A Metamodel for Business Rules with Access Control

Lex Wedemeijer
Department of Management, Science and Technology, Open University, Valkenburgerweg 177, Heerlen, The Netherlands
Lex.Wedemeijer@gmail.com

Keywords: Metamodel, Declarative Business Rules, Access Control, Relation Algebra.

Abstract: Business rules outline the way of working with data in today's organizations. We present a metamodel to support and underpin the rule-oriented language to capture business rules that we developed earlier. Like the language, our metamodel is founded on Relation Algebra. The metamodel is compact, and enables the rule designer to record the business rules in their exact details, and to examine the data of the organization for rule violations. Even though such violations should be signalled to the appropriate stakeholders in the business, the access to such signals is subject to access controls, which constitute a special type of business rule. We account for this requirement in our metamodel, so that it captures regular business rules and access permissions alike, and enables to monitor them for violations. A prototype implementation demonstrated the feasibility of our approach.

1 INTRODUCTION

Rule engineering calls for a rule language that is understandable for the intended user community, yet precise enough for subsequent application development. The importance is well argued in the Business Rules Manifesto (2003).

In an earlier paper (Wedemeijer, 2015), we proposed a concise language to specify declarative business rules that consists of just 5 statements. We also proposed a provisional metamodel, but this has since been found to be inadequate. The lack of a proper metamodel hampered the further development of the rule-oriented language and design tools.

The objective of this paper is to present a metamodel to support our language for declarative and state-oriented business rules, including rules for access control, and to provide a solution to the long-standing problem how to reconcile access rules with regular business rules.

Binary Relation Algebra provides the rigorous foundation for our work. The variant we use can be classified as a Description Logic of SHIO type (Baader et al., 2008).

The paper is organized as follows. Section 2 points out some related work. Section 3 explains the major part of our metamodel in conjunction with four of the language statements. The fifth language statement concerns rule enforcement. This is analysed, and an alternative is proposed in section 4. In section 5, we briefly discuss the rules that apply at the metamodel level. Violation of a metamodel-rule signifies an error in the design of the business model, or in a business rule formula. Section 6 presents the completed metamodel. Its integration with the rule language is underpinned by a prototype implementation. Section 7 concludes the paper.

2 RELATED WORK

Declarative business rules have been analysed and modelled in various contexts and from different viewpoints. Rule metamodels as well as access control models are well described in the literature. We mention some approaches that are related to our work on declarative business rules, without attempting to be exhaustive.

2.1 Metamodels for Business Rules

Object Constraint Language (OMG, 2012) is a language to describe constraints on classes defined in UML models. Its emphasis is on implementation, and we consider it too technical for our purpose. The SBVR current standard (OMG, 2015) describes a metamodel for business rules, but covers a much wider area that the declarative and state-oriented business rules that we focus on. Hence, the metamodel is overly complex and does not fit our purpose.

Wedemeijer L.
A Metamodel for Business Rules with Access Control.
DOI: 10.5220/0006222100460053
In *Proceedings of the Sixth International Symposium on Business Modeling and Software Design (BMSD 2016)*, pages 46-53
ISBN: 978-989-758-190-8

Ecore (Steinberg et al., 2008) is a metamodel for the Eclipse programming environment. Specified as an UML model, its aim is to describe models and to provide run-time support for datamodels in Eclipse. As it aims to also support operations, it is not truly state-oriented. Ecore too covers a much wider area than what we focus on, while a genuine notion to represent business rules is lacking.

The Semantic Web Rule Language (Horrocks et al, 2004) is a rule-based approach that employs Ecore as a basis. Semantic Web approaches use the Open-World Assumption whereas our work is based on the Closed-World Assumption (Ceravolo et al., 2007). This fundamental difference is most evident in the 'total' ruletype that requires inspecting all items of an object (class): the rule may be satisfied under Closed-World Assumption, but unknown under the Open-World Assumption.

RAP (Michels, 2015) is a language and metamodel geared to declarative business rules. However, it aims to provide learning support for students in a course for business rules, and the metamodel, tailored to this goal, is implementation dependent causing RAP to be deficient in its support of rule enforcement and access control.

In summary, few of the approaches discussed above support the state-oriented and declarative properties that we think are fundamental in business rules. With the possible exception of RAP, they are not suited to fit our compact rule language.

