Towards Automated Generation of Regulation Rule Bases using MDA

Deepali Kholkar, Sagar Sunkle and Vinay Kulkarni
Tata Consultancy Services, Pune, India
{*deepali.kholkar, sagar.sunkle, vinay.vkulkarni*}*@tcs.com*

Keywords: Model Driven Architecture, Business Rule Management Systems, Rule Engines, Production Rule Systems, SBVR, CIM, PIM, PSM, Fact-oriented Model, Model Transformation, Formal Compliance Checking, Defeasible Logic.

Abstract: Enterprises today face the problem of complying with ever-increasing regulation. Use of rule engines for implementing compliance is widespread, however, the rule base needs to be encoded manually. We present a method using model-driven architecture (MDA) to automate generation of rules in a rule language, from a platform-independent model derived from a specification given by domain experts. We demonstrate how a Semantics of Business Vocabulary and Rules (SBVR) model of regulation rules can serve as the common source model for generating rules on various categories of rule engine platforms. The approach is illustrated using a real-life case study from the MiFID-2 financial regulation.

1 INTRODUCTION

The regulatory environment in which enterprises operate is steadily growing in complexity as more and more regulations come into force. Enterprises grapple with the problem of implementing compliance while managing costs, time to market, accuracy and correctness. Compliance is mandatory and non-compliance entails heavy penalties and reputational risk. Regulatory compliance therefore figures among the top few concerns of enterprises worldwide[1].

Compliance implementations in enterprise IT systems are broadly classified into two categories: one where rule checks are implemented directly in application code, and the other where rule engines are used to encode and check rules in an orthogonal manner, outside of application code.

The rule engine approach is increasingly being adopted since it allows a declarative specification of rules to be maintained separately from the processes and workflow encoded in applications. Rules maintained in such business rule management systems (BRMS) may originate not just from regulations, but organizational policies, business requirements, or any other source. The business rule approach (Bauer, 2009), as this is called, allows rules to be maintained by users without the need to modify IT systems.

Although BRMS are in widespread use, they require rules to be manually encoded from their natural language source, i.e. legal text of regulations, into the target rule language. Compliance implementation and subsequent change management therefore require intensive involvement of human experts, to encode rules as well as compute impact when there is a change on the regulation or enterprise side. The implementation team needs to be conversant with technicalities of the rule-checking platform, posing a severe challenge for effective participation of domain and legal experts.

Automation is highly desirable to address all of these issues, especially, an approach that allows domain experts to specify rules in domain terms, yet enables automated derivation of the formal specification of rules. Direct derivation of formal specification from a natural language format specified by domain experts is not possible, as also discussed in (Levy and Nazarenko, 2013). We advocate creation of models as an intermediate step.

In this paper, we demonstrate the applicability of the the model-driven engineering process prescribed by Object Management Group (OMG)[2] in their Model-Driven Architecture^{TM} (MDA^{TM}) standard, to the problem of automated rule generation from their natural language specification. Specifically, we show how such an architecture can be implemented using the Semantics of Business Vocabular-

[1]Top Ten Problems Faced by Business, http://www.bmgi.com/resources/articles/top-ten-problems-faced-business

[2]OMG Model Driven Architecture, http://www.omg.org/mda/

Kholkar, D., Sunkle, S. and Kulkarni, V.
Towards Automated Generation of Regulation Rule Bases using MDA.
DOI: 10.5220/0006216406170628
In *Proceedings of the 5th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2017)*, pages 617-628
ISBN: 978-989-758-210-3

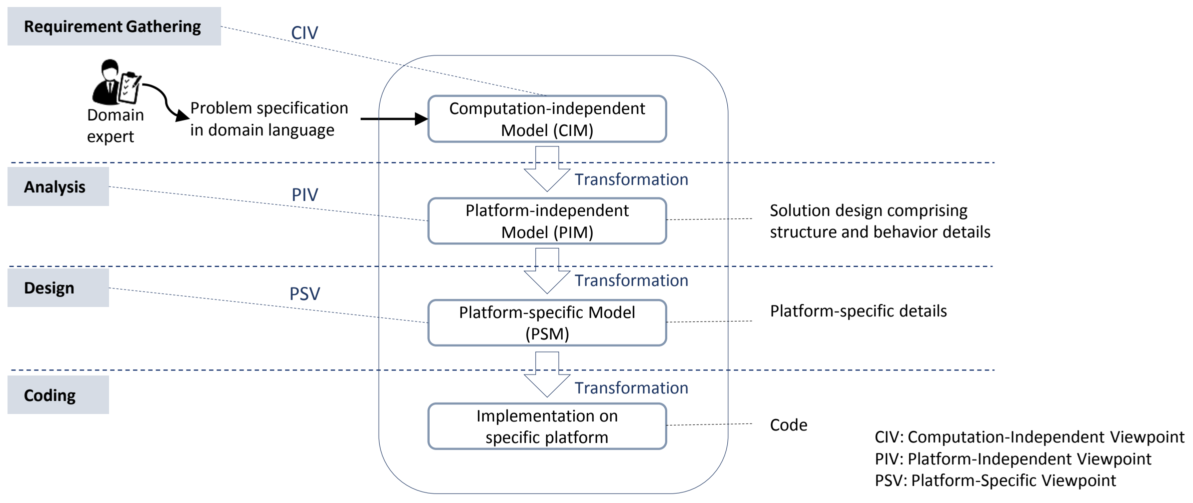
Requirement Gathering
CIV
Domain expert
Problem specification in domain language
Computation-independent Model (CIM)
Analysis
Transformation
PIV
Platform-independent Model (PIM)
Solution design comprising structure and behavior details
Design
Transformation
PSV
Platform-specific Model (PSM)
Platform-specific details
Coding
Transformation
Implementation on specific platform
Code
CIV: Computation-Independent Viewpoint
PIV: Platform-Independent Viewpoint
PSV: Platform-Specific Viewpoint

Figure 1: Layers in MDA.

ies and Rules^{TM} (SBVR^{TM}) standard by OMG[3]. We demonstrate how SBVR is highly suitable for building a platform-independent model of rules from which any platform-specific rule implementation can be derived. We discuss rule generation for two rule platforms, viz. DR-Prolog (Dimaresis, 2007) and JBoss Drools[4].

