Industrial-strength Rule Interoperability using Model Driven Engineering

Marcos Didonet Del Fabro* - Patrick Albert* - Jean Bézivin Frédéric Jouault****

**ILOG, an IBM Company*
9, rue de Verdun
94253 Gentilly Cedex
{mddfabro,albert}@ilog.fr

***AtlanMod (INRIA & EMN)*
4 rue Alfred Kastler
44307 Nantes Cedex 3
{jean.bezivin, frederic.jouault}@inria.fr

ABSTRACT. *Business Rule Management Systems (BRMS) aim at enabling business users automating their business policies. There are several BRMS supporting different languages and capabilities, and almost no available interoperability facility. So far, migration typically follows a manual procedure. In this paper, we present a generic approach based on Model Driven Engineering (MDE) techniques for bridging Business Rules languages; the solution has been fully implemented and tested on different industrial-strength BRMS. The development of such bridges presents many challenges, such as parsing sophisticated languages with different abstraction levels and supporting a large number of artefacts as input and as output. We describe the overall MDE architecture and components as well the as the issues we encountered. In addition, we present the lessons we learned on applying MDE techniques to an industrial project.*

KEY WORDS: *model transformations, business rules, BRMS, interoperability.*

1. Introduction

Business rules are mostly an evolution of production rules (Brownston *et al.* 1985) – an established A.I. technique – used to enable business users to automate policies. The rules are written in Domain Specific Declarative Languages as close as possible to the application domains. The technical details of the rules and application are hidden by a so-called “business-level” layer, enabling the policies owners to create and manage the rules by themselves, with little to no support from software developers – the IT bottleneck is skipped. Being largely independent for IT, the business users gain agility, while a set of tools and process support the rules lifecycle controlling the evolution of the application.

BRMS, or Business Rule Management Systems (Owen 2004), are the products enabling such behaviour. They are made of a number of tools supporting various activities such as rule authoring, validation, documentation, versioning and execution. Well-formalized processes guarantee that new or modified rules are well managed and have little to no risk of breaking the application logic.

A number of BRMS products compete on the market: ILOG JRules (JRules 2008), JBoss Drools (Drool 2008), Yasu QuickRules (Yasu 2008), and though they share common roots, all these systems have heterogeneous set of capabilities and specific languages. Consequently, it is hard to migrate from one system to another.

Rule Interchange Format (RIF) (RIF 2008) and Production Rules Representation (PRR) (PRR 2007) are two standard proposals respectively developed at the W3C and at the Object Management Group (OMG 2008) aiming at providing a level of standardization to the domain. But these necessary initiatives are either not ready or only covering a share of the BRMS languages, thus the interoperability problem is still a problem and will remain for a moment.

In this paper, we present a rule interoperability solution based on Model Driven Engineering (MDE) technologies. Our solution allows creating bridges between rule languages from different Business Rules Management products; it is currently applied to bridging ILOG JRules and JBoss Drools.

We define two bridges: a (reasonably) simple one from DRL – Drools Rule Language (Drools 2007) – to IRL – ILOG Rule Language (JRules 2008), and a much more complex bridge that takes IRL rules as input and generates Business Rules written in the ILOG “Business Action Language” (BAL) (ILOG Language 2008). Such bridges are a composition of several transformations.

Since the BAL contains expressions similar to natural-language, the bridge requires complementary models about the ontology, the terminology and the type of the expressions. The development of such bridges raises several challenges: mapping existing industrial languages with different abstraction levels and complexity; managing the presence of several models as input and as output; and eventually the packaged production of executable and ready-to-use projects.

In both cases, the resulting rules are successfully executed using the same objects sharing regular POJO (Plain Old Java Objects) definitions in Drools and JRules.

The major contributions of this paper are the following. We show how we applied a complex MDE architecture involving a large number of input or output models and transformations - no less than twenty - for developing bridges between two industrial BRMS. Then, we identify some hard issues when developing such bridges, and we explain the adopted solutions. We focus on the specification of the transformations and on the coordination of their execution. This scheme is based on the capture of the abstract syntax by metamodels, on the systematic use of chains of model transformations and on the mappings between concrete and abstract syntaxes.

This paper is organized as follows. Section 2 presents the general architecture of the components of two BRMS. Section 3 presents our rule interoperability solution.

Section 4 describes the applications and the lessons learned. Section 5 presents the related work. Section 6 concludes.

