

# A model driven approach for bridging ILOG Rule Language and RIF

Valerio Cosentino<sup>1,2</sup>, Marcos Didonet del Fabro<sup>3</sup>, Adil El Ghali<sup>1,4</sup>

<sup>1</sup> IBM France

<sup>2</sup> AtlanMod, INRIA & EMN, Nantes, France

<sup>3</sup> Universidade Federal do Paran, Brazil

<sup>4</sup> Lutin UserLab, France

valerio.cosentino@fr.ibm.com, marcos.ddf@inf.ufpr.br,  
elghali@lutin-userlab.fr

**Abstract.** Nowadays many companies run their business using Business Rule Management Systems (BRMS), that offer: a clear separation between decision logic and procedural structure; the ability to modify a rulebase set rather than processes and the reusability of rules across applications. All these factors allow a company to quickly react and align its policies to the ever-changing market needs. Despite these advantages, different BRMSs can be found on the market, each of them implementing a proprietary business rule language (ex.: JBoss uses Drools, IBM uses Ilog Rule Language, etc.). Rule Interchange Format (RIF) is a W3C open standard aiming at reducing the heterogeneity among business rule languages, which makes rules less reusable and interchangeable. Our work is focused on providing an implementation based on a Model Driven approach for bridging Ilog Rule Language to RIF.

## 1 Introduction

The field of BRMS is characterized by a number of normative, open source, or proprietary systems and languages (Ilog JRules, JBoss Drools, etc.), allowing the expression of various solutions to business problems at a high abstraction level, but with heterogeneous sets of capabilities and languages. Rule Interchange Format (RIF [1]) and Production Rules Representation (PRR [2]) are two standard proposals respectively developed at the W3C and at the OMG aiming at providing a level of standardization to the domain. An important difference between RIF and PRR is that RIF is a standard for distribution of shared rules at runtime, whereas PRR is a standard for interchanging design-time rules.

This work presents a case study of bridging a business rule language: the Ilog Rule Language (IRL) to a standard format: the Rule Interchange Format (RIF) and vice versa. We show how to make it possible with the help of Model Driven Engineering (MDE [3]).

RIF is the W3C Rule Interchange Format and it is part of the infrastructure composing the semantic web. It is an XML language for expressing rules that

computers can execute. One of the main goals of RIF is to promote a standard format to interchange rules between existing rule languages.

Because of the serious trade offs in the design of rule language, RIF provides multiple versions, called dialects:

- Core: it is the fundamental RIF language. It is designed to be the common subset of most rule engines
- BLD: it adds to Core dialect logic functions, equality in the then-part, and named arguments
- PRD: adds a notion of forward-chaining rules, where a rule fires and then performs some action (adding, updating or retracting some information)

In this work we have implemented transformations for the Core and PRD dialects. An example of a RIF example is shown in (Fig. 1).

```
Prefix(ex <http://example.com/2008/prd1#>)
(* ex:rule_1 *)
Forall ?customer ?purchasesYTD (
  If And( ?customer#ex:Customer
          ?customer[ex:purchasesYTD->?purchasesYTD]
          External(pred:numeric-greater-than(?purchasesYTD 5000)) )
  Then Do( Modify(?customer[ex:status->"Gold"]) ) )
```

Fig. 1. Example of a rule in RIF

IRL is a formal rule language used inside WebSphere ILOG JRules BRMS. It supports the two different levels of a full-fledged BRMS: the technical level targeted at software developers and the business action language targeted at business users. The technical rules of JRules are written in IRL. The complete specification of IRL can be found in the JRules documentation [4] an example of an IRL rule is provided in (Fig. 2).

```
rule rule_1 {
  when{
    customer : customer.Customer(purchaseYTD : purchase_YTD);
    evaluate(purchaseYTD > supportPackIBM.RifUtil.toDecimal("5000"));
  }
  then{
    modify refresh customer {status = "Gold";}
  }
}
```

Fig. 2. Example of a rule in IRL

## 2 Model Driven Engineering

The primary software artifacts of the MDE approach are models, which are considered as first class entities. Every model defines a concrete syntax and conforms to a meta-model (or grammar, or schema), which defines its abstract syntax. One of the most common operations applied on models is transformation, which consists in the creation of a set of target models from a set of source models according to specific rules.