The following sections outline the principles of MDA and describe their application to the problem of rule base generation, the choice of models in our architecture, their mappings, a detailed illustration of rule generation for DR-Prolog using a case study example. We show how the architecture easily enables mapping and generation onto a different kind of rule platform, i.e. Drools, from the same platform-independent model.

2 OVERVIEW OF MDA

MDA advocates creation of abstract, machine-readable models of the problem and solution space, stored in standardized repositories (Kleppe et al., 2003). These models can then be repeatedly accessed to generate implementation artefacts such as schemas, code, test harnesses, deployment scripts (Kleppe et al., 2003; Kulkarni and Reddy, 2003). Models give a higher-level abstraction over code, that is easier to understand and maintain.

2.1 Principles of MDA

MDA emphasizes separation of concerns, where domain, structural, and platform details are encapsulated in separate layers of abstraction (Kulkarni and Reddy, 2003). A description of the problem in purely domain-specific terms is captured in the computation-independent model (CIM)[5]. The next layer is the platform-independent model (PIM) that captures design details of the solution in terms of structure and behavior. The PIM is devoid of any technology platform details. Finally, there is the platform-specific model (PSM) that is the realization of the PIM on a specific technology platform. The three layers of MDA are illustrated in Figure 1.

Separation of concerns enables details of each aspect to be clearly specified without getting entangled with another. This makes the specification maintainable.

2.2 MDA Layers and SDLC

As can be seen from the definitions of the MDA layers in Section 2.1, each layer corresponds to a phase in the software development lifecycle (SDLC). CIM is the specification of a system created in the Requirements Gathering phase, while PIM corresponds to the Analysis, and PSM to the Design phase (Alhir, 2003), as shown in Figure 1.

[3]Semantics of Business Vocabulary and Business Rules, http://www.omg.org/spec/SBVR/1.2/

[4]RedHat Drools BRMS, http://www.drools.org/

[5]Model Driven Architecture - A Technical Perspective, http://www.omg.org/cgi-bin/doc?ormsc/2001-07-01

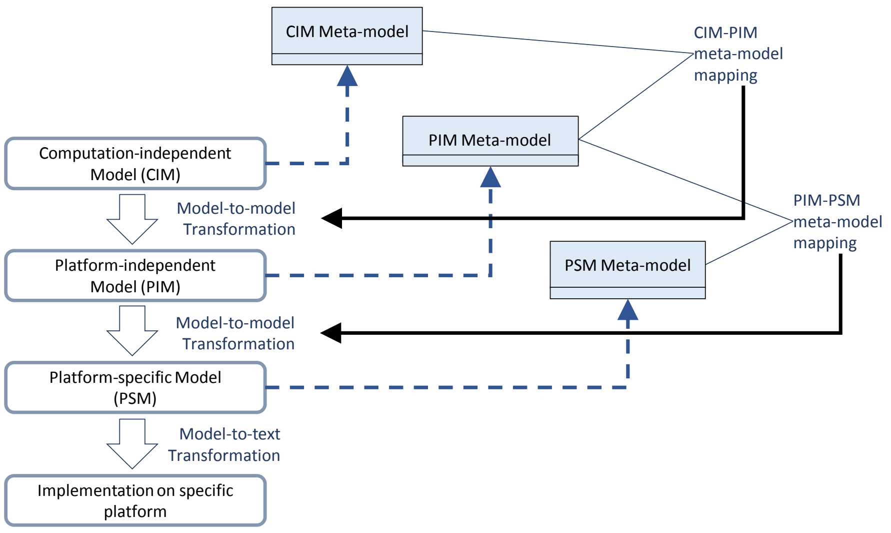
CIM Meta-model
CIM-PIM meta-model mapping
PIM Meta-model
Computation-independent Model (CIM)
Model-to-model Transformation
PIM-PSM meta-model mapping
Platform-independent Model (PIM)
PSM Meta-model
Model-to-model Transformation
Platform-specific Model (PSM)
Model-to-text Transformation
Implementation on specific platform

Figure 2: Model mapping and transformations in MDA.

2.3 Model Mapping and Transformation

The advantage of MDA is that models are machine operable. Successive layers of the model, i.e. PIM and PSM can be derived from the previous layers, i.e. CIM and PIM respectively, by meta-model mapping and automated transformation. This results in productivity and cost benefits. It also enables tracking impact of changes in one layer on another, making change management easier. Moving to new technology platforms or a different design choice requires only the relevant part of the model or mapping to be changed. Downstream model layers can be generated afresh to reflect the changes. For instance, to derive the PSM for a different technology platform, the PIM needs only to be mapped to the meta-model for the new platform, and its PSM generated.

The mapping and transformations between the three layers are illustrated in Figure 2. A mapping is a set of rules for deriving one model layer from another, and is based on the meta-models of the two layers. PIM-PSM mapping can be used to generate a realization of the logical PIM on physical execution infrastructure[6]. If both PIM and PSM are MOF-compliant models, model-to-model transformation techniques and tools such as QVT[7] can be used to generate the PSM. The implementation on the chosen technology platform can then be generated from the PSM using model-to-text transformation. In the next section, we describe our approach of applying MDA to the problem of creating and maintaining a rule base in a BRMS.

3 MDA FOR RULE BASE GENERATION

BRMS or rule engines work on the basic premise that the truth status of defined *rules* needs to be determined, given information available as *facts*.