2. General architecture

A BRMS is composed of several components supporting the definition, management, and execution of business rules. We present business rules and the base concepts of BRMS. We then zoom into the two Business Rules Languages studied and the artifacts involved.

2.1. Rules

Rules are more and more used to help business organizations automating their policies, providing properties such as reliability, consistency, traceability and scalability. Rule languages are mostly made of *If-Then* statements (as shown in Figure 1) involving predicates and actions about sets of business objects. Rules variables are bound to objects of a certain type; when the condition part recognizes that a set of objects satisfies the condition predicates, the action part is triggered.

```
rule "approve"
if
  ?C:Customer.status = “Gold”
Then
  ?C.applyDiscount(20%)
end
```

Figure 1. *A simple Production Rule*

2.2. Business rules

Compared to earlier rules languages such as the seminal OPS5 (Forgy 1981) intended to be used by software developers or knowledge engineers, the business rules approach is an attempt to bring the power of rules programming to business users willing to automate business policies. A trained business is able to evolve the rules implementing its policies without having to ask for IT support.

The main difference is that the rules are expressed in a language close to natural language that can be understood and managed by “business analysts”. These Business-level languages are typically compiled into lower-level technical languages; the simple ‘production rules’ semantics remains unchanged.

The “Business Layer” is composed of additional models supporting the definition of the rules (see Figure 2).

- A Business Object Model (BOM) defines the classes representing the objects of the application domain. A BOM is in fact a simplified ontology defining classes with their associated attributes, predicates and actions.

- A Vocabulary model (VOC) defines the words or phrases used to reference the BOM entities.

- A Business Action Language (BAL) Rule, close to natural language, is typically an If-Then statement that composes the elements of the VOC to build user understandable rules, such as "**If** the age of a human is greater than 18 **Then** the human is major". This rule relates the attributes "age" – a Positive Integer – and Boolean attribute "major" of the class "Human".

The business object model and vocabulary are defined in an early stage of the project, and their projection on an executable layer – the Execution Object Model (XOM) and technical rule language – is implemented by a group of IT specialists. The technical-level layer is used to execute the business rules on the application data within the application architecture.

Figure 2 illustrates how the BOM is implemented by the XOM (typically a set of Java or C# classes), and how the BAL is compiled to the Technical Rules language. The technical rules are applied to the XOM objects.

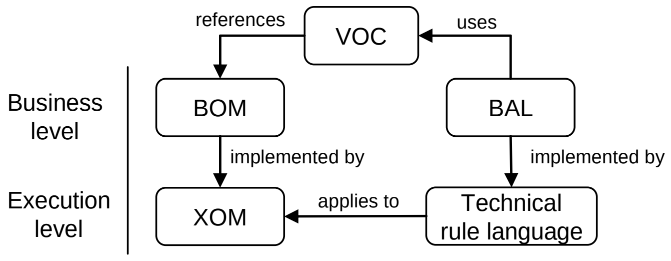

Figure. 2. *Different layers of production rules*

2.3. Business Rule Management Systems

A Business Rule Management System is composed of a large set of components dedicated to enable Business Users managing the whole life cycle of the business rules. We present two BRMS in the following sections, focusing on the rule languages supported by these systems.

2.4 Drools

Drools is an open-source BRMS part of the JBoss foundation. Its rules language is called DRL, for Drools Rule Language (Drools 2003). Though Drools has recently introduced some sort of macros for defining business friendly rules, we have focused our work on the technical rules language, as illustrated in Figure 3.

```
rule "approve"
when
   not Rejection()
   $policy : Policy( approved == false )
then
   System.out.println("approve: " + $policy.getPrice());
   $policy.setApproved(true);
end
```

Figure 3. *Example of rule in DRL*

2.5. ILOG JRules

JRules is the product developed and marketed by ILOG. 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.

2.5.1. Technical rules

The technical rules written in IRL – see Figure 4 – are directly executed by the rule engine. The complete specification of IRL might be found at (IRL 2008).

