Leveraging the Power of IDP with the Flexibility of DMN: a Multifunctional API

RuleML + RR 2021

Simon Vandevelde, Vedavyas Etikala,
Jan Vanthienen, Joost Vennekens
KU Leuven — EAVISE, LIRIS
13 September 2021

Outline

1. DMN
2. IDP
3. DMN-IDPy API
4. Application example

1 DMN

What is DMN?

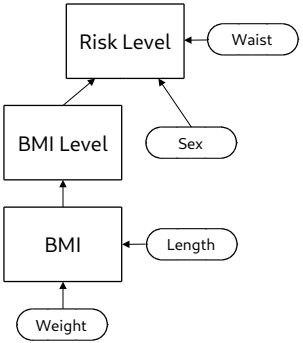
Decision Model and Notation

- Notation standard for decision logic
- Published by OMG Group
- Table-based representation of decisions
- Decision Requirements Diagram
- Main focus: readability, traceability

What is DMN?

Decision Requirements Diagram

- Represents structure of decision model
- Inputs, decision tables, knowledge sources, . . .



What is DMN?

Decision Tables

- Decide outputs (blue) based on inputs (green)
- Rows represent rules
- Columns represent values
- Behaviour defined by Hit Policy
- (S-)Feel in inputs

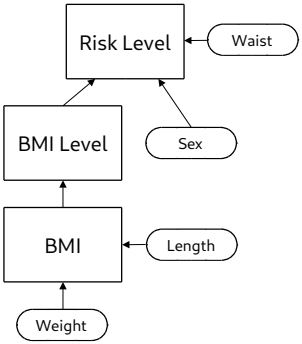
BMI Level		
U	BMI	BMILevel
1	< 18.5	Underweight
2	$[18.5..25]$	Normal
3	$(25..30]$	Overweight
4	> 30	Obese

What is DMN?

(Simple) Friendly Enough Expression Language

- Simple values
- Lists of values
- Numerical comparisons
- Ranges of values
- Calculations

Full DMN example



Risk Level

U	BMILevel	Sex	Waist	RiskLevel
1	Normal	–	–	Low
2	Underweight	–	–	High
3	Overweight	Male	≤ 102	Increased
4	Overweight	Male	> 102	High
5	Overweight	Female	≤ 88	Increased
6	Overweight	Female	> 88	High
7	Obese	Male	≤ 102	High
8	Obese	Male	> 102	Very High
9	Obese	Female	≤ 88	High
10	Obese	Female	> 88	Very High

BMI Level

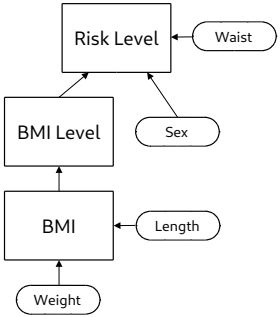
U	BMI	BMILevel
1	< 18.5	Underweight
2	$[18.5..25]$	Normal
3	$(25..30]$	Overweight
4	> 30	Obese

BMI

U	Weight	Length	BMI
1	–	–	Weight/(Length*Length)

Current DMN execution

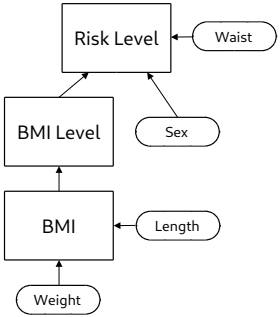
- the “standard” *bottom-to-top*
  - Start at “bottom” inputs
  - Evaluate table by table
  - Finish at top-level variable
- *reasoning on sub-decisions*
- *wildcard mode*



Current DMN execution

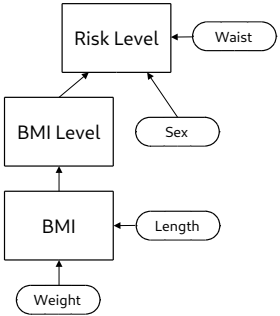
- the “standard” *bottom-to-top*
- *reasoning on sub-decisions*
  - Evaluate only subset of decisions
  - E.g. only calculate BMI

