Logical Data Modeling: Part 8

Terry Halpin
INTI International University

This is the eighth article in a series on logic-based approaches to data modeling. The first article [5] briefly overviewed deductive databases, and illustrated how simple data models with asserted and derived facts may be declared and queried in LogiQL [2, 15, 17], a leading edge deductive database language based on extended datalog [1]. The second article [6] discussed how to declare inverse predicates, simple mandatory role constraints and internal uniqueness constraints on binary fact types in LogiQL. The third article [7] explained how to declare *n*-ary predicates and apply simple mandatory role constraints and internal uniqueness constraints to them. The fourth article [8] discussed how to declare external uniqueness constraints. The fifth article [9] covered derivation rules in a little more detail, and showed how to declare inclusive-or constraints. The sixth article [10] discussed how to declare simple set-comparison constraints (i.e. subset, exclusion, and equality constraints), and exclusive-or constraints. The seventh article [11] explained how to declare subset, constraints between compound role sequences, including cases involving join paths. The current article discusses how to declare exclusion and equality constraints between compound role sequences. The LogiQL code examples are implemented using the free, cloud-based REPL (Read-Eval-Print-Loop) tool for LogiQL that is accessible at https://developer.logicblox.com/playground/.

Compound Exclusion Constraints without Join Paths

Table 1 shows an extract from report of a publishing company containing data about books and their authors, as well as the reviewers of the book if known. Each book is identified by its International Standard Book Number (ISBN). Each author and reviewer is identified by a Person Number (PNr) assigned by the publishing company. Other details about books and persons (e.g. book title, person name), are maintained but are not of interest for this discussion. A blank entry for reviewers indicates that the book has not yet been assigned any reviewer. As an optional exercise, you might like to specify a data model for this example, including all relevant constraints, before reading on.

Table 1 Report extract of data about books, their authors, and their reviewers (if known)

ISBN	*Author PNrs*	*Reviewer PNrs*
1-23456-789-1	1	2, 3
2-55867-246-7	1, 2	3, 5
3-24680-345-2	3	

Figure 1 models this example with a populated Object-Role Modeling (ORM) [12, 13, 14] diagram. The sample facts are shown in fact tables next to the relevant fact types. The preferred reference schemes for Book and Person are depicted by the reference modes "ISBN" and ".Nr" shown in parentheses. The mandatory role dot and Book's role in the fact type Book is authored by Person indicates that each book has at least one author. The uniqueness constraint bar spanning both roles of the binary fact types Book is authored by Person and Book is reviewed by Person indicates that each of these relationships is many-to-many.

In general, an *exclusion constraint* may apply between two or more sequences of roles so long as the corresponding roles are compatible (hosted by the same or compatible types). If a role connector meets the junction of adjacent roles, it applies to the role pair. So in this model the exclusion constraint depicted using a circled "X" connected to the role pairs of the two fact types indicates that the set of (book, person) pairs that populate the authorship predicate is mutually exclusive with the set of (book, person) pairs that populate the reviews predicate.

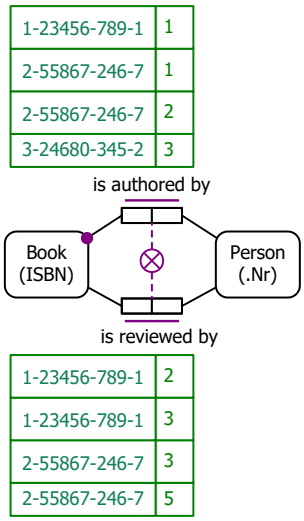

Figure 1 Populated data model for Table 1 in ORM notation.

The NORMA tool [4] verbalizes this subset constraint as follows:

For each Book **and** Person, **at most one of the following holds:**
that Book is authored by **that** Person;
that Book is reviewed by **that** Person.

Figure 2(a) schematizes the same universe of discourse depicted in Table 1 as an Entity Relationship diagram in Barker notation (Barker ER) [3]. The Barker ER schema depicts the primary reference scheme for Book and Person by prepending an octothorpe "#" to the isbn and person nr attributes respectively. The asterisk "*" prepended to the isbn and person nr attributes indicates that these attributes are mandatory. The half solid, half dashed line indicates that authorship relationship is mandatory for Book and optional for Person. A crowsfoot at both ends of the relationship lines indicates that the relationships are many-to-many. Barker ER has no graphical way to display the exclusion constraint.