In the work presented in this paper three main tools have been used:

- Ecore allows to handle and create meta-models in Eclipse Modeling Framework (EMF) [5]
- TCS (Textual Concrete Syntax) ([6],[7]) is a bidirectional mapping tool between meta-models and grammars. It is able to perform both text-to-model (injections) and model-to-text (extractions) translations from a single specification. TCS is used to map context-free concrete syntaxes to meta-models.
- ATL (Atlas Transformation Language) [9] is a model transformation language specified as both a meta-model and a textual concrete syntax. It allows developers to produce a number of target models from a set of source models by writing rules that define how to create target models from source model elements.

## 3 Transformations

The RIF2IRL transformation takes as input a RIF file, a BOM file (Business Object Model [4]) and a Dictionary model. The output generates an IRL model and the related IRL file, generated by a TCS parser.

From the first input, using a TCS parser, a model conforming to an implementation of the RIF meta-model is extracted. A BOM file is passed to the transformation to resolve the domain information declared inside the RIF file/model. The related BOM model is extracted out of the BOM file by a TCS parser and it conforms to the BOM meta-model implemented in [10]. A Dictionary is added as input parameter in the transformation in order to provide a translation from the element names declared in the RIF file to the corresponding ones appearing in the IRL file. The transformation has been implemented in ATL [8] and it defines how to convert RIF to IRL.

The IRL2RIF transformation takes as input a IRL file and two BOMs and returns a RIF file. From the first input, a model conforming to the IRL meta-model is extracted by means of a TCS parser. The first BOM contains the information of the domain model, while the second one is used to map IRL functions to the built-in RIF functions for handling dates, numbers, strings, etc. The output is a RIF model, that is translated into a RIF file by a TCS parser.

## 4 The demonstration

The IRL2RIF and RIF2IRL transformations will be first demonstrated using a set of simple examples (Appendix A) that illustrate the main functionalities of

the transformation. We will then show a real world application of the transformation using data from the Arcelor use-case studied in the ONTORULE project [11]. In the context of the Rule Xchanger application depicted in the (Fig. 3), we will focus on the transformation of IRL rule edited using JRules into RIF rules that will be executed using the Tight engine.