2.2 Models for Access Control

The contexts of access control and regular business rules are generally regarded as different: primary rules outline what must be done to create user value. Access control rules are secondary rules to outline what is, or what is not allowed in doing so.

Role-Based Access Control (ANSI, 2004) was defined in 2004 and is still being developed. It is a standard for access control that describes a strategy for granting permissions to view data and to perform editing operations. The model stipulates separation of concerns: users, sessions and roles on the one hand, and permissions for data-objects and operations on the other, the two contexts being linked by assigning Permissions.

Relation-Based Access Control (Zhang et al., 2010) is a variant that emphasizes this notion of linking-pin. Notice however that the operations and data-objects covered at the right-hand side of Core RBAC may concern data editing operations as performed by business users and thus subject to user assignments. Complications that may result from this duplicity are ignored in RBAC.

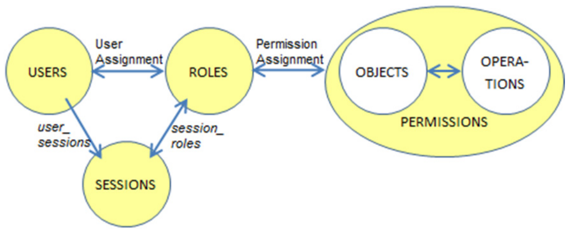

Figure 1: Core RBAC.

Access control rules can, and should be described as regular business rules (Liu et al., 2003). However, access control control are separated from the primary business concepts, and potential matches or links are neglected.

Most access control approaches are preventive in nature, assuming that access attempts without prior permission will automatically fail. In business practice, this is not always how it works. It is often unclear how an approach copes with accesses that have actually occurred without a corresponding permission.

Access control rules may be regarded as rules in their own right. Still, to the best of our knowledge, no standard exists of a joint model combining rules about primary business data (concepts and relations) with secondary rules to control access to that data.

3 METAMODEL

Figure 2 depicts the major parts of the metamodel.

Rectangles represent concepts that in Relation Algebra have no attributes, unlike conventional Relational Algebra. A line with a name represents a relation. Our convention is to let arrowheads point from the domain to the range concept. Although less common, this is useful for designers in writing correct rule formulas. Dotted lines depict specialization-generalization relationships.

The three shaded areas correspond to the language statements as indicated.

Relations in the metamodel are univalent; the exception of [Tuple] *is_in* [Expression] is discussed later. A relation is total if the connecting line begins at the boundary of the domain concept. Otherwise, a line starting out from the interior, the relation is optional. This convention suggests that the relation is total for some specialization that is not depicted explicitly.

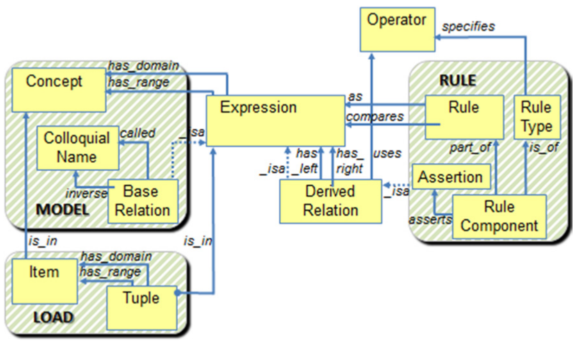

Figure 2: Partial metamodel for Business Rules.

3.1 The MODEL Statement

At the left are the concepts of Concept, BaseRelation and ColloquialName as specified in the MODEL statement. Together, the name, domain and range are unique for each BaseRelation. An inverse name may optionally be given, which must also be unique.

BaseRelation is our preferred name for relations that the designer explicitly specifies. This is to avoid confusion with expressions, constructed by way of Relation-Algebra operations.

Specialization/generalization is captured by way of *_isa* (and inverse name *_asi*) as a reserved ColloquialName, and there are several instances in the metamodel, depicted as dotted lines. A natural restriction for *_isa* relations is that each one must be an injective function.

3.2 Regarding Expressions

Expression is the core concept in our metamodel. It captures binary relations created by way of Relation-Algebra operators. This is where the power of Binary Relation Algebra comes in, and no language statement is needed.

Once base relations are specified, other relations are derived, either by applying a unary operator to one expression, or by applying a binary operator to two expressions. The metamodel captures the two options by way of a left-hand argument and operator, which are always compulsory, and an optional right-hand argument.