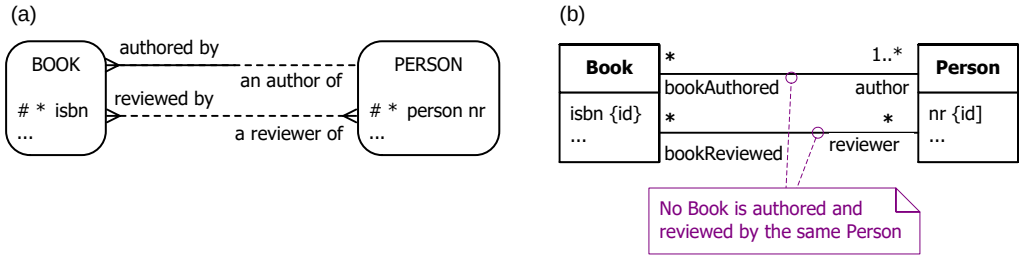

Figure 2 Data schema for Table 1 in (a) Barker ER, and (b) UML notation.

Figure 2(b) schematizes Table 1 as a class diagram in the Unified Modeling Language (UML) [18]. The UML class diagram depicts the primary identification for Book and Person by appending "{id}" to the isbn and nr attributes respectively. The isbn and nr attributes by default have a multiplicity of 1, so are mandatory and single-valued. Names are provided for each association role. The multiplicities at the end of the authorship association indicate that each book has one or more authors, and each person authors zero or more books. The multiplicities at the end of the review association indicate that each book has zero or more reviewers and each person reviews zero or more books. Like Barker ER, UML has no graphic symbol for an exclusion constraint, but here the constraint has been declared informally in a textual note.

The ORM schema in Figure 1 may be coded in LogiQL as shown below. The right arrow symbol "->" stands for the material implication operator "→" of logic, and is read as "implies". An exclamation mark "!" denote the logical negation operator and is read as "it is not the case that". A comma "," denotes the logical conjunction operator "&", and is read as "and". Recall that LogiQL is case-sensitive, each formula must end with a period, and head variables are implicitly universally quantified.

```
Book(b), hasISBN(b:isbn) -> string(isbn).
Person(p), hasPersonNr(p:pNr) -> int(pNr).
isAuthoredBy(b, p) -> Book(b), Person(p).
isReviewedBy(b, p) -> Book(b), Person(p).

// Each book has an author
Book(b) -> isAuthoredBy(b, _).

// No book is authored and reviewed by the same person
isAuthoredBy(b, p) -> !isReviewedBy(b, p).
```

The first line of the LogiQL code declares Book as an entity type whose instances are referenced by ISBNs that are coded as strings. The colon ":" in hasISBN(b:isbn) distinguishes hasISBN as a refmode predicate, so this predicate is injective (mandatory, 1:1). Similarly, the next line declares the entity type Person along with its reference mode. The next two lines declare the typing constraints on the isAuthoredBy and isReviewedBy predicates, each of which is many-to-many.

Comments are prepended by "//", and describe the constraint declared in the code immediately following the comment. An underscore "_" denotes the anonymous variable, is used for existential quantification, and is read as "something". For example, the mandatory role constraint expressed in LogiQL as "Book(b) -> isAuthoredBy(b, _)." corresponds to the logical formula "$\forall b$(Book $b \rightarrow \exists x$ b isAuthoredBy x)".

The exclusion constraint coded as "isAuthoredBy(b, p) -> !isReviewedBy(b, p)." corresponds to the logical formula "$\forall b,p$ [b isAuthoredBy $p \rightarrow \sim$(b isReviewedBy p)]".

To enter the schema in the free, cloud-based REPL tool, use a supported browser such as Chrome, Firefox or Internet Explorer to access the website https://repl.logicblox.com. Alternatively, you can access https://developer.logicblox.com/playground/, then click the "Open in new window" link to show a full screen for entering the code.

Schema code is entered in one or more *blocks* of one or more lines of code, using the *addblock* command to enter each block. After the "/>" prompt, type the letter "a", and click the addblock option that then appears. This causes the addblock command (followed by a space) to be added to the code window. Typing a single quote after the addblock command causes a pair of single quotes to be appended, with your cursor placed inside those quotes ready for your block of code (see Figure 3).

Figure 3 Invoking the addblock command in the REPL tool.

