Temporal Object Modeling: Diagramming Conventions and Design Considerations

Richard Vidgen
UMIST (United Kingdom)

Many applications that use a database management system are required to hold temporal data, although few commercially available DBMSs provide temporal facilities. Furthermore, requirements specification techniques, such as structured methods and Object-Oriented Analysis, typically do not support a temporal modelling notation. This paper describes components of temporal modelling, such as granularity, events, and transaction time and introduces a temporal diagramming notation that is suitable for Object-Oriented analysis. Examples are furnished of class diagrams for time-stamped attributes and associations. A logical relational design for a class diagram is given that could be used as the basis for physical design of a relational database. The future direction is represented by temporal modelling of object reclassifications, which will require access to business model meta-data.

A common business requirement of management information applications is to retain a history of how data have changed over time. This is particularly the case with financial and management accounting applications in which a commonly found requirement is: to report current year results using the current chart of accounts; to report current year results using the previous year's chart of accounts; and to report current year results using a future chart of accounts that will be introduced for the next financial year. Any operational computer system that requires a history of changes to be maintained for audit trail purposes, such as a medical information system, or for trend analysis, such as a history of budget forecasts in a Decision Support System, has a need for temporal modelling of data.

However, many modelling notations push the issue of time to the periphery. For example, in the Structured Systems Analysis & Design Method (SSADM), which is the most widely used system development method in the UK (see Goodland & Slater, 1995 for a description of SSADM Version 4 and Robinson & Berrisford, 1994 for an Object-Oriented approach to SSADM), the logical data structure does not usually feature time as an entity with the consequence that the need to keep history may be recorded separately as a functional requirement. This can result in a system designed on the basis of data structures that take little or no account of the need to hold temporal data. Where a need for history is recognized it might be accomplished through the introduction of naive denormalizations. For example, rather than record all the changes to a customer's credit limit over time, an upper limit of three might be imposed, with limit1, limit2, and limit3 being held as attributes of a Customer entity type. Once a system has been built and a temporal requirement recognized belatedly then less than satisfactory work-arounds are often introduced, such as taking multiple copies of the database and using the different physical versions to produce the required temporal effects.

There has been considerable interest in temporal modelling in data models and databases and a number of surveys have been conducted of the literature, including (McKenzie, 1986; Roddick & Patrick, 1992; Snodgrass 1995). Time semantics have been incorporated in a number of conceptual modelling techniques and Theodoulidis & Loucopoulos (1991) describe a number of approaches, including: the infological data model (Langefors, 1973; Langefors & Sundgren 1975); the conceptual information model (Bubenko, 1977); the Time-extended Entity-Relationship model (Klopprogge, 1983); the Historical Database Model

Manuscript originally submitted April 5, 1995; Revised July 31, 1995; Accepted November 17, 1995 for publication.

(Clifford & Warren, 1983); the Entity Relationship Attribute Event model (Dubois et al., 1986); and TEMPORA, which has been developed as part of the European Union ESPRIT initiative (Theodoulidis et al., 1990; 1991). There has also been considerable work in the area of temporal query languages, such as ERT-SQL (Entity-Relationship Time Structured Query Language) (Snodgrass, 1987), including O-O extensions to query languages (see Snodgrass (1995) for a review of temporal query languages).

The interest in temporal modelling has grown throughout the 1980s and, although grounded in a semantic data modelling and database design tradition, an invitational workshop held at the University of Arizona in March 1994 reported that the participants felt that Object-Oriented data models provide the most appropriate basis for future work, while recognizing that temporal object bases are still in the early stages of development and commercial adoption (University of Arizona, 1994).

The first objective of this paper is to recommend a diagramming notation that can be used to capture temporal business requirements in an object model. Mainstream Object-Oriented analysis and design methods, such as that of Martin & Odell (1992), Coad & Yourdon (1991), and Rumbaugh et al's OMT (1991) do not provide a notation for time-stamping and do not address directly the issue of modelling temporal aspects. Although the simplicity of the time-stamped object model makes it a useful medium for communication with business/user personnel and for the capture of requirements, the designer cannot avoid the complexity of time-stamped data structures when performing logical and physical design. Therefore, the second objective is to show how temporal requirements can be modelled in a non-temporal object model and hence form a basis for implementation in non-temporal environments. Examples of logical relational design for time-stamped data structures are also given since relational technology is still prevalent in practice, particularly in management information system applications. The third objective is to consider what contribution an Object-Oriented (O-O) approach can make to time-stamped data, particularly through the addition of encapsulated behaviour.

The structure of the paper is as follows: in section 2 some basic assumptions about temporal characteristics are described; in section 3 temporal modelling requirements are introduced; in section 4 diagramming notations and expanded class diagrams are developed; section 5 considers how an O-O paradigm might contribute to temporal modelling; section 6 shows how time-stamped data structures can be implemented relationally; and section 7 introduces the temporal modelling of state transitions.

Temporal characteristics

Before looking at temporal object modelling and class diagrams it is appropriate to consider some basic assumptions concerning time.

Granularity

If we are told that "John Smith became the owner of the motor car with registration A123 XYZ on 06-Sep-1994" then it will not be possible to ascertain the ownership of the car at 3:00 pm on 06-Sep-1994 since the change in ownership has been recorded using a granularity of date. If it is a requirement to know more precisely the point in the 24 hour period that transfer of ownership took place then a finer granularity might be introduced, such as minutes:"John Smith became the owner of the motor car with registration A123 XYZ at 15:25 hours on 06-Sep-1994." If we assume that a car must have an owner, then the fact that the previous ownership ended at 15:24 is derivable. However, this leaves an indeterminate period of one minute where the ownership, as reported in a computer system, is indeterminate. This is not because ownership is necessarily in dispute in the real world - it arises as a result of the choice of granularity of time used to record the change in ownership. Yet finer granularities of time may be introduced (milliseconds, microseconds, nanoseconds, etc.) but, assuming that time is infinitely divisible, then the indeterminacy can never be removed entirely.

