Temporal Modeling and ORM

Terry Halpin

Neumont University, Utah, USA.
e-mail: terry@neumont.edu

Abstract: One difficult task in information modeling is to adequately address the impact of time. This paper briefly reviews some popular approaches for modeling temporal data and operations, then provides a conceptual framework for classifying temporal information, and proposes data model patterns to address time-impacted tasks such as modeling histories, and tracking entities across time as they migrate between roles. Special attention is given to capturing the relevant business rules. While the data modeling discussion focuses on Object-Role Modeling (ORM), many of the basic principles discussed can be adapted to other approaches such as Entity Relationship Modeling (ER) and the Unified Modeling Language (UML).

1 Introduction

One challenging aspect of information modeling is to deal appropriately with temporal data and operations. This paper discusses how various temporal issues can be addressed at the conceptual level. The treatment focuses on Object-Role Modeling (ORM), 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. However, much of the discussion can be adapted to other data modeling approaches such as Entity Relationship Modeling (ER) [5] and the class diagramming technique within the Unified Modeling Language (UML) [19].

Unlike ER modeling and UML class diagrams, 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. In addition to ORM, fact-oriented modeling includes a number of closely related approaches, such as Natural language Information Analysis Method (NIAM) [29] and Fully-Communication Oriented Information Modeling (FCO-IM) [1]. For a basic introduction to ORM see [13], and for a thorough treatment see [15]. For a comparison of ORM with UML see [11].

Business rules include constraints and derivation rules. Static rules apply to each state of the information system that models the business domain (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.

Since the 1980s, many extensions to fact-orientation have been proposed to model temporal aspects and processes. The TOP model [10] allows fact types to be qualified by a temporal dimension and granularity. TRIDL [4] includes time operators and action semantics, but not dynamic constraints. LISA-D [18] supports basic updates. Task structures and task transactions model various processes [17], with formal grounding in process algebra. EVORM [24] formalizes first and second order evolution of information systems. Explorations have been made to address reaction rules [e.g. 16], and a proposal has been made to extend ORM with a high level textual language to specify dynamic rules in a purely declarative fashion [3].

Some fact-based approaches that share similarities with ORM include support for modeling temporality or system dynamics. For example, one extended fact-based model caters for different calendric systems and temporal operators [23], T-ORM provides basic support for temporal object evolution [8], the CRL language in TEMPORA enables various constraints, derivations and actions to be formulated on Entity-Relationship-Time (ERT) models [26, 28], and the OSM method includes both graphical and textual specification of state nets and object interactions [9].

Attribute-based methods such as UML and some extensions of ER incorporate dynamic modeling via diagrams (e.g. UML state charts and activity diagrams), with recent approaches such as the Business Process Modeling Notation (BPMN) gaining popularity for workflow modeling. The MADS (Modeling of Application Data with Spatio-temporal features) approach [22] extends ER with deep support for temporal data types and operators. For textual specification of dynamic rules, the most popular approach is the Object Constraint Language (OCL) [21], but the OCL syntax is often too mathematical for validation by nontechnical domain experts.

The rest of this paper is structured as follows. Section 2 reviews some standards and proposals for modeling temporal data and operations. Section 3 discusses conceptual issues and patterns for modeling facts that include temporal information. Section 4 proposes data model patterns, including dynamic rules where necessary, to capture histories of entities as they migrate between roles. Section 5 summarizes the main results, and lists references.

2 Temporal Data Standards and Proposals

Industrial standards and proposals for temporal data typically identify three main temporal data types: instant; duration; and period. An *instant* is a point in time (e.g. 2008 July 4, 2:00 p.m. MDT). A *duration* is a length of time (e.g. 2 weeks): this term is used in both 1S0 8601 [19] and XML schema (www.w3.org/TR/xmlschema11-2/), but is called "interval" in the SQL standard (www.iso.org). A *period* is an anchored duration of time (e.g. 2008 July 4 … 2008 July 7 PST). This term is used in SQL/Temporal (currently on hold), but is called *time interval* in ISO 8601 and *interval* in OWL-Time (www.w3.org/TR/owl-time/). Hence the term "interval" needs to be used with care. In ISO 8601, periods are closed (they have both a start and an end), but in OWL-Time they may be open (e.g. today onwards). In OWL-Time, a period with nonzero extent (if closed, it ends after it starts) is a *proper* period.