Consider the rule illustrated in Figure 4. The rule has the same semantics as the rule from Figure 3, and the rules are quite similar.

```
rule approve {
when {
  not Rejection();
  ?policy : Policy(approved == false);}
then {
  System.out.println("approve: " + ?policy.getPrice());
  ?policy.setApproved(true);
}
```

Figure 4. *Example of rule in IRL*

2.5.2. Business Action Language

The Business Action Language (BAL) hides implementation details, allowing business analyst concentrating on the business logic. The equivalent rule for the previous examples is shown in Figure 7.

```
definitions
  set 'policy' to a policy;
if
  there is no rejection and 'policy' is not approved
then
  print "approve: " + the insurance price of 'policy';
  make it true that 'policy' is approved;
```

Figure 5. *Example of a rule in BAL*

As we can see, as a BAL rule is composed of natural language fragments; it is close enough to natural language that any business analyst can read and modify it. The specification of BAL is found at (ILOG Rule Language 2008).

2.5.2.1. Business Object Model

The Business Object Model (BOM) contains all the concepts manipulated by the business rules; it describes the ontology of the application domain. It is mapped to a so-called execution object model (XOM) that will support the actual application data. In the simplest cases, the BOM can be automatically extracted from the classes definitions defined in Java, C# or XML-Schema. For instance, if the execution objects are Java objects, the business model will define one BOM *class* per Java class. We illustrate in Figure 6 the BOM model of the *Policy* concept.

```
public class Policy {
    public boolean approved;
    public double basePrice;
    public int discountPercent;
    public double insurancePrice;
    public Policy () ;
}
```

Figure 6. *BOM for the policy business object*

2.5.2.2. Vocabulary

The vocabulary (VOC) describes the mapping between the business concepts and the terminology used to write BAL rules. It defines a controlled vocabulary that “closes” the rules syntax on a fixed set of allowed words and fragments.

For instance, the attribute *insurancePrice* is mapped to “*insurance price of {this}*”. The *{this}* token indicates the calling object. An excerpt of the vocabulary of policy is shown in Figure 7. The verbalization of class *Policy* is *policy*.

This closed vocabulary is used by the business rules' editor to propose the possible inputs while a business user creates or modifies a rule.

```
Policy#concept.label = policy
Policy.approved#phrase.action =
      make it {approved} that {this} is approved
Policy.approved#phrase.navigation= {this} is approved
Policy.insurancePrice#phrase.action=
      set the insurance price of {this} to {insurance price}
Policy.insurancePrice#phrase.navigation= {insurance price} of {this}
```

Figure 7. *Vocabulary with the terminology of the Policy class*

2.5.2.3. B2X and project files

The "Business To eXecution" model (B2X) describes how the logical elements represented in the BOM are actually mapped to physical data structures. Although generic BOM mappings are defined towards Java, C# and XML, an application developer might add new B2X constructs to specify the way new BOM elements that have no corresponding XOM construct are implemented with the target programming language.

The B2X illustrated in Figure 8 contains a mapping for a virtual BOM member *InsertAction.* When the corresponding verbalization is used in a BAL rule, the engine calls the expression of the mapping (*insert (object)*).

```
     <method>
       <name>InsertAction</name>
       <parameter type = "java.lang.Object"/>
       <body language = "irl"><![CDATA[insert (object);]]> </body>
    </method>
```

Figure 8. *B2X mapping*

Eventually, in order to produce a ready to use ILOG JRules application, a JRules project has to be created. The *.ruleproject* (RP) file specifies the project parameters, the folder where the rules, the BOM and the vocabulary are stored, a URL, and the output folder. The BAL rules are wrapped in XML files (*.brl* files) that contain the rule properties, URL, folder info and the code of the rules.

3. Rule Interoperability

We present our solution that applies MDE techniques to achieve interoperability across BRMS. As already stated, the central concept in BRMS is the rule language. For that reason, the main goal is to transform a set of rules of a source BRMS, into an executable set of rules (and associated files) in a target BRMS. The interoperability between BRMS is intended to ease the comparison of rule engines, and to automate the migration from one product to another.

In our scenario, we have two complementary objectives: translating a set of rules available in DRL into IRL (interoperability between technical languages) and translating IRL rules into BAL rules (interoperability between a technical and a business language). We first produce IRL files from DRL files, and then we produce the BAL-level project from the IRL files. In order to have a ready-to-run JRules application, it is also necessary to produce the vocabulary, BOM and project files.

We present below the general architecture. Then, we describe the different steps of the bridge, the challenges encountered, and how we handle them.

