Temporal Modeling: Part 1

Terry Halpin
Neumont University

In previous articles (e.g. [6]), I've discussed ways to verbalize static business rules in a language easily understood by business people, and depicted many of these rules graphically in modeling approaches such as Object Role Modeling (ORM) [1], Entity-Relationship Modeling (ER) [2], and the Unified Modeling language (UML) [9, 10]. Most of these verbalization patterns have now been implemented in NORMA (Neumont ORM Architect) [8], an open source tool supporting second generation ORM (ORM 2) [5, 7]. Static rules (constraints, obligations, and derivation rules) apply to each state of the business domain, taken by itself. In contrast, instances of dynamic rules involve at least two states of the business domain (e.g. transitions from one state to the next). While some research has addressed high level verbalization of dynamic rules (e.g. [1]), this work is at a preliminary stage. Rather than moving on to the verbalization of dynamic rules at this point, I'll return to this topic in a later article. Instead, this and the next few articles will examine some related issues involving the impact of *time* in the conceptual modeling of business domains.

Temporal Data

In the database community, various proposals and products have been developed to address temporal data. For example, SQL/Temporal was considered as a temporal extension to SQL (on hold since late 2001, but might be revived) and many DBMS products have incorporated specific support for temporal features. Three basic *temporal data types* may be distinguished [12]:

- *Instant* (point in time, e.g. 2006 December 3, 2:50 p.m. UTC)
- *Interval* (duration of time, e.g. 3 weeks)
- *Period* (anchored duration of time, e.g. 2006 December 3 ... 2006 December 23 MST)

For each of these data types, various *temporal operators* may be defined. For example, date subtraction (−) applied to an ordered pair of dates (e.g. today and your birthdate) results in an interval (e.g. your age), and the overlaps operator applied to two periods yields a Boolean value (True or False) indicating whether those periods overlap in time. Many such temporal operators (e.g. before, after, during) have been defined, and are useful for specifying temporal queries and dynamic rules.

Currently, the only data type with good support from the standard version of most DBMSs is Instant (e.g. Date, Time, Timestamp). SQL-92 added support for Interval, but most commercial DBMSs have yet to provide this support. Period was initially included in the SQL/Temporal proposal for SQL:1999, but this temporal extension for SQL is on-hold and is not even included in SQL:2003. In the meantime, periods may be specified in the SQL standard either by start and end times, or by a start time and an interval. However, sophisticated support for temporal data is available in extensions (e.g. spatio-temporal data cartridges/blades) to some of the major DBMSs, and in specific temporal database products.

In database work, two kinds of time are often distinguished. *Valid time* denotes the time period during which a database fact was, is, or will be valid in the business domain being modeled. In contrast, *transaction time* denotes the time period during which a database fact is/was recorded in the database. In a database table, valid time and transaction time may often be captured by extra columns (e.g. a mandatory startVT and an optional endVT column, or a mandatory startTT and an optional endTT). If the end transaction time is unknown, it is sometimes recorded as the largest timestamp supported.

Different kinds of fact may be recorded with different *granularities of time* (e.g. second, minute, day). Time may also be measured using different *calendars* (e.g. Gregorian, Julian), and different *time zones* (e.g. UTC, Mountain Standard Time). For most modeling tasks, the choice of calendar and time zone is taken to be implicitly understood.

Temporal Object Types

Temporal extensions have been proposed for the relational model [3], as well as higher level data modeling approaches such as ER, ORM, and UML. For example, the MADS approach [11] provides detailed extensions to ER covering both spatial and temporal data. This article however focuses on the use of basic *temporal object types* (e.g. Date or Period) which may be used just like other object types in the business model, and for which basic temporal operators (e.g. –) are assumed to be predefined. For the rest of this article, time is understood to mean valid time, not transaction time.