Most temporal standards draw from ISO 8601, which specifies many temporal terms, calendric systems (e.g. Gregorian, Julian), time zones (e.g. UTC, MDT), date and time formats etc. The SQL standard includes basic support for date, time, datetime and "interval" (in the sense of duration). XML Schema supports these and several other temporal data types (e.g. gYear for Gregorian year). The Time Markup Language (TimeML) covers date/time concepts as well as linguistic expressions to describe events (www.timeml.org/site/index.html). OWL-Time, a working draft to extend the Web Ontology Language OWL (www.w3.org/2004/owl) includes a set of temporal classes and predicates, and logical axioms about these. A subgroup of the Object management Group (OMG) is currently working to provide a unified treatment of basic temporal concepts for use in multiple approaches, including SBVR (Semantics of Business Vocabulary and Business Rules).

Instants are strictly ordered on a time axis, and may be compared using temporal operators such as < (for "is before") and ≤ (for "is at or before").

But many different proposals exist for an appropriate set of *temporal operators between periods* (time intervals). Many make use of *Allen's operators* [1], although some of these proposals [e.g. 7] wrongly construe many of Allen's definitions. Fig. 1 visually depicts Allen's operators as 13 mutually exclusive relationships between an ordered pair of closed, proper periods P_1 and P_2.

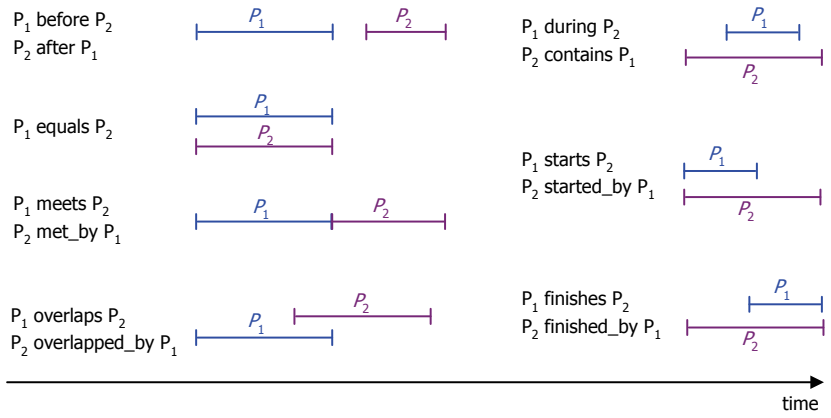

Fig. 1. Allen's operators for comparing closed, proper periods

OWL-Time accurately adopts Allen's operators, but this is unfortunate, as these operators are often poorly named, poorly defined, or poorly chosen. To begin with, overlaps and meets are intuitively understood as symmetric, but here are made asymmetric. If such asymmetric operators are to be used, they should be given intuitive names (e.g. leftOverlaps, rightOverlaps, leftMeets, rightMeets). Moreover, using our intuitive understanding of the terms, the contains, starts and finishes operators are too restrictive (e.g. equals should be treated as a special case of these, and starts should be a special case of during etc.).

Allen's before and after operators between periods require that the end of P_1 < the start of P_2. This could be acceptable if we adopt a quantized view of time, where time is composed of atomic *chronons*, since that would allow the periods to be contiguous.

However if we assume that time is continuous, this would not allow the periods to be contiguous, which goes against our common sense notion of "before". For example, one would normally agree that the Jurassic period is before the Trassic period (it immediately preceded it) and that yesterday is before today. Since OWL-Time and many other approaches leave the question open whether time is discrete or continuous, this choice of "before" is bound to confuse.

