# Modeling Dynamic Rules in ORM

Herman Balsters[1], Andy Carver[2], Terry Halpin[2], and Tony Morgan[2]

[1] University of Groningen, The Netherlands
H.Balsters@rug.nl
[2] Neumont University, Utah, USA
{andy, terry, tony.morgan}@neumont.edu

**Abstract.** This paper proposes an extension to the Object-Role Modeling approach to support formal declaration of dynamic rules. Dynamic rules differ from static rules by pertaining to properties of state transitions, rather than to the states themselves. In this paper, application of dynamic rules is restricted to so-called single-step transactions, with an old state (the input of the transaction) and a new state (the direct result of that transaction). Such restricted rules are easier to formulate (and enforce) than a constraint applying historically over all possible states. In our approach, dynamic rules specify an elementary transaction type indicating which kind of object or fact is being added, deleted or updated, and (optionally) pre-conditions relevant to the transaction, followed by a condition stating the properties of the new state, including the relation between the new state and the old state. These dynamic rules are formulated in a syntax designed to be easily validated by non-technical domain experts.

## 1 Introduction

Object-Role Modeling (ORM) is a fact-oriented approach for modeling, transforming, and querying information in terms of the underlying facts of interest, where facts and rules may be verbalized in language readily understandable by non-technical users of the business domain. In contrast to Entity-Relationship (ER) modeling [4] and Unified Modeling Language (UML) class diagrams [18], ORM models are attribute-free, treating all facts as relationships (unary, binary, ternary etc.). ORM includes procedures for mapping to attribute-based structures, such as those of ER or UML. We use the term "ORM" to include a number of closely related dialects, such as Natural language Information Analysis Method (NIAM) [27] and Fully-Communication Oriented Information Modeling (FCO-IM) [1]. For a basic introduction to ORM see [13], for a thorough treatment see [8]. For a comparison of ORM with UML see [10].

Business rules include constraints and derivation rules. *Static rules* apply to each state of the information system that models the business domain, and may be checked by examining each state individually (e.g. each person was born on at most one date). *Dynamic rules* reference at least two states, which may be either successive (e.g. no employee may be demoted in rank) or separated by some period (e.g. invoices ought to be paid within 30 days of being issued). While ORM provides richer graphic support for static rules than ER or UML provide, ORM as yet cannot match UML's support for dynamic rules.

R. Meersman, Z. Tari, P. Herrero et al. (Eds.): OTM Workshops 2006, LNCS 4278, pp. 1201 – 1210, 2006.

Since the 1980s, many extensions to ORM have been proposed to model temporal aspects and processes. The TOP model [7] allows fact types to be qualified by a temporal dimension and granularity. TRIDL [3] includes time operators and action semantics, but not dynamic constraints. LISA-D [16] supports basic updates. Task structures and task transactions model various processes [15], with formal grounding in process algebra. EVORM [22] formalizes first and second order evolution of information systems. Some explorations have been made to address reaction rules [e.g. 14], and some proposals suggest deriving activity models from ORM models ([23]).

Some fact-based approaches that share similarities with ORM have developed deep support for modeling system dynamics. For example, the CRL language in TEMPORA enables various constraints, derivations and actions to be formulated on Entity-Relationship-Time (ERT) models [24, 25], and the OSM method includes both graphical and textual specification of state nets and object interactions [6].

Various attribute-based methods such as UML and some extensions of ER incorporate dynamic modeling via diagrams (e.g. UML state charts and activity diagrams). For textual specification of dynamic rules, the most popular approach is the Object Constraint Language (OCL) [19, 26], but the OCL syntax is often too mathematical for validation by non-technical domain experts. Olivé suggests an extension to UML to specify temporal constraints, but this is limited to rules about creation of objects [20]. Substantial research has been carried out in providing logical formalizations for dynamic rules, typically using temporal logics or Event-Condition-Action (ECA) formalisms (e.g. de Brock [2], Lipeck [17], Chomicki [5], and Paton & Díaz [21]). Many works also describe how to implement dynamic rules in software systems.

However, to our knowledge, no one has yet provided a purely declarative means to formulate dynamic constraints in a textual syntax suitable for non-technical users. This paper provides a first step towards such support for dynamic rules in ORM by addressing *single-step transactions*, with an old state (the input of the transaction) and a new state (resulting from that transaction). Our dynamic rules specify an *elementary transaction type* indicating the kind of object or fact being added, deleted or updated, and (optionally) pre-conditions relevant to the transaction, followed by a condition on the new state, including its relation to the old state. These dynamic rules are formulated in a syntax designed for easy validation by non-technical domain experts. Our aim is to identify basic rule patterns rather than provide a complete, formal grammar.

