A RuleML - DMN Translator

Adrian Paschke and Simon Könnecke

Freie Universität Berlin, Germany
paschke AT inf.fu-berlin.de, simon.koennecke AT fu-berlin.de

Abstract. Rule markup languages are the vehicle for using rules on the Web and other distributed systems. They allow deploying, executing, publishing and communicating rules in a network. The Object Management Group (OMG) standardized the Decision Modelling and Notation (DMN). Decision tables describe the business logic of a decision making process. We form a bridge by providing a translation framework to translate DMN decision tables to RuleML, a standardized rule interchange language.

1 Introduction

RuleML [1] is an overarching family of Web rule languages [4] which enable standardized machine-interpretation, automated processing and translation between various other rule standards [11], e.g. between RuleML and W3C RIF [2] and rule execution languages [5], as well as theorem provers [16]. RuleML 1.02 spans complementary rule families [1] such as Deliberation RuleML (e.g. used for decision rules), Reaction RuleML (e.g. used for reaction rules and production rules[12,14,9]) and Consumer RuleML (e.g. used for for embedding RuleML into other host languages).

With its semantic style mechanism[2] RuleML supports references to explicitly (pre-)defined *Semantic Profiles* [9] specifying the intended semantics and possible translations of RuleML into other representations.

This paper demonstrates a translator for the OMG Decision Model and Notation 1.1 (OMG DMN[3] into RuleML 1.02. DMN is a standard approach for describing and modeling repeatable decisions within organizations. Our contribution enables representation and semantic interpretation of DMN in RuleML, hence enabling further translations of RuleML DMN representations into other representation and execution languages using the various translators available for RuleML. We will demonstrate this by an implemented translator service[4] from DMN to RuleML. Furthermore, the translator service can translate from RuleML into the executable Prova[5] rule language. RuleML translation

[1] http://ruleml.org
[2] examples can be found in http://de.slideshare.net/swadpasc/paschke-ruleml2014keynote
[3] http://www.omg.org/spec/DMN/
[4] see http://wiki.ruleml.org/index.php/RuleML_Implementations
[5] http://www.prova.ws

enables, e.g. to use the growing number of DMN modelling tools[6] as user interfaces / editors for the various RuleML-supported rule engines which can act as platform-specific execution environments for the DMN models.

The further paper is structured as follows: Section 2 describes the background for using RuleML's Semantic Profiles for OMG DMN. DMN decision tables are introduced in section 3. Section 4 summarizes the DMN profile as basis for the DMN-RuleML translator. Section 5 describes the implementation of the translator service. Section 6 demonstrates the translator and finally section 7 concludes this demo paper.

2 Background

Since Reaction RuleML 1.0 so called *Semantic Profiles* have been introduced in RuleML and further adopted in the overall RuleML 1.02 family. They are used to define the intended semantics for knowledge interpretation (typically a model-theoretic semantics), reasoning (typically entailment regimes and proof-theoretic semantics), or for execution (e.g., operational semantics such as selection and consumption policies and windowing techniques in complex event event processing). Profiles can further detail the syntax and semantics of a RuleML rule base and provide necessary information about the intended semantics for modular RuleML knowledge representations [13] as required for interchange, translation, inference, and verification and validation with test cases [10,7].

Various pre-defined semantic profiles [7] exist, which can be referenced as semantic styles[8] in RuleML, e.g. logical rule semantics [8] such as Herbrand semantics for first order logic and horn logic as well as Tarski semantics for horn logic, as used e.g., in PSOA RuleML[9]. This also includes semantic profiles supporting the semantically safe translation of other rule standards into RuleML, making RuleML an overarching "lingua franca" to interchange rules and integrate with other markup languages. For instance, the First-Order-Deontic-Alethic Semantic Profile (FODAL semantic profile)[10] defines a two-layered modal Kripke semantics for deontic and alethic modal first order formulas which can be used to represent, e.g. business rules as in the OMG standard Semantics of Business Vocabulary and Rules (OMG SBVR)[11], and to map from the SBVR FODAL representations into Modal Reaction RuleML via the profile's translation function. This profile has been used, e.g., as basis for the mapping from OMG SBVR into OASIS LegalRuleML [12] with a Reaction RuleML translator [15].