If time is regarded to be continuous, it is better to define "*before*" between periods so that the end of $P_1 \leq$ the start of P_2. With this definition, it follows that yesterday is before today, even if we adopt the ISO 8601 definition of calendar day as a "time interval starting at midnight and ending at the next midnight, the latter being also the starting instant of the next calendar day" [11]. Note that with this definition, any given midnight occurs on exactly two calendar days!

With this definition of calendar day, and using Allen's operators, yesterday meets today (yesterday's end is today's start) but does not overlap with today (since Allens' overlap requires yesterday's end to precede today's start). It seems preferable to define periods to *overlap* if and only if they have an instant in common. Apart from being symmetric, this is consistent with the way overlaps is defined in set theory and mereology. As illustrated later, it is also useful to distinguish between *trivial overlap* (where periods have exactly one instant in common) and *nontrivial overlap*.

Allen's *meets* operator is flawed, not only in being asymmetric, but in failing to cater for discrete time. If time is discrete, we should define periods to meet if they are contiguous (the end chronon of one immediately precedes the start chronon of the other). Note that this is one way to avoid the temporal version of the classic problem about where the midpoint goes when a line is divided in two [11, p. 110].

Whether or not time is continuous, we can measure time only to a limited accuracy, which effectively makes it discrete for information modeling purposes. Moreover, when recording information, we often choose a coarser temporal granularity than is physically attainable (e.g. we might track a patient's blood pressure at most daily or hourly). Pragmatically, we often juxtapose periods of a coarse granularity when tracking history rather than treating the end of one period to be the start of the next. For example, when updating an employee's salary, the new salary period is typically set to one day after the previous salary period. This avoids problems such as assigning two different salaries to an employee at the instant his/her salary is updated.

Another problem with Allen's operators is that they are often of little use pragmatically. In practice, one often needs instead to apply constraints involving our intuitive notions or overlapping, nonoverlapping, containment, etc.

Recently we investigated OWL-Time from an ORM perspective, and found it to be seriously deficient. Apart from its unwise adoption of Allen's operators, OWL-Time's axiomatic development appears to be problematic (partly because of its silence on the discrete/continuous time issue), and is incomplete. As a simple example of the latter, Fig. 2 shows an ORM schema for a fragment of the OWL metaschema dealing with duration descriptions. The inclusive-or constraint (circled dot) and preferred external uniqueness constraint (circled double-bar), are not captured in OWL-Time, but are clearly needed. As a general comment about OWL itself, OWL models are much easier to formulate if generated from ORM rather than working directly in OWL.

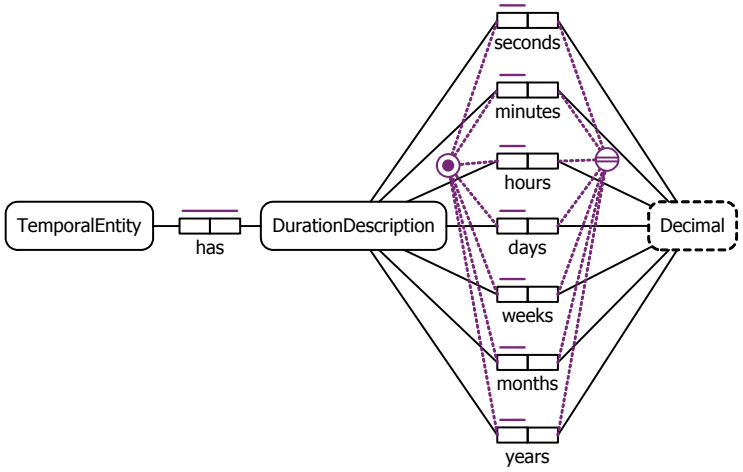

Fig. 2. ORM schema for duration descriptions in OWL-Time (constraints added)

3 Conceptual Modeling of Temporal Facts

At the conceptual level, and ORM in particular, basic *temporal object types* (e.g. Date or Period) may be used in models like other object types, with relevant temporal operators (e.g. –, overlaps) predefined for the type. For ORM, we introduce some useful classifications. A temporal object type is once-only or repeatable. A *once-only temporal object* is a single instant or period. Once-only types (e.g. Year(CE), Month(ym)) are useful for recording when an individual event (non-repeatable) happened or will happen (e.g. the election of the next US president). A *repeatable temporal object* corresponds to a set of instants/periods. Repeatable object types (e.g. Weekday(.code), MonthOfYear(.nr)) are useful for modeling schedules (e.g. a workout routine).