Temporal object types may be classified into two kinds: once-only and repeatable. Each *once-only temporal object* is either a single *instant* or a single *period* of time, measured to a specified accuracy or temporal granularity (relative to an assumed calendar and time zone). Some simple examples are depicted in Figure 1, using ORM notation. Here "Year(CE)" indicates that the object type Year is identified by a CE (Common Era, formerly known as AD for Anno Domini) value. For example, Albert Einstein was born in the year 1879 CE. In this sense, a year is a 12 month period in the time stream. Likewise, a date is a period of one day in the time stream (e.g. 2006 Dec 5), a month is a period of approximately 30 days in the time stream (e.g. 2006 Dec), an hour is a period of 60 minutes in the time stream (e.g. 2006 Dec 5, 9 a.m. – 10 a.m). A time is an instant in the time stream measured to a specified granularity, such as a second, e.g. 2006-12-05 14:35:27 (i.e. 2006 Dec 5, 14 h 35 m 27 s). Object types like these are useful for recording when an individual event happened or will happen (e.g. the birth of Einstein, or the next Olympic Games).

Figure 1. Examples of once-only temporal object types.

Each *repeatable temporal object* corresponds to a set of instants or time periods, measured to a specified granularity (relative to an assumed time zone). Some simple examples are depicted in Figure 2, using ORM notation. Here a weekday is a day slot of any week (e.g. Sunday), a monthOfyear is a month slot of any year (e.g. January, or Month 1), and an hour slot is a one hour slot of a weekday (e.g. Monday starting at 9 a.m. and ending at 10 a.m.). Object types like these are useful when modeling schedules (e.g. a workout routine, a crop planting schedule, or a conference program).

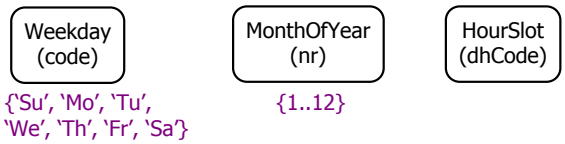

Figure 2. Examples of repeatable temporal object types.

Sometimes periods are modeled in more detail, by explicitly specifying their start and end times, to the desired level of temporal granularity (year, month, day, hour, minute etc.). For example, if periods are closed (we know both start and end) and the granularity is day, then they may be modeled in ORM as shown in Figure 3 and in UML as Figure 3(b). In ORM, a circled double line indicates an external uniqueness constraint used for the preferred identification scheme. As a non-standard extension to UML, the preferred identifier is shown by appending "{P}" to the identifying attributes.

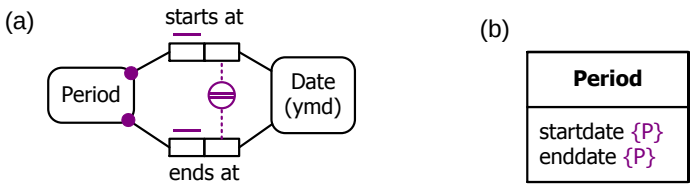

Figure 3. Closed periods modeled in (a) ORM and (b) UML.

Intervals (unanchored durations of time) may be modeled as a simple object type with a unit for the chosen granularity. As a simple example, if we choose to measure ages of fossils in year units, this may be modeled in ORM as shown in Figure 4(a) and in UML as shown in Figure 4(b). In ORM 2, the colon ":" indicates that "y" is a unit of a given type (display here suppressed, but can be expanded to "y: Interval").

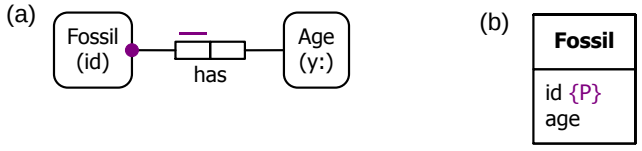

Figure 4. Modeling an interval in (a) ORM and (b) UML.