Now copy the schema code provided above to the clipboard (e.g. using Ctrl+C), then paste it between the quotes (e.g. using Ctrl+V), and then press the Enter key. You are now notified that the block was successfully added, and a new prompt awaits your next command (see Figure 4). By default, the REPL tool also appends an automatically generated identifier for the code block. Alternatively, you can enter each line of code directly, using a separate addblock command for each line.

```
      1  Welcome to the LogicBlox playground!
      2
     />
     />
7-42 />  addblock 'Book(b), hasISBN(b:isbn)  ->  string(isbn).
     ..  Person(p), hasPersonNr(p:pNr)  ->  int(pNr).
     ..  isAuthoredBy(b, p) ->  Book(b), Person(p).
     ..  isReviewedBy(b, p) ->  Book(b), Person(p).
     ..
     ..  // Each book has an author
     ..  Book(b)  ->  isAuthoredBy(b, _).
     ..
     ..  // No book is authored and reviewed by the same person
     ..  isAuthoredBy(b, p)  ->  !isReviewedBy(b, p).
     ..  '
     =>
         Successfully added block 'block_1Z331EYJ'
7-42 />
```

Figure 4 Adding a block of schema code.

The data in Table 1 may be entered in LogiQL using the following delta rules. A delta rule of the form *+fact* inserts that fact. Recall that *plain*, *double quotes* (i.e. ",") are needed here, not single quotes or smart double quotes. Hence it's best to use a basic text editor such as WordPad or NotePad to enter code that will later be copied into a LogiQL tool.

```
+Book(b), +hasISBN(b:"2-55867-246-7"), +Person(p1), +Person(p2), +Person(p3),
 +hasPersonNr(p1:1), +hasPersonNr(p2:2), +hasPersonNr(p3:3),
 +isAuthoredBy(b, p1), +isReviewedBy(b,p2), +isReviewedBy(b,p3).

+Book(b), +hasISBN(b:"1-23456-789-1"), +Person(p1), +Person(p2), +Person(p3),  +Person(p5),
 +hasPersonNr(p1:1), +hasPersonNr(p2:2), +hasPersonNr(p3:3),  +hasPersonNr(p5:5),
 +isAuthoredBy(b, p1), +isAuthoredBy(b, p2), +isReviewedBy(b,p3), +isReviewedBy(b,p5).

+Book(b), +hasISBN(b:"3-24680-345-2"), +Person(p3),  +hasPersonNr(p3:3),
 +isAuthoredBy(b, p3).
```

Delta rules to add or modify data are entered using the exec (for 'execute') command. To invoke the exec command in the REPL tool, type "e" and then select exec from the drop-down list. A space character is automatically appended. Typing a single quote after the exec command and space causes a pair of single quotes to be appended, with your cursor placed inside those quotes ready for your delta rules. Now copy the lines of data code provided above to the clipboard (e.g. using Ctrl+C), then paste it between the quotes (e.g. using Ctrl+V), and then press the Enter key. A new prompt awaits your next command (see Figure 5).

```
7-42 />  exec '+Book(b), +hasISBN(b:"2-55867-246-7"), +Person(p1), +Person(p2), +Person(p3),
     ..    +hasPersonNr(p1:1), +hasPersonNr(p2:2), +hasPersonNr(p3:3),
     ..    +isAuthoredBy(b, p1), +isReviewedBy(b,p2), +isReviewedBy(b,p3).
     ..
     ..  +Book(b), +hasISBN(b:"1-23456-789-1"), +Person(p1), +Person(p2), +Person(p3),  +Person(p5),
     ..    +hasPersonNr(p1:1), +hasPersonNr(p2:2), +hasPersonNr(p3:3),  +hasPersonNr(p5:5),
     ..    +isAuthoredBy(b, p1), +isAuthoredBy(b, p2), +isReviewedBy(b,p3), +isReviewedBy(b,p5).
     ..
     ..  +Book(b), +hasISBN(b:"3-24680-345-2"), +Person(p3),  +hasPersonNr(p3:3),
     ..    +isAuthoredBy(b, p3).
     ..  '
7-42 />
```

Figure 5 Adding the data below the program code.