The rest of this paper is structured as follows. Section 2 focuses on rules involving updates to a single role in a functional binary fact type. Section 3 extends the examples of Section 2 to show how history can be added. Section 4 examines rules involving the addition of instances of non-functional fact types. Section 5 discusses a more complex case involving derivation. Section 6 briefly discusses fact deletion. Section 7 summarizes the main results, suggests topics for further research, and lists references.

## 2 Updating Single-Valued Roles in a Functional Fact Type

Our first sub-case is a functional (*n*:1 or 1:1) binary fact type. Fig. 1 shows a functional fact type in ORM 2 notation [11], where role names may be displayed in square brackets and used to verbalize rules in attribute-style [9].

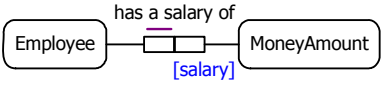



**Fig. 1.** In each state, each employee has at most one salary

Suppose a dynamic constraint requires that salaries of employees must not decrease. We show two alternative expressions for this constraint, using the reserved words **old** and **new** to refer to situations immediately before and after the transition.

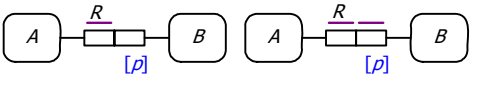
(a) **Context**: Employee
**new** salary >= **old** salary

(b) **For each** Employee,
**new** salary >= **old** salary

Here the *context* of the constraint is the object type Employee, and the elementary transaction *updates* the salary of the employee. The presence of the **new** and/or **old** keywords signals that the prospective transaction is an update (rather than an addition or deletion); and this all implies that the rule is applicable only when there is in fact an "old" marital status (of the same student) to update. The constraint is violated if and only if it evaluates to false (like SQL check-clauses). So if the employee had no prior salary, the inequality evaluates to unknown and hence is not violated. In this case we record only a "snapshot" of the current salary (i.e. no salary history) which allows a simple constraint structure. A later example considers salary history.

Specification of Employee as the context is sufficient in this case because the fact type is *n*:1. While the rule may be specified in relational style, using a predicate reading, an attribute style formulation using a role name is often more convenient. Each transaction is always considered to be isolated (serializable).

Constraints of this kind are fairly common in business systems. Generalizing from the example above to any functional binary fact type of the form *A R*'s *B*, with *B*'s role name *p* (denoting the property or "attribute" of *A* being constrained), we obtain the constraint formulation pattern in Fig. 2, where Θ denotes the required relationship between the values of the property *p* after and before the transition.




**Context**: *A* | **For each** *A*,
**new** *p* Θ **old** *p*

**Fig. 2.** General pattern for updating a named, single-valued role on *n*:1 and 1:1 relationships

Our dynamic-constraint language should also be able to handle constraints that involve a table of state-transitions. A simple example involves marital states:

| *From \ To* | *Single* | *Married* | *Widowed* | *Divorced* |
|---|---|---|---|---|
| *Single* | 0 | 1 | 0 | 0 |
| *Married* | 0 | 0 | 1 | 1 |
| *Widowed* | 0 | 1 | 0 | 0 |
| *Divorced* | 0 | 1 | 0 | 0 |

The matrix shows which updates to a given student's marital status are possible. There is no functional or deterministic relationship between an old state and a new state that

can or cannot follow. One simple solution involves that sort of construct which, in programming languages, is commonly called a case or switch statement:

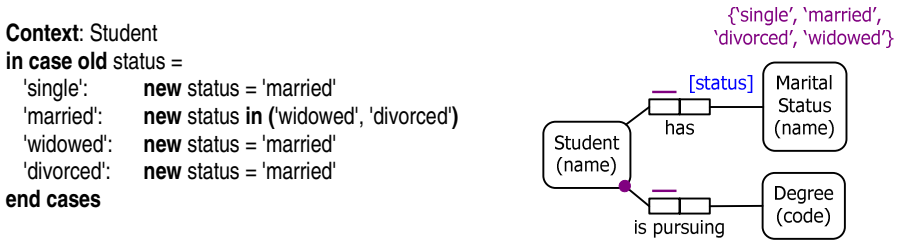



**Fig. 3.** Updating the marital status of a student

One may alternatively specify impossible transitions for any given case., e.g. the divorced case could be reworded as "**new** status **not in** ('single', 'divorced', 'widowed')". To say the **new** status must *not* equal some value, one uses <> instead of =.