Rule engines can be classified into the following three principal kinds[8] based on the reasoning algorithm used

1. Pure inferencing engines such as Prolog, DR-Prolog. These either use forward chaining i.e. data-driven inferencing about rules based on available data or information, or backward chaining, i.e. goal-driven inferencing beginning with a *goal* or query given by the user, and testing for the truth value of its contained goals one by one.
2. Production Rule Systems (PRS) such as JBoss Drools that combine inferencing using forward or backward chaining or both, called hybrid reasoning, and take actions based on the conclusions drawn.
3. Reactive rule engines that do complex event processing, i.e. detect events from available information, and react to them.

[6]Model Driven Architecture - A Technical Perspective, http://www.omg.org/cgi-bin/doc?ormsc/2001-07-01

[7]Meta Object Facility (MOF) 2.0 Query/View/Transforma tion Specification, http://www.omg.org/spec/QVT/1.2/

[8]Production Rule Representation^{TM}, http://www.omg.org/spec/PRR/

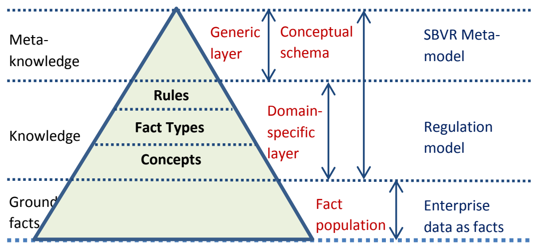

Figure 3: Layers of a fact-oriented model.

Rule languages for these engines differ in their syntax, but are based on a common paradigm called fact-oriented modeling (FOM). A fact-oriented model captures rules as compositions of *facts*. A *fact* is a relation between *concepts*. The layers of a fact-oriented model are illustrated in Figure 3.

Similar to application code, rules in these languages need to be coded by development teams conversant with language syntax, whereas their requirements are understood only by domain experts. Rules are the most critical component of any business application and drive all the processes in the organization. It is therefore imperative that domain experts are able to directly specify, maintain, and control business rule repositories. Production rule systems though widely used, are unfortunately not usable by domain experts.

We opine that MDA is highly applicable to this problem scenario, and provides an alternative to manual encoding of rules. We propose use of a model-driven architecture for automated generation of rule bases. A high-level specification of rules in domain language, given by domain experts forms the CIM of rules. The common conceptual model used by all rule engines, comprising *rules* dependent on *facts*, constitutes a PIM of rules, since it gives a *structure* for definition of rules. Individual rule languages denote the PSM of rules. The PIM is common to all rule languages and can be mapped to the PSM for individual rule languages. The implementation in individual rule language can then be generated from this PIM-PSM mapping. This offers the possibility of switching to or maintaining different implementations of a rule base on various platforms using a common PIM.

The characteristics each model layer must have, as specified by OMG, are listed below.

- CIM should be able to express details about the problem domain in the language of the domain expert, with no references to how they should be implemented, whether in terms of design or technology.
- PIM needs to capture high-level design details of the solution. These include structure, relations, and behavior of various entities, also details of implementation, without being bound to a specific technology platform.
- PSM should be able to capture low-level design of the solution as the platform-specific interpretation of the PIM.

In the next few sections, we discuss our choice of modeling languages for each layer.

3.1 The CIM and PIM Layers

Several general-purpose rule languages and notations exist, such as SBVR, Production Rule Representation (PRR), RuleML, SWRL, W3C RIF. SWRL, RuleML and W3C RIF are specially designed for capturing ontologies. SWRL combines the capabilities of RuleML and OWL. PRR has been explicitly devised to create a generic representation of rules that addresses all types of production rule systems.

SBVR was devised by OMG as a standard for capturing the vocabulary used by a business domain, definitions and relations between terms, and business rules governing the domain. SBVR is a fact-oriented modeling notation (Nijssen, 2007; Halpin, 2007), that captures rules as compositions of facts. This is the same conceptual model as that used by all rule engines, making SBVR the natural choice of model.

SBVR has a MOF-compliant meta-model, and also its own controlled natural language notation for specifying the model, called SBVR Structured English (SE). SBVR SE is a restricted subset of natural language, with a well defined set of keywords that connect natural language phrases denoting concepts and their relations. We use SBVR SE for the CIM, since it is a structured, yet near-natural language notation in which the vocabulary and rules for any domain can be specified, fulfilling the criteria for a CIM.

SE is intended as a means to populate the SBVR model, hence its elements have direct correspondence with the SBVR meta-model. A translation scheme from SE to an SBVR model can thus be worked out. It therefore follows that we use SBVR as the PIM. SBVR SE and SBVR model have often been clubbed in literature and classified as a CIM (Diouf et al., 2007). We choose to treat SE as a CIM notation, and the SBVR model as a PIM since it captures structure and behaviour of domain entities using a specific meta-model. A concept model captures the structure, while rules captured as logical formulations built on the concept model encode the behaviour.

We choose SBVR as PIM because its semantic fact-oriented model captures the complete dependence hierarchy of rules on fact types and concepts, crucial for the inferencing done by rule engines. We choose SBVR over other languages for its inherent mapping to SE. Any other rule language as PIM