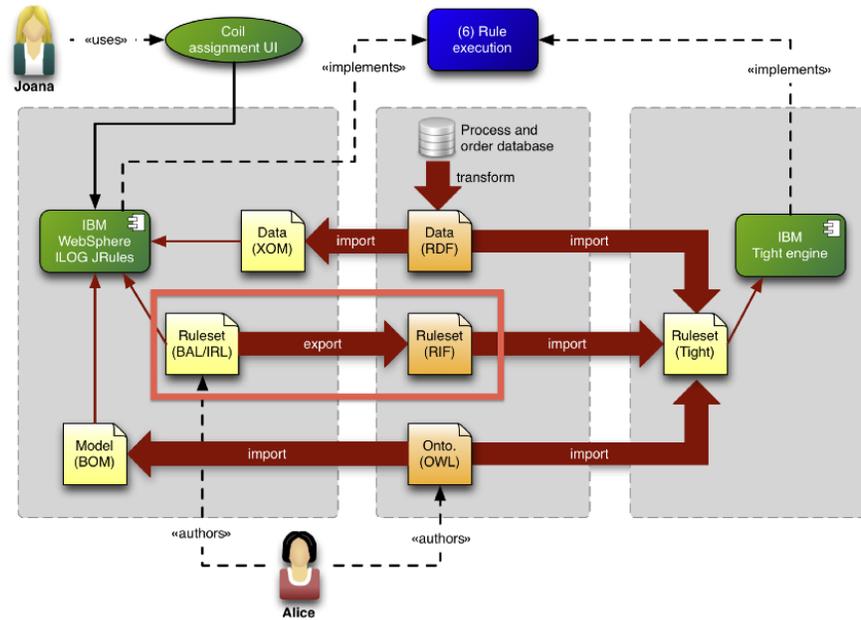


Fig. 3. Rule Xchanger scenario

## 5 Conclusion

The transformation we intend to demonstrate is a key component in the interoperability between two rule languages that has been used in some real world applications such as the Arcelor use-case. It shows that rule application developed using commercial BRMS such as JRules can be exported to other platforms that support the RIF standard.

## References

1. Rule Interchange Format <http://www.w3.org/TR/rif-overview/>
2. Production Rules Representation <http://www.omg.org/spec/PRR/1.0/>
3. Schmidt, D.C. Model-Driven Engineering. IEEE Computer 39 (2), Feb. 2006.

4. JRules documentation <http://pic.dhe.ibm.com/infocenter/dmanager/v7r5/index.jsp>
5. Eclipse Modeling Framework <http://www.eclipse.org/modeling/emf/>
6. Textual Concrete Syntax <http://wiki.eclipse.org/TCS>
7. Jouault F., Bézivin J., Kurtev I., TCS: a DSL for the Specification of Textual Concrete Syntaxes in Model Engineering. In proc. of GPCE'06, Portland, Oregon, USA, pp 249-254
8. Atlas Transformation Language <http://www.eclipse.org/at1>
9. Jouault F, Allilaire A, Bézivin J, Kurtev I. ATL: a Model Transformation Tool. Science of Computer Programming 72(3, Special Issue on Second issue of experimental software and toolkits EST):31-39, 2008
10. Didonet Del Fabro M., Albert P., Bézivin J., Jouault F., Industrial-strength Rule Interoperability using Model Driven Engineering, Technical report 6747, INRIA, Nov. 2008. <http://hal.inria.fr/docs/00/34/40/13/PDF/RR6747.pdf>
11. Gonzalez-Moriyon G., Final steel industry public demonstrators, ONTORULE Deliverable D5.5, Jan. 2012

## A Annex

```

Forall ?x (
  If And (?x#ex:Customer
    ?x[ex:status->"normal"]
    ?x[ex:discount->10] )
  Then Do ( Retract( ?x[ex:discount->10] ) )
  (Assert ?x[ex:discount->0] ) )

rule p0_r0 {
  when{
    X : customer.Customer ();
    evaluate((X.status == "normal" && X.discount == 10));
  }
  then{
    X.discount = 0;
  }
}

****

Forall ?x (
  If And (?x#ex:Customer
    ?x[ex:status->"normal"]
  Then Do ( Modify ( ?x[ex:discount->0] ) ) )

rule p0_r0 {
  when{
    X : customer.Customer ();
    evaluate((X.status == "normal"));
  }
  then{
    modify X {discount = 0;}
  }
}

```

```

BOM

package customer;

public class Customer
{
    public int discount;
    public java.lang.String status;
    public Customer();
}