We can generalize similar kinds of constraints in the manner shown in Fig. 4. In the constraint, *B1*, *B2*, *B3*, etc. represent possible *B*s that may play role *p*.


**Fig. 4.** General pattern for enumerated values

## 3 Examples of Historical Facts

Our earlier constraint that an employee's salary could not decrease required only a "snapshot" view of salary. We now extend this simple case by requiring a salary *history* (see Fig. 5). The **new** keyword is not required here because we add a fact rather than update an existing fact. We assume here the existence of a function **previous** that can return the existing salary most recently added for any specific employee.


**Fig. 5.** Salary example with history

Returning to the specific example of recording a student's marital status, a similar extension to add a history of marital status values for each student is shown in Fig. 6.

**Context**: Student
**For each** status **added**
**if before:**
Student acquired **some** MaritalStatus on **some** Date
**then after:**
**in case previous** status =
'single': status = 'married'
'married': status **in** ('widowed', 'divorced')
'widowed': status = 'married'
'divorced': status = 'married'
**end cases**

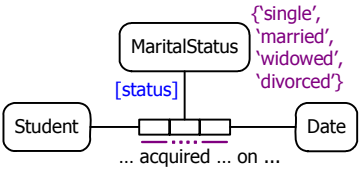



**Fig. 6.** Marital status example with history

We generalize this example to a pattern for recording history of some kind of thing acquiring some property at some instant, as shown in Fig. 7.

**Context**: A
**For each** p **added**
**if before**:
A acquired **some** B at **some** Tag
**then after**:
**in case previous** p =
B1: p **in** (B2, B3, ...)
- - etc
**end cases**

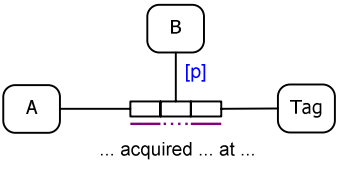



**Fig. 7.** General pattern for enumerated values with history

As before, *B1*, *B2*, etc. represent possible *B*s that may play role *p*. "Tag" represents some value that consistently increases (or perhaps decreases) for a given *A* as new facts are added. Tag values equipped with an ordering criterion are isomorphic to a linearly ordered time-stamping mechanism. Dates and times obviously fit this role, but so also do other kinds of sequenced identifiers such as "incident number", "version number" or other kinds of sequenced identifiers. We assume that the Tag history is complete for each *A* (we add a new fact to a history of all previous facts of the same type for that *A*, and never add a fact that is "earlier" than some existing fact).

The dynamic constraint above applies to a situation where new facts are added to an existing history. If we require a constraint on the addition of the first fact of this type for each object of type *A*, then we need a separate constraint without the "before" condition given above. For the first addition of a fact of this type, if we do not care about the role *p* added, then we do not need to specify the additional constraint.

## 4 Adding Instances of a Non-functional Fact Type

We now consider *adding fact instances* to a *non-functional fact type* (no single-role uniqueness constraint), such as the Seating occupies Table association in Fig. 8, which shows a model fragment extracted from a restaurant application.

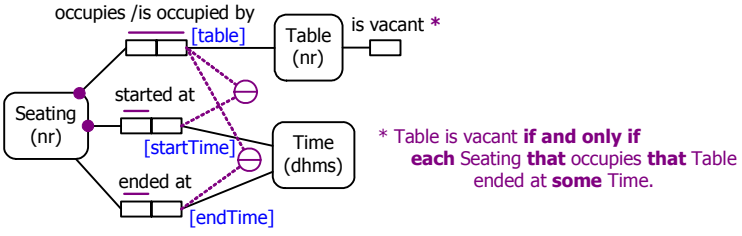



**Fig. 8.** Fragment of an ORM schema about restaurant seatings

A seating is the allocation of a party (of one or more customers) to one or more vacant tables. Each seating starts at some instant and eventually ends at some instant. The circled bars depict the external uniqueness constraints that when a seating starts or ends, any given table is assigned to at most one seating. The asterisked rule is a derivation rule for the snapshot fact type Table is vacant. Notice that the model maintains a *history* of seatings (e.g. for each table we record all the seatings it was previously allocated to). To ensure that no seatings that overlap in time occupy the same table, the following rather complex textual constraint could be added:

**For each** $\text{Seating}_1$, $\text{Seating}_2$:
**if** $\text{Seating}_1$.startTime <= $\text{Seating}_2$.startTime
**and** ($\text{Seating}_2$.startTime <= $\text{Seating}_1$.endTime **or not exists** $\text{Seating}_1$.endTime)
**then no** Table **that** is occupied by $\text{Seating}_1$ is occupied by $\text{Seating}_2$

Instead of this static rule, if we ignore the possibility of people changing their tables during a seating, the user interface itself can ensure that only tables that are currently vacant may be selected for a seating (for multiple screens used in parallel, an appropriate locking mechanism is assumed). Fig. 9 shows the relevant schema fragment and the dynamic rule that a table may be assigned to a seating only if it is vacant *at that time*. The *context* for the constraint is the *fact type* Seating occupies Table. The elementary transaction *adds* an *instance* of this fact type. The reserved words **before** and **after** denote the states just before and after the transaction, **needed** indicates the precondition is necessary for the fact addition to take place (not just for this constraint), and **the** is scoped to the transaction instance.

**Context**: Seating is occupied by Table
**For each fact added**
**needed before: the** table is vacant
**after: the** table is **not** vacant

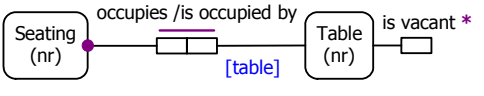



**Fig. 9.** A dynamic constraint for adding instances to the seating fact type

Note that the ability to specify a fact type for the context leads to a much more natural formulation than would be obtained if the context had to be specified as an object type or class, as is the case with OCL. Fig. 10 models the seating example in UML in less detail (e.g. UML has no graphic notation for uniqueness constraints on attributes).

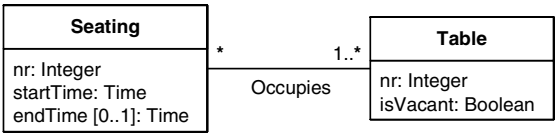



**Fig. 10.** A UML class diagram for the seating example

To specify in OCL the dynamic constraint that a table may be assigned to a seating only if it is vacant at that time, we could try to state this in the context of the Seating class using an operation addTable (taking a parameter *t* of type Table) thus:

```
context Seating::addTable(t:Table)
pre: (t.isVacant)
post: not(t.isVacant) and (table → includes(t))
```

This however is *incorrect* OCL, because it introduces a side effect: the update operation addTable(*t*: Table) updates not only an object from the class Seating, but also the value of the parameter *t*. If we use Table as the context, we may rephrase our constraint as "A seating can be allocated to a table only if that table is vacant", and introduce an update operation allocateSeating(s: Seating) within the class Table thus:

```
context Table::allocateSeating(s: Seating)
pre: isVacant
post: not isVacant and (seating → includes(s))
```

This constraint, though free of side effects from the point of view of the Table class, is still arguably not free of side effects as seen from the Seating class, since invocation of allocateSeating to some specific table $t_0$ and seating $s_0$ would result in the property "$s_0$.table → includes($t_0$)" changing the value of $s_0$. These side effects result from trying to specify the addition of a complete fact within the context of one specific class. A possible resolution is to introduce an auxiliary class *C* (e.g. representing the full model), associated with both the Seating and Table classes, as shown in Fig. 11. Addition of the fact "$s_0$ occupies $t_0$" could then be represented within the context of class *C*. This rather roundabout and artificial solution is needed in order to add complete facts simply because OCL requires that any rule context must be a class.

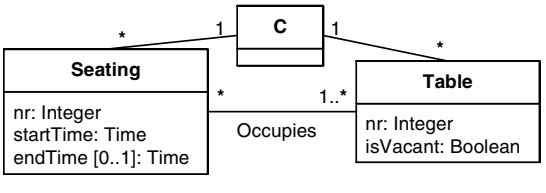



**Fig. 11.** Introducing an artificial class to provide the context for a side-effect free rule

## 5 A More Complex Case Involving Derivation

Let us now consider the case of a transaction dealing with operations on one or more accounts (see Fig. 12). Accounts can be augmented by having a deposit or interest added, or they can be diminished by a fee charge or withdrawal. Simple transactions

refer to an operation on one account only, while transfer transactions deal with two accounts, where a money amount is transferred from the first account to a second account. We record historical information of all transactions, from which the current account balances may be derived. We assume that an account exists prior to any transaction on it, and that on the event that an account is opened, its balance is set to zero.