The Operator concept is the set of operators such as inverse, negation, union, intersection, and composition, to be used in DerivedRelations. The actual list of available operators is implementation-dependent, and is easily expanded. For instance, a unary operator called 'total' may be implemented to derive all items in the domain concept that do not partake in a certain expression.

Apart from BaseRelations and DerivedRelations, the metamodel also provides for IdentityRelations and nominals. A designer can refer to the Identity Relation for a concept in any language statement without having to define it first.

Nominal expressions are denoted as constant values, i.e. a single or multiple pairs. Although this resembles a population of tuples, nominals are fixed expressions that come without extensions and so cannot be edited.

3.3 The LOAD Statement

At the lower left in figure 2 are the Item and Tuple concepts and their associated relations that record the populations of Concepts and Expressions. A designer needs to specify tuples for BaseRelations only, because Relation Algebra will then determine all tuples for all Expressions according to the expressions' derivation formula. This applies at loadtime, but later at runtime as well.

By exception, the relation in the metamodel from Tuple to Expression is not univalent. An essential property of Relation Algebra is that one tuple can be member of multiple Expressions. By implication, one (set of) tuples can be loaded into more than one BaseRelation at once.

3.4 The RULE Statement

The Rule statement of our language is captured by Rule, RuleType, RuleComponent and Assertion, with corresponding relations.

One variant of the rule statement restricts a single expression, potentially by specifying several constraints at once. For example, stating that an expression MUST BE FUNCTION means that two rule-types apply, 'univalent' and 'total'. Applying objectification as a design pattern (Halpin, 2006), we reify the link between Rule and RuleType into the concept RuleComponent.

The other variant of rule statement compares an expression with another, using the MUST IMPLY comparison.

A minor language improvement is to introduce new comparators MUST INCLUDE, MUST EXCLUDE, and MUST EQUAL, variants that also compare two expressions. The designer can then pick the comparator that makes a rule easy to understand.

Each comparison gives rise to a single Rule-Component, except MUST EQUAL that combines the two variants MUST IMPLY and MUST INCLUDE.

Each RuleComponent is uniquely identified by its rule and type of rule. It comes with exactly one assertion, the derived relation containing all tuples that violate the RuleComponent. Rules need not be unique, which is why a single assertion may link to more than one rule component.

3.5 The EXPLAIN Statement

The metamodel accounts for the EXPLAIN statement of our language by one concept, Text, related to the Concept, BaseRelation, and Rule concepts. We specify the relations as optional because textual explanations contribute to the users' understanding of the metamodel, but not to its formal consistency, correctness, or completeness. And although some form of organization or coherence in the texts may be desirable, we consider such nice-to-have documentary features beyond the scope of this paper.

3.6 regarding the ENFORCE Statement

Ideally, a business adheres to all of its rules at all times, and no violations occur. In the context of our metamodel, it means that all assertions ought to remain unpopulated. In a live business environment, rules may sometimes be violated, so that some assertions will record violations. If our purpose was only to develop a declarative, state-oriented metamodel for business rules, then the passive capability to record violations would suffice. However, a running business calls for more than that. The problems and alternatives for rule enforcement are discussed in the next section.

3.7 Discussion

The metamodel presented in figure 2 accounts for four of the five statements of our language. Consisting of only 12 concepts and 20 relations (*_isa* relations included), it is compact but enables the rule designer to model business concepts and relations, and to specify declarative business rules about them.

Rule violations are recorded as (sets of) tuples in assertions and are avialable for inspection by the business stakeholders in charge of remedying the violations.

4 RULE ENFORCEMENT AND ACCESS CONTROL

The metamodel will record rule violations, but does not enforce the rules. No mechanism is provided to guarantee rule adherence in the running database environment. Indeed, it is implementation-dependent how violations will be determined or prevented. This section explains how we extended the metamodel to support rule enforcement, albeit not in full.

4.1 Analysis of ENFORCE Strategies

We now take a closer look at the three strategies to specify enforcements, proposed in our language as:

```
ENFORCE Rule AS Reject/Report/Resolve
```

ENFORCE AS REJECT is easy to handle. Whenever an attempt is made to change the current state of the database such that a new rule violation would emerge, then the change is rejected and the database state remains unchanged. It reflects the view that the population of the database after the change, with this violation, cannot possibly be true in the real world. In practice, this view may be valid for some rules, but certainly not for all.