---

[6] http://openjvm.jvmhost.net/DMNtools/

[7] see e.g. http://wiki.ruleml.org/index.php/Predefined_Semantic_Styles_of_RuleML_1.02

[8] http://wiki.ruleml.org/index.php/Semantic_Styles_of_RuleML_1.02 (@style attribute)

[9] http://ruleml.org/1.02/profiles/HornPSOA-Tarski [3]

[10] http://ruleml.org/1.02/profiles/fodal

[11] http://www.omg.org/spec/SBVR/

[12] https://www.oasis-open.org/committees/legalruleml/

The translator described in this paper is based on a semantic profile for representing and translating OMG DMN (1.1) in RuleML. DMN 1.1 defines the simple friendly enough expression language (S-FEEL) for the purpose of giving standard executable semantics to many kinds of expressions in decision model [6, p. 92].

3 DMN Decision Table

The purpose of DMN is to provide the constructs that are needed to model decisions [6, p. 21]. Decision logic can be express in DMN with a decision table. A decision table is a tabular representation with a name, hit policy, allowed values, default values and 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 [6, p. 72]. Refer to figure 1 for a sample decision table.

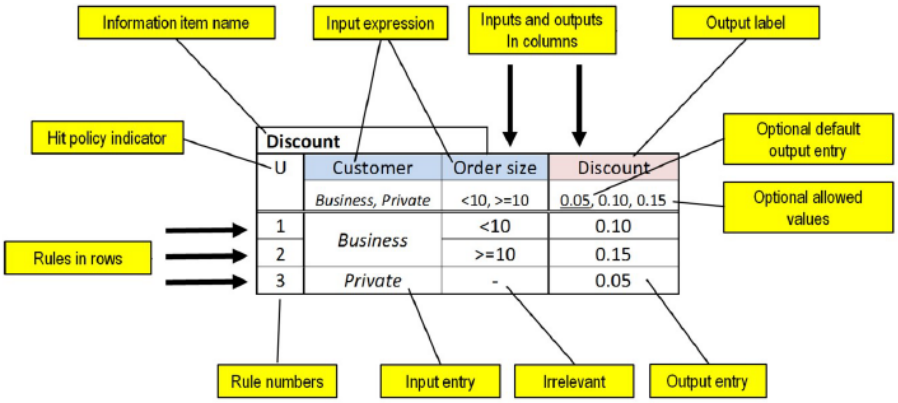

Fig. 1. Horizontal decision table example [6, p. 73]

A decision table normally contains several rules. As a default, rules do not overlap. If rules overlap, meaning that more than one rule may match a given set of input values, the hit policy determines how the output entries will be chosen [6, p. 91]. To list possible hit policies: unique, rule order or collection. Unique is the default hit policy of decision tables and means there are no overlapping rules. Rule order provide all valid output entries ordered by the defined rule sequence. Collection returns randomly all valid output entries. Collection can also come with an aggregation function like min, max, sum or count.

4 Semantic Profile for DMN Representation in RuleML

In this section we describe the basics of the RuleML semantic profile[13] used for the RuleML-DMN translator in the demo. As representation language for DMN we apply a (multi-sorted) logic programming syntax with a horn logic Herbrand semantics, extended by built-in comparators, arithmetical operations, and (data) types. [8]

4.1 Syntax and Semantics

The core alphabet Σ is defined as follows:

1. A signature $S = (\overline{P}, \overline{F}, arity, \overline{c})$.
   - $\overline{P}$ is a finite sequence of predicates symbols $(P_1, \cdots, P_n)$.
   - $\overline{F}$ is a finite sequence of functions symbols $(F_1, \cdots, F_m)$.
   - $\overline{c}$ is a finite sequence of constant symbols.
2. A set of variables V.
3. Connectives / operators: $\neg, \wedge, \vee, \Leftarrow, :=$.

The serialization of the core alphabet Σ in RuleML is stated in table 1.

