Formal Semantics of Dynamic Rules in ORM

Herman Balsters[1], Terry Halpin[2]

[1] University of Groningen, The Netherlands
e-mail: H.Balsters@rug.nl
[2] Neumont University, Utah, USA.
e-mail: terry@neumont.edu

Abstract: This paper provides formal semantics for an extension of the Object-Role Modeling approach that supports 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 we restrict application of dynamic rules 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). These dynamic rules further specify an elementary transaction type by indicating which kind of object or fact (being added, deleted or updated) is actually allowed. Dynamic rules may declare pre-conditions relevant to the transaction, and a condition stating the properties of the new state, including the relation between the new state and the old state. In this paper we provide such dynamic rules with a formal semantics based on sorted, first-order predicate logic. The key idea to our solution is the formalization of dynamic constraints as static constraints on the database transaction history.

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 are verbalized in language understandable by nontechnical users of the business domain. In contrast to attribute-based modeling approaches such as Entity Relationship (ER) modeling [5] and class diagramming in the Unified Modeling Language (UML) [17], ORM models are attribute-free, treating all facts as relationships (unary, binary, ternary etc.). For example, instead of the attributes Person.isSmoker and Person.birthdate, ORM uses the fact types Person smokes and Person was born on Date.

Other fact-oriented approaches closely related to ORM include CogNIAM (www.pna-group.com), Fully-Communication Oriented Information Modeling (FCO-IM) [2], and the Semantics of Business Vocabulary and Business Rules (SBVR) [19] specification recently approved by the Object Management Group. A basic introduction to ORM may be found in [11] and a thorough coverage in [12]. The version of ORM discussed in this paper is ORM 2 [10], as supported by the NORMA tool [7].

Business rules include constraints and derivation rules. *Static rules* (also known as state 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 moon orbits at most one planet). *Dynamic rules* reference at least two states, which may be

either successive (e.g. no employee may be demoted in rank—this kind of dynamic rule is known as a transition constraint) or separated by some period (e.g. invoices ought to be paid within 30 days of being issued). ORM is richer than ER or UML in its ability to depict static constraints graphically, but unlike UML it currently has no graphic notation (e.g. activity diagrams) to specify business processes. To capture dynamic rules, UML supplements its graphical notations with formulae in the Object Constraint Language (OCL) [18, 24], but the OCL syntax is often too mathematical for validation by nontechnical users.

Since the 1980s, many extensions to fact-oriented approaches have been proposed to model temporal aspects and processes (e.g. [4, 8, 13, 14, 15, 21, 22]). For a brief review of such work see [1], where we in conjunction with two colleagues introduced to ORM a purely declarative means to formulate dynamic constraints on *single-step transactions*, with an old state (the input of the transaction) and a new state (resulting from that transaction). Such 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 nontechnical domain experts. In this paper, we focus on providing a formal semantics for the basic rule patterns for dynamic rules found in [1]. Such a formalization supports further understanding of dynamic rules, and also provides a step to further tool support.

Substantial research has been carried out to provide logical formalizations of dynamic rules, typically using temporal logics (e.g. [9], ch. 8) or Event-Condition-Action (ECA) formalisms (e.g. de Brock [3], Lipeck [16], Chomicki [6], Paton & Díaz [20], Snodgrass[23]). Our approach differs from previous work by *treating a dynamic rule as a special kind of static rule on the transaction history*. We define the semantics of a dynamic rule by making explicit the log of all previous transaction instances pertaining to that specific rule. Snapshot data are maintained in the user database, whereas historical data are kept in the log database. This *logging semantics* allows us to formalize dynamic constraints in a static way, using first-order predicate logic. ORM model fragments associated with dynamic rules may also be fully formalized in this way, as first described in [9] using unsorted predicate logic; for ease of readability, we now use sorted predicate logic. This enables us to offer the full semantics of ORM models, including both static and dynamic rules, in one coherent framework.

The rest of this paper is structured as follows. Section 2 provides a simple example of how a graphical ORM model (with no dynamic rules) may be transformed into a logical theory. Section 3 shows how to formalize dynamic rules over updates to single-valued roles in functional fact types. Section 4 extends this case to capture history. Section 5 considers additions of instances to nonfunctional fact types. Section 6 examines a more complex case involving derivation. Section 7 summarizes the main contributions, notes some further research options, and provides a list of cited references for further reading.