Fact Types and Events

In this article, the word "*fact*" means a proposition taken to be true by the business. A model of the business domain comprises a schema (denoting the business structure in terms of fact types and rules) and a population (business state, i.e. a set of fact instances). As the business changes over time, the set of recorded sentences describing it also changes. ORM represents all *fact types* as relationship types (e.g. Person smokes, Person was born on Date, Person plays Sport for Country). ER and UML represent fact types as relationship types (associations, e.g. Person drives Car) or attributes (e.g. Person.isSmoker, Person.birthdate). Regardless of how we represent fact types in a model, for the purposes of making decisions about how they are impacted by time it is convenient to distinguish at least *four kinds of fact type*:

- *Definitional* (truth of instances is a matter of definition rather than depending on an event)
- *Once-only* (instances correspond to a single event)
- *Repeatable* (instances may correspond to multiple events)
- *Time-deictic* (the meaning of instances depends on the time of utterance/inscription)

Definitional facts are completely devoid of any temporal aspect. They are necessarily true by definition, rather than depending on any event in the business domain. Figure 5(a) shows an ORM diagram for the definitional fact type PolygonShape has NrSides along with a sample fact population. Figure 5(b) shows the same fact type as a UML class diagram (using {P} as a non-standard notation for preferred identifier).

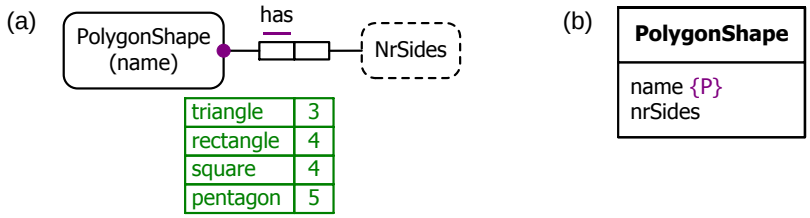

Figure 5. Example of a definitional fact type in (a) ORM and (b) UML.

Once-only facts correspond to a single event. Their truth derives from an event that can never be repeated in that business domain. For example, I was born in Australia. This is true because of my birth event (a state of affairs that happened many years ago). Whether or not reincarnation is possible, let us agree to ignore such possibilities for the business domain we are currently modeling (see Figure 6), where each employee has exactly one birth event. Figure 6(a) depicts the fact type Employee was born in Country in ORM, along with a sample population. Figure 6(b) depicts this as a many:1 association in UML, and Figure 6(c) depicts this as a single-valued attribute in UML.

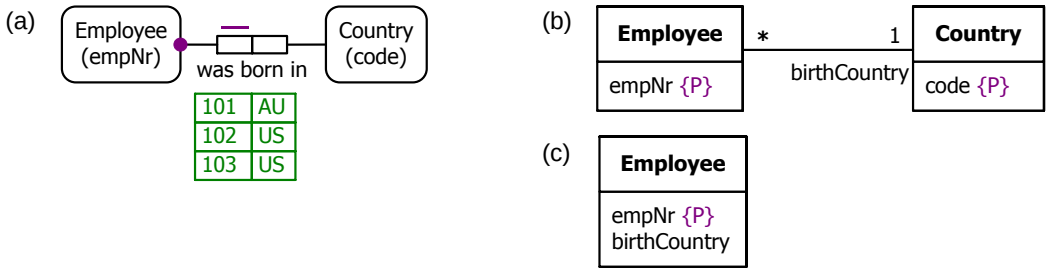

Figure 6. Example of a once-only fact type in (a) ORM and (b), (c) UML.

Repeatable facts may correspond to many events, in the sense that their truth may derive from any one of a set of events. For example, I have visited Belgium several times. The fact that I have visited Belgium is made true by any one of those visits. Because the same employee may repeatedly visit the same country, the fact type Employee visited Country is said to be repeatable. Figure 7(a) depicts the fact type Employee visited Country in ORM, along with a sample population. Figure 7(b) depicts this as a many:many association in UML, and Figure 7(c) depicts this as a multi-valued attribute in UML.