The ENFORCE AS REPORT statement is an obligation to report violations to the stakeholder(s) in charge of monitoring rule compliance, and also a permission for the stakeholder(s) to read them. But the obligation to report specifies compulsory action, which is inconsistent with our purpose of declarative language and metamodel. Also, it tacitly assumes that the metamodel contains a stakeholder concept; this is even an explicit assumption in the RBAC standard discussed above.

In practice however, the stakeholder role is assumed by business people, and this role, or these business people, may already be captured as a concept. In such cases, adding a Stakeholder concept to the metamodel results in undesired redundancy. Each change in organization, stakeholdership, or monitoring responsibilities, would require an update in the metamodel. In our experience (Wedemeijer, 2002), such changes in the way of doing business are rather frequent, much more so than changes in rules, relations or concepts.

ENFORCE AS RESOLVE as the third strategy also poses serious difficulties. In our earlier paper we pointed out that this strategy is not declarative but imperative, and not state- but transaction-oriented in character. Moreover, it assumes that the stakeholder who performs the data edit to resolve the violation, has permission to access that data.

Enforcements are rules about rules. Enforcement rules in many practical business environments are phrased as imperative rules, whereas we are looking for declarative business rules. They constitute rules in their own right and so should be handled in much the same way as ordinary business rules. Because rule enforcement strategies implicate that rule violations need to be accessed, we turn our attention to access control. Our aim is to capture and integrate the rules for data access into the metamodel as regular business rules.

4.2 Permission to Access

Rule enforcement is dependent upon access control. To report violations to some stakeholder, assumes that the stakeholder is permitted to read the violations, and that proper edit permits are granted for taking corrective actions.

In accordance with the RBAC standard, our access control rule is simply: "access to data requires permission to access that data". This applies to all expressions, not only to the assertions associated to some business rules.

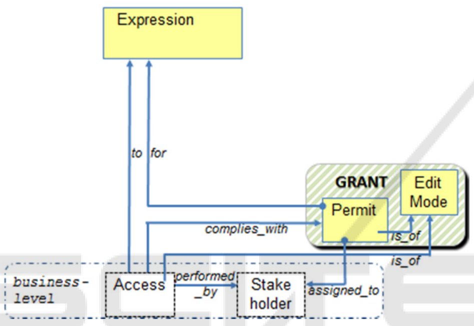

Figure 3: Permit and EditMode concept in the metamodel. Access and Stakeholder belong to the business level.

Figure 3 shows how the metamodel is expanded with a Permit and EditMode concept to cover the 'permission assignment' relation of RBAC. A permit allows a user to access the data recorded for that expression(s). Obviously, an expression cannot be validly accessed if no permits are granted for it, and so its contents remains hidden to the users.

EditMode is optional, to tailor permissions to various modes of data editing (add, change, delete). The default access mode is 'read' which does not refer to an editmode. Expressions in general cannot be edited, and editmodes should be made to apply only to BaseRelations or (the IdentityRelation of) Concepts. Evidently, an edit permit automatically implies read permission for that expression.

4.3 Access Control Rules

Access control requires keeping track of all data access. But keeping score of all data accesses comes down to record the online transaction processing as performed in the running business database environment: we do not suggest to duplicate this within the context of our rule metamodel which would become very volatile.

Instead, we assume that some concept of Access is defined at the business model level (figure 3). In the running database, each access should be recorded: to which expression, and compliant to which permit. These relations, transgressing the business/metamodel boundary, support the decision-making process in the operational database to allow, or deny access.

Next, we also assume two relations: [Access] *to* [Expression] and [Access] *complies_with* [Permit]. Both relations will be functions (univalent and total), if the Access concept is defined suitably.

Of course, other and more complicated concepts and rules for data access may exist in more realistic business environments. And there is more to it. Access to an expression requires that the access comes under a permit for that exact expression:

```
RULE 123-read-control AS
   [Access] to [Expression]
MUST IMPLY
   [Access] complies_with [Permit]
     composition
   [Permit] for [Expression]
```

The statement follows the language conventions set out in our earlier paper (Wedemeijer, 2015). The rule-identifier, 123-read-control, is arbitrary.