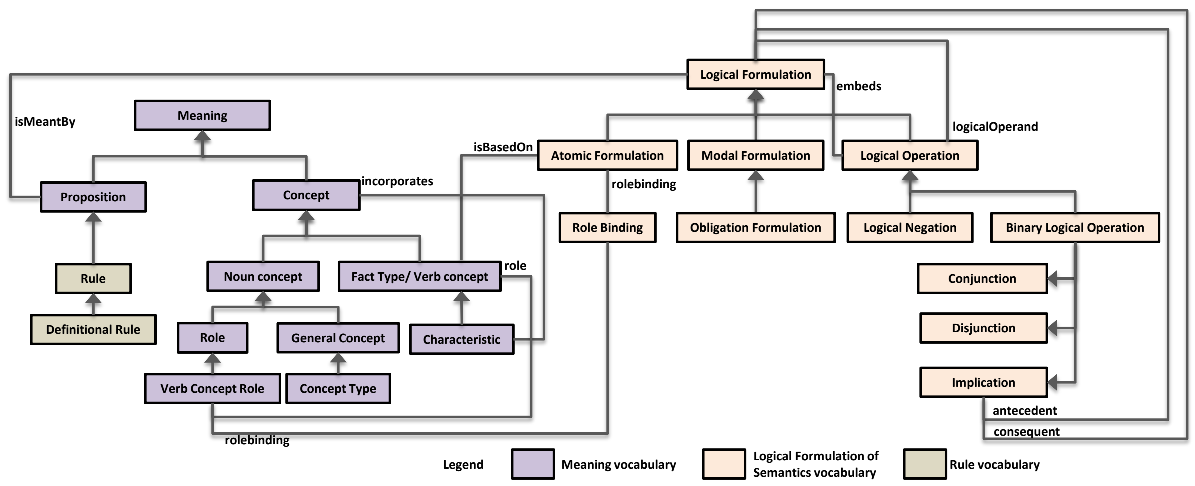

Figure 4: SBVR meta-model.

would require designing the CIM-PIM mapping from SE. SBVR captures both structure and behavior in a single notation, whereas when PRR is used to capture rules, structure needs to be defined using UML class models.

The expressiveness of SBVR is sufficient to capture a generic platform-independent representation of rules. The generic SBVR meta-model can be mapped to the platform-specific conceptual models of DR-Prolog as well as a PRS, as we illustrate in the next few sections. The SBVR meta-model subset we use is shown in Figure 4, and described in detail in the next section.

3.2 SBVR Meta-model

We use a subset of the OMG SBVR meta-model[9] for capturing regulation rules, shown in Figure 4. The meta-model comprises three sections, as shown in the figure.

1. **Meaning Vocabulary:** This is the meta-model for capturing structure or the body of *concepts*. *Noun concepts* denote entities, while *verb concepts*, also called *fact types*, signify relations. Fact types take the form *role verb role*, where *role* denotes a noun concept. *General concepts* and *concept types* specialize concepts and help create concept hierarchies. Attributes of a concept are captured as *characteristics*.

2. **Logical Formulation of Semantics Vocabulary:** This section comprises *logical formulations* of fact types. Compound logical formulations e.g. *conjunctions*, *implications*, *negations* are composed of *atomic formulations*. Each atomic formulation is based on a fact type from the body of concepts.

3. **Rule vocabulary**: This section specifies rules, based on logical formulations. We use three types of rules: *rules* to denote obligations, *definitional rules* to denote necessity formulations, and *operational rules* to denote actions to be executed. A rule inherits from *Proposition*, that is *meant by* a logical formulation that is a formal expression of the rule in terms of fact types.

SBVR thus provides a comprehensive meta-model for capturing the *semantics* of rules as logical formulations over fact types and concepts.

The next subsection discusses platform-specific models for rules.

3.3 The PSM Layer

We select DR-Prolog and JBoss Drools as our target platforms for rule generation. Each is representative of a separate class of rule engines, viz. backward-chaining and production rule systems respectively. Our objective is to demonstrate that MDA helps generate code onto multiple platforms using a single PIM.

The DR-Prolog and Drools rule definition meta-models are the PSMs for our MDA for rules. DR-Prolog attempts to answer queries by working backwards from the query through its constituent goals to the available information to see whether the query is satisfied. Drools and other production rule systems use the hybrid reasoning Rete algorithm[10] to in-

[9]Semantics of Business Vocabulary and Business Rules, http://www.omg.org/spec/SBVR/1.2/

[10]RedHat Drools BRMS, https://www.drools.org/learn/documentation.html

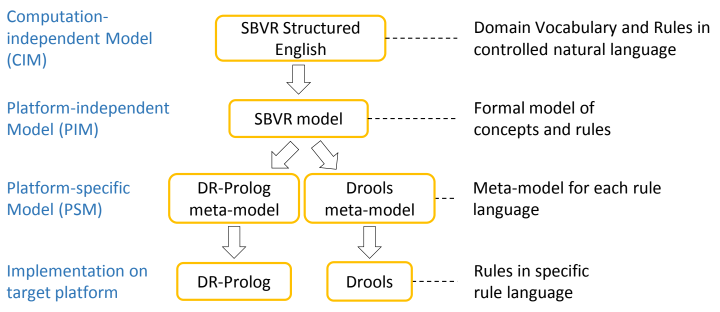
Computation-independent Model (CIM)
SBVR Structured English
Domain Vocabulary and Rules in controlled natural language
Platform-independent Model (PIM)
SBVR model
Formal model of concepts and rules
Platform-specific Model (PSM)
DR-Prolog meta-model
Drools meta-model
Meta-model for each rule language
Implementation on target platform
DR-Prolog
Drools
Rules in specific rule language

Figure 5: Layers of MDA for Rule base generation.

fer which rules of the form ***when** condition **then** action* apply, and trigger the corresponding action which changes the state of the system. Applicable rules have to be freshly recomputed on the new system state and this cycle continues.

Although rule syntax and execution semantics differ in the two rule engines, their rule definition meta-models follow the fact-oriented modeling paradigm and therefore easily map to the SBVR meta-model. We create the PIM-PSM meta-model map for the specific platform, say, DR-Prolog, and use it for transformation of the PIM instance to PSM instance. We then translate the PSM instance to rules in DR-Prolog syntax. Our model driven architecture for rule base generation is depicted in Figure 5.

MDA also allows multiple PIM-PSM layers to be used, with each PSM becoming the PIM for the next layer. For instance, it was possible to use SBVR as PIM and PRR as a PSM, since it captures the model for the PRS class of systems. The PRR model then becomes the PIM for a Drools PSM.