```

Fig. 4. Example 1

```

(* calculateFib1 *)
Group (
  Forall ?f such that (?f #ex:Fib)
    (IF Exists ?n ?v (And (?f[ex:number->?n]
      ?f[ex:value->?v]
      pred:numeric-equal(?n 1)
      pred:numeric-equal(?v 0)))
    Then Do ( Modify (?f[ex:value]->1))))

package calculateFib1 {
  rule calculateFib1_r0 {
    when{
      f : fib.Fib (n : number; v : value);
      evaluate(n == 1 && v == 0);
    }
    then{
      modify f (value = 1;);
    }
  }
}
*****
(* calculateFib2 *)
Group (
  Forall ?f such that (?f #ex:Fib)
    (IF And (Exists ?n ?v (And (?f[ex:number->?n]
      ?f[ex:value->?v]
      pred:numeric-equal(?n 2)
      pred:numeric-equal(?v 0))))
    Then Do ( Modify (?f[ex:value]->1))))

package calculateFib2 {
  rule calculateFib2_r0 {
    when{
      f : fib.Fib (n : number; v : value);
      evaluate(n == 2 && v == 0);
    }
    then{
      modify f (value = 1;);
    }
  }
}
*****
(* MakeRecursiveGoal *)
Group (
  Forall ?f ?f1 such that (?f #ex:Fib)
    (If And (Exists ?n ?v ?n1 (And (?f[ex:number->?n]
      ?f[ex:value->?v]
      pred:numeric-greater-than(?n 1)
      numeric-equal(?v 0)
      (INeg (Exists ?f1 (And (?f1 #ex:Fib
        ?f1[ex:number->?n1]
        pred:numeric-equal(?n1 pred:numeric-subtract(?n 1)
        )))) )))
    Then Do( Assert(ex:Fib(pred:numeric-subtract(?n 1))))))

package MakeRecursiveGoal {
  rule MakeRecursiveGoal_r0 {
    when{
      f1 : fib.Fib (n1 : number);
      f : fib.Fib (v : value; n : number);
      evaluate(n > 1 && v == 0 && !(n1 == n - 1));
    }
    then{
      insert(new fib.Fib (n - 1));
    }
  }
}
*****
(* computeValue *)
Group (
  Forall ?f ?f1 ?f2 such that (?f #ex:Fib)
    (If And (Exists ?n ?v (And (?f[ex:number->?n]
      ?f[ex:value->?v]
      pred:numeric-greater-than(?n 2)
      pred:numeric-equal(?v 0)
      ?f1 #ex:Fib
      ?f1[ex:number->?n1]
      ?f1[ex:value->?v1]
      pred:numeric-equal(?n1 pred:numeric-subtract(?n 1))
      pred:numeric-not-equal(?v1 0)
      ?f2 #ex:Fib
      ?f2[ex:number->?n2]
      ?f2[ex:value->?v2]
      pred:numeric-equal(?n2 pred:numeric-subtract(?n 2))
      pred:numeric-not-equal(?v2 0))))
    Then Do ( Modify (?f[ex:value]->pred:numeric-add(?v1 ?v2))))

package computeValue {
  rule computeValue_r0 {
    when{
      f2 : fib.Fib(n2 : number; v2 : value);
      f1 : fib.Fib(n1 : number; v1 : value);
      f : fib.Fib(v : value; n : number);
      evaluate(n > 2 && v == 0 && n1 == n - 1 && v1 != 0 && n2 == n - 2 && v2 != 0);
    }
    then{
      modify refresh f {value = v1 + v2;};
    }
  }
}
*****
(* setResult *)
Group (
  Forall ?f such that (?f #ex:Fib)
    (IF And (Exists ?n ?v (And (?f[ex:number->?n]
      ?f[ex:value->?v]
      pred:numeric-not-equal(?v 0))))
    Then Do (pred:print(pred:concat("Fib" ?n "=" ?v))))

package setResult {
  rule setResult_r0 {
    when{
      f : fib.Fib(v : value; n : number);
      evaluate(v != 0);
    }
    then{
      out.println(supportPackIBM.RifUtil.concat("Fib", n, "=", v));
    }
  }
}