If access involves editing, then an extra restriction is that the permit allows the correct editmode:

```
RULE 456-edit-control AS
   [Access] is_of [EditMode]
MUST IMPLY
   [Access] complies_with [Permit]
     composition
   [Permit] is_of [EditMode]
```

4.4 Access Control for Stakeholders

A next extension is to account for stakeholders who actually perform the data accesses. In ordinary business models, stakeholders go under a wide variety of names: users, employees, departments, or whatever. Most approaches for access control, and the RBAC standard is no exception, capture such concepts in a separate business model. Our approach leaves it to the business designer to specify in the business datamodel who the permissions are granted to, and to specify the volatile relations of these concepts with Permit. This allows the designer to merge this important aspect into the overall business model, or to employ a secondary model.

To outline how this works out in practice, we assume for the sake of this paper that a single Stakeholder concept is identified in the business datamodel as in figure 3. We also assume the [Access] *performed-by* [Stakeholder] relation which is univalent and total, and relation [Permit] *assigned-to* [Stakeholder] which may be many-to-many. The rule that access requires permission now comes down to:

```
RULE 789-access-control AS
  [Access] performed-by [Stakeholder]
MUST IMPLY
  [Access] complies_with [Permit]
    composition
  [Permit] assigned-to [Stakeholder]
```

Notice how these rules apply at the level of the business model, and violations will appear in a regular Assertion. From a business point of view, the only peculiarity is that the Expression and Permit concept and contents are fixed at the metamodel level, and cannot be edited on the fly.

Combined, the rules 123, 456 and 789 support all three enforcement strategies, as follows. If no permit was issued for a rule, then viewing violations is impossible so no violations should emerge, which comes down to the "reject" strategy. If some permit was issued, then stakeholders with a proper permit can obtain a report of the violations, which is the "report" strategy. If an edit permit was issued, then the stakeholder may proceed to add, change or delete a violating tuple, executing the "resolve" strategy.

4.5 GRANT Statement Replaces ENFORCE

As the ENFORCE language statement is inadequate, we now propose as alternative the GRANT statement. It records permit-identifiers, and relates them to one or more expressions in the metamodel. One variant issues permits for reading, the other for editing:

```
GRANT Permit FOR Expression(s)/Rule(s)

GRANT Permit FOR BaseRelation(s)
 IN EditMode
```

For the convenience of the business designer, a single permit may be granted for several expressions and (assertions corresponding to) rules at once, or one edit permit for multiple base relations. Once the permits are established in the metamodel, they can be assigned to any roles or users as defined in the business model, at load-time or at runtime of the operational database.

The statement specifies access control for all expressions, not only for rule assertions. Rather, an enforcement strategy can be inferred from the permissions granted for assertions.

If no permit is granted for an assertion, then violations cannot be viewed and so should not exist. Hence, the rule ought to never be violated, and the reject strategy applies. A read permit granted for an assertion means that rule violations are possible, and some stakeholders are probably able to view them: this is the report strategy. If an edit permit is granted, then the resolve-strategy applies.

4.6 Discussion

Our way of granting access permissions has great advantage. The point is that a distinct assertion will record the violations of the access-control rule. A distinct read permit is required to inspect access violations, which must be defined in the metamodel and then assigned to stakeholders in the business.

For example, access to some data may be performed by a stakeholder without the proper permission, resulting in a violation of rule 789-access-control. One possible way to resolve the violation is by having the permission assigned to the stakeholder belatedly. This is a great feature for data auditors, and rarely supported in other approaches for Access Control.

The granting of permissions resembles the RBAC standard that also envisions separation of concerns, but ignores the potential overlap of (user) roles with objects in the business model. Our approach allows such overlap, and puts the business designer in charge of avoiding possible duplicity.

The main advantage is that access control is now relegated to the level of business model, where permits can be assigned, changed and withdrawn at any time, without affecting the metamodel. Thus, access rule violations can be handled in the same way as violations of ordinary business rules.

One may argue that a mere permission to read violations, does not ensure that a stakeholder will actually do so. But this is equally true for the ENFORCE AS REPORT statement: there is no recordkeeping of violations that have or have not been been viewed. Would a rule be imposed that each violation must be viewed, then each emerging violation constitutes a violation of this rule. This is inferior design: the simultaneous emergence of two violations is a clear update-anomaly.