Periods may be modeled by explicitly indicating their start and end times (if known), using an external uniqueness constraint to provide an identifier. Durations may be modeled by a simple object type with a unit for the chosen temporal granularity, e.g. Age(y:).

For modeling purposes, a fact is a proposition taken to be true by the business, and a fact type is a set of possible fact instances. We classify fact types as definitional, once-only, or repeatable. *Definitional facts* are true by definition, so have no temporal aspect. For example, the fact type PolygonShape has NrSides is definitional. Each *once-only fact* corresponds to a single event. Its truth is determined by an event that can never be repeated in the business domain (e.g. Terry Halpin was born in Australia). So ignoring reincarnation, the fact type Person was born in Country is once-only. *Each repeatable fact* corresponds to a set of events. Its truth is determined by any one of a non-unit-set of repeatable events (e.g. Terry Halpin visited Mexico). So the fact type Person visited Country is repeatable. For each once-only or repeatable fact type in a model, we need to determine what (if any) temporal information is needed.

An event may be a *point event* (occurs at an instant) or a *period event* (has nonzero duration, e.g. your reading of this paper). For once-only fact types relating to point events, if we wish to record when at least some instances of those events occurred, add a temporal fact type of the desired granularity (e.g., for Person was born in Country, add Person was born on Date, or Person was born in Year etc.). For once-only fact types relating to period events, to record when at least some instances of those events occurred, add temporal fact types of the desired granularity to note the start and end (if known) of the period (e.g. FirstReading started at Time(dhm), FirstReading ended at Time(dhm)). Here FirstReading may be modeled as an objectification of Person first read Paper, or as a coreferenced type identified by FirstReading is by Person, FirstReading is of Paper. If Period is explicitly introduced (e.g. FirstReading occupied Period) then the start and end predicates are attached to Period. If we are not interested in distinguishing start and end, we may model it as for a point event using coarse granularity (e.g. FirstReading occurred on Date).

While once-only fact types are unchangeable, repeatable fact types may be *changeable* (e.g. Patient has Temperature, Patient is allergic to Drug). For such fact types, if we are interested only in the current *snapshot* then no remodeling is needed (simply update the fact populations as required). To maintain *history* of a changeable fact type that is *functional*, we may simply insert into its key the relevant role played by a temporal object type of the desired granularity (e.g. Patient(.nr) at Hour(dh) had Temperature(°C:)). This flattened approach may be remodeled using nesting or coreferencing in the usual way. For example, use the fact type TemperatureMeasurement recorded Temperature, where TemperatureMeasurement is either an objectification of Patient had temperature taken at Hour or is coreferenced by TemperatureMeasurement is of Patient and TemperatureMeasurement is at Hour. As a further alternative, a simple identifier may be introduced for the measurement object type, e.g. TemperatureMeasurement(.nr).

To maintain *history of nonfunctional fact types* that are changeable, the previous patterns may be modified to include a distinguishing temporal role (e.g. startdate or starttime), to distinguish different events that make the same fact true. Consider for example, the report of country visits shown in Fig. 3. For each visit, the start date is known and possibly the end date is known ("?" denotes a null). Employee 102 visited The Netherlands twice, and we wish to retain a record of both visits, so we cannot model visits by the simple fact type Employee visited Country.