Events

Changes in data are attributed to events. For example, the event "car bodywork colour changes" results in a new value for the data item colourOfBodywork; the event "product price changes" results in a new value for the data item productPrice. An event is assumed to have a duration of one unit of time of the finest granularity defined to the temporal object model. Thus, although events should not be considered to be instantaneous (of zero duration), this assumption means that they do not need to be included as permanent aspects of the temporal object model. This is obviously a simplification that might be acceptable for some data items, such as productPrice, where a discrete change might well be a reasonable assumption. For other items, such as colourOfBodywork the assumption of a discrete change might not be sufficient. There can be a significant duration in changing the colour of a car of hours or even days. During this period the car is neither one colour nor another and if this is to be captured in the object model then it will be necessary to introduce a state that allows cars to be in the state "being re-sprayed" together with two events such as "re-spray begins" and "re-spray ends." Whether this is necessary can only be decided based upon an analysis of the requirements that the information system is to satisfy.

Frequency

In addition to the granularity of the time interval it is also useful to specify how frequently the value of a data item may change. For example, a granularity of date could be used to

record changes in product prices, while specifying a maximum frequency of monthly. This would ensure that any price change for a specific product is effective for a minimum period of one month.

Transaction time

Recording the transaction time will make it possible to distinguish between when the event was recorded and when it was (will become) effective. Using the example of a product price change, there are three situations of interest:

- *transaction time = effective time:* the time at which the product price change is recorded occurs at the same time as the price change is effective (contemporaneous);
- *transaction time < effective time*: the time at which the product price change is recorded occurs before the price change becomes effective (future-dated);
- *transaction time > effective time*: the time at which the product price change is recorded occurs after the price change becomes effective (back-dated).

In this paper it is assumed that the time the transaction occurred is always recorded; it will not be modelled explicitly in the object model. Also, although we refer to object histories, it should be apparent that this term also includes future values.

Temporal modelling

Object model calendar classes

Calendar classes are used for specifying temporal characteristics, such as granularity and frequency (figure 1). Classes are shown as soft boxes and class names begin with an upper case letter (e.g., Hour), attribute names begin with a lower case letter (e.g., hourOfDay) and are shown in the middle section of the class box, and methods are shown in the bottom section of the class box. Three dots are used to show that methods and/or attributes have been suppressed for diagrammatical representation (de Carteret & Vidgen, 1995). The cardinality of an association is specified using minimum and maximum values. For example, each instance of the class Month must be associated with one and only one instance of Year, while each instance of Year may be associated with many instances of Month (Martin & Odell, 1992; de Carteret & Vidgen, 1995). The calendar class can be generalized (Theodoulidis et al., 1991) and modelled using a recursive structure; it should also cater for user-defined time, such as retail periods. Date has been used rather than Day in figure 1, and Time rather than Second, reflecting their common, if ambiguous, general usage. The calendar classes are part of the business object model and will either be supplied by software vendors or constructed by the developers - either way, the calendar should be treated as a reusable class structure.

Meta-model classes

If price is a mandatory attribute for product then price will be *continual* over time. Other data items might be optional, such as telephone number, and therefore have gaps in their history, thus exhibiting *discontinual* temporal behaviour. Assuming that a product can have only one price at any one time then it is *single-valued*. If a person can have more than one telephone number and, assuming that this is of

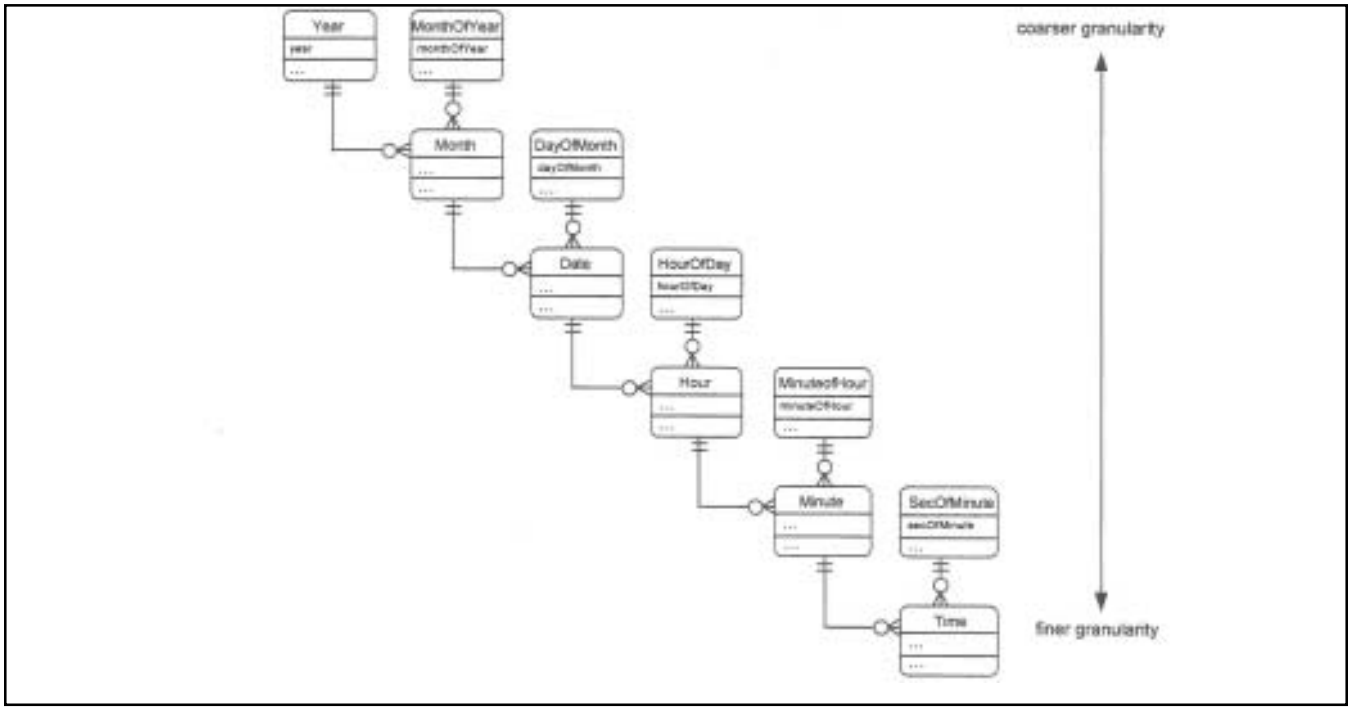

Figure 1: calendar classes (fragment)