5 METAMODEL-RULES

In any model of data, rules apply to ensure correct ness and consistency within the modelled context. A rule violation signifies that some data is flawed, and should be remedied by the stakeholder. In our metamodel, violation of a metamodel-rule signifies that some feature of the business model is flawed, and should be remedied by the designer.

5.1 Cardinality

Univalence applies to almost all relations in the metamodel. The exception in relation [Tuple] *is_in* [Expression] was already pointed out. A second exception is the [Permit] *for* [Expression] relation. Notice that arbitrary cardinalities may apply to relations involving a Stakeholder, but that concept is part of the business model, not the metamodel.

Many, but not all relations are total. An important optional relation concerns permits that not always involve an editmode. Other relations that are not total are: the inverse name of BaseRelation; the right-hand side in a derivation formula for a derived relation; and the relations for explanatory texts. The three *_isa* relations in our metamodel combine univalence, totality, and injectivity.

5.2 Compound Rules

Compound rules involve more than just one relation.

In our metamodel, an example is the short loop between Rule and Expression: the *compares* relation must always refer to a different expression than the *as* relation, so as to avoid trivial and contradictory formulas.

A more interesting example is the requirement of referential integrity. Not only must the items in a tuple be on record for some concept, they must exist in the correct concept. This boils down to two compound rules, one about domains and the other about ranges:

```
RULE 0-referential-integrity-domain AS
  [Tuple] is_in [Expression]
    composition
  [Expression] has_domain [Concept]
MUST EQUAL
  [Tuple] has_domain [Item]
    composition
  [Tuple] is_in [Concept]
```

Another consistency rule for a metamodel loop concerns [Rule] *compares* [Expression]. Whenever a rule uses *compares*, then must its corresponding assertion, as a DerivedRelation, have a *has_right* [Expression] clause that uses that exact same expression.

A more intricate rule concerns the Operator concept. The operator used in the assertion for a rule component, is fully determined by that components' RuleType. For example, if a rule component expresses that some expression is 'total', then the corresponding assertion must use the specific 'total' unary operator.

Apart from compound rules that apply to loops in the model, other compound rules can be pointed out. For instance, a compound uniqueness (identifying) rule applies for BaseRelations: each is uniquely identified by its ColloquialName, plus its domain and range concept. Likewise, each Tuple is uniquely identified by the combination of its domain item, and its range item.

Still another kind of rule concerns *_isa* hierarchies: [Assertion] *_isa* [DerivedRelation], composed with [DerivedRelation] *_isa* [Expression], must coincide with the [Assertion] *_isa* [Expression] relation. The latter relation is not depicted, but it does exist as a proper specialization/generalization.

6 THE INTEGRATED VIEW

Figure 4 depicts the compact metamodel to support our business rules language. The ENFORCE statement of our earlier language version was found to be imperative, and too volatile. Focusing on access control only, we defined a GRANT statement for access permits, to regain the declarative and state-oriented character of the language and to reduce volatility.

6.1 Language and Metamodel

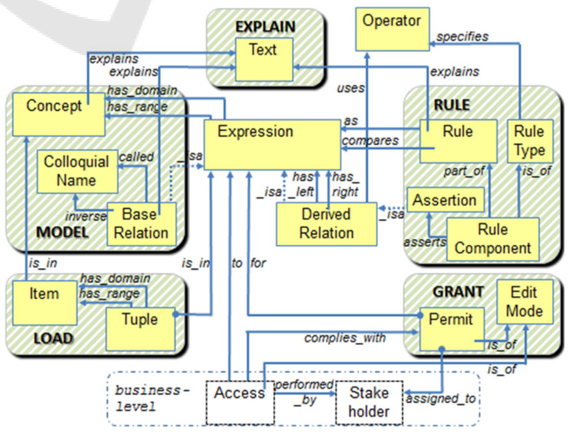

Figure 4: Full Metamodel for Business Rules.

The current language supports five statements: MODEL, RULE, EXPLAIN, LOAD and GRANT. Jointly, they enable a business designer to set up a model, to specify rules, to provide explanations, to load initial data, and to determine access control. The supporting metamodel is expressive yet compact, as twelve concepts suffice to capture a design.