Visit:

empNr	*countryCode*	*startdate*	*enddate*
101	NL	2000-01-01	2000-01-15
101	CA	2008-02-15	?
102	NL	2007-06-05	2007-06-20
102	BE	2007-06-20	2007-06-25
102	NL	2008-06-08	?

Fig. 3. Record of visits to countries by employees

Let us assume that for any given date, an employee may start visiting or end visiting at most one country (if this is not true, replace Date by Instant). Fig. 4 shows basic ORM schemas for this situation, in (a) nested, (b) coreferenced, and (c) flattened form. Other solutions are to introduce a simple identifier for Visit, or an ordinal number as part of the identifier (e.g. Fred's 2nd visit to France). ORM's current relational mapping algorithm (Rmap) maps (a) and (b) to (d), and (c) to (e).

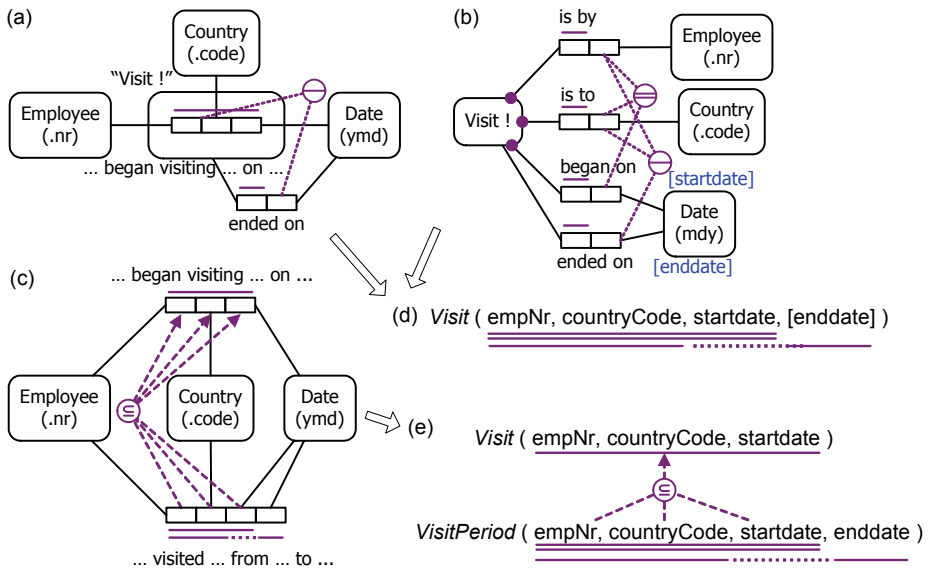

Fig. 4. Basic ORM schemas for modeling the data in Fig. 3

In verbalizing the data in the Fig. 3 report, it seems most natural to use a quaternary for row 1 and a ternary from row 2, leading to the flattened solution (Fig. 4(c)). But Rmap punishes the modeler for this choice by mapping to the 2-table relational schema (Fig. 4(e)) instead of the simpler 1-table schema (Fig. 4(d)) obtained from the nested or coreferenced schemas. As an enhancement to the NORMA tool [6] for ORM 2, we are modifying Rmap to allow retention of the flattened schema while still offering the single table relational map by default (the user may override this choice).

If we also want to talk about visits (e.g. to record the main purpose of a visit as business or pleasure), the nested or coreferenced solutions are far preferable (compare adding the fact type Visit is for VisitPurpose with adding the quaternary Employee began visiting Country on Date for main- VisitPurpose together with another 3-part subset constraint).

Two temporal constraints need to be added to the schemas in Fig. 4. The first is a value-comparison constraint that is most easily understood using schema Fig. 4(b). This constraint may be depicted graphically in ORM 2 [15, p. 290] or verbalized textually as: **For each** Visit, **existing** enddate ≥ startdate. The "**existing**" qualification applies the condition only where an end date does exist.