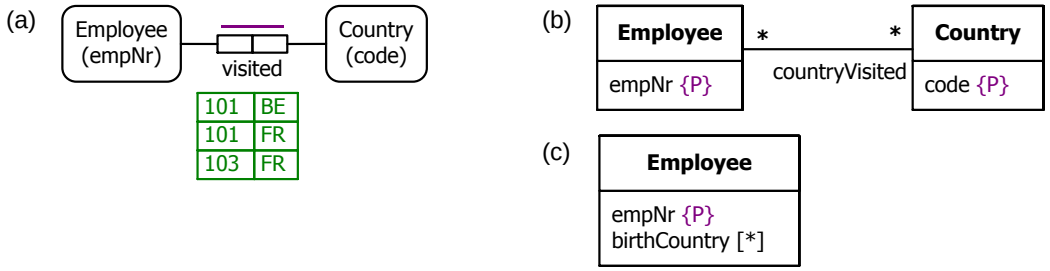

Figure 7. Example of a repeatable fact type in (a) ORM and (b), (c) UML.

I define a proposition as that which is asserted when a sentence is uttered or inscribed. The meaning of *time-deictic* sentences depends on when they are uttered/inscribed. For example, Student 1001 is *currently* enrolled in Course CS440 might be true today and false tomorrow. A proposition (e.g. Employee 101 visited Belgium) must be true or false (and hence is a truth-bearer). A state of affairs (e.g. Employee 101 visiting Belgium) is actual (occurs or exists in the actual world) or not. A state of affairs may be possible or impossible. Some possible states of affairs may be actual (occur in the actual world). States of affairs are thus truth-makers, in that true propositions are about actual states of affairs. In sympathy with the correspondence theory of truth, the relationship between propositions and states of affairs is one of correspondence rather than identity.

Definitional facts may be viewed as corresponding to states of affairs only in a trivial sense (if at all); since they have no temporal aspect, they need no special attention in this regard. In contrast, once-only and repeatable facts correspond to states of affairs that are either single events (once only) or event sets (repeatable). *For each once-only or repeatable fact type in a model, we need to determine what (if any) temporal information is needed*.

Once-only Fact Types

If a fact type is once-only and we wish to record for at least some of its fact instances when its underlying event (that made the fact instance true) occurred, then (if not already present in the model) add a functional fact type to record when the event occurred. To do this, relate a "key" of the original fact type to a temporal object type of the desired temporal granularity. Here "key of a fact type" means an object type that plays a mandatory, functional role in that fact type. A functional role is constrained by a simple uniqueness constraint. If we wish to record the event for each of the original fact instances, the key's role in that temporal fact type should be mandatory.

For example, consider the once-only fact type Employee was born in Country. Suppose that for each instance of this fact type we want to know when the underlying birth event occurred to a granularity of one day. The key object type is Employee (its role is mandatory and functional), so add the mandatory, functional fact type Employee was born on Date. Figure 8(a) depicts this in ORM showing a sample population. Figure 8(b) depicts this in UML using a birthdate attribute. If we instead chose the temporal granularity to be a year, then we would replace Date(ymd) by Year(CE), or birthdate by birthyear, and replace the date values by year values (1946, 1970, 1970).