The next section describe the method for generating the rule base in the chosen rule language, using this architecture.

4 METHOD FOR RULE BASE GENERATION

The method for rule base generation comprises creation of the CIM and PIM of regulation rules using SBVR, mapping the PIM i.e. SBVR meta-model to the PSM, i.e. meta-model of the chosen rule language, and finally, PIM to PSM transformation from SBVR to the rule language. These steps are detailed in the next couple of sections.

4.1 Create CIM and PIM of Regulation Rules in SBVR

We use the Eclipse Modeling Framework (EMF) (IBM, 2016) model-to-model conversion tools and code generators to generate code for an SBVR editor. We convert the MOF-compliant SBVR meta-model available on the OMG SBVR website to EMF Ecore format and use it as the basis for generating editor code.

We build the CIM and PIM of regulation rules using SBVR in the following steps

1. Domain experts mark in the NL regulation text, the statements representing rules to be checked, definitions of terms used in the rules, and data descriptions relevant to the rules.
2. The domain experts then write each NL rule statement in the controlled natural language SBVR Structured English (SE). This is the CIM of the regulation. SBVR SE is written using a restricted English vocabulary and specific font styles, viz. the term font for designating noun concepts, general concepts, concept types and roles; Name font for individual concepts or names; *verb* font for designations of fact types; and keyword font for other words in definitions and statements.
3. We mapped the SE meta-model to the SBVR meta-model. We create the SBVR PIM corresponding to the captured SE statements manually using the SBVR editor. Automation of the CIM-PIM translation from SE to SBVR model is our ongoing work and its description is outside the scope of this paper.

The next section describes PIM to PSM transformation, from SBVR to our chosen rule language, DR-Prolog.

Table 1: SBVR to DR-Prolog mapping.

SBVR element	**DR-Prolog construct**	**DR-Prolog syntax**
Element of guidance	Defeasible rule	defeasible(*rule name*, obligation, *rule consequent*, [*rule antecedent*]).
Logical formulation	Rule	
Implication (consequent, antecedent)	Implication rule (consequent, antecedent)	fact(*consequent*):- fact(*antecedent*).
Conjunction (operands)	Conjunction rule (operands)	fact(*conjunction*):- fact(*operand1*), fact(*operand2*).
Disjunction (operands)	Disjunction rule (operands)	fact(*disjunction*):- fact(*operand1*). fact(*disjunction*):- fact(*operand2*).
Negation (operand)	Negation rule (operand)	fact(*negation*):- fact(not(*operand*)).
Atomic formulation *based on* verb concept	Predicate	fact(*verbConcept(role-list)*).
Concept (Delimiting characteristic, characteristics)	Predicate (key attribute,attribute-list)	fact(*concept(key attribute,attribute-list)*).
General concept (Specialization of concept)	Implication (consequent, antecedent)	fact(*concept1(key attribute,attribute-list)*:- fact(*concept2(key attribute,attribute-list)*).
Characteristic of concept	Unary predicate (Delimiting characteristic of concept)	fact(*characteristic(key attribute of concept)*)
Delimiting characteristic of concept	Primary key of predicate	fact(*concept(**key attribute**, attribute-list)*).
Fact type (concept-list)	Predicate (concept-list)	fact(*factType(list of key attributes of concepts)*).
Ground facts	Ground facts	fact(*concept(key value, value-list)*).

4.2 Transform PIM to PSM: SBVR to DR-Prolog

In order to transform rules from SBVR PIM to DR-Prolog, we create the PIM-PSM map between SBVR and DR-Prolog meta-models, shown in the first two columns of Table 1. The conceptual model of DR-Prolog has almost a one-to-one correspondence with the SBVR meta-model.

We have written a custom program to use this PIM-PSM map to generate the DR-Prolog PSM instance model from the SBVR PIM instance model. From the DR-Prolog PSM, another program generates rules in the corresponding textual DR-Prolog syntax shown in the third column of Table 1.

In the next section, we illustrate our approach using a real-life case study example from the MiFID-2 regulation[11].

[11]MiFID2: http://ec.europa.eu/finance/securities/isd/mifid2/index_en.htm

5 CASE STUDY

The MiFID-2 (Markets in Financial Instruments Directive) regulation lays down obligations on financial institutions regarding the types of transactions that must be included/ excluded in reporting trades to the regulatory body. We illustrate here the relevant excerpt from the original regulation text and the chain of models created for the regulation rules.

5.1 Regulation Text

The original regulation text containing inclusion and exclusion rules for transactions is shown below.

Meaning of transaction

1. *For the purposes of Article 26 of Regulation (EU) No 600/2014, the conclusion of an acquisition or disposal of a financial instrument referred to in Article 26(2) of Regulation (EU) No 600/2014 shall constitute a transaction.*