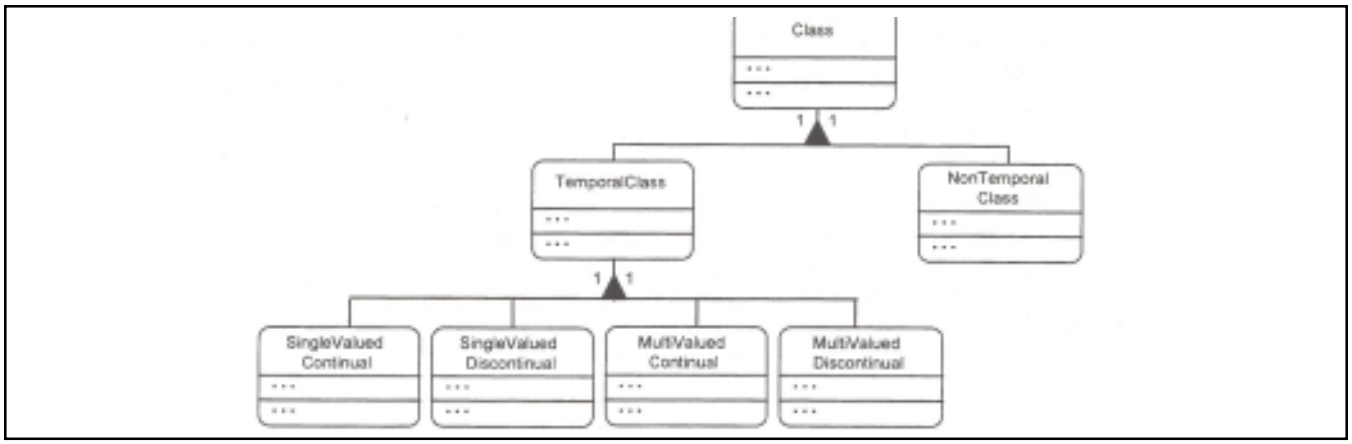

Figure 2: Temporal Classes in the Meta-model

interest to the modeller, then telephone number is *multi-valued.*

The combinations of continual and discontinual, and single-valued and multi-valued yield four patterns of temporal behaviour that will be used to model time-stamped attributes and associations. The four patterns are: single-valued/continual; single-valued/discontinual; multi-valued/continual; and multi-valued/discontinual. These patterns of behaviour are needed as part of the meta-model (figure 2).

Figure 2 contains a generalization/specialization structure in which the cardinality of the subclassifications is shown as 1..1, indicating, for example, that Class must be subclassified as TemporalClass or NonTemporalClass, but not both (a subclass group that is mandatory and mutually exclusive). By contrast, a subclass group cardinality of 0..n would indicate optional and independent subclasses. The temporal meta-classes in figure 2 is needed to ensure that time-stamped attributes and associations conform to the different patterns of temporal behaviour.

Diagramming conventions

In this section temporal class diagrams are introduced for mandatory and optional attributes, one to many associations, many to many associations, and recursive associations. In the interests of diagrammatic clarity the method section of the class boxes has been suppressed.

Mandatory attribute

Figure 3a shows a Person class, in which the mandatory attribute *address* has been marked for time-stamping with the following class and attribute specification:

Person
name
address (t)
started by: person changes address
ended by: derived
granularity: date
max. back-date: unlimited
max. forward-date: 90 days
type: single-valued/continual
max. frequency: daily
telephoneNo (o)
dateOfBirth

The temporal characteristics of the address attribute are shown in italics. Address is a mandatory attribute reflecting the requirement that "each person *must* reside at one address." This is modelled by a single-valued/continual pattern of temporal behaviour. The *started* by clause specifies the event(s) that can cause a history object to be created, in this case "person changes address." The *ended by* clause is derivable from the next started by time. The *granularity* is specified as date, *back-dating* is unlimited, and *future-dating* is limited to three months. The *frequency* is specified as daily, thus restricting address changes to one per day.

Although the user and analyst would communicate using a notation such as that in figure 3a, the designer will need to consider the implementation details as shown in figure 3b, to which a temporal class, AddressHistory has been added. This class will require that its instances behave according to the rules of a single-valued/continual pattern of temporal behaviour. Person has a minimum cardinality of 1 with respect to AddressHistory since the attribute address is

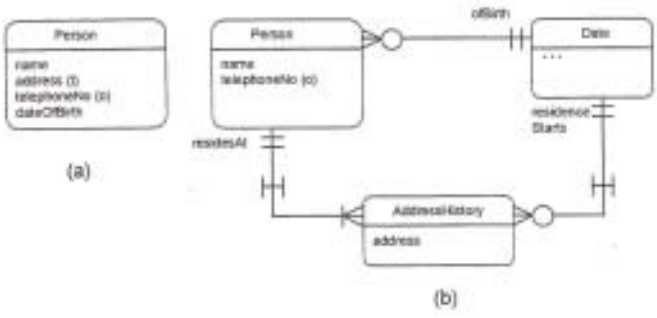

Figure 3: time-stamped mandatory attribute

mandatory for Person. The association of AddressHistory with Date is used to record the date the residence starts - the date the residence ends is not needed since this fact can be derived from the next start date. In figure 3b the attribute dateOfBirth has been replaced by an association between Date and Person. This is desirable since Date has now been recognized explicitly as a Calendar class.