2 Formalizing Basic ORM models as Logical Theories

An ORM model includes both schema (structure) and population (instances). In [9], one of us provided a detailed algorithm for translating any ORM model into a set of formulae in unsorted predicate logic (with identity, and using mixfix predicates and numeric quantifiers). We now use basically the same approach, but employ sorted logic. While there is no space here to cover the full algorithm, we illustrate the basic approach with a simple example.

The ORM model shown in Fig. 1 includes a schema with one elementary fact type Employee has Salary and a population of three fact instances. Fig. 1(a) is in compact form, abbreviating the reference schemes for Employee and Salary in parentheses. These reference schemes may be automatically expanded to the existential fact types shown in Fig. 1(b). In ORM 2, unit-based reference schemes (e.g. USD:) also involve a unit dimension (in this case, Money) but for simplicity we ignore this aspect here.

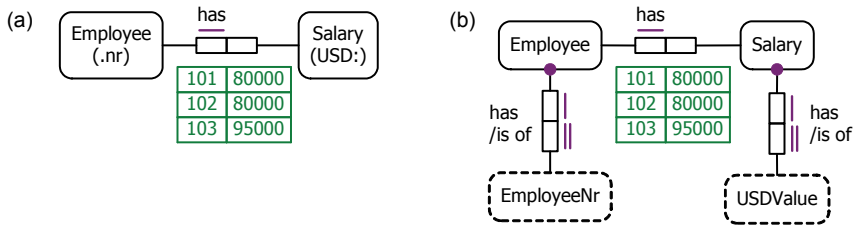


Fig. 1. A simple ORM model in (a) compact and (b) expanded form

The model may be formalized as indicated below. Object types are typed as entity types (solid line) or value types (broken line), and the top level entity types are declared mutually exclusive. For simplicity, we omit classifications of value types here and relevant axioms for numeric operators. The predicates are then typed. Although our sorted logic notation uses short predicate names (e.g. "has"), different fact types are always distinguished by typing the object variables. If one wishes to avoid predicates with different semantics being assigned the same short name (cf. Horse runs Race with Person runs Company), full fact type readings may be used instead to name the predicates. Type predicates are placed in prefix position; all other predicates are mixfix.

Object Types: ∀x:Employee x is an entity; ∀x:Salary x is an entity
∀x:EmployeeNr x is a value; ∀x:USDValue x is a value
∀x:Employee ∀y:Salary x ≠ y

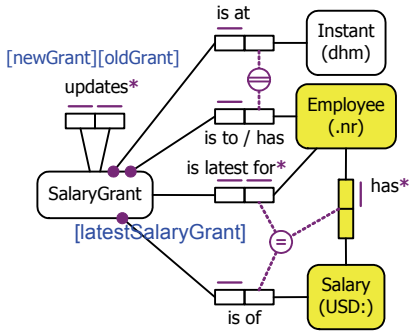
Fact Types: ∀x ∀y (x has y → [(Employee x & Salary y) ∨ (Employee x & EmployeeNr y)
∨ (Salary x & USDValue y)]
∀x:Employee ∀y:EmployeeNr (x has y ≡ y is of x)
∀x:Salary ∀y:USDValue (x has y ≡ y is of x)

Constraints: ∀x:Employee $\exists^{0..1}$y:Salary x has y
∀x:Employee $\exists^{1}$y:EmployeeNr x has y
∀x:EmployeeNr $\exists^{0..1}$y:Employee x is of y
∀x:Salary $\exists^{1}$y:USDValue x has y; ∀x:USDValue $\exists^{0..1}$y:Salary x is of y

Population: ∃x:Employee ∃y:Salary ∃z:EmployeeNr ∃w:USDValue
(x has y & x has z & y has w & z = 101 & w = 80000)
etc. for the other 2 rows of data

3 Updating Single-Valued Roles in a Functional Fact Types

We now consider formalization of dynamic rules added to ORM models, starting with the case of updates to a functional (*n*:1 or 1:1) binary fact type. The dynamic constraint on the salary fact type in Fig. 1 requires that salaries of employees must not decrease. Using the syntax introduced in [1], where **old** and **new** to refer to situations immediately before and after the transition, this constraint may be stated textually as: **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.