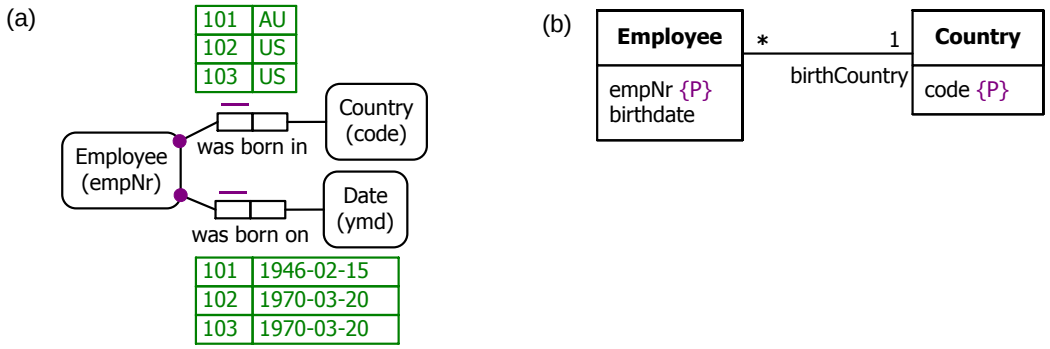

Figure 8. Adding a temporal fact type to record when once-only facts are "made true".

The example just discussed includes one temporal fact type (involving a temporal object type such as Date), and one non-temporal fact type (in this case a spatial fact type). Both these fact types are mandatory for employees. In other situations, either or both of these fact types might be optional for employees. This leads to three other basic patterns, as shown in Figure 9.

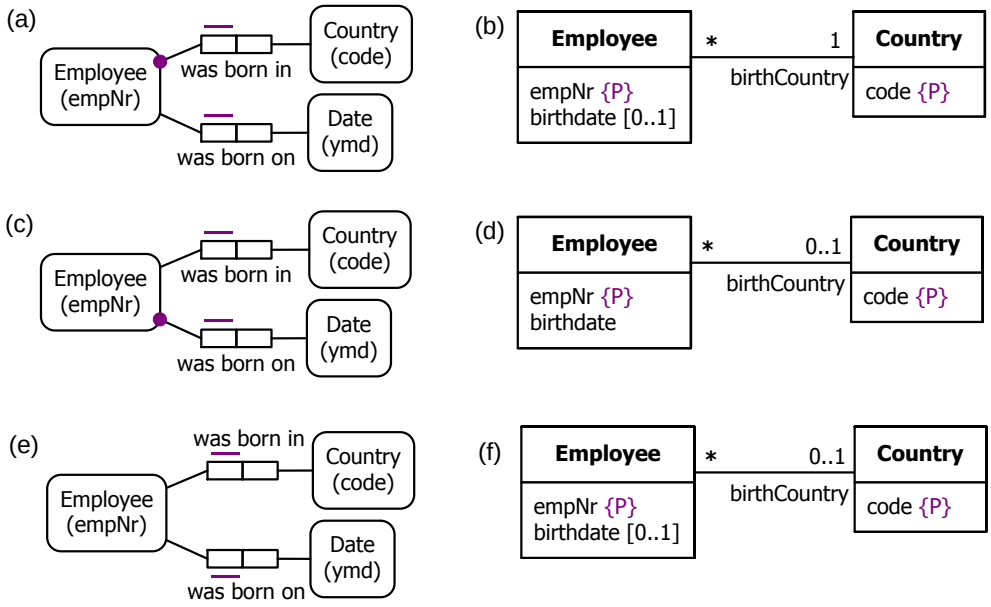

Figure 9. One or both of the facts is optional for employees.

If both facts are optional for employees, but one is known only if the other is known this leads to two other patterns as shown in Figure 10. The subset constraints depicted graphically in ORM are captured textually by OCL constraints in UML.