An instance of the class AddressHistory is created whenever the event "person changes address" occurs. To avoid duplicate or inconsistent address histories, such as two instances of addressHistory that record the fact that "John Smith resides at 1 Beech Grove on 1 January 1996" and "John Smith resides at 10 Ash Drive on 1 January 1996" an "I" symbol is added to each of the associations that AddressHistory has with Person and Date to prevent further instances of AddressHistory being associated with the same instance-pairing of Person and Date.

Optional attribute

The attribute telephoneNo is optional and has a single-valued/discontinual pattern of temporal behaviour, reflecting the business requirement that, at any one time, "each person may have one telephone number." Assume that the Person class with temporal characteristics defined is now:

Person
name
address
telephoneNo (o) (t)

started by:	person is contactable on telephone number
ended by:	person ceases to be contactable on telephone number
granularity:	time
max. back-date:	unlimited
max. forward-date:	unlimited
type:	single-valued/discontinual
max. frequency:	daily

dateOfBirth

When modelling this situation in detail (figure 4b) an association is needed with Date to record when the telephone number is no longer valid since this fact cannot necessarily be derived, as reflected by the minimum cardinality of zero for Person with respect to TelephoneNoHistory. The model shows also that the contactForEnds date is optional as this date will not typically be known at the time a new telephone number is recorded.

Mandatory one to many association

Associations are marked for time-stamping by the addition of a box on the association line (figure 5a) (this is similar to the time-stamping notation used in TEMPORA as described by Theodoulidis et al., 1991). The mandatory one to many association has two components: the fact that each motor car must have one owner requires single-valued/continual behaviour; and the fact that each person may own many motor cars requires multi-valued/discontinual behaviour:

Person owns MotorCar ownedBy Person (t)

started by:	car ownership changes
ended by:	derived
granularity:	time
max. back-date:	30 days
max. forward-date:	0 days

Person owns MotorCar

type:	multi-valued/discontinual
max. frequency:	unlimited

MotorCar ownedBy Person

type:	single-valued/continual
max. frequency:	daily

Each time that the event "car ownership changes" occurs its effect will be captured in the OwnershipHistory class (figure 5b). As with the mandatory attribute, no end date is needed since this is derivable from the start date. By specifying the granularity as time and the frequency of MotorCar is ownedBy Person as daily the time of sale can be captured accurately while requiring that the period of ownership span at