2. *An acquisition referred to in paragraph 1 shall include:*

   (a) *a purchase of a financial instrument;*

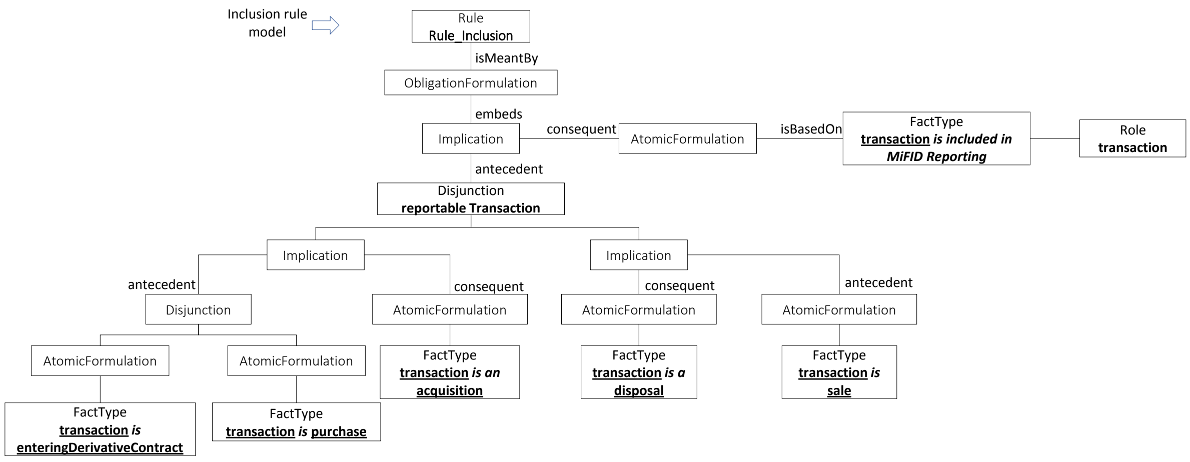

Figure 6: SBVR model of rules.

(b) *entering into a derivative contract in a financial instrument.*

3. *A disposal referred to in paragraph 1 shall include:*

(a) *sale of a financial instrument;*

(b) *closing out of a derivative contract in a financial instrument.*

..

4. *A transaction for the purposes of Article 26 of Regulation (EU) No 600/2014 shall not include:*

(a) *a securities financing transaction as defined in Regulation [Securities Financing Transactions]*

(b) *a contract arising exclusively for clearing or settlement purposes;*

(c) *an acquisition or disposal that is solely a result of custodial activity;*

The next sub-section illustrates the CIM created by writing the above natural-language regulation rules in Structured English.

5.2 CIM of Regulation Rules in SE

The inclusion and exclusion rules from the regulation text are encoded in SBVR SE as below.

Rule_Inclusion: It is obligatory that transaction *is included in MiFID reporting* if the transaction *is an* acquisition or a disposal.

Rule_Exclusion: It is obligatory that transaction *is excluded from MiFID reporting* if the transaction *is a* securities financing transaction or clearing or settlement contract or an acquisition or disposal *arising from* custodial activity.

Here, the keywords *It is obligatory that* denote the obligation modality of the rule. The rule is built upon fact types *transaction is included in MiFID reporting*, *transaction is an acquisition*, and *transaction is a disposal*. Transaction, acquisition, and disposal are concepts; *is included in MiFID reporting* is a characteristic of a transaction.

Acquisition and disposal are high-level concepts defined in terms of other concepts, e.g. purchase and sale. These definitions are captured as definitional rules as follows Acquisition *is a* purchase or entering a derivative contract. Disposal *is a* sale or closing a derivative contract.

The next section shows the SBVR model constructed from these SE rules.

5.3 PIM of Rules in SBVR

The SBVR model corresponding to the above SE rules is created using the SBVR editor, and shown in Figure 6. This PIM of rules is programmatically translated to a DR-Prolog model of MiFID rules and from there to MiFID rules in DR-Prolog syntax, illustrated in the next sub-section.

5.4 Translated Regulation Rules in DR-Prolog

The inclusion and exclusion rules in DR-Prolog syntax generated from the SBVR model illustrated in Figure 6 as per the mapping shown in Table 1 are shown below.

defeasible (rule_inclusion, obligation, includeInMiFIDReporting (TransRef), [reportableTransaction(TransRef)]).

defeasible (rule_exclusion, obligation, excludeInMiFIDReporting(TransRef), [exclusionTransaction(TransRef)]).

Since the antecedent *reportableTransaction* of the inclusion rule is a disjunction of *acquisition* and *disposal*, the implications or simple DR-Prolog rules specifying this relation follow.
fact (reportableTransaction(TransRef)) :- fact (acquisition(TransRef)).
fact (reportableTransaction(TransRef)) :- fact (disposal(TransRef)).

Definitional rules get translated as simple DR-Prolog rules, as
fact (acquisition(TransRef)) :- fact (purchase(TransRef)).
fact (acquisition(TransRef)) :- fact (enteringDerivativeContractInFI(TransRef)).

The generic SBVR-to-DR-Prolog meta-model mapping and translator can thus be used to translate any SBVR model of rules to a rule base in DR-Prolog in an automated manner.

In the next section, we discuss insights and lessons learnt in applying our approach.

5.5 Discussion and Lessons Learnt

In the course of development of this architecture and during its application, we came across several important considerations that are discussed here.

In theory, writing SE rules is not hard for domain experts, with a little training, given that SE is a restricted subset of English with well-defined keywords. However, there are several ways in which rules can be expressed in SE, with the same meaning, compounding the risk of making syntactic errors. We created an SE editor that checks syntax, to get around this problem. Even so, manual encoding of SE rules, as well as identifying rule statements to be encoded from the voluminous NL text of regulations are effort-intensive tasks. Intelligent assistance in these tasks to significantly reduce experts' burden is needed, so that their time is utilized efficiently. Towards this end, we are working on automated rule extraction using natural-language processing and machine learning techniques. These techniques are semi-automated, and extract SE rules from NL text of regulations, so that domain experts can validate them. Availability of bilingual corpora is important for the accuracy and success of these techniques. Initially some effort may need to be expended, to create such corpora.

Manually translating SE rules to SBVR as we did for the case study is even more cumbersome and error-prone than coding SE rules, since SBVR syntax is complex. Similar to SE, SBVR also provides several ways of encoding a rule. We are therefore working on automating this step. This makes correctness of SE rules all the more important. To avoid errors, the SE-SBVR mapping and translation must be validated, again using a bilingual corpus. Since the SBVR model can be built in several ways, we found that deciding on modeling standards and guidelines upfront was essential to maintain consistency in the model. This is necessary even when translation from SE is automated.

Although automation speeds up the process, domain experts' review can become a bottleneck when applying the approach in the real world. Other practical considerations are managing multiple users working on creating various parts of the model at the same time, allowing sharing of models, and preventing duplication.