3.1. General architecture

The architecture follows the usual MDE pattern: **inject**, **transform, extract**, and relies on four core MDE practices and technologies: Domain Specific Languages (DSLs) (Kurtev *et al.* 2006), metamodeling (Kurtev *et al.* 2006), model transformations (Jouault *et al. 2005*) and projections across technical spaces (i.e., injections and extractions) (Kurteve *et al.* 2002). DSLs and metamodeling are used to design adapted rule languages; model transformations are used to produce a set of output models from a set of input models; injections allows translating the input rules (and related artifacts) into models (e.g., text-to-model translation); and extractions allows translating models into the output artifacts (e.g., model-to-text translation). Though the current implementation only supports Drools and JRules, the architecture is modularized in order to easily accept other rules languages.

We rely on the tools developed at the AtlanMod Inria project:

- TCS (Textual Concrete Syntax) (Jouault *et al.* 2006). TCS is bidirectional, i.e., it provides text-to-model and model-to-text translations.

- KM3 (Jouault *et al.* 2006), a simple metamodel specification language, is used to define the metamodels.

- ATL (Jouault *et al.* 2006) is used to implement the many model transformations. ATL has a development environment integrated into Eclipse (EMP 2008), such as textual editor, and debugger.

- AM3 (Allilaire *et al.* 2006), the model management scripting environment, enabling us to execute chains of “load, transform and save” units.

3.2. DRL to IRL

This bridge produces an ILOG JRules project including rules written in the IRL language from a set of files written in DRL.

3.2.1. KM3 and TCS definition

The KM3 metamodels are based on the online specification of Drools and JRules (respectively expressed as XSD schemas and EBNF notation). For the sake of maintainability, we could have designed more clever KM3 models, we have chosen to create model elements sticking to the specifications.

Still, in the Drools case, we had to refactor the KM3 to take into account the nested nature of XSD schemas. For instance, <*choice*> elements in the XML schemas have been transposed into trees in KM3. In both cases The TCS models directly map the KM3 metamodels to the syntax.

3.2.2. Transformation specification

A transformation rule, expressed in ATL, has the following signature: `create OUT : IRL from IN : DRL;` It takes one DRL model as input and produces one IRL model as output. The transformation is relatively simple, since both languages are similar. There are a few DRL expressions that are not natively supported by IRL (the opposite is also true, but in this paper we concentrate on one direction). For example, DRL conditions might be connected by an *OR* predicate – such as in "`not Rejection() OR Policy(approved == false)`" – which is not possible in IRL. To transpose the "OR" semantics, we first run an endogenous refactoring transformation that transforms every *OR* predicate into two rules, with the same conclusions.

3.2.3 Chaining and parameterization

A model management script chains the different operations, executing the whole bridge in one automated step; it executes the following sequence of actions:

1. **Injection:** DRL textual syntax into DRL models.
2. **Refactoring:** DRL models are transformed into new DRL models
3. **Transformation:** the DRL models are transformed into the IRL models.
4. **Extraction:** IRL models are projected in the IRL textual syntax.

3.3. IRL to BAL

The central transformation produces an ILOG JRules Project including rules written in the BAL syntax. As this bridge is far more complex than the former – it includes additional input and output models, such as the BOM, the VOC and the B2X – it has been designed as a composition of a large number of transformations.

3.3.1. KM3 and TCS definition

We have created five KM3 metamodels, for the BOM, VOC, BAL, B2X and project files, and three TCS schemas, because only the BOM, VOC and BAL have textual syntax. Though they lack a well documented formal specification, the BOM and VOC models are relatively simple; we have been able to create the metamodels by testing successive alternatives in the editor provided by JRules.

As BAL is not context-free, and TCS only supports context-free expressions, the creation of the TCS for BAL has been quite challenging. A BAL rules is essentially a composition of small fragments of free text. It is thus possible in BAL to write literally any kind of expression with very few restrictions. In order to use TCS, we support a large subset of BAL and its associated KM3 metamodel. Consider the two expressions below:

```
set '<variable>' to <value>
set the insurance price to 300,00
```

The first expression defines a variable, while the second assigns a value to a field. To make thinks more complex, any kind of terminology is supported by JRules, such as *set's* and *to's* within sets, commas, *ifs*, etc.

Another challenge is the specification of operators. The operators are textual operators, such as "*is bigger than*", "*is smaller than*", and they may be composed as in: "*is bigger than* X *and smaller than* Y". To be able to parse these multi-word operators, we have created a class for each: *IsBiggerThan*, *IsSmallerThan*, etc. Though this approach proved to be practical, is has required writing much more code on the IRL to BAL transformation, and requires more modifications to create a TCS for another language (e.g., French).

