Fact Calculus: Using ORM and Lisa-D to Reason About Domains

S.J.B.A. Hoppenbrouwers, H.A. (Erik) Proper, and Th.P. van der Weide

Institute for Computing and Information Sciences, Radboud University,
Toernooiveld 1, 6525 ED Nijmegen, The Netherlands, EU
{S.Hoppenbrouwers, E.Proper, Th.P.vanderWeide}@cs.ru.nl

Abstract. We propose to use ORM and Lisa-D as means to formally reason about domains. Conceptual rule languages such as Lisa-D, RIDL and ConQuer allow for the specification of rules in a semi-natural language format that can more easily be understood by domain experts than languages such as predicate calculus, Z or OCL. If one would indeed be able to reason about properties of domains in terms of Lisa-D expressions, then this reasoning would be likely to be better accessible to people without a background in formal mathematics, such as "the average" domain expert. A potential application domain for such reasoning would be the field of *business rules*. If we can reason about business rules formulated in a semi-natural language format, the formal equivalence of (sets of) business rules (i.e. various paraphrasings) can be discussed with domain experts in a language and a fashion that is familiar to them.

1 Introduction

We will propose and explore initial ideas about a fact-based approach to reasoning based entirely on concepts that are familiar to a domain expert, as modeled through ORM/Lisa-D. We show how ORM models [5], combined with and covered by non-graphical Lisa-D expressions [6], can be subject to reasoning using an alternative type of reasoning rule. As will be explained, these reasoning rules are based on "information descriptors" [6]. However, the system is still rooted in classical predicate logic. This paper proposes to use ORM and Lisa-D as a means to formally reason about domains. Conceptual rule languages such as Lisa-D [6], RIDL [9] and ConQuer [1] allow for the specification of rules in a semi-natural language format that can be more easily understood by domain experts than languages such as predicate calculus, Z [11] or OCL [12].

A long term ambition of ours is to relate formal reasoning to styles of (communication about) reasoning close to reasoning in *contextualized* (i.e. domain-related) Natural Language (NL) [8], without loosing formal functionality. Metaphorically, fact calculus (and our communication-oriented approach to representing it) could be presented as a "next generation approach of dealing with symbol-based reasoning" as opposed to reasoning systems more directly related to formal logic. If one would indeed be able to reason about properties of domains in terms of Lisa-D expressions, then this reasoning would be likely to be better accessible to people without a background in formal mathematics, such as "the average" domain expert. A potential application domain for such reasoning would be the field of *business rules*. When reasoning can be done at the level of business rules formulated in a semi-natural language format, the equivalence of (sets of) business rules can be discussed with domain experts in a language that is familiar to them.

The aim of the modeling process as we see it [8] is to find a representation mechanism for sentences from some domain language. The result may be seen as a signature for a formal structure. Each elementary fact type is a relation in this structure. Assuming a set of variables, we can introduce the expressions over this signature [2]. This signature forms the base of a formal theory about a domain. The constraints then are seen as the axioms of this theory.

In classic reasoning (for example, natural deduction [3]), formal variables are used, through which we can formulate statements and try to prove them from the axioms, using a conventional reasoning mechanism (containing for example *modus ponens*). In this paper we explore an alternative approach, based on specific models (populations), and focus on properties of some particular population.

Statements about a current population can be represented as Lisa-D statements. For clarity's sake, we include "translations" in regular English for every Lisa-D statement. The translations sometimes leave out redundant information that is nevertheless vital in reasoning about information descriptors. Though such translation cannot currently be produced deterministically or even automatically, we do intend to explore ways of achieving this in the future.

For example, consider the following statement, that could be a query:

EACH Student living in City 'Elst' MUST ALSO BE attending Course 'Modeling'
Each student that lives in the city of Elst also attends the course Modeling

2 Fact Calculus

2.1 Starting from Predicate Calculus

In existing approaches, reasoning in terms of a conceptual model is closely related to reasoning in predicate calculus [2, 6]. This form of reasoning is instance related, for example transitivity of the relation f is expressed as:

$$\forall_{x,y,x}\left[f(x,y) \wedge f(y,z) \Rightarrow f(x,z)\right]$$

Using instances leads to a style of reasoning that may be qualified as *reasoning* **within** *the Universe of Discourse*. Using a conceptual language such as Lisa-D provides the opportunity to reason without addressing particular instances. This may be characterized as *reasoning* **about** *the Universe of Discourse*.