OMG's provision of SE as a controlled natural-language interface to SBVR is an invaluable feature that makes it possible for domain experts to interact with and review the generated model, that would be hard to do directly with SBVR. Since SE is usable by domain experts, it provides a way to create the CIM in OMG's MDA process. SE is the key link that helps create a model from natural-language specifications.

OMG's MDA process gives a method for using models and model-based techniques effectively for generating code. The most important consideration in choice of modeling languages is their expressiveness with respect to the requirements of the problem context. However, all features needed in the problem context may not be available in each modeling language chosen for our architecture. Workarounds need to be implemented by specifying construct-specific transformation rules in the CIM-PIM and PIM-PSM model mapping to take care of such cases, so that there is no loss of information. For instance, real-world applications make use of temporal constructs that can be modeled in SE. However, temporal constructs are not directly supported by SBVR and DR-Prolog, but are encoded using available arithmetic functions and variables to denote time. On the other hand, Drools supports temporal constructs, which makes it better suited as a PSM for real-world business applications.

An important consideration in using model-based techniques is model validation. In our approach, we validated the models and transformations manually, as well as by testing the generated rules using the rule engine and test bed data. Any discrepancies with expected results were manually traced back through the model chain to arrive at the source of the error.

Domain experts validated the SE rules written by us for our case study application. They validated the generated rules by examining the results of rule execution on their test data comprising a set of actual transactions from one of their systems. They examined the rules executed, their success/ failure status,

Table 2: SBVR to Drools mapping of constructs.

SBVR element	Drools construct	Drools syntax
Element of guidance	Rule	**rule** *rule name attribute-list* **when** *antecedent-list* **then** *action-list* **end**
Logical formulation	Constraint	
Implication	Rule	**rule** *rule name attribute-list* **when** *antecedent-list* **then** *action-list* **end**
Conjunction (operands)	Conjunction (constraints)	*condition***,** *condition*
Disjunction (operands)	Disjunction (constraints)	*condition* **or** *condition*
Negation (operand)	Negation (constraint)	**not**(*condition*)
Concept (characteristics)	Type (attributes)	**declare** *concept name attribute-list*
Characteristic of Concept	Attribute of Type	*attribute name* **:** *Type*
Delimiting characteristic of Concept	Key attribute of Type	**@key** *attribute name* **:** *Type*
General Concept (Specialization of Concept)	Type extends type	**declare** *concept name* **extends** *concept name*
Fact type	Attribute within source Type, of type target Type	**declare** *source-concept-name target-concept-name* **:** *Type*

and the data facts output by the compliance engine in support of the success/ failure result, and verified these for correctness. The quality of generated rules depends upon the quality of SE rules input to the framework. We may need to help domain experts reviewing SE rules by providing tools for organising and presenting the rules for easy readability and editing.

Choice of modeling languages and correct specification of mapping and model transformation rules between layers is a one-time investment in setting up a model-based automated engineering process. The initial investment in time and cost as well as the prospect of change are the principal deterrents to the proliferation of MDE in practice. Both can be overcome only by increased usage of MDE, coming up with better tools, sharing of artefacts such as models and mappings, and reporting on experience.

In the next section, we discuss the extensibility of this architecture for generation onto other rule platforms, using Drools as an example.

6 GENERATION ONTO MULTIPLE RULE PLATFORMS

The general rule specification in SBVR can be mapped to another rule PSM, to generate an implementation on a different rule platform. We now illustrate a mapping of the SBVR meta-model to the Drools rule language meta-model, with the objective of demonstrating that rules on the Drools platform can be generated from the same SBVR PIM of rules. The implementation of this SBVR-Drools translation is work in progress.

The Drools rule engine too expects specification in a structure similar to DR-Prolog, i.e. rules and facts. The SBVR rule meta-model therefore maps directly to the Drools rule model. We create an SBVR-Drools meta-model map, i.e. our PIM-PSM map, illustrated in Table 2. This map can be used to generate a Drools PSM instance from the SBVR PIM of rules. Rules in corresponding Drools syntax shown in Table 2 can then be generated from the Drools PSM instance, as we did for DR-Prolog.

Drools being a PRS, expects additionally, a procedural specification of rule consequents or actions that change the system state. We therefore need special interpretation for the rule actions in Drools. This is done by defining special transformation rules for operative business rules defined in SBVR to action specifications for Drools and similar PRS. Consequents of operative business rules in SBVR are interpreted as action specifications by extracting the Drools function name and parameters from the consequent logical formulation and its operands respectively. To be able to do this, the rule consequent should be modeled accordingly by the user in the SBVR model.

The same SBVR PIM of rules for a problem domain can thus be translated to rule bases on multiple platforms, using the generic PIM-PSM mapping for each platform. The next section discusses related work.

7 RELATED WORK

We discuss related work under three heads, viz.

model-driven rule generation, formal representations of regulations, and SBVR-related work.

7.1 Model-driven Rule Generation

A model-driven rule generation approach has been proposed in (Diouf et al., 2007), that suggests using a combination of MDA and Ontology Definition Metamodel (ODM) for generating rules. Use of SBVR as CIM and the authors' own proprietary general purpose rule language as PIM is proposed, however, the language or its mapping to SBVR or PSMs was yet to be worked out.

7.2 Representations of Regulation Rules

Compliance checking approaches in the literature (Governatori and Rotolo, 2013; Awad et al., 2010; Governatori et al., 2009; Governatori, 2005) use formal representations of regulation rules. A system for defeasible logic representation of regulations is presented in (Dimaresis, 2007) that we use as the compliance engine in our work. In all of these, experts need to directly code rules in the rule language, high-level models or specification-based generative methods are not used.