3.3.2. BOM, VOC, B2X and RP refactoring

The JRules environment can automatically generate the initial BOM and the default vocabulary from the Java or C# XOM classes. However, as IRL provides more constructs than BAL, as for example *insert* or *retract* actions and *for*, *while*, *if* and *try-catch* statements, we had to add specific vocabulary to express them at the BAL level, and specific translations from BOM to XOM.

The solution has been to generate B2X mappings as needed for all non-supported expressions occurring in the rules. This has translated into the definition of three elementary transformations:

- **Extending** the BOM to add virtual methods, for instance an *InsertAction* method;

- **Extending** the VOC to add verbalizations, for instance "*Policy.InsertAction = add policy*;"

- **Extending** the initial B2X mapping to add the new mapping expressions.

3.3.3. Transformation specification

The IRL2BAL transformation has different issues. The first one is to find the correct verbalization (terminology) for every IRL expression. The transformation must search the corresponding expressions in the BOM and in the vocabulary. We create a set of ATL helpers that navigate through the IRL expressions from left to right, and produce an expression from right to left. This means, for example, that *policyInsurance.cost* is translated as "*the cost of the insurance policy*". The vocabulary is navigated recursively to find the correct verbalizations.

BAL has also pre-defined patterns that are not in the BOM. Operations over *Strings* are the most common patterns. For instance, expressions such as *str1.indexOf("str") != -1* must be transformed into (*str1 contains "str"*). We had to do an inventory of all the patterns supported by the BAL and to implement them in the transformation rules.

This transformation has different complex issues, and it deserves itself a separated study. However, in this paper we focus on the overall complexity of the transformations and metamodels.

3.3.4. Project files

A rule project in JRules has additional files storing project-relative properties. These files are not at the core of the bridge, but they are necessary to obtain a ready-to-use project. For instance, we define a helper transformation that wraps the BAL rules into an XML model (BRL), which is later extracted into an XML file that is directly used by JRules.

4. Application

As shown in Figure 9, the solution has reached a level of complexity in terms of the number and types of operations performed. It may be considered as a significant deployment of an MDE application in a real environment to solve a practical problem. As a side consideration, we can observe here the need to get tools to handle complex networks of transformations. The inventory of the transformations required, as well as their correct chaining is one of the central challenges for producing such a bridge.

The bridge described executes *twenty four* operations, some of which are complex in nature and involve a large number of metamodel classes, grammars or transformation rules. They can be categorized as follows:

- **Generation:** of BOM and VOC by JRules from the Java Classes definitions.

- **Injection**: input files for DRL, BOM, VOC and optionally IRL.
- **Transformations**:
  - **Refactoring**: endogenous model transformation for DRL and IRL
  - **Augmentation:** growing endogenous model transformation for BOM, VOC, B2X and Rule Project.
  - **Traduction**: exogenous model transformation for DRL2IRL, IRL2BAL and BAL2BRL.
  - **XML-ification:** a special kind of exogenous model transformation toward XML, for B2X, BRL and Rule Project.
- **Extraction**: toward ad-hoc files for IRL, BOM, VOC and BAL, and toward XML for B2X, BRL and Rule Project.

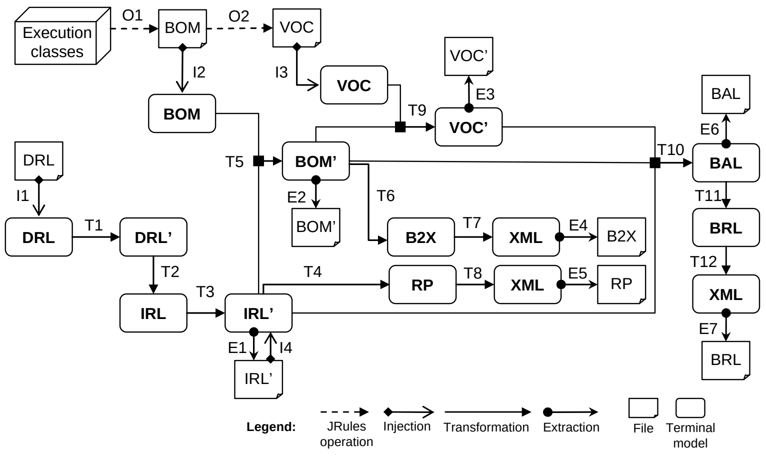

Figure 9. *DRL → IRL → BAL complete process*

Table 1 below provides some quantitative information about some of the metamodels, TCS models and transformations.