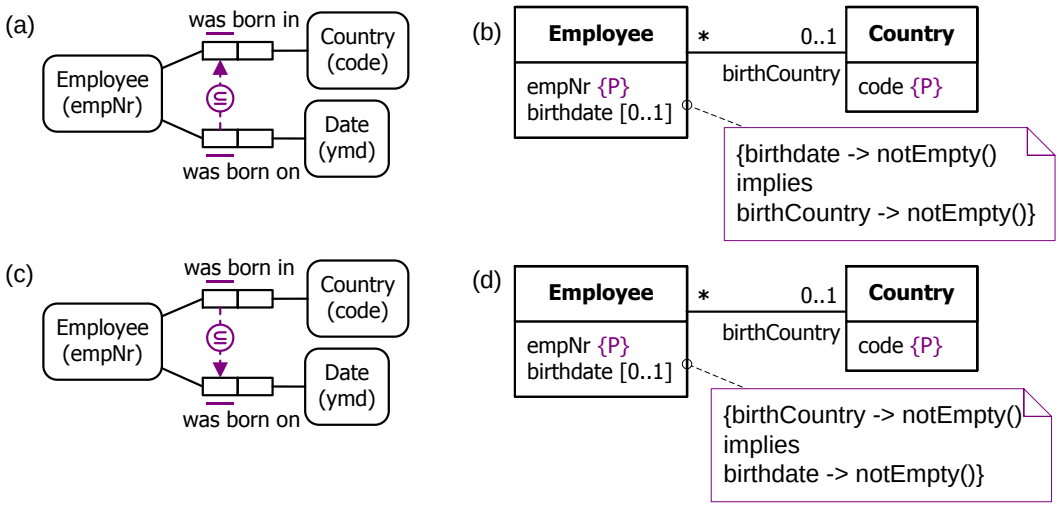

Figure 10. Subset constraints between optional roles.

In this article, the term "event" covers both *point events* (which occur at a single instant of time, e.g. a horse winning a race) and *period events* (which occur over a non-zero period of time, e.g. a rock concert). If the event described in the once-only fact is a period event and we wish to record the start and/or end of that period, then add temporal fact types of the desired temporal granularity. As an example, suppose the fact type Employee first visited Country is used to record the event when people first visit various countries, and suppose we always know the date when such visits started but only optionally know when they ended. For example, this could record details about my first visit to Belgium, my first visit to Japan, and so on. Figure 11 shows one way to model this in ORM and UML, objectifying the *m:n* visit association and attaching fact types for the start date (mandatory) and end date (optional). Alternatively, these two temporal fact types could be replaced by a single fact type linking to Period as a compositely identified object type or class. In industrial versions of ER that do not support objectification, the FirstCountryVisit type may be modeled as a co-referenced type with defining associations to Employee and Country.

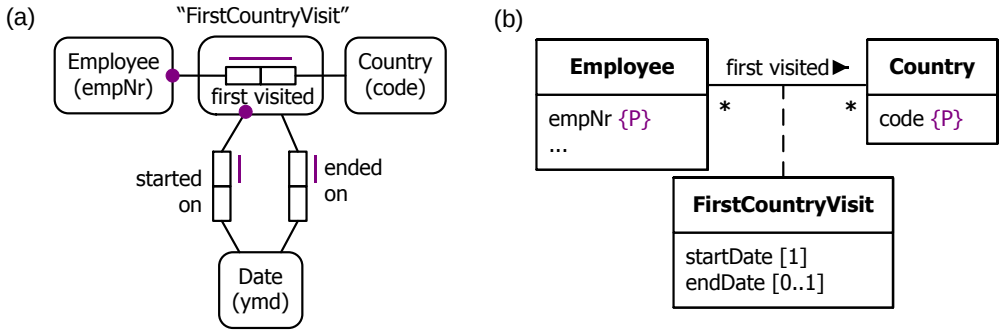

Figure 11. Recording an open period for a period event in (a) ORM and (b) UML.

Although cases like this typically involve an *m:n* association, similar patterns may be used for n:1 associations. For example, suppose the fact type Employee first lived in Country is used to record where people lived in the first part of their lifetime (e.g. I first lived in Australia). This fact type is *n:1* rather than *m:n*. In ORM 2 as well as UML, such associations may be objectified, and we may attach the temporal information in the same way as previously (see Figure 12(a) and (b)). In modeling approaches that do not allow such

objectification, the start and end date information may be attached directly to Employee (see Figure 12(c) for an ORM version of this alternative).