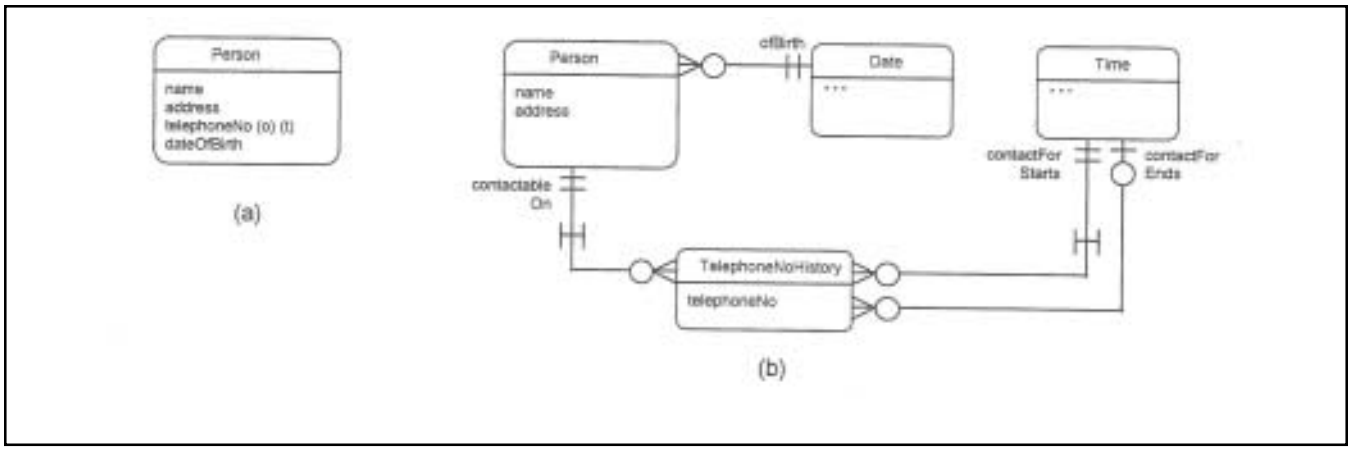

Figure 4: time-stamped optional attribute

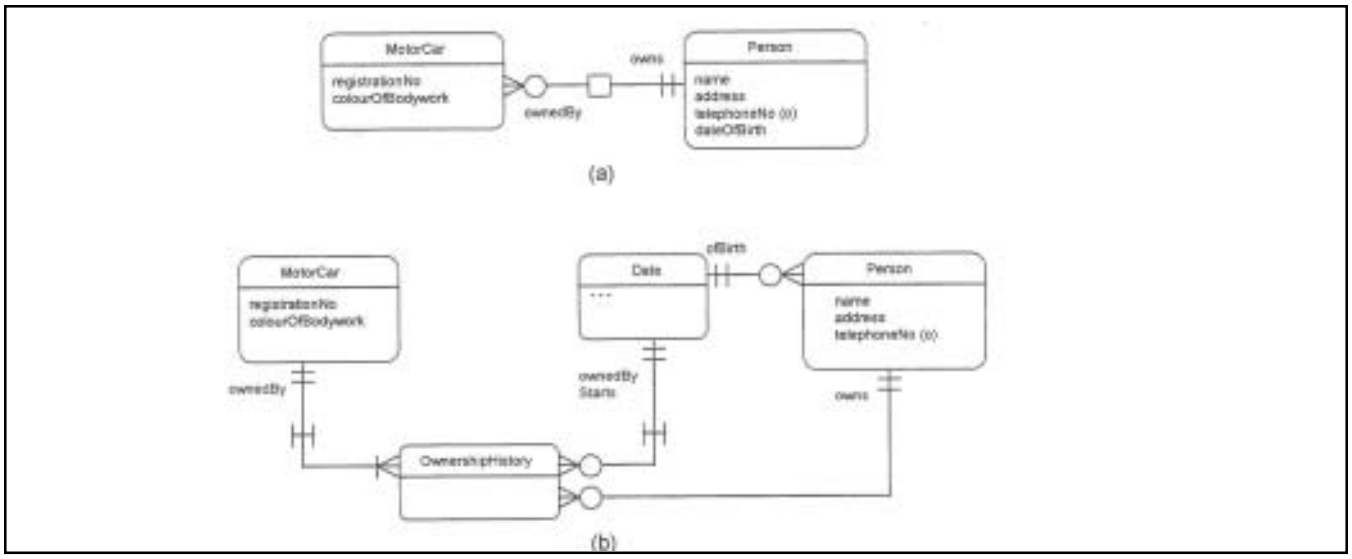

Figure 5: time-stamped mandatory one to many association

least two days. A forward dating of zero requires that all changes to ownership be either contemporaneous with the change of ownership occurring, or be back-dated.

Optional one to many association

In this case the association between car and owner is optional since not all cars will have an owner at all times. With optional associations a second event needs to be captured - "car ownership ceases" as it is no longer possible to deduce when the ownership ceased. The association specification is:

Person owns MotorCar
type: multi-valued/discontinual
max. frequency: unlimited
MotorCar ownedBy Person
type: single-valued/discontinual
max. frequency: daily

As with the optional attribute, an additional association is needed to record when the ownership ends. It is worth noting that time-stamped attributes can be promoted to time-stamped associations. For example, the time-stamped attribute telephoneNo shown in section 4.2 could be made into an independent entity and then remodelled as in figure 6b. However, this is not a temporal issue - it is a reflection of the fact that telephone numbers might be a thing of interest in their own right.

Many to many association

The expanded time-stamped many to many association (figure 7b) looks remarkably similar to the optional one to many model (figure 6b). However, 6a and 7a have different cardinalities, as reflected by the additional "I" symbol on the association between Person and OwnershipHistory in figure 7b. The association specification is:

Person owns MotorCar
type: multi-valued/discontinual
max. frequency: unlimited
MotorCar ownedBy Person
type: multi-valued/discontinual
max. frequency: daily

To make the ownership of cars mandatory it is necessary to change the minimum cardinality of MotorCar with respect to OwnershipHistory from zero to one. To make the many to many association fully mandatory then in addition change the minimum cardinality of Person with respect to OwnershipHistory from zero to one. However, this will not be sufficient to enforce the required temporal behaviour, since a person is required to have only one ownership history entry and therefore ownership of a motor car could cease without a new owner being appointed. The OwnershipHistory class should therefore inherit the appropriate behaviour from the temporal classes shown in figure 2.

Recursive associations

Recursive associations are a powerful way of meeting the requirements of management information systems, particularly in the case of the bill of materials and the hierarchy (figure 8a).

The association in figure 8a is fully optional since some organization units cannot have juniors (the leaf nodes) and one organization unit cannot have a senior (the root node). If it is assumed that all organization units with the exception of the root, must report to one senior then the date when an organization unit stopped reporting to a particular senior can be deduced. If it is possible for organization units to exist, temporarily at least, without a senior then it will be necessary to introduce an end date. This situation can be demonstrated