	TCS size	**KM3 size**	**Dev. time**
BOM	21 templates	21 classes	3 days
VOC	16 templates	18 classes	2 days
DRL	98 templates	102 classes	1.5 weeks
IRL	179 templates	185 classes	1.5 weeks
BAL	191 templates	194 classes	4 weeks

Table 1. *Size of KM3 and TCS of the main DSLs*

Table 2 below provides the size of the two main transformations, one refactoring and two augmentation transformations.

	Transformation size	**Dev. time**
DRL2IRL	1005 lines, 61 rules	2 weeks
DRLRefactor	1412 lines, 81 rules	
IRL2BAL	2396 lines, 90 rules	3 weeks
BOM augmentation	677 lines, 32 rules	2 weeks
VOC augmentation	778 lines, 31 rules	1 week

Table 2. *Size of the transformations*

Both bridges have been tested on the two standard business rules benchmarks: *manners* and *waltz* (Academic Benchmarks 2007), on an “insurance claim” demonstration (Drools Examples 2008), and on the translation of a banking application with more than 100 rules.

The rules are executed over the Java objects provided with the examples. We have run the *manners* benchmark with settings for 16, 32, 64, 128, 256 and 512 objects. In all cases, the results are the same, and the same number of rules is fired in the three implementations: DRL, IRL and BAL. The correctness of the transformations are empirically validated (by extensive testing), we did not apply any formal validation technique. The generated rules correctly preserve the operational semantics of their original models.

4.1. Lessons learned

Following this experimentation, we conclude that MDE tools have reached a reasonable level of maturity allowing their use in the context of industrial projects. They show good performances, little bugs, and are available as Eclipse open source projects. However, installing and using these tools may be rather complicated for a non specialist. The major advantage of MDE is the possibility to concentrate on the problem specification and apply a declarative and modular approach using a small set of principles and tools.

One of the major conclusions is that the correct chaining of transformations is a key factor of success, because the bridge must be easy to configure and to run, acting over several input and output models. A transformation cannot be fired until all its parameters are loaded. Consequently, the dependency relations shown in the schema of Figure 9 must be respected during the execution.

Coupling the scripts and of the parameterization of the execution has proved very important. However, an integrated repository, using graphical interfaces integrated within Eclipse would help a lot. We can observe here the need to get tools to handle complex networks of transformations. This is a field where MDE has still to provide new solutions.

KM3 enables the definition of the metamodels in a practical way. On the syntax side, thought we could eventually reach our objective, we found more difficulties with TCS, because of its context-free limitation. Such a limitation has introduced an unwanted level of complexity in our models. This means we have to do compromises between supporting all the expressions and the complexity of the development.

Declarative model transformations are a concise and practical approach. The ATL transformations are particularly elegant in the DRL to IRL bridge, because the languages have similar semantics. However, the metamodels have specific details that have required considerable development time.

As far as we know, the bridge requiring such a large number of transformation expressions have not been deeply explored before. Some of them -- such as expression patterns – are generic enough to be used in other solutions, with just minor adaptations. However, even by using declarative rules, the transformations produced are still large. Current solutions still need to improve their modularization capabilities. For instance, we would like to have transformations and libraries separated by packages, and with easy ways to navigate through the transformation code.

5. Related work

There are several BRMS in the market, such as ILOG JRules, Drools, Haley, Yasu QuickRules, etc. We are not aware of any industrial solution (at least publicly available) of bridges between different BRMS, and especially using advanced MDE techniques. A first work before this one has been the translation of PRR into IRL (Abouzhara *et al.* 2007). Though our project has a much greater size and complexity, the former has provided us with inspiration and reasonable hope for convergence.

Model transformations (and related concepts) are widely used in solutions of different degree of complexity. Several uses cases of model transformations may be found at (ATL Use cases 2008). There are several other tool suites similar to AMMA (e.g., Epsilon (Kolovos 2008) or oAW (oAW 2009)). The closest one is oAW, which is also available on the Eclipse.org platform. AMMA and oAW share the same model-centric vision and only differ on some implementation choices.

6. Conclusions

In this paper, we have presented a practical approach to provide interoperability between BRMS. The utilization of MDE techniques enabled to successfully develop two bridges amongst rule languages with different degrees of expressiveness.