The second constraint requires that no two visits by the same employee overlap *nontrivially* in time (the data in rows 3 and 4 of Fig. 3 indicate that trivial overlap is allowed in this business domain). This constraint cannot be captured graphically in ORM 2 but can be specified textually in either static form or dynamic form. The static form is complex (cf. the restaurant seating example in [3]), whereas given the value-comparison constraint, the dynamic form of the overlap constraint may be rendered simply: **For each** Employee, **existing previous** Visit.enddate ≤ **added** Visit.startdate. For discussion on the semantics underlying such syntax, see [3].

Note that if we have *complete* knowledge of all visit periods by an employee, we could derive the quaternary in Fig. 4(c) from the two ternaries Employee began visiting Country on Date and Employee ended a visit to Country on Date, with a pair subset constraint between the Employee-Country role pairs (from the enddate fact type to the startdate fact type), by ordering visit periods sequentially. However, if we have incomplete knowledge we cannot derive the quaternary, and the two ternaries solution must be rejected (e.g. the ternary solution allows a population of the two tuples <101, NL, 2000-01-01, ?> and <101, NL, ?, 20008-02-15>, but the employee might have made two visits, not one visit. This raises a fine point about the notion of *elementarity* of facts. Assuming complete knowledge, and the derivation possibility by ordering visit periods, is the quaternary fact "Employee 101 visited the Country 'NL' from the Date '2000-01-01' to the Date '2000-01-15'" elementary? We leave further investigation of this issue as a research topic.

Sometimes, business rules require no overlap (trivial or nontrivial). For example, in modeling pay awards, it is normal to require for each JobPosition that no two (startdate, enddate) periods overlap. And if we modify the country visit example to country habitation, where on a given date a person may start or end residing in at most one country, the country role is excluded from the identification scheme for habitations, and no overlap is allowed.

As a final note before ending this section, one difference between the ORM and CogNIAM (www.pna-group.com) flavors of fact-oriented modeling is that ORM forbids the inclusion of nulls in asserted (non-derived) facts. For example, ORM ignores the null in verbalizing the ternary fact on row 2 of Fig. 3, whereas CogNIAM allows this row to be verbalized as a quaternary including the null. In ORM we have found it useful to be able to specify additional constraints on derived fact types, where nulls are allowed in fact populations, but have found it safer to avoid nulls in asserted fact types (requiring any asserted fact to be either elementary or existential). Which of these approaches is better in this regard is left as a topic for further discussion.

4 Modeling History of Migration between Role Subtypes

In previous work [14], we outlined a general approach for modeling histories of entities as they migrated from one role subtype to another. In this section, after a brief review of some basic concepts, we now extend that work.

A type is *rigid* if each instance of it must remain in that type for the duration of that instances's lifetime (e.g. Person, Tree), otherwise the type is a *role type* (e.g. Employee, Cricketer). Over time, an entity may move from one role type to another. Suppose each role has specific details of interest and we want to maintain this history of an entity as it changes roles. We now classify role subtypes as once-only or repeatable. With a *once-only role subtype*, objects can never return to play that role again once they have left the subtype (e.g. Child, SinglePerson). With a *repeatable role subtype* objects can return to play that role again (e.g. Employee, MarriedPerson).

Histories involving transitions between once-only role subtypes may be modeled using a *successive disjunctions pattern*. For example, Adult is a subtype of TeenagerOrAdult which in turn is a subtype of ChildOrTeenagerOrAdult. Subtype specific de-

tails may now be easily retained (e.g. Adult has favorite- Book, TeenagerOrAdult as a teen had favorite- PopGroup, ChildOrTeenagerOrAdult as a child had favorite- Toy). This arrangement automatically caters for the linear transition order from role to role.

If the roles are once-only, then an alternative solution is to use what we call the *once-only role playing pattern*, augmented by a *dynamic constraint* to constrain the possible role transitions. For example, the child-teenager-adult example may be modeled as shown in Fig. 5.