Derivation rule for the snapshot fact type
is provided by the equality constraint
which may be formulated in attribute form thus:

* **For each** Employee,
salary = latestSalaryGrant.salary.

* SalaryGrant is latest for Employee **iff**
SalaryGrant is to Employee **and** is at **some** Instant_1
and not exists some SalaryGrant_2 **that** is to **the same** Employee **and** is at **some** $\text{Instant}_2 > \text{Instant}_1$.

* SalaryGrant_2 updates SalaryGrant_1 **iff**
SalaryGrant_1 is to Employee_1 **and** SalaryGrant_2 is to Employee_1
and SalaryGrant_1 is at Instant_1 **and** SalaryGrant_2 is at Instant_2
and not exists some SalaryGrant_3 **that** is to Employee_1 **and** is at Instant_3
and $\text{Instant}_3 > \text{Instant}_1$ **and** $\text{Instant}_3 < \text{Instant}_2$.

Fig. 2. Logging semantics for update salary rule

While the user schema is confined to Employee has Salary, for which only a current snapshot is required (no history), in the background we add fact types to maintain a log of salary grants, as shown in Fig. 2 (unshaded portion). The strict order on Instant enables us to define the notions of *latest* as well as *updating* of an old salary grant by a new one as shown. The employee-salary fact type is now derivable from the equality constraint, as shown. The dynamic constraint may now be reformulated as the following static constraint (no action is needed if the salary grant is the first for the employee): SalaryGrant_2 updates SalaryGrant_1 **only if** SalaryGrant_2.salary >= SalaryGrant_1.salary.

The additional object types and fact types may be formalized as discussed in the previous section. The graphical constraints are also trivially formalized. For example, the external uniqueness constraint and the join equality constraint are expressible as:

∀x:Instant ∀y:Employee $\exists^{0..1}$z:SalaryGrant (z is at x & z is to y)

∀x:Employee ∀y:Salary [x has y ≡ ∃z:SalaryGrant(z is latest for x & z is of y)]

The derivation rules in Fig. 2 are expressed in FORML 2, our formal ORM 2 textual language that is a sugared version of our underlying logical syntax designed for consumption by nontechnical domain experts. Type names are used for sorted variables, with subscripts added as needed to distinguish variables of the same type. Where not stated explicitly, head clause variables are implicitly universally quantified, and variables introduced in the body clause are existentially quantified (cf. Horn clauses). Functional style in dot notation may be used, using role names as function names. For example, the join equality constraint formulated above may be reformulated in functional style as

∀x:Employee x.salary=x.latestSalaryGrant.salary

and then sugared to the FORML 2 rule: **For each** Employee, salary = latestSalaryGrant.salary. The other derivation rules in Fig. 2 are equivalent to the following:

∀x:SalaryGrant ∀y: Employee [x is latest for y ≡ (x is to y & ∃z:Instant x is at z &
~∃w:SalaryGrant ∃u:Instant (w is to y & u >z))]

∀x,y:SalaryGrant [y updates x ≡ (x.employee=y.employee & ~∃z:SalaryGrant
(z.employee=x.employee & y.instant>z.instant & z.instant>x.instant))]

The key result is that the dynamic constraint For each Employee, **new** salary >= **old** salary may be recast as the following static constraint:

∀x,y:SalaryGrant (y updates x → y.salary >= x.salary)

Generalizing from this example to any functional binary fact type of the form *A R*'s *B*, with *B*'s role name *r* (denoting the "attribute" of *A* being constrained), we obtain the dynamic constraint pattern **For each** *A*, **new** *r* Θ **old** *r*, where Θ denotes the required relationship between the values of *r* after and before the transition. Our logic specifications for the salary example may be easily adapted to cover this general pattern.

This approach may be easily extended to formalize simple state transition rules such as the dynamic rule for marital status transitions shown in Fig. 3. As in the previous example, the presence of the **new** and/or **old** keywords signals that the prospective transaction is an update (rather than an addition or deletion). In this case, the logging subschema (unshaded portion) is based on MaritalStatusAssignment. The formalization is similar to that in the previous example, allowing the dynamic rule to be reformulated as the following static rule. This example may be generalized to updates of an enumerated role on a functional fact type *A R's B* in an obvious way.