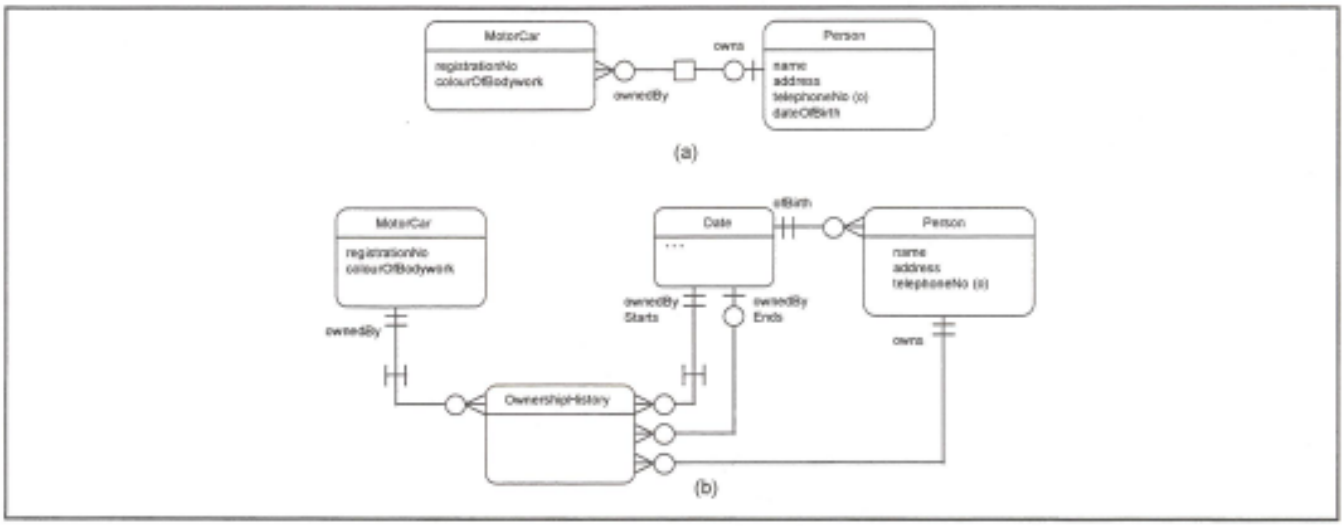

Figure 6: time-stamped optional one to many association

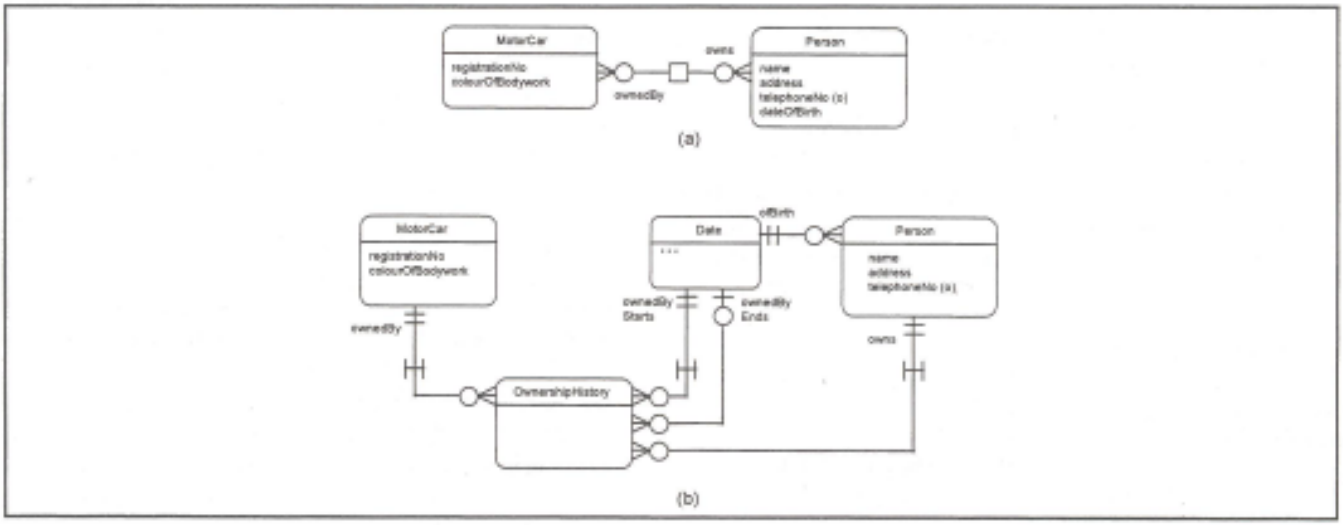

Figure 7: time-stamped optional many to many association

more clearly by the introduction of a subclass of OrganizationUnit as in figure 8b. The association between OrganizationUnit and OrganizationUnitWithSenior is then treated in exactly the same way as the non-recursive mandatory association in figure 5b (figure 8c). Different cardinalities of recursion, the many to many (bill of materials) and the one to one (chain) can be resolved similarly.

Temporal modelling and the management of complexity

One of the appeals of adopting an Object-Oriented paradigm is that it could provide a way of coping with the complexity of a data model that has temporal capabilities. It is common practice to assume that a method exists to return the contents of each attribute of a class, this method having the same name as the attribute. Thus to find out a person's address involves invoking the default method address, supported by the class Person. With respect to time-stamped attributes we assume that the method address will have the following defaults:

address	returns current address
address (timepoint)	returns address as of timepoint¨
address (timepoint1, timepoint2)	returns a set of addresses for the period starting timepoint1 and ending timepoint2

This can be shown diagrammatically by adding methods to the class box (figure 9).

In the case of the many to many association between MotorCar and Person, where is the method to sit? It seems to belong to neither MotorCar nor to Person, but to sit in

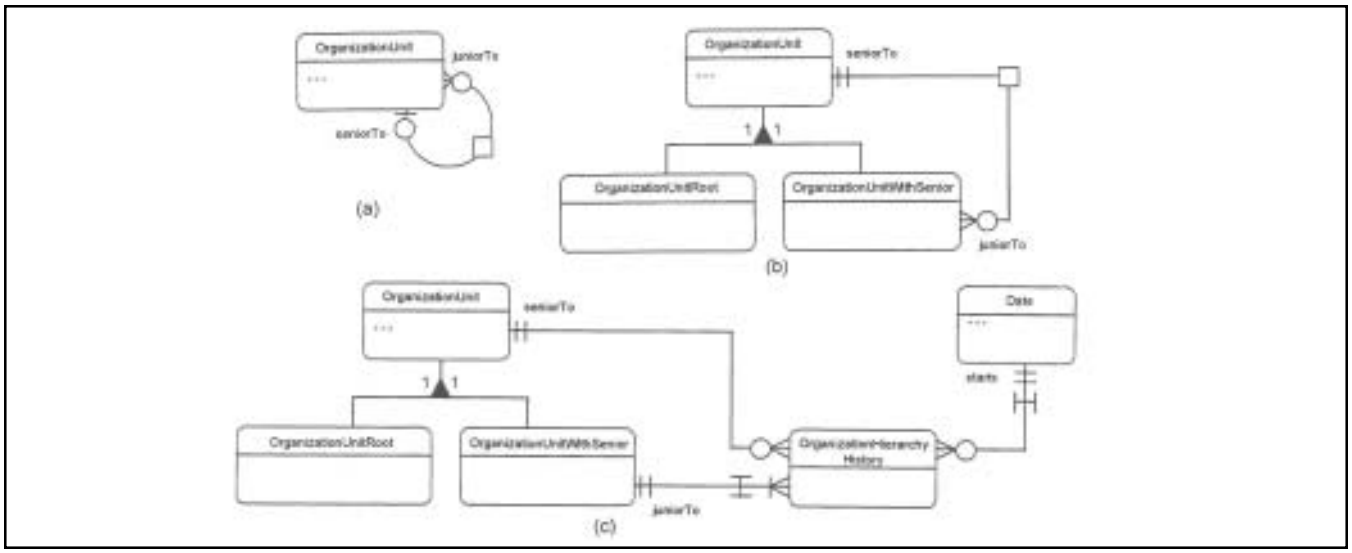

Figure 8: time-stamped recursive one to many association

between them.