Now that the data model (schema plus data) is stored, you can use the *print* command to inspect the contents of any predicate. For example, to list all the recorded books, type "p" then select print from the drop-down list, then type a space followed by "B", then select Book from the drop-down list and press Enter. Alternatively, type "print Book" yourself and press Enter. Figure 6 shows the result. By default, the REPL tool prepends a column listing automatically generated, internal identifiers for the returned entities. Similarly, you can use the print command to print the extension of the other predicates.

```
7-42 />  print Book
     =>
```

10000000004	2-55867-246-7
10000000005	1-23456-789-1
10000000007	3-24680-345-2

Figure 6 Using the print command to list the extension of a predicate.

As discussed in previous articles, to perform a query, you specify a derivation rule to compute the facts requested by the query. For example, the following query may be used to list the person number of each person who has authored a book and reviewed a book. The rule's head _(pNr) uses an anonymous predicate to capture the result derived from the rule's body. The head variable *pNr* is implicitly universally quantified. The variable *p* introduced in the rule body is implicitly existentially quantified.

```
_(pNr) <- isAuthoredBy(_, p), isReviewedBy(_, p), hasPersonNr(p:pNr).
```

In LogiQL, queries are executed by appending their code in single quotes to the *query* command. To do this in the REPL tool, type "q", choose "query" from the drop-down list, type a single quote, then copy and paste the above LogiQL query code between the quotes and press Enter. The relevant query result is now displayed as shown in Figure 7.

```
7-42 />  query '_(pNr) <-  isAuthoredBy(_, p), isReviewedBy(_, p), hasPersonNr(p:pNr).'
     =>
```

2
3

Figure 7 A query to list the student number, name and grade in CS100 for those students with a grade in CS100.

To test the exclusion constraint, try adding the following delta rule to assert that the book with ISBN 3-24680-345-2 is reviewed by the person with person number 3.

```
+Book(b), +hasISBN(b:"3-24680-345-2"), +Person(p3), +hasPersonNr(p3:3), +isReviewedBy(b, p3).
```

Since the database already includes a fact asserting that this book is authored by the same person, this update attempt is rejected because it violates the exclusion constraint (see Figure 8).

```
7-42 />  exec '+Book(b), +hasISBN(b:"3-24680-345-2"), +Person(p3), +hasPersonNr(p3:3), +isReviewedBy(b, p3).'
     =>
Exception in command: ERROR
--cut-here----------------------ERROR-REPORT-(begin)-----------------cut-here--
Error: Constraint failure(s):
block_1Z331EYJ:10(1)--10(43):
    false <-
      Exists b::Book,p::Person .
        isAuthoredBy(b,p),
        isReviewedBy(b,p).
(1) b=[10000000007],p=[10000000000]
```

Figure 8 Asserting that a book is authored and reviewed by the same person violates the exclusion constraint.

Compound Exclusion Constraints involving Join Paths

A role sequence appearing as an argument in a set-comparison constraint may be projected from a path involving one or more conceptual joins. The previous article [11] discussed an example of a subset constraint with a join path. An example of an exclusion constraint involving a join path is shown in Figure 9. This is a screenshot from the NORMA tool showing both an ORM schema and the verbalization of the exclusion constraint. The schema is part of an information system for managing details about research papers submitted to a conference. The exclusion constraint ensures that people cannot review any research paper authored by someone from their own institute. This schema applies to a single conference at a time (no history is kept).

The first argument of the exclusion constraint consists of the first and last roles of the *join path* that traverses from ResearchPaper via the authorship predicate to Institute via Person (where the *conceptual join* of the two roles hosted by Person in this path requires the persons in the two predicates to match). The second argument of the exclusion constraint consists of the first and last roles of the join path that traverses from ResearchPaper via the reviews predicate to Institute via Person. The constraint requires, for each population of the database, that the set of (research paper, institute) pairs projected from the first join path must be mutually exclusive with the set of (research paper, institute) pairs projected from the second join path.