7.3 SBVR and Fact-oriented Modeling

Several approaches use SBVR for encoding rules, such as semi-automated approaches to generate SBVR from natural language descriptions (Bajwa et al., 2011; Levy and Nazarenko, 2013; Njonko and Abed, 2012), expression of anti-money laundering rules in SBVR (Abi-Lahoud et al., 2013), precise capture of legal rules to reveal inconsistencies (Johnsen and Berre, 2010), but the SBVR models are not used for automated generation of rule bases in a rule language. Rules are written in SE and manually translated to rules in a BRMS in (Levy and Nazarenko, 2013). Requirements for translation from SBVR to Formal Contract Logic (FCL), a proprietary defeasible logic language are defined in (Kamada et al., 2010). The source SBVR and desired target FCL specification are given, however, the mapping or transformation between the two specifications is not dealt with.

8 CONCLUSION AND FUTURE WORK

The ideas proposed in MDA have been long been hailed for their promise of productivity enhancement and efficient change management, especially for large systems. MDA has worked well for technical problem spaces e.g. software development and evolution through code generation. We explored if this idea can work for business problem spaces as well e.g. regulatory compliance.

Modeling technologies are evolving to be usable by domain experts who are the real owners of business requirement model repositories. SBVR and Business Process Modeling Notation (BPMN) are two cases in point. We described an approach for generation of rule bases using SBVR Structured English as a CIM of rules and the corresponding SBVR model as PIM. We demonstrated how the SBVR model serves as a general-purpose PIM for rules, that can be used to map to various platform-specific rule language metamodels, even if they have different execution semantics, like DR-Prolog and Drools.

We illustrated automated generation of the rule implementation in DR-Prolog using the PIM-PSM mapping. We also illustrated the PIM-PSM mapping for the Drools platform. Actual generation of rule implementation in Drools using this mapping is part of ongoing work, as is automated translation of the SBVR SE CIM to SBVR PIM.

However, our generators described in this work are custom-written programs. We plan to implement specification-based PIM to PSM and PSM to rule language transformation using QVT and/ or Eclipse model transformation tools in order to fully exploit the power of MDA.

We plan to conduct a detailed study of the expressive power and adequacy of SBVR, Drools and DR-Prolog using a bigger real-world case study, as part of future work.

REFERENCES

Abi-Lahoud, E., Butler, T., Chapin, D., and Hall, J. (2013). Interpreting regulations with sbvr. Fodor, P., Roman, D., Anicic, D., Wyner, A., Palmirani, M., Sottara, D., Lvy, F., eds.: Joint Proceedings of the 7th International Rule Challenge, the Special Track on Human Language Technology and the 3rd RuleML Doctoral Consortium, Seattle, USA, July 11 -13, 2013. Volume 1004 of CEUR Workshop Proceedings., CEUR-WS.org.

Alhir, S. S. (2003). Understanding the model driven architecture.

Awad, A., Weidlich, M., and Weske., M. (2010). Consistency checking of compliance rules. In *BIS 2010: 106-11.*

Bajwa, I., Lee, M., and Bordbar, B. (2011). Sbvr business rules generation from natural language specification.

AAAI Spring Symposium: AI for Business Agility. pp. 28. AIII.

Bauer, E. (2009). The business rules approach.

Dimaresis, N. (2007). A system for modal and deontic defeasible reasoning. In *Int. J. Cooperative Inf. Syst. 14(2-3): 181-216*.

Diouf, M., Maabout, S., and Musumbu, K. (2007). Merging model driven architecture and semantic web for business rules generation. In *Web Reasoning and Rule Systems, First International Conference, RR 2007, Innsbruck , Austria, June 7-8, 2007, Proceedings*, pages 118–132.

Governatori, G. (2005). Representing business contracts in ruleml. In *Int. J. Cooperative Inf. Syst. 14(2-3): 181-216*.

Governatori, G., Hoffmann, J., Sadiq, S., and Weber, I. (2009). Detecting regulatory compliance for business process models through semantic annotations. In *Ardagna, D., Mecella, M., Yang, J., eds.: Business Process Management Workshops. Volume 17 of Lecture Notes in Business Information Processing. 517*. Springer Berlin Heidelberg.

Governatori, G. and Rotolo, A. (2013). A conceptually rich model of business process compliance. In *APCCM 2010: 3-12*.

Halpin, T. (2007). Fact oriented modeling past, present and future. In *Conceptual Modelling in Information Systems Engineering, J.Krogstie, A. L. Opdahl, and S. Brinkkemper (eds.) pp. 19-38*. Berlin Heidelberg: Springer-Verlag.

IBM (2016). Eclipse modeling framework.

Johnsen, A. and Berre, A. (2010). A bridge between legislator and technologist - formalization in sbvr for improved quality and understanding of legal rules. International Workshop on Business Models, Business Rules and Ontologies, Bressanone, Brixen, Italy.

Kamada, A., Governatori, G., and Sadiq, S. (2010). Transformation of sbvr compliant business rules to executable fcl rules. RuleML 2010: 4th International Web Rule Symposium. Number 6403, Springer (2010) 153161.

Kleppe, A., Warmer, J., and Bast, W. (2003). *MDA explained - the Model Driven Architecture: practice and promise*. Addison Wesley object technology series. Addison-Wesley.

Kulkarni, V. and Reddy, S. (2003). Separation of concerns in model-driven development. *IEEE Softw.*, 20(5):64–69.

Levy, F. and Nazarenko, A. (2013). Formalization of natural language regulations through sbvr structured english (tutorial). Morgenstern, L., Stefaneas, P., Levy, F., Wyner, A., Paschke, A. (eds.) RuleML 2013. LNCS, vol. 8035, pp. 19-33. Springer, Heidelberg.

Nijssen, G. (2007). SBVR: Semantics for business.

Njonko, P. and Abed, W. E. (2012). From natural language business requirements to executable models via sbvr. Systems and Informatics (ICSAI), 2012 International Conference on. IEEE.