To the best of our knowledge, this is the first approach implementing a solution for transformations with such a large number of transformations. The discovery, development and chaining of complex transformations has been a challenging task. The presence of several transformations, models and metamodels showed that megamodeling is a crucial issue when dealing with large projects. The scripting language is a first step that helped a lot, together with its parameterization. However, care should be taken to control the inherent complexity of the solution. We think more work can still be done on that area, especially in the specification of generic megamodeling platforms.

On top of their expected practical usefulness, implementing these bridges has revealed both the strength and some limits of available MDE tools. We have greatly appreciated the power of metamodeling associated with declarative programming and have suffered from the main limit relative to parsing of non context-free grammars. We have also faced different transformation problems that have not been explicitly handled in the literature (e.g., the navigation through patterns of expressions).

Our approach has been empirically validated by executing the bridge on a set of well-known benchmarks for business rules, and on a demonstrative example. The bridges produced the expected results.

There are several possibilities for future work, such as the creation of similar bridges amongst different rule languages (e.g., Yasu or Haley). The parsing can be internationalized into different languages, which is a common requirement of industry. Finally, we plan to study how to use MDE techniques to parse context-aware grammars, or even natural language.

Acknowledgements: This work has been partially supported by French ANR IdM++ project.

7. References

Abouzhara A, Barbero M. Implementing two business rule languages: PRR and IRL. Ref. site: http://www.eclipse.org/m2m/atl/usecases/PRR2IRL/. March07

Academic Benchmark performances. Ref. site: http://blogs.ilog.com/brms/2007/10/22/academic-benchmark-performance/, 27-12-2007

Allilaire F, Bézivin J, Brunelière H, Jouault F. Global Model Management. In proc. of eTX Workshop at the ECOOP 2006, Nantes, France

ATL Use Cases. Ref. site: http://www.eclipse.org/m2m/atl/usecases/. April 08

Brownston L, Farrell R, Kant E. Programming Expert Systems in OPS5 Reading, Massachusetts: Addison-Wesley, (1985)

Drools Examples. Ref. site: http://download.jboss.org/drools/release/4.0.4.17825.GA/ drools-4.0.4-examples.zip. 31-03-2008

Eclipse Modeling Project. Ref. site: http://www.eclipse.org/modeling/. April 08

Forgy C. OPS5 User's Manual, Technical Report CMU-CS-81-135 (Carnegie Mellon University), 1981

ILOG Rule Languages. Ref. site: http://www.ilog.com/products/jrules/documentation/jrules67/. April 08

ILOG JRules. Ref. site: http://www.ilog.com/products/jrules/index.cfm. April 08

JBoss Drools. Ref. site: http://www.jboss.org/drools/. April 08

Jouault F, Bézivin J. KM3: a DSL for Metamodel Specification. In proc. of 8th FMOODS, LNCS 4037, Bologna, Italy, 2006, pp 171-185

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

Jouault F, Kurtev I. Transforming Models with ATL. In proc. of the Model Transformations in Practice Workshop at MoDELS 2005, Montego Bay, Jamaica, pp 128-138

Kolovos D, Paige R, Polack F. A Framework for Composing Modular and Interoperable Model Management Tasks, In proc. of Workshop on Model Driven Tool and Process Integration (MDTPI), EC-MDA 2008, Berlin, Germany

Kurtev I, Bézivin J, Aksit M. Technological Spaces: An Initial Appraisal. In proc. of CoopIS, DOA'2002 Federated Conferences, Industrial track, 2002, Irvine, California, USA

Kurtev I, Bézivin J, Jouault F, Valduriez P. Model-based DSL Frameworks. In proc of. Companion of OOPSLA 2006, October 22-26, 2006, Portland, OR, USA, pp 602-616. 2006

OpenArchitectureWare (oAW). Ref. site: http://www.openarchitectureware.com/. Jan. 2009

OMG. Object Management Group. Ref. site: http://www.omg.org. April 08

Owen J. Business rules management systems. Extract business rules from applications, and business analysts can make changes without IT breaking a sweat. Infoworld, 25-06-2004

Production Rule Representation (PRR), Beta 1, Document Number: dtc/2007-11-04 Ref. site: http://www.omg.org/spec/PRR/1.0/

Rule Interchange Format (Working Group). Ref. site: http://www.w3.org/2005/rules/wiki/RIF_Working_Group. April 08

Yasu Technologies. Ref. site : http://www.yasutech.com/. April 08