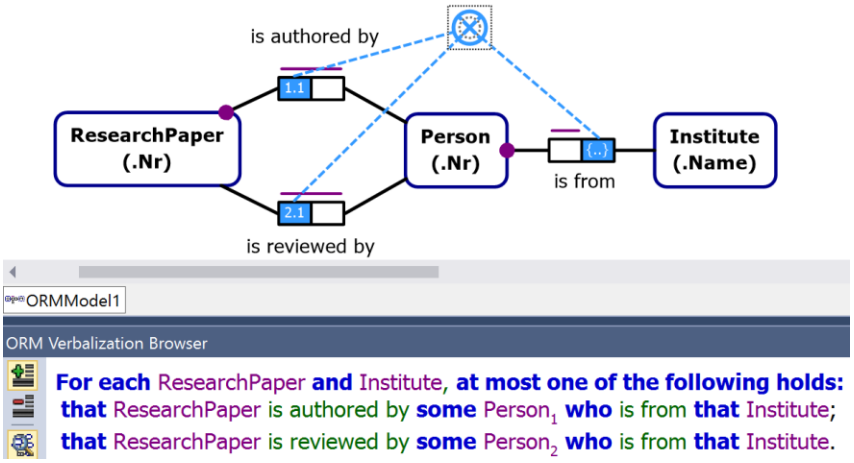

Figure 9 ORM schema with an exclusion constraint involving join paths.

Neither Barker ER nor UML provide graphical support for such constraints, and are ignored for this example. The schema for this example may be coded in LogiQL as follows. The exclusion constraint is implemented by first deriving the predicates hasAnAuthorFrom(rp, i) and hasAReviewerFrom(rp, i) and then declaring these to be mutually exclusive.

```
ResearchPaper(rp), hasResearchPaperNr(rp:rpn) -> int(rpn).
Person(p), hasPersonNr(p:pNr) -> int(pNr).
Institute(i), hasInstituteName(i:in) -> string(in).
isAuthoredBy(rp, p) -> ResearchPaper(rp), Person(p).
isReviewedBy(rp, p) -> ResearchPaper(rp), Person(p).
instituteOf[p] = i -> Person(p), Institute(i).

// Each research paper has an author
ResearchPaper(rp) -> isAuthoredBy(rp, _).
// Each person is from an institute
Person(p) -> instituteOf[p] = _.

// Nobody may review a paper authored by someone from the same institute
hasAnAuthorFrom(rp, i) <- isAuthoredBy(rp, p), instituteOf[p] = i.
hasAReviewerFrom(rp, i) <- isReviewedBy(rp, p), instituteOf[p] = i.
hasAnAuthorFrom(rp, i) -> ! hasAReviewerFrom(rp, i).
```

Compound Equality Constraints

A *compound equality constraint* restricts the populations of a set of two or more compatible sequences of two of more roles to be equal. As a simple example, consider the report extract shown in Table 2. The data is maintained by a weather bureau that records the minimum and maximum temperatures of some Australian capital cities for some months of the current year.

Table 2 Report extract of monthly temperature extremes (if known) in ºC for some Australian capital cities.

City	*Month*	*Minimum Temperature*	*Maximum Temperature*
Brisbane	January	20	36
Brisbane	February	20	33
Sydney			
…	…		

Figure 10 shows an ORM model for this example. The equality constraint ensures that the population of the set of (city, month) pairs for the minimum temperature predicate must always match the population of the set of (city, month) pairs for the maximum temperature predicate. So for any given city and month, either both the extreme temperatures are measured or none are.

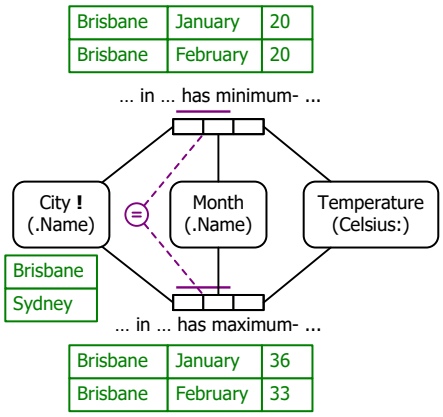

Figure 10 ORM model for Table 2.

Neither Barker ER nor UML provide graphical support for such constraints, and are ignored for this example. The schema for this example may be coded in LogiQL as follows. The equality constraint is coded as two subset constraints, one in each direction.

```
City(c), hasCityName(c: cn)  ->  string(cn).
Month(m), hasMonthName(m:mn)  ->  string(mn).
minCelsiusFor[c, m] = t  ->  City(c), Month(m), int(t).
maxCelsiusFor[c, m] = t  ->  City(c), Month(m), int(t).

// For each city and month, min and max temperatures are both recorded or none are.
minCelsiusFor[c, m] = _  ->  maxCelsiusFor[c, m] = _.
maxCelsiusFor[c, m] = _  ->  minCelsiusFor[c, m] = _.
```

Conclusion

The current article discussed how to declare compound exclusion and compound equality constraints, including a case involving join paths. Future articles in this series will examine how LogiQL can be used to specify business constraints and rules of a more advanced nature. The core reference manual for LogiQL is

accessible at https://developer.logicblox.com/content/docs4/core-reference/. An introductory tutorial for LogiQL and the REPL tool is available at https://developer.logicblox.com/content/docs4/tutorial/repl/section/split.html. Further coverage of LogiQL may be found in [15].