First it should be noted that a precise formulation of domain rules may require a way of referring to general instances in order to describe concisely their relation. In daily practice people use NL mechanisms to make such references. However, in many situations domain rules can be nicely formulated without addressing any particular instance. As an example, consider the rule (phrased in NL):

When a car is returned, then its official documents should also be returned

In this sentence, without explicitly addressing any particular car, the subtle use of the reference *its* provides a sufficient reference.

The main idea behind Lisa-D, as present in its early variant RIDL [9] is a functional, variable-less description of domain-specific information needs. In Lisa-D the mechanism of variables is of a linguistic nature. Variables are special names that can be substituted once they are evaluated in a context that generates values for this variable. The set comprehension construct is defined in that way. Lisa-D expressions are based on a domain-specific lexicon, that contains names for the elements that constitute a conceptual schema. The lexicon contains a name for each object type, and also provides names for the construction mechanism in a conceptual schema (such as the roles and object types it contains).

2.2 Information Descriptors

The names in the lexicon are on par with the words from which NL sentences are constructed. Lisa-D sentences are referred to as information descriptors. The base construction for sentences is juxtaposition. By simply concatenating information descriptors, new information descriptors are constructed. Before describing the meaning of information descriptors, we will first discuss how such descriptors will be interpreted.

The semantics of Lisa-D can be described in various ways. The simplest form is its interpretation in terms of set theory. Other variants use bags, fuzzy sets, or probabilistic distributions. In order to make a bridge with predicate calculus, we will view information descriptors as binary predicates.

Let D be an information descriptor, and P a population of the corresponding conceptual schema. We then see this information descriptor as a binary predicate. Let $\mathcal{V}$ be a set of variables, then we write $P \models x \llbracket D \rrbracket y$ to express that in population P there is a relation between x and y via D, where $x, y \in \mathcal{V}$. For each object type O we introduce a unary predicate: $P \models O(x)$ iff $x \in P(O)$ and for each role R of some fact type F we introduce a binary predicate: $P \models R(x, y)$ iff $y \in P(F) \wedge x = y(R)$. Note that the instances of a fact type (y) are formally treated as functions from the roles of fact type F to instances of the object types involved in a role. In other words, $y(R)$ yields the object playing role R in fact y.

At this point we can describe the meaning of elementary information descriptors as follows. Let o be the name of object type O and r the name of a role R, then o and r are information descriptors with semantics:

$$x \llbracket o \rrbracket y \triangleq O(x) \land x = y \qquad\qquad x \llbracket r \rrbracket y \triangleq R(x,y)$$

Roles involved in fact types may actually receive multiple names. This is illustrated in figure 1. A single role may, in addition to its 'normal' name, also receive a *reverse role name*. Let v be the reverse role name of role R, then we have:

$$x \llbracket v \rrbracket y \triangleq R(y,x)$$

Any ordered pair of roles involved in a fact type may receive a *connector name*. The connector names allow us to 'traverse' a fact type from one of the participating object types to another one. If c is the connector name for a role pair $\langle R, S \rangle$, then the semantics of the information descriptor c are defined as:

$$x \llbracket c \rrbracket z \triangleq \exists_y \left[R(x,y) \land S(y,z) \right]$$

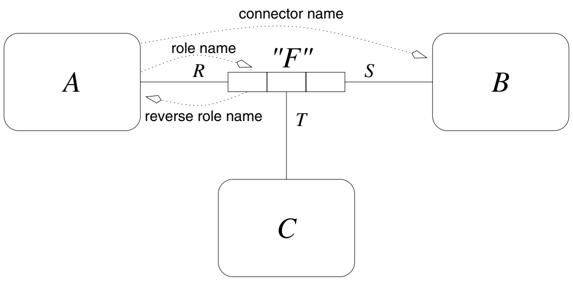

Fig. 1. Role names

Elementary information descriptors can be composed into complex information descriptors using constructions such as *concatenation*, *conjunction*, *implication*, *disjunction* and *complement*. These may refer to the fronts alone or both fronts and tails of descriptors. In this paper we will use:

$$\begin{aligned}
x \llbracket D_1 \; D_2 \rrbracket y &\triangleq \exists_z \left[x \llbracket D_1 \rrbracket z \land z \llbracket D_2 \rrbracket y \right] \\
x \llbracket D_1 \,\textsf{AND ALSO}\, D_2 \rrbracket y &\triangleq \exists_z \left[x \llbracket D_1 \rrbracket z \right] \land \exists_z \left[x \llbracket D_2 \rrbracket z \right] \land x = y \\
x \llbracket D_1 \,\textsf{MUST ALSO BE}\, D_2 \rrbracket y &\triangleq \exists_z \left[x \llbracket D_1 \rrbracket z \right] \Rightarrow \exists_z \left[x \llbracket D_2 \rrbracket z \right] \land x = y \\
x \llbracket D_1 \,\textsf{OR IS}\, D_2 \rrbracket y &\triangleq \exists_z \left[x \llbracket D_1 \rrbracket z \right] \lor \exists_z \left[x \llbracket D_2 \rrbracket z \right] \land x = y \\
x \llbracket D_1 \,\textsf{BUT NOT}\, D_2 \rrbracket y &\triangleq \exists_z \left[x \llbracket D_1 \rrbracket z \right] \land \neg \exists_z \left[x \llbracket D_2 \rrbracket z \right] \land x = y
\end{aligned}$$

where D_1 and D_2 are information descriptors and x, y and z are variables. Some example expression would be:

Person working for Department 'I&KS'
People working for department 'I&KS'

Person (working for Department 'I&KS' AND ALSO owning Car of Brand 'Seat')
People working for department 'I&KS' who also own a car of brand Seat

Person (working for Department 'I&KS' MUST ALSO BE owning Car of Brand 'Seat')
People who, if they work for department 'I&KS', also own a car of brand 'Seat'

Person (owning Car of Brand 'Seat' OR IS living in City 'Nijmegen')
People who own a car of brand Seat, or live in the city of Nijmegen

Person (working for Department 'I&KS' BUT NOT living in City 'Amsterdam')
People working for department 'I&KS', but who do not live in the city of Amsterdam

Correlation operators form a special class of constructs:

$$x\,[\![D\,\mathsf{THAT}\,o]\!]\,y \triangleq x\,[\![D\;o]\!]\,y \wedge x = y$$
$$x\,[\![D\,\mathsf{MUST\ BE\ ANOTHER}\,o]\!]\,y \triangleq x\,[\![D\;o]\!]\,y \wedge x \neq y$$

where D is an information descriptor and o is the name of an object type. Some examples of its use are:

Person working for Department having as manager THAT Person
People who work for a department that has that person as a manager

Person owning Car having Brand being of Car being owned by MUST BE ANOTHER Person
People who own a car of the same brand as another person's car

To make some Lisa-D expressions more readable, we also introduce dummy words AN and A which have no real meaning:

$$\mathsf{AN}\,P \;\triangleq\; \mathsf{A}\,P \;\triangleq\; P$$

Using these 'dummy words' the last two Lisa-D expressions can be re-phrased as:

Person working for A Department having as manager THAT Person
Person owning A Car having Brand being of A Car being owned by MUST BE ANOTHER Person

In Lisa-D many more constructions exist to create complex information descriptors. However, in this paper we limit ourselves to those constructions that are needed for the considerations discussed below. Note again that the natural language likeness of the Lisa-D expressions used in this paper can be improved considerably. For reasons of compactness, this paper defined fact calculus directly in terms of (verbalizations of) information descriptors. However, in the original Lisa-D path expressions were used as the underlying skeleton for rules, where the information descriptors 'merely' serve as the flesh on the bones. Using linguistic techniques as described in e.g. [7, 4] this 'flesh' can obtain a more natural structure. Future work will also concentrate on improved verbalizations of path expressions.

2.3 Rules

Lisa-D has a special way of using information descriptors to describe rules that should apply in a domain. These rules can be used to express constraints and/or business rules. We will use the more general term *rule* for such expressions. These rules consist of information descriptors that are interpreted in a boolean way;

i.e. if no tuple satisfies the predicate, the result is false, otherwise it is true. This leads to the following semantics for rules:

$$\begin{array}{rcl@{\qquad}rcl}
[\![\mathsf{EACH}\,D]\!] & \triangleq & \forall_x \exists_y \, [x \, [\![D]\!] \, y] & [\![\mathsf{SOME}\,D]\!] & \triangleq & \exists_{x,y} \, [x \, [\![D]\!] \, y] \\
[\![R_1 \,\mathsf{AND}\, R_2]\!] & \triangleq & [\![R_1]\!] \wedge [\![R_2]\!] & [\![R_1 \,\mathsf{OR}\, R_2]\!] & \triangleq & [\![R_1]\!] \vee [\![R_2]\!] \\
[\![R_1 \,\mathsf{IMPLIES}\, R_2]\!] & \triangleq & [\![(\mathsf{NOT}\, R_1)\,\mathsf{OR}\, R_2]\!] & [\![\mathsf{NOT}\, R_1]\!] & \triangleq & \neg \, [\![R_1]\!] \\
[\![\mathsf{NO}\, D]\!] & \triangleq & [\![\mathsf{NOT}\,\mathsf{SOME}\, D]\!] & & &
\end{array}$$

where D is an information descriptor and R_1, R_2 are rules.

Note that the $\exists$ and $\forall$ quantifications in the $\mathsf{EACH}\,D$ and $\mathsf{SOME}\,D$ constructs range over all possible instances. Limiting a variable to a specific class of instances is done similar to set theory, where:

$$\forall_{x \in D} \, [P(x)] \;\triangleq\; \forall_x \, [x \in D \Rightarrow P(x)] \;\triangleq\; \forall_x \, [D(x) \Rightarrow P(x)]$$

In our case we would typically write: $\mathsf{EACH}\,T\,\mathsf{MUST\ ALSO\ BE}\,C$ where T is an information descriptor representing the domain over which one ranges and C is the condition.

In the context of rules, we will also use the following syntactic variations of THAT and $\mathsf{MUST\ BE\ ANOTHER}$:

$$\begin{array}{rcl}
x \, [\![D \,\mathsf{THAT}\, o]\!] \, y & \triangleq & x \, [\![D \,\mathsf{MUST\ BE\ THAT}\, o]\!] \, y \\
x \, [\![D \,\mathsf{MUST\ BE\ ANOTHER}\, o]\!] \, y & \triangleq & x \, [\![D \,\mathsf{ANOTHER}\, o]\!] \, y
\end{array}$$

3 Examples of Rule Modeling

In this section we provide some illustrations of ways one can reason within the fact calculus. At the moment most of the reasoning can be done by 'jumping down' to the level of predicate calculus. It indeed makes sense to also introduce derivation rules at the information descriptor and rules level. This is, however, beyond the scope of the short discussion provided in this paper.

3.1 Example: Trains and Carriages

As a first example of the use of fact calculus to reason about domains, consider the following two rules:

EACH Train MUST ALSO BE consisting of A Carriage
Each train consists of carriages

EACH Carriage MUST ALSO BE having A Class
Each carriage has a class.

Using predicate calculus based inference, one could infer:

EACH Train MUST ALSO BE consisting of A Carriage having A Class
Each train consists of carriages that have a class.

3.2 Example: Return of Cars and Papers

Now consider the rule (phrased in NL) that was mentioned earlier in this paper.

When a car is returned, then its official documents should also be returned.

First we note that this rule may be characterized as an action rule; it *imperatively* describes what actions are required when returning a car. From our perspective, however, we prefer to focus on *declarative* characterizations of the underlying Universe of Discourse, as also advocated in [10–article 4].

As a first step, we formulate a declarative characterization. In some cases a positive formulation works best; in this case we choose to use negation, and derive from the action rule the situation that the follow up action of this rule is intending to avoid:

NO Car being returned AND ALSO having (AN Official-paper BUT NOT being returned)
In no case a car may be returned, if papers belonging to that car are not also returned.

In order to make this expression more readable it is sensible to re-balance the order of appearance of the types in the expression. By focussing on the role of the Official-papers we would get:

NO Official-paper of A Car being returned BUT NOT being returned
In no case may papers of a returned car not also be returned.

Using predicate calculus we can formally declare the equivalence of these statements, while the latter formulation (paraphrasing) is more natural.

Via a negative to positive transformation we can also obtain:

EACH Official-paper from A Car being returned MUST ALSO BE being returned
All papers from returned cars must also be returned.