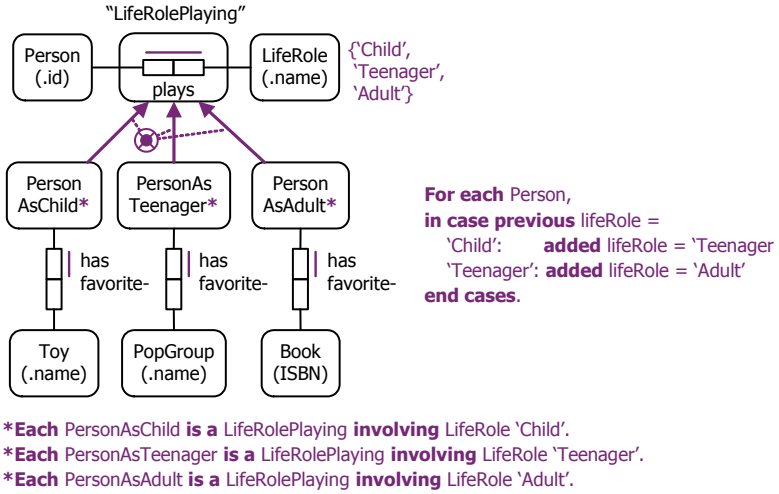

Fig. 5. Example of once-only role playing pattern with dynamic rule

If a role subtype is repeatable, the previous approaches cannot record history of multiple playings of the same role by the same object. To address this problem, we provide what we call the *repeatable role playing pattern*, which includes the start-time of a role playing as part of its natural identifier. One version of this is shown in Fig. 6 (minus the dynamic rule). This assumes that a person may begin or end a given role at most once on the same date (if this is not true, replace Date by Instant). This pattern allows that a person may begin or end multiple roles on the same date. Alternative versions of the pattern introduce either simple identifiers, or ordinal numbers as partial identifiers, for RolePlaying. A concrete example is given in Fig. 7

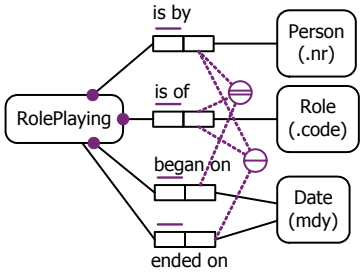

Fig. 6. One version of the repeatable role playing pattern

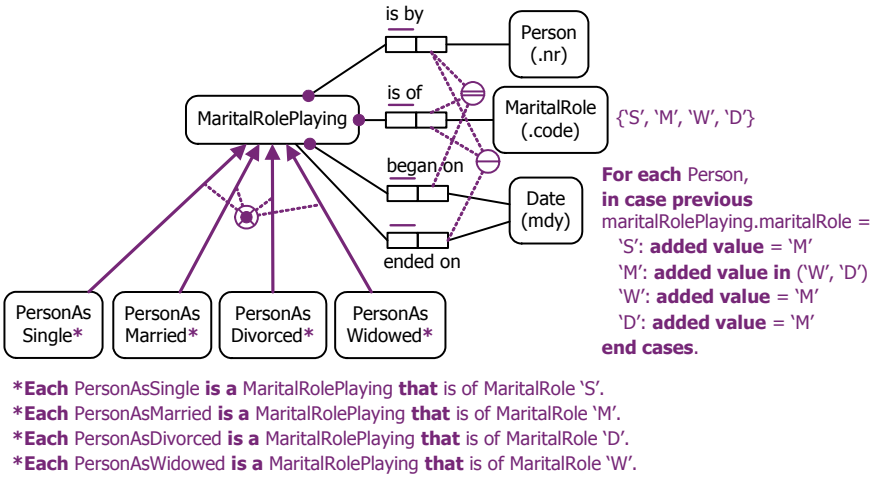

Fig. 7. Example of repeatable role playing pattern with dynamic rule

5 Conclusion

This paper reviewed some approaches to temporal data and operations, identified problems with Allen's operators and OWL-Time, suggested conceptual ways to classify temporal information, raised some issues regarding elementarity, and proposed modeling heuristics and data model patterns to address time-impacted tasks such as modeling histories, and tracking entities across time as they migrate between roles.