  → Sex and Waist are irrelevant
- *wildcard mode*



Current DMN execution

- the “standard” *bottom-to-top*
- *reasoning on sub-decisions*
- *wildcard mode*
  - Evaluate decision with partial info
  - E.g. set *Sex* as wildcard
  - Returns all possible solutions



2 IDP

IDP

IDP system: knowledge-based reasoning engine

$\rightarrow$ knowledge is modeled in extended FOL

$$
\begin{aligned}
&(\mathit{BMI} < 18.5 \Rightarrow \mathit{BMILevel} = \mathit{Underweight}) \\
\wedge&(18.5 \leq \mathit{BMI} < 18.5 \Rightarrow \mathit{BMILevel} = \mathit{Normal}) \\
\wedge&(25 \leq \mathit{BMI} < 30 \Rightarrow \mathit{BMILevel} = \mathit{Overweight}) \\
\wedge&(30 \leq \mathit{BMI} \Rightarrow \mathit{BMILevel} = \mathit{Obese})
\end{aligned}
\tag{1}
$$

BMI Level		
U	BMI	BMILevel
1	< 18.5	Underweight
2	$[18.5..25]$	Normal
3	$(25..30]$	Overweight
4	> 30	Obese

IDP

IDP supports various inference tasks to reason on knowledge

- Model expansion
  - input: an assignment of values
  - output: compute complete assignment, such that KB is satisfied
- Propagation
- Optimization
- Explanation
- . . .

IDP

IDP supports various inference tasks to reason on knowledge

- Model expansion
- Propagation
  - input: an assignment of values
  - output: (in-)equalities that are now implied by the KB
  - E.g. of the form “BMILevel $\neq$ Obese”
- Optimization
- Explanation
- . . .

IDP

IDP supports various inference tasks to reason on knowledge

- Model expansion
- Propagation
- Optimization
  - input: an assignment of values
  - output: compute complete assignment, such that KB is satisfied AND specific term is max-/minimised
- Explanation
- . . .

“Complex” Systems based on DMN

DMN could be used to build knowledge-based AI

Cobot: example

Cobot that assists operator in assembly
→ knowledge can probably be modeled in DMN

However, to actually apply it, the current execution methods are not sufficient!
Can we make IDP-based API to support more methods?

Goals of API

Goals of API:

- Support reasoning on DMN, by using IDP as “execution engine”
- Allow DMN models to be used in more ways
- Support features necessary for generic implementations

3 DMN-IDPy API

DMN-IDPy FEATURES

Features:

- Bottom-to-top calculation
- Reasoning with incomplete information
- Relevance
- Multidirectional Reasoning
- Known Variables
- Variable type and values
- Optimization

Bottom-to-top

The “standard” DMN execution method
→ can be performed by model expansion!

```
spec = DMN('bmi.dmn')
spec.set_value('weight', 74)
spec.set_value('length', 1.79)
spec.set_value('sex', 'Male')
spec.set_value('waist', 90)
```

→

```
>>>> spec.model_expand(1)
Model 1
==========
riskLevel:={->Low}
waist:={->104}
BMILevel:={->Normal}
bmi:={->23.09540900720951}
sex:={->Male}
weight:={->74}
length:={->1.79}
```

Reasoning with Incomplete Information

- We might not know all information
- We might not need all decisions
- E.g. to calculate BMI, we only need weight and height

→ reason with incomplete information by propagating!

```
spec.set_value('Weight', 74)
spec.set_value('Length', 1.79)
spec.propagate()
```

→

```
>>> spec.value_of('BMI')
23.09540900720951
```

Relevance

Sub-goal of API: allow development of “generic” tools

- It should be possible to query inputs necessary for decision
- avoids hard-coding a specific order!
- Not only “direct inputs”: all *upstream* variables