Remind that Expression, DerivedRelation and Operator need not be specified by the designer as they come for free by virtue of Relation Algebra. And although a Stakeholder and Access concept are depicted, these concepts and relations are not part of the metamodel. Permit assignation to stakeholders, and actual access of data, should be recorded at the business level. In our experience, this will considerably reduce the number of changes in the metamodel.

6.2 Demonstration

The metamodel and rules constitute a business model just like any other, perhaps with a somewhat peculiar context. Hence, it can be expressed in our rule-oriented language, and captured as a regular datamodel in the metamodel itself, in a reflective fashion.

Feasibility of our approach is demonstrated in this way, by implementing the metamodel and its complete set of rules in a prototyping rule-based engineering environment. The result is available for download at wiki.tarski.nl/index.php/Research_hub.

7 CONCLUSION

The metamodel defines the information structure underlying our language for declarative business rules, and also covers rules for access control.

Binary Relation Algebra is used as theoretical fundament for exact rule specifications. This formalism however does not support numerical, temporal, or spatial capabilities for rules. Nor is the metamodel designed for performance or scalability, and no efficient algorithm is proposed to determine rule violations. Deployment will call for a distinct development step to transform the business model to a proper database schema that takes requirements into account such as performance, data distribution, federation across hardware platforms, security, interoperability etc.

The metamodel ensures separation of concerns, so that business users can add, edit and delete the operational data, provided that a proper permit was assigned to them. If not, the violation of the access control rule is captured in a rule assertion, just like any other business rule violation.

A salient point of the metamodel is that dependence on role and permit assignments is minimal. Thus, it provides a stable environment to capture and describe business rules. Volatility due to everyday changes in organizations is relegated from the metamodel to the level of the business model.

Our approach handles the primary business rules and the rules for access control in exactly the same way, an elusive goal in business rules engineering attested to in the Business Rules Manifesto. The metamodel for business rules with access control presented in this paper indicates how this goal may be reached.

REFERENCES

ANSI, 2004. *Role Based Access Control.* American Nat. Standard for Information Technology INCITS 359. available at csrc.nist.gov/groups/SNS/rbac/

Baader, Horrocks, Sattler, 2008. *Description logics*. In: van Harmelen, Lifschitz, Porter. (eds.). Elsevier Handbook of Knowledge Representation, pp 135–179

Business Rules Manifesto, 2003. *Version 2.0.* available at www.businessrulesgroup.org/brmanifesto.htm

Ceravolo, Fugazza, Leida, 2007. *Modeling Semantics of Business Rules* Inaugural IEEE-IES Digital Eco-Systems and Technologies Conference, pp. 171-176

Halpin, 2006. *Objectification of relationships.* Advanced Topics in Database Research. Vol 5 no 5 pp 106-123

Horrocks, Patel-Schneider, Boley, Tabet, Grosof, Dean, 2004. *SWRL: A Semantic Web Rule Language Combining OWL and RuleML*. Nat.Research Council of Canada. available at www.w3.org/Submission/SWRL/

Liu, Sun, Barjis, Dietz, 2003. *Modelling dynamic behavior of business organisations.* Knowledge-Based Systems, pp. 101-111

Michels, 2015. *Development Environment for Rule-based Prototyping*. Dissertation. available at portal.ou.nl/documents/114964/31994420/Development_Environment_for_Rule-based_Prototyping.pdf

Object Management Group, 2012. *Object Constraint Language.* available at www.omg.org/spec/

Object Management Group, 2015. *SBVR: Semantics of Business Vocabulary and Business Rules*, Version 1.3. available at www.omg.org/spec/

Steinberg, Budinsky, Paternostro, Merks, 2008. *Eclipse Modeling Framework, 2nd Ed.* Addison-Wesley

Wedemeijer, 2002. *Exploring Conceptual Schema Evolution.* Dissertation. ISBN 90-5681-142-8. available at repository.tudelft.nl/

Wedemeijer, 2015. *A Language to Specify Declarative Business Rules.* Springer Lecture Notes in Business Information Processing. Vol 220, pp 82-101

Zhang, Giunchiglia, Crispo, Song, 2010. *Relation-Based Access Control: An Access Control Model for Context-Aware Computing Environment*. Wireless Personal Communications. Vol 55, pp 5-17