While the graphic depiction of ORM models has been implemented in the NORMA tool, the detailed syntax for textual specification of temporal and dynamic rules (including scheduling) and the generation of code from such textual rules is still a work in progress. We plan to extend the NORMA tool to support such rules, and also implement a mapping from ORM to OWL, work on which has already begun. It may also be worthwhile considering graphical extensions to ORM to directly support some temporal aspects (e.g. marking types as once-only or repeatable).

References

1. Allen, J. 1983, 'Maintaining Knowledge about Temporal Intervals', *Communications of the ACM* 26, 11, 832-843.
2. Bakema, G., Zwart, J. & van der Lek, H. 2000, *Fully Communication Oriented Information Modellin*g, Ten Hagen Stam, The Netherlands.
3. 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.
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. Curland, M. & Halpin, T. 2007, 'Model Driven Development with NORMA', *Proc. 40th Int. Conf. on System Sciences (HICSS-40)*, IEEE Computer Society, January 2007.
7. Date, C., Darwen, H. & Lorentzos, N. 2003, *Temporal Data and the Relational Model*, Morgan Kaufmann, San Francisco.
8. Edelweiss, N., de Oliveira, J., de Castilho, J., Montanari, E. & Pernici, B. 1994, 'T-ORM: Temporal aspects in objects and roles', *Proc. First International Conference. on Object-Role Modeling*, eds T. Halpin & R. Meersman, University of Queensland, pp. 18-27.
9. Embley. D. W. 1998, *Object Database Development*, Addison-Wesley.
10. 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.
11. Halpin, T. 2005, 'Information Modeling in UML and ORM: A Comparison', *Enc.of Inf. n Science and Technology*, vol. 3, ed. M. Khosrow-Pour, IGI, Hershey, pp. 1471-5.
12. 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.
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. 2007, 'Subtyping Revisited', *Proc. CAiSE'07 Workshops*, *vol. 1*, eds. B. Pernici & J. Gulla, Tapir Academic Press, pp. 131-141.
15. Halpin, T. & Morgan, T. 2008, *Information Modeling and Relational Databases*, *Second Edition*, Morgan Kaufmann, San Francisco.
16. Halpin, T. & Wagner, G. 2003, 'Modeling Reactive Behavior in ORM'. *Conceptual Modeling – ER2003*, Proc. 22nd ER Conference, Chicago, October 2003, Springer LNCS.
17. ter Hofstede, A. H. M. 1993, 'Information Modelling in Data Intensive Domains', PhD thesis, University of Nijmegen.
18. 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.
19. ISO 2004, *ISO 8601:2004(E): Data elements and interchange formats—Information interchange—Representation of dates and times*. ISO, Geneva.
20. Object Management Group 2003, *UML 2.0 Superstructure Specification*. Online at: www.omg.org/uml.
21. Object Management Group 2005, *UML OCL 2.0 Specification*. Online at: http://www.omg.org/docs/ptc/05-06-06.pdf.
22. Parent, C., Spaccapietra, S. & Zimanyi, E. 2006, *Conceptual Modeling for Traditional and Spatio-Temporal Applications*, Springer, Berlin.
23. Petrounias, I. & Loucopoulos, P. 1994, 'Time Dimension in a Fact-Based Model', *Proc. First International Conference. on Object-Role Modeling*, eds T. Halpin & R. Meersman, Key Centre for Software Technology, University of Queensland, pp. 1-17.
24. Proper, H. A. 1994, 'A Theory for Conceptual Modeling of Evolving Application Domains', PhD thesis, University of Nijmegen.
25. Snodgrass, R. 2000, *Developing Time-Oriented Database Applications in SQL*, Morgan Kaufmann, San Francisco.
26. Sowa, J. 2000, *Knowledge Representation*, Brooks/Cole, Pacific Grove.
27. 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.
28. 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.
29. Wintraecken J. 1990, *The NIAM Information Analysis Method: Theory and Practice*, Kluwer, Deventer, The Netherlands.