References

1. Abiteboul, S., Hull, R. & Vianu, V. 1995, *Foundations of Databases*, Addison-Wesley, Reading, MA.
2. Aref, M., Cate, B., Green T., Kimefeld, B., Olteanu, D., Pasalic, E., Veldhuizen, T. & Washburn, G. 2015, 'Design and Implementation of the LogicBlox System', *Proc. 2015 ACM SIGMOD International Conference on Management of Data*, ACM, New York. http://dx.doi.org/10.1145/2723372.2742796.
3. Barker, R. 1990, *CASE*Method: Entity Relationship Modelling*, Addison-Wesley, Wokingham.
4. Curland, M. & Halpin, T. 2010, 'The NORMA Software Tool for ORM 2', P. Soffer & E. Proper (Eds.): *CAiSE Forum 2010*, LNBIP 72, pp. 190-204, Springer-Verlag Berlin Heidelberg 2010.
5. Halpin, T. 2014, 'Logical Data Modeling: Part 1', *Business Rules Journal*, Vol. 15, No. 5 (May, 2014), URL: http://www.BRCommunity.com/a2014/b760.html.
6. Halpin, T. 2014, 'Logical Data Modeling: Part 2', *Business Rules Journal*, Vol. 15, No. 10 (Oct., 2014), URL: http://www.BRCommunity.com/a2014/b780.html.
7. Halpin, T. 2015, 'Logical Data Modeling: Part 3', *Business Rules Journal*, Vol. 16, No. 1 (Jan., 2015), URL: http://www.BRCommunity.com/a2015/b795.html.
8. Halpin, T. 2015, 'Logical Data Modeling: Part 4', *Business Rules Journal*, Vol. 16, No. 7 (July, 2015), URL: http://www.BRCommunity.com/a2015/b820.html.
9. Halpin, T. 2015, 'Logical Data Modeling: Part 5', *Business Rules Journal*, Vol. 16, No. 10 (Oct., 2015), URL: http://www.BRCommunity.com/a2015/b832.html.
10. Halpin, T. 2016, 'Logical Data Modeling: Part 6', *Business Rules Journal*, Vol. 17, No. 3 Mar., 2016), URL: http://www.BRCommunity.com/a2016/b852.html.
11. Halpin, t. 2016, 'Logical Data Modeling: Part 7', *Business Rules Journal*, Vol. 17, No. 7 (July, 2016), URL: http://www.brcommunity.com/a2016/b866.html.
12. Halpin, T. 2015, *Object-Role Modeling Fundamentals*, Technics Publications, New Jersey.
13. Halpin, T. 2016, *Object-Role Modeling Workbook*, Technics Publications, New Jersey.
14. Halpin, T. & Morgan, T. 2008, *Information Modeling and Relational Databases*, *2nd edition*, Morgan Kaufmann, San Francisco.
15. Halpin, T. & Rugaber, S. 2014, *LogiQL: A Query language for Smart Databases*, CRC Press, Boca Raton. http://www.crcpress.com/product/isbn/9781482244939#.
16. Halpin, T. & Wijbenga, J. 2010, 'FORML 2', *Enterprise, Business-Process and Information Systems Modeling*, eds. I. Bider et al., LNBIP 50, Springer-Verlag, Berlin Heidelberg, pp. 247–260.
17. Huang, S., Green, T. & Loo, B. 2011, 'Datalog and emerging applications: an interactive tutorial', *Proc. 2011 ACM SIGMOD International Conference on Management of Data*, ACM, New York. http://dl.acm.org/citation.cfm?id=1989456.
18. Object Management Group 2013, *OMG Unified Modeling Language (OMG UML)*, version 2.5 RTF Beta 2. Available online at: http://www.omg.org/spec/UML/2.5/Beta2/PDF/.
19. OMG, 2012, *OMG Object Constraint Language (OCL), version 2.3.1*. Retrieved from http://www.omg.org/spec/OCL/2.3.1/.