Relevance

```
>>> spec.dependencies_of('BMILevel')
{'BMI': 0, 'Weight': 1, 'Length': 1}

>>> spec.dependencies_of('BMILevel')
 {'BMILevel': 0, 'BMI': 1, 'Weight': 2, 'Length': 2,
  'Sex': 0, 'Waist': 0}
```

Multidirectional Reasoning

- “Standard” DMN approaches always work bottom to top
- But we can do more!
- Using IDP, we can reason in any direction!
- “Unlocks” many more uses for single DMN model

Multidirectional Reasoning

Example: input BMI directly

```
spec.set_value('BMI', 31)
spec.propagate()
```

→

```
>>> spec.value_of('BMILevel')
Obese
```

Multidirectional Reasoning: example

- We have calculated *BMILevel = Overweight*
- Logical next question:
  - "What should weight be for healthy BMI?"
- Easy solution: *guess* until you find correct answer
  - Only easy for small models
  - Never 100% precise

→ Apply DMN model "in reverse"!

Multidirectional Reasoning

Example: using model “in reverse”
→ e.g. calculate weight based on length and BMI:
→ knowledge is already present

```
spec.set_value('BMI', '25')
spec.set_value('Length', 1.79)
spec.propagate()
```

→

```
>>> spec.value_of('Weight')
80.1025
```

Multidirectional Reasoning

Simply “reverse” DMN table?

BMI Level		
U	BMI	BMILevel
1	< 18.5	Underweight
2	$[18.5..25]$	Normal
3	$(25..30]$	Overweight
4	> 30	Obese

BMI Level		
U	BMILevel	BMI
1	Underweight	< 18.5
2	Normal	$[18.5..25]$
3	Overweight	$(25..30]$
4	Obese	> 30

Output can now be range, list of values, ...

(sidenote: this would change meaning of table – this is just an example, not what we actually do)

Multidirectional Reasoning: caveat

In standard execution, every set of inputs has single solution
$\rightarrow$ now, it can be a solution space
$\rightarrow$ two ways of overcoming this:

- Model expansion to generate specific solution
- Optimization (see later)

Known variables

DMN-IDPy allows variable to be assigned a value at any time

- Either by user (*set_value*)
- Or by system, when propagating
- We need to know what variables have a *known* value

```
spec.set_value('Length', 1.79)
spec.set_value('Weight', 79)
spec.propagate()
```

→

```
>>> spec.is_certain('BMI')
True
>>> spec.is_certain('Sex')
False
```

Variable type and values

Variables in DMN are typed, e.g. *String, Int, Float, ...*

```
>>> spec.type_of('Sex')
String
>>> spec.possible_values_of('Sex')
['Male', 'Female']
```

DMN-IDPy allows us to query this information

Optimization

Find solution with highest or lowest value for a term
→ E.g. “What should my target weight be to be healthy?”

```
>>> spec.set_value('Length', 1.79)
>>> spec.set_value('BMILevel', 'Normal')
>>> spec.maximize('Weight')
Model 1
==========
RiskLevel:={->Low}
BMILevel:={->Normal}
BMI:={->25}
Weight:={->80.1025}
Length:={->1.79}
```

4 Application example

Application

Example

Consider an “interactive DMN execution engine”

1. interpret any DMN model
2. ask user what they want to calculate
3. query all necessary inputs
4. output value

$\rightarrow$ . . . primitive chat bot?

Application

→ Using API: 23 lines of Python

```
>>> python bot.py bmi.dmn
Which variable to calculate? ['RiskLevel', 'BMILevel', 'BMI']
> Risk Level
The following variables are still unknown:
['Weight', 'Length']
Value for Weight (Real) unknown.
> 79
Value for Length (Real) unknown.
> 1.79
Calculated value for Risk Level:
Normal
```

Thank you for your attention.

Any questions?

For further questions or discussion:
s.vandevelde@kuleuven.be

For more information on DMN-IDPy:
https://cdmn.readthedocs.io/en/latest/DMN_guide.html