```

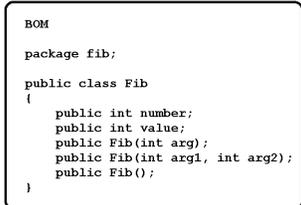


Fig. 5. Example 2

```

(* startSort *)
Group (
  Forall ?c
    (If Exists ?Id (And (?c #ex:Control
                        ?c[ex:id->?Id]
                        pred:numeric-equal(?Id 0)))
      Then Do ( Modify (?c[ex:id->1])))

package startSort {
  rule startSort_r0 {
    when{
      c : sort.Control(Id : id);
      evaluate(Id == 0);
    }
    then{
      modify refresh c {id = 1;}}
}
*****
(* startDisplay *)
Group (
  Forall ?c
    (If Exists ?Id (And (?c #ex:Control
                        ?c[ex:id->?Id]
                        pred:numeric-equal(?Id 1)))
      Then Do ( Modify (?c[ex:id->2])))

package startDisplay {
  rule startDisplay_r0 {
    when{
      c : sort.Control(Id : id);
      evaluate(Id == 1);
    }
    then{
      modify refresh c {id = 2;}}
}
*****
(* switchPosition *)
Group (
  Forall ?first ?second ?c
    (If And (Exists ?Id ?p1 ?p2 ?v1 ?v2 (And (?c #ex:Control
                                             ?c[ex:id->?Id]
                                             pred:numeric-equal(?Id 2)
                                             ?first #ex:Element
                                             ?first[ex:position->?p1]
                                             ?first[ex:value->?v1]
                                             ?second #ex:Element
                                             ?second[ex:position->?p2]
                                             ?second[ex:value->?v2]
                                             pred:numeric-greater-than(?v2 ?v1)
                                             pred:numeric-less-than(?p2 ?p1))))
      Then Do (Modify (?first[ex:position->?p2])
              Modify (?second[ex:position->?p1])))

package switchPosition {
  rule switchPosition_r0 {
    when{
      second : sort.Element(v2 : value; p2 : position);
      first : sort.Element(p1 : position; v1 : value);
      c : sort.Control(Id : id);
      evaluate(Id == 2 && v2 > v1 && p2 < p1);
    }
    then{
      modify refresh first {position = p2;}
      modify refresh second {position = p1;}}
}
*****
(* incrementPosition *)
Group (
  Forall ?e1 ?e2 ?c
    (If And (Exists ?Id ?p1 ?p2 ?v1 ?v2 (And (?c #ex:Control
                                             ?c[ex:id->?Id]
                                             pred:numeric-equal(?Id 1)
                                             ?e1 #ex:Element
                                             ?e1[ex:position->?p1]
                                             ?e1[ex:value->?v1]
                                             ?e2 #ex:Element
                                             ?e2[ex:position->?p2]
                                             ?e2[ex:value->?v2]
                                             pred:numeric-greater-than(?v2 ?v1)
                                             pred:numeric-equal(?p2 ?p1))))
      Then Do (Modify (?e2[ex:position->pred:numeric-add(?p1 1)])))

package incrementPosition {
  rule incrementPosition_r0 {
    when{
      e2 : sort.Element(v2 : value; p2 : position);
      e1 : sort.Element(v1 : value; p1 : position);
      c : sort.Control(Id : id);
      evaluate(Id == 1 && v2 > v1 && p2 == p1);
    }
    then{
      modify refresh e2 {position = p1 + 1;}}
}
*****
(* displayElement *)
Group (
  Forall ?c ?e
    (If And (Exists ?Id (And (?c #ex:Control
                              ?c[ex:id->?Id]
                              pred:numeric-equal(?Id 2)
                              ?e #ex:Element
                              ?e[ex:value]->?v
                              ?p ?e[ex:position]->?p
                              pred:print(pred:concat("value" ?v "is at position" ?p))))
      Then Do ( ?v ?e[ex:value]->?v
              ?p ?e[ex:position]->?p
              pred:print(pred:concat("value" ?v "is at position" ?p))))

package displayElement {
  rule displayElement_r0 {
    when{
      e : sort.Element();
      c : sort.Control(Id : id);
      evaluate(Id == 2);
    }
  }
}

```

```

BOM
package sort;

public class Control
{
  public int id;
  public Control(int arg);
  public Control();
}

public class Element
{
  public int position;
  public int value;
  public Element(int arg1, int arg2);
  public Element();
}

```