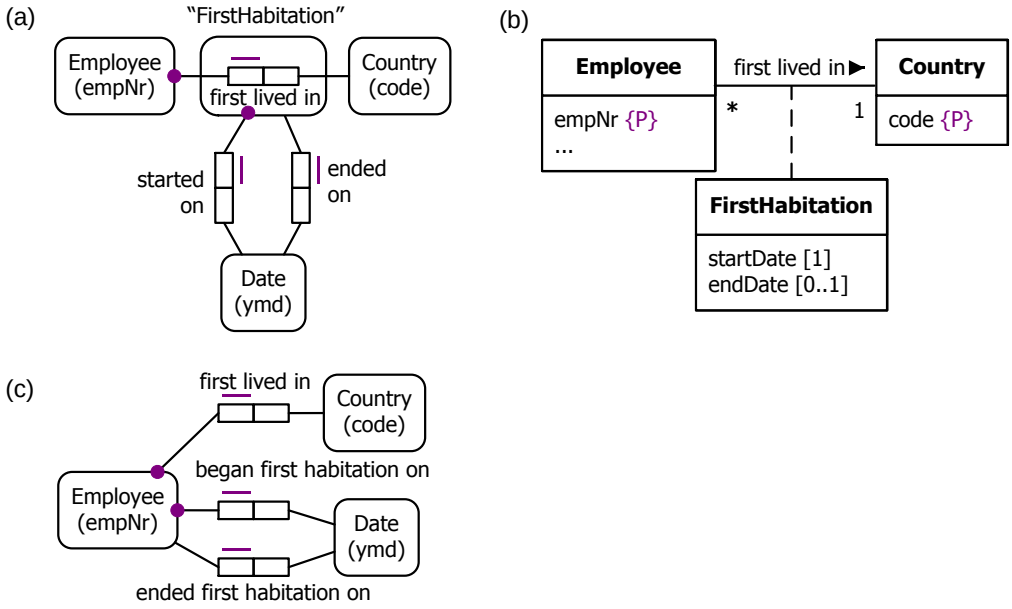

Figure 12. Period events modeled by an *n:1* association.

From a modeling perspective, catering for temporal aspects of once-only fact types is trivial, as the examples so far discussed demonstrate. Once we consider repeatable fact types and time-deictic fact types however, the story becomes much more complicated, as will be discussed in later articles.

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, pp. 1201-10.
2. Chen, P. P. 1976, 'The entity-relationship model—towards a unified view of data'. *ACM Transactions on Database Systems*, 1(1), pp. 9–36.
3. Date, C. J., Darwen, H. & Lorentzos, N. A. 2003, *Temporal Data and the Relational Model*, Morgan Kaufmann, San Francisco.
4. Halpin, T. 2001, *Information Modeling and Relational Databases*, Morgan Kaufmann, San Francisco.
5. 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.
6. Halpin T. 2006, 'Verbalizing Business Rules (Part 16)', *Business Rules Journal*, Vol. 7, No. 10 (Oct. 2006), URL: http://www.BRCommunity.com/a2006/b313.html.
7. Halpin, T. & Curland, M. 2006, 'Automated Verbalization for ORM 2', *On the Move to Meaningful Internet Systems 2006: OTM 2006 Workshops*, eds. R. Meersman, Z. Tari, P. Herrero et al., Montpellier. Springer LNCS 4278, pp. 1181-90.
8. NORMA website: http://sourceforge.net/projects/orm.
9. Object Management Group 2003, *UML 2.0 Infrastructure*, URL: http://www.omg.org/uml.
10. Object Management Group 2006, *Semantics of Business Vocabulary and Rules Interim Specification*. URL: www.omg.org/cgi-bin/doc?dtc/06-03-02.
11. Parent, C., Spaccapietra, S. & Zimanyi, E. 2006, *Conceptual Modeling for Traditional and Spatio-Temporal Applications: The MADS Approach*, Springer, Berlin.
12. Snodgrass, R. T. 2000, *Developing Time-Oriented Database Applications in SQL*, Morgan Kaufmann, San Francisco.