∀x,y:MaritalStatusAssignment [y updates x → ((x.maritalStatus ='single' & y. maritalStatus='married') ∨ (x. maritalStatus='married' & (y.maritalStatus='widowed' ∨ y.maritalStatus='divorced') ∨ (x.maritalStatus='widowed' & y.maritalStatus='married') ∨ (x.maritalStatus='divorced' & y.maritalStatus = 'married'))]

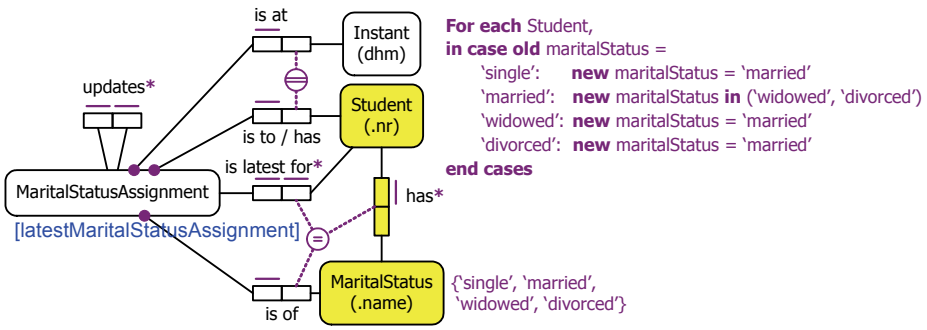


Fig. 3. Updating the marital status of students

4 Examples of Historical Facts

We now extend the salary snapshot case considered earlier to the case where salary *history* is required in the user schema. The dynamic constraint is now specified using the keywords "**added**", "**previous**" and "**existing**", as shown in Fig. 4(a) (for simplicity, reference schemes are omitted). The **added** keyword indicates we are adding a fact rather than updating an existing fact. The "**previous**" function returns the previous salary (if it exists) of the employee, while the qualification "**existing**" applies the condition only if a previous salary for the employee does exist.

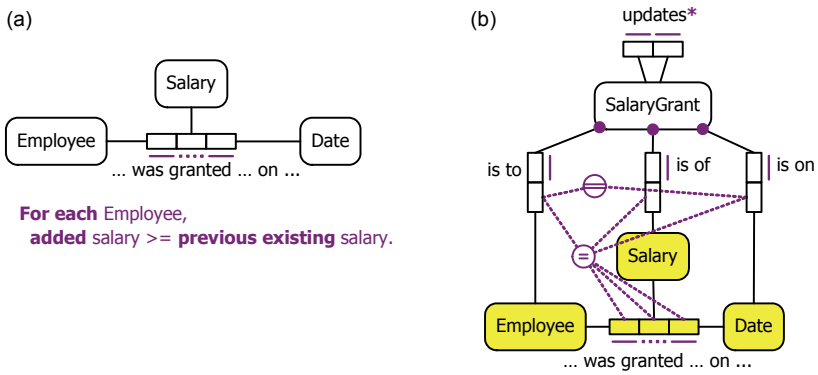


Fig. 4. Salary example with history

This dynamic rule syntax is much simpler than that used in [1], but still relies on a formal semantics being provided for the keywords. To address this need, we first objectify the ternary as SalaryGrant. In ORM 2, this is handled as situational nominalization [12, sec. 10.5], with SalaryGrant in 1:1 correspondence with the ternary, as enforced by the join equality constraint in Fig. 4(b). The case may now be handled as in the earlier case, with Date replacing Instant. In effect, this case is simpler, because the logging data is already available from the application database.

This example may be generalized to any historical fact type of the form $R(A_1.. A_n)$, where a uniqueness constraint spans n-1 roles, one of which is played by a temporal object type such as Date or Instant that is used to order the history.

5 Adding Instances of a Nonfunctional Fact Type

We now consider *adding fact instances* to a *nonfunctional fact type* (no single-role uniqueness constraint), such as the Seating was allocated Table association in Fig. 5, which shows a model fragment extracted from a restaurant application. A seating is the allocation of a party (of one or more customers) to one or more vacant tables. The asterisked rule is a derivation rule for the snapshot fact type Table is vacant.