3.3 Example: Car Registration

As another example, we consider the reasoning that leads to the conclusion that: each person has a license plate, under the assumption that each person has a car and that each car has a license plate. We are thus given the following rules:

1. Each person must own a car
2. Each car must be registered by a licence plate

The resulting rules will serve as *domain axioms* for our reasoning system:

EACH Person MUST ALSO BE owning Car
Each person must own a car

EACH Car MUST ALSO BE being registered by License plate
Each car must be registered by a licence plate

The reasoning rule we apply to combine these two rules is the *First Implication Rule*:

$$\frac{D_1 \text{ MUST ALSO BE } D_2\ D_3 \qquad D_3 \text{ MUST ALSO BE } D_4}{D_1 \text{ MUST ALSO BE } D_2\ (D_3 \text{ MUST ALSO BE } D_4)}$$

where D_1, D_2, D_3 and D_4 are information descriptors. Before proceeding with the example, we first will show the validity of the First Implication Rule (see figure 2). Suppose $x\,[\![D_1]\!]\,p$ for some x and p. Then we can conclude from the first rule that also $x\,[\![D_2\ D_3]\!]\,r$ for some r, and thus for some q we have $q\,[\![D_3]\!]\,r$. Applying the second rule leads to the conclusion $q\,[\![D_4]\!]\,y$ for some y. Combining the arguments leads to: $x\,[\![D_2\ (D_3\ \textsf{MUST ALSO BE}\ D_4)]\!]\,y$.

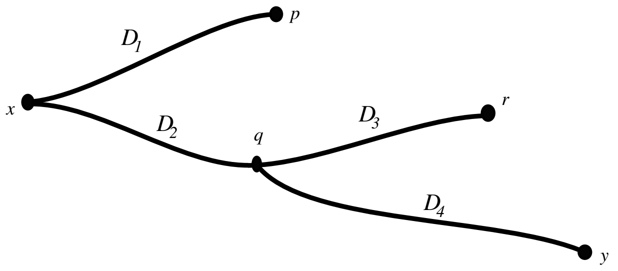

Fig. 2. Situation

Applying the above two reasoning rules by setting:

D_1 = Person	D_2 = owning
D_3 = Car	D_4 = being registered by License plate

we obtain the validity of the following expression in each population:

EACH Person MUST ALSO BE owning (Car MUST ALSO BE being registered by License plate)

Each person must own a car, and that car must be registered by a licence plate.

Furthermore, we also have the following *Absorbtion Rule*:

$$\frac{o\ \textsf{MUST ALSO BE}\ D}{o\ D}$$

where D is an information descriptor, while o is an object type name. This rule allows us to rewrite the previous expression to:

EACH Person MUST ALSO BE owning Car being registered by License plate

Each person must own a car that must be registred by a licence plate.

3.4 Example: Car Ownership

Next we focus on uniqueness within binary relationships. The following rule expresses the rule that a person may not own more than one car:

NO Car being owned by A Person owning MUST BE ANOTHER Car

No car may be owned by a person who also owns another car

With respect to binding rules, the concatenation operator has the highest priority. Furthermore, the MUST BE ANOTHER operator has a higher priority than the NO operator. As a consequence, we get the following binding structure:

NO ((Car being owned by A Person owning) MUST BE ANOTHER Car)

We first will transform this rule into a positive formulation. This positive formulation is better suited for reasoning, but is less obvious to understand from an intuitive point of view.

EACH Car being owned by Person owning Car MUST BE THAT Car
Each car that is owned by a person owning a car, must be this person's only car

Next we add the rule that a licence plate may be associated with at most one car, or in its positive formulation:

EACH Licence plate registering A Car being registered by A Licence plate MUST BE THAT License plate
Each licence plate that registers a car, must be a unique licence plate for that car

In order to combine these two uniqueness constraints, we apply the following reasoning rule:

$$\frac{\textsf{EACH}\, D_1\ o_1\ D_2\, \textsf{MUST BE THAT}\, o_2 \qquad \textsf{EACH}\, D_3\, \textsf{MUST BE THAT}\, o_1}{\textsf{EACH}\, D_1\ D_3\ D_2\, \textsf{MUST BE THAT}\, o_2}$$

where D_1, D_2 and D_3 are information descriptors, while o_1 and o_2 are names of object types. We first prove the validity of this rule. Presume the rules $\textsf{EACH}\, D_1\ o_1\ D_2\, \textsf{MUST BE THAT}\, o_2$ and $\textsf{EACH}\, D_3\, \textsf{MUST BE THAT}\, o_1$, and assume $x\, [\![D_1\ D_3\ D_2]\!]\, y$, then from the definition of concatenation we know that for some p and q we have $x\, [\![D_1]\!]\, p\, [\![D_3]\!]\, q\, [\![D_2]\!]\, y$. From $p\, [\![D_3]\!]\, q$ we conclude that also $p\, [\![o_1]\!]\, q$, and thus $x\, [\![D_1\ o_2\ D_2]\!]\, y$. Applying the first rule yields $x\, [\![o_2]\!]\, y$. Applying this reasoning rule by setting:

$D_1 =$ License plate registering	$o_1 =$ Car
$D_2 =$ being registered by License plate	$o_2 =$ Licenseplate
$D_3 =$ Car being owned by Person owning Car	

we get:

EACH Licence plate registering A Car being owned by A Person owning A Car being registered by A Licence plate MUST BE THAT License plate
Each licence plate that registers a car (owned by a person), must be a unique licence plate for that car owned by that person

As a negative formulated sentence:

NO Licence plate is registering A Car being owned by A Person owning A Car being registered by ANOTHER Licence plate
No licence plate may register a car (that is owned by a person), that is also registered by another licence plate

4 Discussion and Conclusion

In this paper we proposed to use ORM and Lisa-D as a means to formally reason about domains. We explored some aspects of such reasoning. During conceptual modeling, a modeler can add consistency rules that describe the intended populations of the conceptual schema. We assumed that such rules can be formulated in the conceptual language Lisa-D. We discussed reasoning with Lisa-D expressions. Especially, we speculated that such reasoning may well be closer to the way people naturally reason about specific application domains than more traditional forms of (formal) reasoning. If this is indeed the case, we may be able to "reverse the modeling process", and focus on sample reasoning in the application domain, deriving from explicit reasoning examples an underlying system of reasoning rules and domain-specific axioms.

In the near future the original definition of Lisa-D will be adapted to better suit the needs for formal reasoning about domains. More work is also needed in providing more natural verbalisation/paraphrasing of Lisa-D expressions, more specifically the verbalisation of *path expressions* as mentioned in section 2.2. A link to formal theorem proving tools will be considered as well.

References

1. A.C. Bloesch and T.A. Halpin. ConQuer: A Conceptual Query Language. In B. Thalheim, ed., *Proc. of the 15th Intl Conference on Conceptual Modeling (ER'96)*, volume 1157 of *LNCS*, pages 121–133, Cottbus, Germany, EU, 1996.
2. P. van Bommel, A.H.M. ter Hofstede, and Th.P. van der Weide. Semantics and verification of object-role models. *Information Systems*, 16(5):471–495, 1991.
3. H. B. Enderton. *A Mathematical Introduction to Logic.* Academic Press, New York, New York, USA, 1972.
4. P.J.M. Frederiks. *Object-Oriented Modeling based on Information Grammars.* PhD thesis, University of Nijmegen, Nijmegen, The Netherlands, EU, 1997.
5. T.A. Halpin. *Information Modeling and Relational Databases, From Conceptual Analysis to Logical Design.* Morgan Kaufman, San Mateo, California, USA, 2001.
6. A.H.M. ter Hofstede, H.A. (Erik) Proper, and Th.P. van der Weide. Formal definition of a conceptual language for the description and manipulation of information models. *Information Systems*, 18(7):489–523, October 1993.
7. J.J.A.C. Hoppenbrouwers. *Conceptual Modeling and the Lexicon.* PhD thesis, Tilburg University, Tilburg, The Netherlands, EU, 1997. ISBN 90-5668-027-7
8. S.J.B.A. Hoppenbrouwers, H.A. (Erik) Proper, and Th.P. van der Weide. Fundamental understanding of the act of modelling. Proceedings of the 24th International Conference on Conceptual Modeling (ER2005), Klagenfurt, Austria, EU.
9. R. Meersman. The RIDL Conceptual Language. International Centre for Information Analysis Services, Control Data Belgium, Inc., Brussels, Belgium, 1982.
10. R.G. Ross, editor. *Business Rules Manifesto.* Business Rules Group, November 2003. Version 2.0. `http://www.businessrulesgroup.org/brmanifesto.htm`
11. J.M. Spivey. *Understanding Z: A Specification Language and its Formal Semantics.* Cambridge University Press, Cambridge, United Kingdom, EU, 1988.
12. J. Warmer and A. Kleppe. *The Object Constraint Language: Getting Your Models Ready for MDA.* Addison-Wesley, Reading, Massachusetts, USA, 2nd edition, 2003.