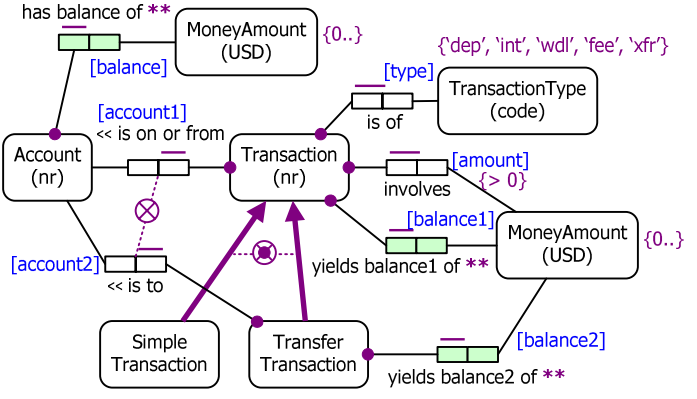



**Fig. 12.** An example involving historical and derived snapshot data

We now use dynamic rules to describe a transition from an old state to a new state for both a simple and a transfer transaction. The order of the components is irrelevant.

**Context:** Account
**For each instance added**
balance = 0

**Context**: SimpleTransaction
**For each instance added**
**in case** type =
'dep', 'int': Transaction.balance1 = (**old** account1.balance + amount )
'wdl', 'fee': Transaction.balance1 = (**old** account1.balance - amount )
**end cases**
**new** account1.balance = Transaction.balance1

**Context**: TransferTransaction
**For each instance added**
balance1 = (**old** account1.balance – amount) **and**
balance2 = (**old** account2.balance + amount) **and**
**new** account1.balance= balance1 **and**
**new** account2.balance= balance2

## 6 Deleting Instances of a Non-functional Fact Type

So far, all our example rules apply to either state-updates or fact-additions; we now briefly consider an example that applies to fact-deletions. A common situation involves a constraint on the length of time that a certain history of events or entity states is kept. For example, a history of payments made to companies might need to be retained for at

least 2 years. This constraint might be expressed as follows. We assume here the existence of functions such as **today** and various operations on date values.

**Context**: MoneyAmount is paid to Company on Date
**For each fact deleted**
**needed before: the** date < **today** – 2 years

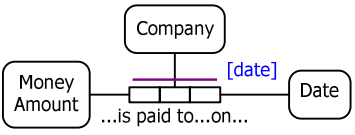



**Fig. 13.** A dynamic constraint for deleting instances from the payment fact type

## 7 Conclusion

This paper proposed an extension to ORM supporting purely declarative specification of dynamic rules restricted to single-step transactions, using syntax designed to be easily validated by non-technical domain experts. These dynamic rules specify an elementary transaction type indicating which kind of object or fact is being added, deleted or updated, any pre-conditions relevant to the transaction, and the relationship between the new state and the old state. By collaborating with other researchers in the ORM community, we intend to incorporate the identified rule patterns with enhancements to previous work on ORM textual languages to provide a formal grammar for a standard textual language for ORM, intended to express static and dynamic rules (constraints and derivations), as well as conceptual queries.

Other future research may be directed at adding actual operations to the ORM-language, explicitly modeling single-step transactions as well as other dynamic rules, which may be alethic or deontic [12]. We also plan to extend the NORMA tool to generate code from dynamic rules. In this context, we hope to provide translations of our dynamic rules to the UML/OCL framework, where our declaration of dynamic rules would be closer to the business level of modeling, and the resulting translation to UML/OCL would be closer to the specification level of the software engineer.

*Acknowledgement.* This paper benefited from discussion with Matt Curland.

## References

1. Bakema, G., Zwart, J. & van der Lek, H. 2000, *Fully Communication Oriented Information Modellin*g, Ten Hagen Stam, The Netherlands.
2. de Brock, E. O. 2000, 'A General Treatment of Dynamic Integrity Constraints'. *Data and Knowledge Engineering, 32*(3): 223-246.
3. Bruza, P. D. & van der Weide, Th. P 1989, 'The Semantics of TRIDL', Technical Report 89-17, Department of Information Systems, University of Nijmegen.
4. Chen, P. P. 1976, 'The entity-relationship model—towards a unified view of data'. *ACM Transactions on Database Systems*, 1(1), pp. 9–36.
5. Chomicki, J. 1992, 'History-less Checking of Dynamic Integrity Constraints', *ICDE 1992*: 557-64.
6. Embley. D. W. 1998, *Object Database Development*, Addison-Wesley.
7. Falkenberg, E. D. & van der Weide, Th. P. 1988, 'Formal Description of the TOP Model'. Technical Report 88-01, Department of Information Systems, University of Nijmegen.