Table 1. Serialization of Core Logic Programming Syntax in RuleML

Symboles of Σ	**RuleML Serialization**
Variables t_i	<Var> t_i </Var>
Constant t_j	<Ind> t_j </Ind> or <Data> t_j </Data>
$F_i(t_1, \cdots, t_n)$	<Exp> <Func> F_i </Func> $t_1 \cdots t_n$ </Exp>
$\neg P_i$	<Neg>P_i</Neg>
$P_i \wedge P_j$	<And>$P_i P_j$</And>
$P_i \vee P_j$	<Or>$P_i P_j$</Or>
$P_i \Leftarrow P_j$	<Implies><And>P_j</And>P_i</Implies>
$t_i := t_j$	<Equal>t_i t_j</Equal>
$P_i(t_1, \cdots, t_n)$	<Atom><Rel>P_i</Rel> $t_1 \cdots t_n$</Atom>

Note, the equal operation assigns values from t_j to t_i.

[13] Note: a semantic profile is interpreted as a substructure of an expanded profile semantic multi-structure and hence can be further extended by other semantic profiles, i.e. additional expressiveness can be added to the core syntax by combinations with further profiles. For instance, a profile extension with multi-sorted structures, where the members of the universe of discourse is structured into multiple sort domains. To support such profile extensions, we deviate in our signature definitions from the standard definitions in terms of just sets and define them in terms of sequences, where the members can occur multiple-times, but are assigned with different sorts by a sort function.

We further extend the alphabet Σ with built-in comparators $(=, !=, <, \leq, >, \geq)$, arithmetical operations $(+, -, \cdot, /, \text{mod}, \hat{})$ and (data) types.

For the interpretation of this syntax the existing semantic profiles for horn logics with Herbrand semantics[14] can be used as basis.

Further we extend the basic syntax of Σ to represent DMN S-FEEL expressions and interpret them as a logic program. In the following we will exemplify this representation and also show example serializations in RuleML syntax.

Data Types and Built-ins Built-in Operators and Comparators will be serialized as predicates. We use Semantic Web Rule Language built-ins (SWRLB)[15] and XML Schema data types[16] to extend the alphabet Σ.

Data types are referenced in RuleML in `<Data>` elements using XML Schema data types. More general types defined in external type systems can be referenced by the @type attribute and interpreted, e.g. by semantic profiles for sorted logics. Table 2 gives examples for the serialization of data types into RuleML. The function $f_{type}(c_i) \mapsto d$ determined the type, where d is a possible data type. Therefore, the alphabet Σ can differ between types of constant symbols.

Table 2. Translation of S-FEEL data types to RuleML

Data Types	**RuleML serialization**
boolean	`<Data ... xsi:type="xs:boolean">...</Data>`
number	`<Data ... xsi:type="xs:double">...</Data>`
string	`<Data ... xsi:type="xs:string">...</Data>`
date	not yet supported

Operations The SWRLB provide arithmetical and comparison operation we simple extend the alphabet Σ by the operation. The predicate's name is the name of the operation, the first parameter obtains the result of the operation, the last two the parameters donate the values of the operands e. g. the listing 1.1 describe a serialization of an arithmetical operation $Result := 2 + 3$. The comparison operation will be serialized the same way without result variable.

Listing 1.1. Sample serialization of swrlb:add.

```
<Atom>
    <Rel iri="swrlb:add">add</Rel>
    <Var>Result</Var>
    <Data ...>2.0</Data>
    <Data ...>3.0</Data>
</Atom>
```

SWRLB extend the Σ with the arithmetical operation $+, -, \cdot, /, \text{mod}, \hat{}$.

[14] http://ruleml.org/1.02/profiles/Horn-Herbrand

[15] https://www.w3.org/Submission/2004/SUBM-SWRL-20040521/

[16] https://www.w3.org/TR/2004/REC-xmlschema-2-20041028/datatypes.html

4.2 DMN Representation in Horn Logic

Qualified Name Qualified Names are variables in S-FEEL expression. The translation function $\tau_{qn}(Exp, Y, \tau_x)$ will parse all qualified names q_i of Exp and add a predicate *context* with q_i as individual and a unique variable Z_i. Exp^* is Exp where all q_i will be replaced by Z_i. After parsing and replacing the function $\tau_x(Exp^*, Y)$ will be applied, where τ_x is one of the other defined translation functions. The qualified name translation function can be described as $\tau_{qn}(Exp, Y, \tau_x) = context(q_0, Z_0) \wedge .. \wedge context(q_n, Z_n) \wedge \tau_x(Exp^*, Y)$, where n is the number of qualified names in the expression Exp.