Conceptually this can be thought of as a method owns implemented by the association Person owns MotorCar and a method ownedBy implemented by MotorCar is ownedBy Person. In an O-O implementation this might involve the establishment of a class OwnershipHistory to support the methods owns and ownedBy, which would behave with the same defaults as in the example of the address attribute above and, possibly, only be accessible by invoking methods attached to Person or MotorCar.

The complexity of the data structures needed to implement the time-stamping is hidden through encapsulation; the default should be to return a value for the current time, thus involving no additional complexity of message passing where temporal behaviour is not required for a particular operation.

Relational implementation of a temporal data structure

Some of the work of maintaining valid history can be accomplished through the imposition of representational uniqueness constraints. Consider the time-stamped optional one to many association and the time-stamped optional many to many association. These can be seen to have the same shape

Figure 9: Methods

(figure 6b, 7b), with a subtle difference in association constraints (the "I" symbol). The relational model requires that a unique identifier be declared explicitly for each table (the primary key) and, therefore, in a relational implementation unique identifiers will be required for MotorCar, Date, and Person.

Assuming that registrationNo can be used to identify uniquely each occurrence of MotorCar and that personNo identifies persons uniquely, then a relational design (table 1) can be developed for the optional one to many association (figure 6b) and the many to many association (figure 7b), paying particular attention to primary keys so as to stop records being added in a relational implementation that would violate the requirements of the different patterns of temporal behaviour. In table 1 primary keys are underlined and foreign keys shown explicitly, together with an associated Foreign clause (see de Carteret & Vidgen, 1995 for a full description of this notation). Relational implementation will, of course, need to be supported by program code that enforces the different temporal behaviour patterns shown in figure 2.

Temporal modelling of reclassifications (state changes)

Temporal modelling of reclassifications can also be achieved, but is more involved than attributes and associations since access to a meta-model is needed. Figure 10 shows a time-stamped generalization/specialization structure. The addition of a white square to the black generalization/specialization triangle indicates that a history of reclassifications is required. The reclassification history can be modelled as in figure 11, in which the temporal class ReclassificationHistory records the different subclassifications of Employee that each employee takes. For example, Jones might join on 01-Jan-1992 as an hourly employee, be

one to many (optional)	many to many (mandatory and optional)
MotorCar registrationNo colourOfBodywork Person personNo address (o) telephoneNo dateOfBirth Foreign: dateOfBirth∅Date OwnershipHistory registrationNo dateOwnershipStart personNoOwns (o) dateOwnershipEnd Foreign: registrationNo∅ MotorCar Foreign: dateOwnershipStart∅Date Foreign: dateOwnershipEnd∅Date Foreign: personNoOwns∅ Person	MotorCar registrationNo colourOfBodywork Person personNo address (o) telephoneNo dateOfBirth Foreign: dateOfBirth∅ Date OwnershipHistory registrationNo dateOwnershipStart personNoOwns (o) dateOwnershipEnd Foreign: registrationNo∅MotorCar Foreign: dateOwnershipStart∅ Date Foreign: dateOwnershipEnd∅Date Foreign: personNoOwns ∅Person

Table 1: Logical relational design for time-stamped 1:n and m:n associations

reclassified on 01-Jul-1992 as a contract employee, on 01-Jan-1993 as a salaried employee, and leave the organization on 15-Jan-1993. The meta-class Class contains as instances all the valid business classes, such as Employee and HourlyEmployee, and would need to be available to the business model. Although not covered in this paper, different patterns of temporal behaviour can be defined for the different cardinalities of subclass groups in the same way that they were for attributes and associations in section 4 above.

If the ReclassificationHistory class can be specified with temporal characteristics that would allow future-dating then it would be possible to specify future reclassification dates, such as the end date for a contract, that would trigger a reclassification at the appropriate time.

Reclassifications should conform to the valid state transitions defined for the class in question. Figure 12 shows a state transition diagram for Employee which could also need to be available when creating instances of the temporal class ReclassificationHistory.

Holding different states implies that two levels of deletion will be needed: the first relates to deletions that occur as part of the life-cycle, a virtual delete, and the second an absolute temporal delete that removes all trace of an instance. The virtual delete would allow the DBMS to report that an object does not exist at time *x*, although the object would show up in enquiries with appropriate effective dates (back-dated or future-dated). There are further issues concerned with aggregation and states. Rumbaugh et al. (1991) point out that aggregation is an "and-relationship", in which the aggregate state is one state from the first diagram, and a state from the second diagram, and a state from each other diagram (p. 99). Clearly this situation will be complicated by a requirement to maintain state change histories.

Summary

The need to record what happens to attributes, associations, and reclassifications over time is a common requirement in management information applications and in any application in which a temporal audit trail must be maintained. A