8. Halpin, T. 2001, *Information Modeling and Relational Databases*, Morgan Kaufmann, San Francisco.
9. Halpin, T. 2004, 'Business Rule Verbalization', *Information Systems Technology and its Applications*, Proc. ISTA-2004, (eds Doroshenko, A., Halpin, T. Liddle, S. & Mayr, H), Salt Lake City, Lec. Notes in Informatics, vol. P-48, pp. 39-52.
10. Halpin, T. 2005, 'Information Modeling in UML and ORM: A Comparison', *Encyclopedia of Information Science and Technology*, vol. 3, ed. M. Khosrow-Pour, Idea Publishing Group, Hershey PA, USA, pp. 1471-5.
11. Halpin, T. 2005, 'ORM 2', *On the Move to Meaningful Internet Systems 2005: OTM 2005 Workshops*, eds R. Meersman, Z. Tari, et al., Cyprus. Springer LNCS 3762, pp 676-87.
12. Halpin, T. 2006, 'Business Rule Modality', *Proc. CAiSE'06 Workshops,* eds T, Latour & M. Petit, Namur University Press, pp. 383-94.
13. Halpin, T. 2006, 'ORM/NIAM Object-Role Modeling', *Handbook on Information Systems Architectures*, *2nd edn*, eds P. Bernus, K. Mertins & G. Schmidt, Springer, Heidelberg, pp. 81-103.
14. Halpin, T. & Wagner, G. 2003, 'Modeling Reactive Behavior in ORM'. *Conceptual Modeling – ER2003*, Proc. 22nd ER Conference, Chicago, October 2003, Springer LNCS.
15. ter Hofstede, A. H. M. 1993, 'Information Modelling in Data Intensive Domains', PhD thesis, University of Nijmegen.
16. ter Hofstede, A. H. M., Proper, H. A. & Weide, th. P. van der 1993, 'Formal definition of a conceptual language for the description and manipulation of information models', *Information Systems*, vol. 18, no. 7, pp. 489-523.
17. Lipeck, U. W. 1990, 'Transformation of Dynamic Integrity Constraints into Transaction Specifications', *Theor. Comput. Sci. 76*(1): 115-142.
18. Object Management Group 2003, *UML 2.0 Superstructure Specification*. Online at: www.omg.org/uml.
19. Object Management Group 2005, *UML OCL 2.0 Specification*. Online at: http://www.omg.org/docs/ptc/05-06-06.pdf.
20. Olivé, A. 2003, 'Integrity Constraints Definition in Object-Oriented Conceptual Modeling Languages', *Proc. ER2003*, Springer LNCS, pp. 349-362.
21. Paton, N. W. & Díaz, O. 1999, 'Active Database Systems', *ACM Computing Surveys*, 31(1): 63-103.
22. Proper, H. A. 1994, 'A Theory for Conceptual Modeling of Evolving Application Domains', PhD thesis, University of Nijmegen.
23. Proper, H. A., Hoppenbrouwers, S. J. B. A., & Weide, th. P. van der 2005, 'A Fact-Oriented Approach to Activity Modeling', *On the Move to Meaningful Internet Systems 2005: OTM 2005 Workshops*, eds R. Meersman, Z. Tari, P. Herrero et al., Cyprus. Springer LNCS 3762, pp 666-75.
24. Theodoulidis C., Loucopoulos P. & Kopanas, V. 1992, 'A Rule Oriented Formalism for Active Temporal Databases', *Next Generation CASE Tools*, eds K. Lyytinen & V.-P Tahvanainen, IOS Press, Amsterdam.
25. Theodoulidis C., Wangler B., & Loucopoulos P. 1992, 'The Entity-Relationship-Time Model', *Conceptual Modelling, Databases, and CASE: An Integrated View of Information Systems Development*, ch. 4, pp. 87-115, John Wiley & Sons.
26. Warmer, J. & Kleppe, A. 2003, *The Object Constraint Language, 2nd Edition*, Addison-Wesley.
27. Wintraecken J. 1990, *The NIAM Information Analysis Method: Theory and Practice*, Kluwer, Deventer, The Netherlands.