This model maintains a *history* of seatings (for each table we record all the seatings it was previously allocated to). The value-comparison constraint (circled ">" with dots) verbalizes as: **For each** Seating, **existing** endTime > startTime. To ensure that no seatings that overlap in time occupy the same table, the dynamic rule in Fig. 5 declares 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 was allocated Table, and the elementary transaction involves the *addition* of 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.

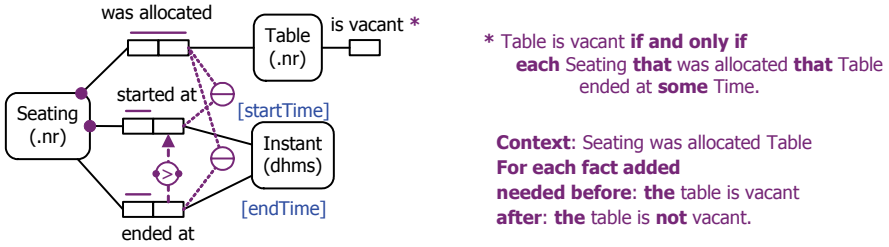


Fig. 5. Fragment of an ORM schema maintaining history of restaurant seatings

One static alternative to the dynamic rule was presented in [1], but this expression is extremely complex. A simpler static constraint formulation is possible using our logging semantics approach. The unshaded portion of Fig. 6 introduces TableSeating in 1:1 correspondence with Seating was allocated to Table, using the derivation rules shown to determine its start and end times. The value comparison constraint **For each** TableSeating, **existing** endtime > startime is omitted since it is implied. The updates and latest table seating predicates may be defined similarly to the previous examples, where one table seating updates another if and only if both seatings are for the same table and there is no intermediate seating for that table. Given the value comparison constraint, the dynamic rule to ensure no overlap may now be simply formulated as the following static constraint:

$$\forall x,y:\text{TableSeating}\ (y \text{ updates } x \rightarrow y.\text{startTime} > x.\text{endTime})$$

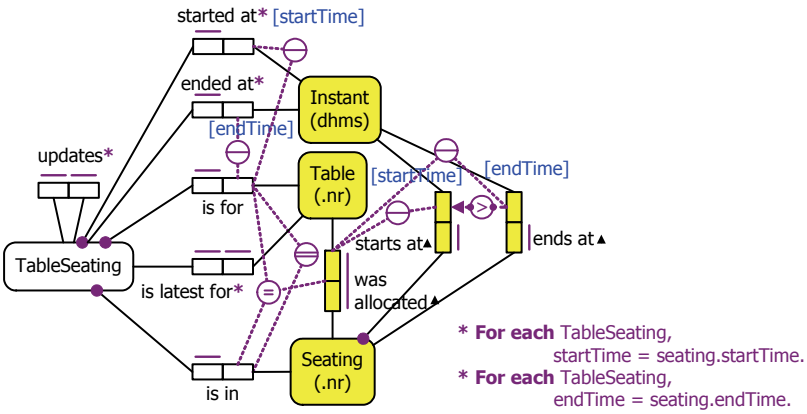


Fig. 6. Adding the logging subschema for Fig. 5

6 A More Complex Case Involving Derivation

In [1], a more complex case dealt with account transactions. We have space here to consider only transfer transactions, a basic schema for which is shown in Fig. 7. Transfer transactions transfer funds from one account to another. 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. The following dynamic constraint may be specified on transfer transactions:

Context: TransferTransaction
For each instance added
newFromBalance = (**old** fromAccount.balance – amount) **and**
newToBalance = (**old** toAccount.balance + amount) **and**
new fromAccount.balance= newFromBalance **and**
new toAccount.balance = newToBalance

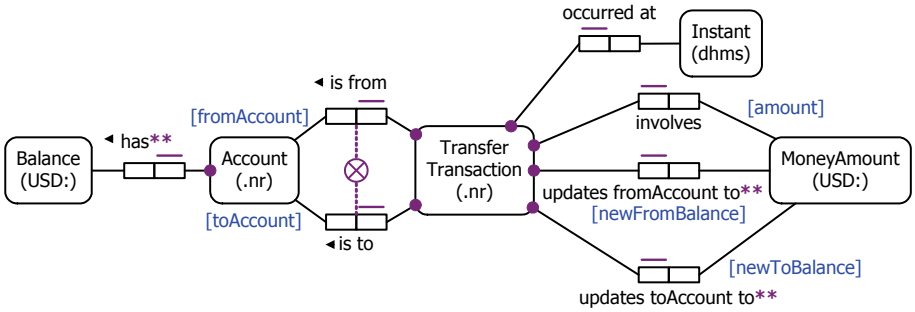


Fig. 7. An example involving historical and derived snapshot data

As with normal banking operations, history is maintained for all transactions in the user database, so the data needed for our logging semantics are essentially already there. Similar techniques to those discussed earlier can be applied to replace the dynamic rule with a static constraint.