Interval The translation function of interval expressions to Σ can be defined as follows:

$$\tau_{interval}(Exp^*, X) = \begin{cases} Exp^* = [a..b] & : a \geq X \wedge X \leq b \\ Exp^* = (a..b) & : a > X \wedge X < b \end{cases}$$

Note, for the reason of simplification we only show two of the nine valid S-FEEL interval mappings. a and b are placeholders for variables or constants symbols.

Literal expression The translation function for S-FEEL literal expression l can be defined as $\tau_{literal}(Exp^*, X) = X := Exp^*$.

Unary Test S-FEEL unary test expression means the first operand is implicitly given. The operation and second operand are explicitly defined in unary test expression for example "< 12". When an S-FEEL unary test is translated to Σ there can be determined the following function:

$$\tau_{unary}(Exp^*, X) = \begin{cases} not(Exp^*) & : \neg(\tau_{unary}(Exp^*)) \\ Exp^* \text{ is a interval} & : \tau_{interval}(Exp^*, X) \\ Exp^* = - & : \text{true} \\ \text{otherwise} & : f_{concat}(X, Exp^*) \end{cases}$$

The variable X is the implicit operand. The function $f_{concat}(X, Exp^*)$ concatenates X and Exp^*.

Decision Table, Output Expression and Rule The attributes of a decision table, output expression or rule will be mapped as a fact to the knowledge base. The first term of fact will be the decision tabel id. The mapping of an decision table appears as: $decisionTable(\text{"}tableId\text{"}, \text{"}tableLabel\text{"}, \text{"}hitPolicy\text{"}, \text{"}aggregation\text{"})$.

Input Expression The input expression has a literal expression l and will be translated as follows: $inputExpression(\text{"}tableId\text{"}, \text{"}ColNo\text{"}, Result) \Leftarrow \tau_{qn}(l, Result, \tau_{literal})$, where $ColNo$ is the column position at the decision table.

Input Entry The input entry S-FEEL expression can describe multiply unary tests separated by commas. For each unary test t_i will be translated as $inputEntry($ "$tableId$", "$ColNo$", "$RuleNo$") $\Leftarrow inputExpression($"$tableId$", "$ColNo$", $Value)$ $\wedge\, \tau_{qn}(t_i, Value, \tau_{unary})$, where $ColNo$ column position of the related input expression and $RuleNo$ is the corresponding rule position.

Output Entry Output entry defines the result value via a literal expression l. The translation can be describe as $outputEntry($"$tableId$", "$label$", $\cdots, Value) \Leftarrow$ $\wedge_{i=0}^{n} inputEntry($"$tableId$", i, "$RuleNo$") $\wedge\, \tau_{qn}(l, Value, \tau_{literal})$, where n is the number of input entries of the related rule, dots at $outputEntry$ predicate are placeholder for the attributes of the output entry and $RoleNo$ the number of the rule order.

5 RuleML DMN Translator Service

The translation service is split in two Java programs, one called rule translation service (RTS) and other called rule translation web service (RTWS). RTS runs by specifying four parameters, input language l_{in}, output language l_{out}, input data d_{in} and output d_{out}. RTS translate always from or to RuleML. The case where l_{in} or l_{out} not RuleML the service will translate l_{in} to RuleML and than to l_{out}. Note, the languages l needed to be listed at RTS. RTS provide syntax mapping between S-FEEL to RuleML, DMN to RuleML and RuleML to Prova. DMN and RuleML as XML based formatted will by parsed JAXB[17]. S-FEEL will parsed by ANTRL[18]. RTWS wrappers RTS to provide an REST API of the service. RTWS based on Java Spark[19] a lightweight web framework to create microservices.

