduct">
        <ProductType>STANDARD LOAN</ProductType>
        <Amount>350000</Amount>
        <Rate>0.0395</Rate>
        <Term>360</Term>
    </inputData>
    <inputData name="BureauData">
        <CreditScore>649</CreditScore>
        <Bankrupt>false</Bankrupt>
    </inputData>
</decisionModelInstance>
```

Lending-originalOutput.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<kList xmlns="methodandstyle/dmnInstance" xmlns:fn="http://www.w3.org/2005/xpath-functions" xmlns:n1="http://www.omg.org/spec/DMN/20151101/dmn.xsd" xmlns:n2="
methodandstyle/dmnInstance" xmlns:n3="methodandstyle/FunctionLib" xmlns:tns="methodandstyle/dmnExecute" xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <node name="ApplicantData" type="inputData">
    <Monthly>
      <Income>6000</Income>
      <Expenses>2000</Expenses>
      <Repayments>0</Repayments>
    </Monthly>
    <Age>35</Age>
    <ExistingCustomer>true</ExistingCustomer>
    <MaritalStatus>M</MaritalStatus>
    <EmploymentStatus>EMPLOYED</EmploymentStatus>
  </node>
  <node name="RequestedProduct" type="inputData">
    <ProductType>STANDARD LOAN</ProductType>
    <Amount>350000</Amount>
    <Rate>0.0395</Rate>
    <Term>360</Term>
  </node>
  <node name="BureauData" type="inputData">
    <CreditScore>649</CreditScore>
    <Bankrupt>false</Bankrupt>
  </node>
  <node name="ApplicationRiskScore" type="decision">90</node>
  <node name="Pre-bureauRiskCategory" type="decision">HIGH</node>
  <node name="BureauCallType" type="decision">FULL</node>
  <node name="Post-bureauRiskCategory" type="decision">LOW</node>
  <node name="RequiredMonthlyInstallment" type="decision">1680.880325608555</node>
  <node name="Pre-bureauAffordability" type="decision">true</node>
  <node name="Eligibility" type="decision">ELIGIBLE</node>
  <node name="Strategy" type="decision">BUREAU</node>
  <node name="Post-bureauAffordability" type="decision">true</node>
  <node name="Routing" type="decision">ACCEPT</node>
</kList>
```

My Implementation In Trisotech Cloud





Spaces in Names Cause Syntax Errors

- Validation is critical for any execution software
  - ItemDefinition names with spaces
  - Pointers to nowhere
  - Names not in scope

DMN Level 3 Implementation… For Real

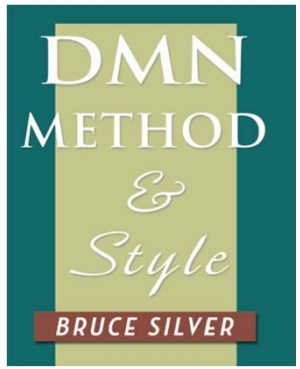
- Oracle
  - FEEL and boxed expressions, not yet DRD
  - Release later this year as part of Oracle Process Cloud Service
- Trisotech
  - Working DMN editor supporting FEEL and boxed expressions, XML export
  - Execution currently limited to S-FEEL decision tables
- OpenRules
  - (Beta) Open source, full FEEL (no spaces in names), XML model import
  - No boxed expressions yet, but committed to full CL3 implementation
- RedHat (DROOLS)
  - Full chapter 10 FEEL parser/execution, open source under Apache license (in development)
- WfMC
  - DMN Technology Compatibility Kit to verify support for FEEL Level 3 features
  - Test suite, validation routines, and more

Also…

- Sapiens Decision
  - Standalone DMN modeler with FEEL in development, map to TDM for governance and production use
- Signavio
  - Level "2a" today (FEEL-equivalent features but proprietary syntax), export to DROOLS
  - Committed to CL3 eventually but not top priority today
- FICO
  - Free DMN CL2 editor today
  - "*CL3 compliance is a secondary objective and FICO may decide to use SRL rather than FEEL for the primary decision logic language in DMS. We might provide SRL-FEEL conversion for openness but SRL has a good deal more functionality than FEEL so not all models created with DMS would be capable of representation in pure DMN.*"
- Camunda
  - November release will be CL2 (DRD + S-FEEL), waiting for customer demand for CL3

In Conclusion…

- The promise of DMN – Level 3 implementation – is real!
- It's happening this year
- Vendors who thought Level1/Level2 is enough may begin to rethink
- If you want to contribute your own ideas
  - Join the RTF!
    - Contact gary.hallmark@oracle.com
  - Join the WfMC DMN TCK!
    - Contact KSwenson@us.fujitsu.com



- Learn how to use DMN
  - DMN Method and Style https://www.amazon.com/dp/0982368151
  - DMN training and certification http://methodandstyle.com/product/dmntraining/
    - Next class live/online July 12-14 from 11am-4pm ET each day

- For copy of this presentation, email bruce@brsilver.com