7 Conclusion

This paper outlined an approach to provide a formal semantics for a proposed extension to Object-Role Modeling that supports declaration of dynamic rules. Dynamic rules differ from static rules by pertaining to properties of state transitions, rather than to the states themselves. We have restricted application of dynamic rules to so-called single-step transactions. Dynamic rules further specify an elementary transaction type by indicating which kind of object or fact (being added, deleted or updated) is actually allowed. Dynamic rules are equipped with preconditions 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.

We have provided such dynamic rules with a formal semantics based on sorted, first-order predicate logic. The key idea to our solution is the formalization of dynamic constraints as static constraints on the database transaction history. This *logging semantics* for dynamic rules makes explicit the log of all previous transaction instances pertaining to those specific rules. Snapshot data are maintained in the user database, whereas historical data are kept in the log database. This approach avoids the need to consider more complex logics such as temporal logics, while at the same time conforming in part to industrial database approaches that utilize log files to manage transactions. Future research options include extending this framework to cover other kinds of transactions (e.g. deletions) as well as dynamic rules involving more complex temporal expressions.

References

1. Balsters, H., Carver, A., Halpin, T., Morgan, T. 2006, Modeling Dynamic Rules in ORM, *On the Move to Meaningful Internet Systems 2006: OTM 2006 Workshops*, eds R. Meersman, Z. Tari, P. Herrero et al., Montpellier. Springer LNCS 4278, 1201-1210.
2. Bakema, G., Zwart, J. & van der Lek, H. 2000, *Fully Communication Oriented Information Modellin*g, Ten Hagen Stam, The Netherlands.
3. de Brock, E. O. 2000, 'A General Treatment of Dynamic Integrity Constraints'. *Data and Knowledge Engineering, 32*(3): 223-246.
4. Bruza, P. D. & van der Weide, Th. P 1989, 'The Semantics of TRIDL', Technical Report 89-17, Department of Information Systems, University of Nijmegen.
5. Chen, P. P. 1976, 'The entity-relationship model—towards a unified view of data'. *ACM Transactions on Database Systems*, 1(1), pp. 9–36.
6. Chomicki, J. 1992, 'History-less Checking of Dynamic Integrity Constraints', *ICDE 1992*: 557-64.
7. Curland, M. & Halpin, T. 2007, 'Model Driven Development with NORMA', *Proc. 40th Int. Conf. on System Sciences (HICSS-40)*, IEEE Computer Society, January 2007.

8. 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.
9. Girle, R. 2003, *Possible Worlds*, McGill-Queen's University Press, Montreal.
10. Halpin, T. 1989, 'A Logical Analysis of Information Systems: static aspects of the data-oriented perspective', doctoral dissertation, University of Queensland. Available online at http://www.orm.net/Halpin_PhDthesis.pdf.
11. Halpin, T. 2005, 'ORM 2', *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 676-87.
12. Halpin, T. 2006, 'ORM/NIAM Object-Role Modeling', *Handbook on Information Systems Architectures*, *2nd edition*, eds P. Bernus, K. Mertins & G. Schmidt, Springer, Heidelberg, pp. 81-103.
13. Halpin, T. & Morgan, T. 2008, *Information Modeling and Relational Databases*, *Second Edition*, Morgan Kaufmann, San Francisco.
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. Object Management Group 2007, *Semantics of Business Vocabulary and Business Rules (SBVR) Specification*. Online at: http://omg.org/technology/documents/ bms_spec_catalog .htm#SBVR.
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. Snodgrass, R.T. 1994, 'TSQL2Language specification', *SIGMOD Record 23(1)*, 65-86.
25. Warmer, J. & Kleppe, A. 2003, *The Object Constraint Language, Second Edition*, Addison-Wesley.