We explain three ways to translate a DMN file "table.dmn" to a RuleML file "table.ruleml". Firstly as command line execution, secondly as Java library call, thirdly as HTTP call.

Command-Line Interface RTS can be invoked as command on command line. We will translate the DMN file to RuleML file as follow: java -jar ruletranslatorservice.jar dmn ruleml table.dmn table.ruleml. The result will printed to file "table.ruleml".

Java Library RTS can be used as library in other projects. The input of a translation can be a File, a InputStream or a String. Here we load the file "table.dmn", mark it with input language "DMN", translate it to "RuleML" and store the result to "table.ruleml" by calling the function translate of RuleTranslationService.

[17] https://jaxb.java.net/
[18] http://www.antlr.org/
[19] http://sparkjava.com/

```
//...
public static void main(String[] args) {
    RuleTranslatorService rts = RuleTranslatorServiceFactory.
        createTranslatorService();

    try {
        rts.translate(new File("table.dmn"), "DMN", "RuleML", new
            FileOutputStream("table.ruleml"));
    } catch (UnknownRulesLanguageException ex) {
        // ...
    } catch (RuleTranslatorException ex) {
        // ...
    }
}
// ...
```

REST Translation For RTWS the translation languages l needs to be provide in the URL call, e.g. the URL of the DMN to RuleML translation service is http://localhost/api/translate/dmn/ruleml/ (We assume that the RTWS runs on localhost). The body of the HTTP post call contains the raw DMN file "table.dmn". Successfully requesting RTWS gives a response with a HTTP status code 200 and RuleML serialization in the body, otherwise an error has occurred.

REST Execution RTWS also provides an execution method, e.g. a request with HTTP post message to the URL http://localhost/api/execute/ with the body:

```
{language: "dmn", parameters: {Customer: "Business", "Order␣Size": 5},
    source: <table.dmn>, query: ["inputEntry(DecisionTable,␣Column,␣Row)
    "]}
```

The service will translate the DMN file to RuleML and from RuleML to Prova source code. Parameters of the body get translated to facts like $context(\text{“}Customer\text{”}, Value) \Leftarrow Value := \text{“}Business\text{”}$ and will be translated to Prova source code. Furthermore, Prova source code will extend by ":- solve(inputEntry(DecisionTable, Column, Row))". The invocation of Prova source code by RTWS will be a response as a JSON object with the result of the query.

6 DMN-RuleML Translation Demo

This section demonstrates how a DMN decision table in figure 1 gets serialized in RuleML. We will examine the decision table "Discount", the column "Customer" (DMN input expression), the row starts with 1 (DMN rule), the cell Business at DMN rule (DMN input entry), column "Discount" (DMN output) and the cell below with value 0.10 (DMN output entry).

The **decision table "Discount"** can be defined in DMN as follows:

```
<decision id="decisionDiscount" name="Discount">
    <decisionTable id="decisionTable" hitPolicy="UNIQUE">
        ... <!-- all input expressions and output expressions -->
        ... <!-- all rules -->
    </decisionTable>
</decision>
```

RTS translates the decision table "Discount" to RuleML as follows:

```
<RuleML xmlns="http://ruleml.org/spec"
        xmlns:dmn="http://www.omg.org/spec/DMN/20151101/dmn.xsd"
        xmlns:swrlb="http://www.w3.org/2003/11/swrlb"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <Assert>
        <formula>
        <Rulebase iri="dmn:DecisionTable">
            ... <!-- signatures -->
            <Atom keyref="decision">
              <Rel>Decision</Rel>
              <Ind>decisionDiscount</Ind> <!-- decision id -->
              <Ind>Discount</Ind> <!-- decision label -->
            </Atom>
            <Atom keyref="decisionTable">
              <Rel>decisionTable</Rel>
              <Ind>decisionTable</Ind> <!-- decision table id -->
              <Ind>UNIQUE</Ind> <!-- Hit Policy -->
              <Ind/> <!-- Aggregation -->
            </Atom>
            ... <!-- all input expressions and output expressions -->
        </Rulebase>
        </formula>
        ... <!-- Rulebases for each rule -->
  </Assert>
</RuleML>
```