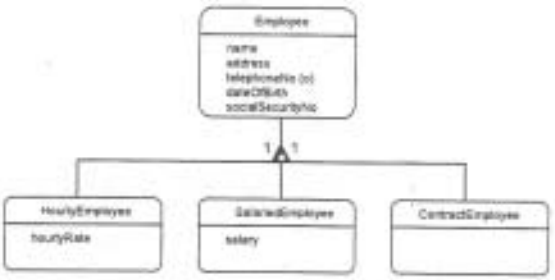

Figure 10: time-stamped subclassification

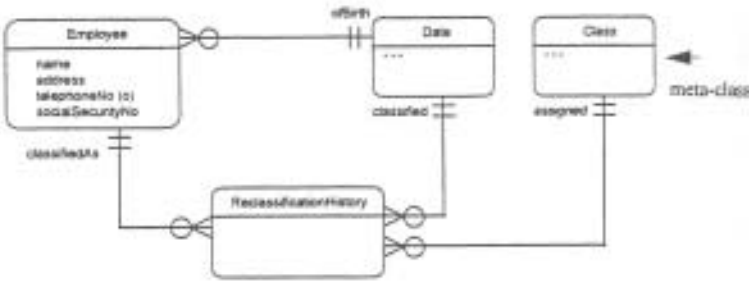

Figure 11: implementation of subclassification history

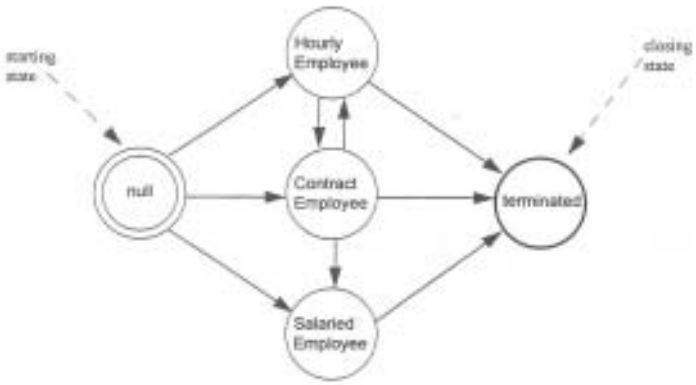

Figure 12: state transition diagram for Employee

Class diagrams that include notations for time-stamping and specification of temporal characteristics (a calendar) and temporal classes have been proposed. One benefit of the notation is that it is simple and should support communication between user and analyst/developer. A further benefit of invoking an O-O paradigm for temporal modelling is that encapsulation allows the complexity of the temporal data structures to be hidden and for temporal features to be ignored when not required.

A logical relational design has been included to demonstrate how a time-stamped object model can be approximated in a relational DBMS. Ideally database management systems would have temporal facilities allowing an easier migration from design to implementation, but, in the absence of a temporal DBMS, then CASE tools should be able to generate automatically structures similar to those presented in this paper from an object model. In modelling reclassifications, the need for meta-data to be available in the implementation environment was noted. Recommendations for practice arising from this paper are: firstly, systems analysis diagramming conventions be extended to include time-stamping and temporal specifications and secondly, commercial CASE tool support be developed for automatic generation of classes and relational schemas that support temporal data structures. Future research should consider the contribution that an O-O approach can make to the theory and the practice of temporal modelling, particularly in the areas of state change histories and aggregations.

Acknowledgements

The author thanks the anonymous referees for their constructive and helpful comments on an earlier version of this paper. Thanks are also due to Carrie de Carteret and Richard Veryard for their patient discussion, support and encouragement.

References

Bubenko, J., (1977). The temporal dimension in information modelling. In: Nijssen, G.M., editor. *Architecture and Models in DBMSs.* North-Holland, Amsterdam.

de Carteret, C., & Vidgen, R., (1995). *Data Modelling for Information Systems.* Pitman.

Clifford, J., & Warren, D.S., (1983). Formal Semantics for Time in Databases. *ACM Transactions on Database Systems,* 8(2): 214-254.

Coad P., & Yourdon, E., (1991). *Object-Oriented Analysis* (2nd edition). Yourdon Press, Prentice-Hall.

Dubois, E., Hagelstein, J., Lahou, E., et al., (1986). The ERAE Model: a case study. In: Olle, T., Sol, H., & Verrijn-Stuart, A., editors. *Information Systems Design Methodologies: Improving the Practice* (CRIS-3). North-Holland, Amsterdam.

Goodland, M., & Slater, C., (1995). *SSADM Version 4: a practical approach.* McGraw-Hill.

Klopprogge, M. R., (1983). TERM: an approach to include the time dimension in the E-R model. In P. Chen, editor. *Entity-Relationship Approach to Information Modelling and Analysis.* Elsevier Science/North Holland.

Langefors, B., (1973). *Theoretical Analysis of Information Systems.* Student Literature and Auerbach, Lund, Sweden.

Langefors, B., & Sundgren, B., (1975). *Information Systems Architecture.* Petrocelli/Charter, New York.

Martin, J., & Odell, J., (1992). Object-Oriented Analysis and Design. Prentice-Hall. Englewood Cliffs.

McKenzie, E., (1986). Bibliography: temporal databases. *SIGMOD 15*(4):40 - 52 Robinson, K., & Berrisford G., (1994). Object-Oriented SSADM. Prentice-Hall International.

Roddick, J., & Patrick, J., (1992). Temporal Semantics in Information Systems - a survey. *Information Systems,* 17(3): 249-267.

Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorensen, W., (1991). Object-Oriented Modeling and Design. Prentice-Hall, Englewood Cliffs, New Jersey.

Snodgrass, R., (1987). The Temporal Query Language TQuel. *ACM Transactions on Database Systems*, 12 (2): 247-298

Snodgrass, R., (1995). Temporal Object-Oriented Databases: a critical comparison. In: Kim, W., editor, Modern *Database Systems: the object model, interoperability, and beyond.* The ACM Press, New York.

Theodoulidis, B., Wangler, B., & Loucopoulos, P., (1990). *Requirements specification in TEMPORA.* Second Nordic Conference on Advanced Information Systems Engineering (CAiSE-90). Kista, Sweden.

Theodoulidis, B., Loucopoulos, P., & Wangler, B., (1991). A Conceptual Modelling Formalism for Temporal Database Applications. Information Systems, 16(4): 401-416.

Theodoulidis, C., & Loucopoulos, P., (1991). The Time Dimension in Conceptual Modelling. *Information Systems,* 16(3): 273 - 300.

University of Arizona, (1994). *Towards an Infrastructure for Temporal Databases.* Report of an invitational ARPA/NSF workshop (TR 94-01).

Richard Vidgen obtained a first degree in Computer Science & Accounting and a MSc in Accounting , both of which were awarded by the University of Manchester. He then developed and supported financial applications software for MSA Inc. (now Dunn & Bradstreet Software), followed by a number of years working as a freelance consultant, engaged in the design and implementation of IT solutions for the banking and insurance industries. In 1992 he was appointed to a lectureship at the University of Salford, where he completed a Ph.D. in the area of information systems quality. He is currently a lecturer in the Department of Computation at UMIST researching data and object modelling requirements engineering and IS quality.