We can see that decision and decision table attributes get represented as fact in knowledge base. All input expressions and output definitions get translated to the same rulebase. **Input expression** "Customer" can be defined in DMN as follows:

```
<input id="input1" label="Customer">
  <inputExpression id="inputExpression1" typeRef="string">
    <text>input1</text>
  </inputExpression>
</input>
```

Translation of the input expression is translated to a Rule $inputExpression(\cdots, Value) \Leftarrow context(\text{“}input\text{”}, S) \wedge Value := S$.

```
<Implies>
  <And>
    <Atom>
      <Rel>context</Rel>
      <Ind>input1</Ind>
      <Var>Storage10</Var>
    </Atom>
    <Equal>
      <Var>Value</Var>
      <Var>Storage10</Var>
    </Equal>
  </And>
  <Atom keyref="inputExpression">
    <Rel>inputExpression</Rel>
    <Ind>decisionTable</Ind> <!-- decision table id -->
    <Ind>0</Ind> <!-- column number -->
    <Var>Value</Var>
  </Atom>
</Implies>
```

Output Expression "Discount" has only attributes. Therefore a fact will be added to the rulebase.

```
<output id="output1" label="Discount" name="output1" typeRef="string"/>

<Atom keyref="outputExpression">
  <Rel>outputExpression</Rel>
  <Var>Result</Var>
  <Ind>decisionTable</Ind> <!-- decision table id -->
  <Ind>output1</Ind>  <!-- output expression id -->
  <Ind>output1</Ind>  <!-- output expression name -->
  <Ind>Discount</Ind> <!-- output expression label -->
  <Ind>string</Ind>   <!-- output expression type -->
</Atom>
```

Rule 1 contains all cells in the row. Starting by input entries "Business", < 10 and output entry 0.10. Each rule translated will be translated to new rulebase.

```
<rule id="row-950612891-1">
    <description>Discount for small business customer</description>
    ... <!-- inputEntry and outputEntry ... -->
</rule>
```

All attributes of DMN rule and description will translated to a fact.

```
<Rulebase>
  <oid><Ind>row-950612891-1</Ind></oid>
  <Atom keyref="rule">
    <Rel>rule</Rel>
    <Ind>decisionTable</Ind> <!-- decision table id -->
    <Ind>row-950612891-1</Ind> <!-- rule id -->
    <Ind/> <!-- rule label -->
    <Ind>Discount for small business customer</Ind> <!-- rule
        description -->
  </Atom>
  ... <!-- Here stays InputEntry and OutputEntry serializiation ... -->
</Rulebase>
```

Input entry and **output entry** are always related to a rule. Therefore the translation will be added to the same rulebase as the rule. The input entry has an ID and a S-FEEL expression. The input entry is related to input expression. Input expression provides the value that each input entry compares with the related S-FEEL expression.

```
<inputEntry id="UnaryTests_0c1o054"> <text><![CDATA["Business"]]></text>
    </inputEntry>
```

In this case, the input entry gets compared with the value of input expression "Customer". Therefore, inputExpression predicate retrieves the value of input. This predicate *inputEntry*("*decision*", 0, 0) get only true, when the input is equal to "Business". Note, the S-FEEL serialization is in this case only the atom with relation "equal".

```
<Implies>
  <And>
    <Atom keyref="inputExpression">
      <Rel>inputExpression</Rel>
      <Ind>decisionTable</Ind>
      <Ind>0</Ind>
      <Var>Value</Var>
    </Atom>
    <Atom>
      <Rel iri="swrlb:equal">equal</Rel>
      <Var>Value</Var>
      <Data xsi:type="xs:string">Business</Data>
    </Atom>
```

```
  </And>
  <Atom keyref="inputEntry">
    <Rel>inputEntry</Rel>
    <Ind>decisionTable</Ind> <!-- decision table id -->
    <Ind>0</Ind> <!-- input column number -->
    <Ind>0</Ind> <!-- rule number -->
  </Atom>
</Implies>
```

Output entry describes the output value of the output expression, when all input entries turned true. DMN output entry is only a definition of the output value. Defined in the text tag as S-FEEL expression.

```
<outputEntry id="LiteralExpression_065u3ym"> <text>0.10</text> </
    outputEntry>
```

The output translation is related to the rule 1, therefore customer need to be a business customer and the order size should be below 10. If the two constraints are satisfied then the customer receives a 10% discount; in other words the predicate outputEntry will be true if the constraints are satisfied. The representation of the constrains in RuleML in the body of the rule includes all input entries connected with a logical $\wedge$.

```
<Implies>
  <And>
    ... <!-- inputEntry(decisionTable, 0, 0) and
          inputEntry(decisionTable, 1, 0) -->
    <Equal>
      <Var>Value</Var>
      <Data xsi:type="xs:double">0.1</Data>
    </Equal>
  </And>
  <Atom keyref="outputEntry">
    <Rel>outputEntry</Rel>
    <Ind>decisionTable</Ind> <!-- decision table id -->
    <Ind>output1</Ind> <!-- output expression id -->
    <Ind>0</Ind> <!-- output colum number -->
    <Ind>LiteralExpression_065u3ym</Ind> <!-- output entry id -->
    <Var>Value</Var>
  </Atom>
</Implies>
```

In summary, decision table, output expressions, rules will be serialized as facts. Input expression, input entry, output expression and output entry will serialized as horn clauses. The knowledge base is complete, so that queries can be answered after adding the context predicates. Variables should be defined only once.

7 Summary

In this paper we have demonstrated a translator service for mapping OMG DMN (v. 1.1) into RuleML (v. 1.02). This enables the use of the growing set of modelling tools for OMG DMN as user interfaces together with the rich set of RuleML-supported rule engines as execution environments. Furthermore, it adds another standard to the RuleML interchange framework, as a basis for semantically-safe rule translations by semantically composing the underlying semantic profiles.

Acknowledgement

This work has been partially supported by the “InnoProfile-Transfer Corporate Smart Content" project funded by the German Federal Ministry of Education and Research (BMBF) and the BMBF Innovation Initiative for the New German Länder - Entrepreneurial Regions.

References

1. Athan, T., Boley, H., Paschke, A.: Ruleml 1.02: Deliberation, reaction and consumer families. In: Proceedings of the RuleML 2015 Challenge, the Special Track on Rule-based Recommender Systems for the Web of Data, the Special Industry Track and the RuleML 2015 Doctoral Consortium hosted by the 9th International Web Rule Symposium (RuleML 2015), Berlin, Germany, August 2-5, 2015. (2015), http://ceur-ws.org/Vol-1417/paper6.pdf
2. Boley, H.: RIF ruleml rosetta ring: Round-tripping the dlex subset of datalog ruleml and rif-core. In: Rule Interchange and Applications, International Symposium, RuleML 2009, Las Vegas, Nevada, USA, November 5-7, 2009. Proceedings. pp. 29–42 (2009), http://dx.doi.org/10.1007/978-3-642-04985-9_6
3. Boley, H.: A rif-style semantics for ruleml-integrated positional-slotted, object-applicative rules. In: Rule-Based Reasoning, Programming, and Applications - 5th International Symposium, RuleML 2011 - Europe, Barcelona, Spain, July 19-21, 2011. Proceedings. pp. 194–211 (2011), http://dx.doi.org/10.1007/978-3-642-22546-8_16
4. Boley, H., Paschke, A., Shafiq, M.O.: Ruleml 1.0: The overarching specification of web rules. In: Semantic Web Rules - International Symposium, RuleML 2010, Washington, DC, USA, October 21-23, 2010. Proceedings. pp. 162–178 (2010), http://dx.doi.org/10.1007/978-3-642-16289-3_15
5. Hallmark, G., de Sainte Marie, C., Fabro, M.D.D., Albert, P., Paschke, A.: Please pass the rules: A rule interchange demonstration. In: Rule Representation, Interchange and Reasoning on the Web, International Symposium, RuleML 2008, Orlando, FL, USA, October 30-31, 2008. Proceedings. pp. 227–235 (2008), http://dx.doi.org/10.1007/978-3-540-88808-6_24
6. Object Management Group: Decision model and notation (2014), http://www.omg.org/spec/DMN/1.1/Beta
7. Paschke, A.: The ContractLog Approach Towards Test-driven Verification and Validation of Rule Bases - A Homogeneous Integration of Test Cases and Integrity Constraints into Evolving Logic Programs and Rule Markup Languages (RuleML) . International Journal of Interoperability in Business Information Systems 10 (2005), http://ruleml.org/reaction/docs/ContractLog_VVI.pdf
8. Paschke, A.: Reasoning Web. Semantic Technologies for the Web of Data: 7th International Summer School 2011, Galway, Ireland, August 23-27, 2011, Tutorial Lectures, chap. Rules and Logic Programming for the Web, pp. 326–381. Springer Berlin Heidelberg, Berlin, Heidelberg (2011), http://dx.doi.org/10.1007/978-3-642-23032-5_6
9. Paschke, A.: Reaction ruleml 1.0 for rules, events and actions in semantic complex event processing. In: Rules on the Web. From Theory to Applications - 8th International Symposium, RuleML 2014, Co-located with the 21st European Conference on Artificial Intelligence, ECAI 2014, Prague, Czech Republic, August 18-20, 2014. Proceedings. pp. 1–21 (2014), http://dx.doi.org/10.1007/978-3-319-09870-8_1

10. Paschke, A., Athan, T.: Law test suites for semantically-safe rule interchange. In: Proceedings of the 16th International Workshop on Non-Monotonic Reasoning (NMR'16) (2016), http://nmr2016.cs.uct.ac.za/proceedings_nmr2016_online.pdf
11. Paschke, A., Boley, H.: Rule Markup Languages and Semantic Web Rule Languages. In: Giurca, A., Gasevic, D., Taveter, K. (eds.) Handbook of Research on Emerging Rule-Based Languages and Technologies: Open Solutions and Approaches, pp. 1–24. IGI Publishing (May 2009)
12. Paschke, A., Boley, H.: Rules Capturing Events and Reactivity. In: Giurca, A., Gasevic, D., Taveter, K. (eds.) Handbook of Research on Emerging Rule-Based Languages and Technologies: Open Solutions and Approaches, pp. 215–252. IGI Publishing (May 2009)
13. Paschke, A., Boley, H.: Distributed rule-based agents with rule responder and reaction ruleml 1.0. In: Proceedings of the RuleML 2014 Challenge and the RuleML 2014 Doctoral Consortium hosted by the 8th International Web Rule Symposium, Challenge+DC@RuleML 2014, Prague, Czech Republic, August 18-20, 2014. (2014), http://ceur-ws.org/Vol-1211/paper9.pdf
14. Paschke, A., Boley, H., Zhao, Z., Teymourian, K., Athan, T.: Reaction ruleml 1.0: Standardized semantic reaction rules. In: Rules on the Web: Research and Applications - 6th International Symposium, RuleML 2012, Montpellier, France, August 27-29, 2012. Proceedings. pp. 100–119 (2012), http://dx.doi.org/10.1007/978-3-642-32689-9_9
15. Ramakrishna, S., Paschke, A.: A process for knowledge transformation and knowledge representation of patent law. In: Rules on the Web. From Theory to Applications - 8th International Symposium, RuleML 2014, Co-located with the 21st European Conference on Artificial Intelligence, ECAI 2014, Prague, Czech Republic, August 18-20, 2014. Proceedings. pp. 311–328 (2014), http://dx.doi.org/10.1007/978-3-319-09870-8_23
16. Zou, G., Peter-Paul, R., Boley, H., Riazanov, A.: Psoatransrun: Translating and running PSOA ruleml via the TPTP interchange language for theorem provers. In: Proceedings of the RuleML2012@ECAI Challenge, at the 6th International Symposium on Rules, Montpellier, France, August 27th-29th, 2012 (2012), http://ceur-ws.org/Vol-874/paper6.pdf

All links were last followed on June 17, 2016.