Temporal Entity-Relationship Models—a Survey

Heidi Gregersen and Christian S. Jensen

TR-3

A TimeCenter Technical Report

Title	Temporal Entity-Relationship Models—a Survey
Author(s)	Heidi Gregersen and Christian S. Jensen
Publication History	September 1996. Technical Report R-96-2039, Aalborg University. January 1997. A TIMECENTER Technical Report.

TIMECENTER Participants

Aalborg University, Denmark
Michael H. Böhlen
Renato Busatto
Heidi Gregersen
Christian S. Jensen (codirector)
Kristian Torp

University of Arizona, USA
Hong Lin
Richard T. Snodgrass (codirector)

Individual participants
Curtis E. Dyreson, James Cook University, Australia
Michael D. Soo, University of South Florida, USA
Andreas Steiner, ETH Zurich, Switzerland

The TIMECENTER icon on the cover combines two "arrows." These "arrows" are letters in the so-called *Rune* alphabet used one millennium ago by the Vikings, as well as by their precedessors and successors, The Rune alphabet (second phase) has 16 letters. They all have angular shapes and lack horizontal lines because the primary storage medium was wood. However, runes may also be found on jewelry, tools, and weapons. Runes were perceived by many as having magic, hidden powers.

The two Rune arrows in the icon denote "T" and "C," respectively.

Abstract

The Entity-Relationship (ER) Model, using varying notations and with some semantic variations, is enjoying a remarkable, and increasing, popularity in both the research community, the computer science curriculum, and in industry. In step with the increasing diffusion of relational platforms, ER modeling is growing in popularity. It has been widely recognized that temporal aspects of database schemas are prevalent and difficult to model using the ER model. As a result, how to enable the ER model to properly capture time-varying information has for a decade and a half been an active area of the database research community. This has led to the proposal of almost a dozen temporally enhanced ER models.

This paper surveys all temporally enhanced ER models known to the authors. It is the first paper to provide a comprehensive overview of temporal ER modeling, and it thus meets a need for consolidating and providing easy access to the research in temporal ER modeling. In the presentation of each model, the paper examines how the time-varying information is captured in the model and present the new concepts and modeling constructs of the model. A total of 20 different design properties for temporally enhanced ER models are defined, and each model is characterized according the these properties[1].

***Keywords*— Conceptual modeling, entity-relationship models, database design, temporal databases, temporal data models, design criteria for temporal ER models, time semantics.**

1 Introduction

The Entity-Relationship (ER) Model [2], in its different versions, with varying syntax and with some semantic variations, is enjoying a remarkable, and increasing, popularity in both the research community and in industry. The model is easy to comprehend and use. An ER diagram provides a good overview of database design, and the model's focus on the structural aspects of database schemas, as opposed to their behavioral aspects, also appears to match the levels of ambition for documentation adopted by many users.

The ER model may be used for different but related purposes, namely for analysis—i.e., for modeling a mini-world—and for design—i.e., for describing the database schema of the a computer system. As a third alternative, the ER model may be supported directly by a DBMS. In that case, it may be used as an implementation model. However, although graphical and textual ER query languages have been proposed by the research community, the ER model is rarely used as an implementation model. Rather, the typical use seems to be one where the model is used primarily for design, with the design diagrams also serving as analysis diagrams, and where the constructed diagrams are mapped to a relational platform. In step with the increasing diffusion of relational platforms in industry, ER modeling is growing in popularity.

The use of ER modeling is supported by a wealth of textbook material. For example, most introductory database textbooks (e.g., [10, 25, 3]) contain chapters on ER modeling, and several complete books exist (e.g., [1, 32]) that are devoted entirely to ER modeling.

Companies either develop their own ER diagrams from scratch, or they purchase and modify generic, standard diagrams[2]. Indeed, generic diagrams for a variety of types of applications are commercially available, e.g., the FSDM from IBM.

Some companies build ER diagrams using only simple drawing tools. Other companies use one of the many commercially available tools that are more sophisticated and better support the building of diagrams and also map diagrams to implementation platforms. Such tools are either stand-alone, e.g., SmartER from Knowledge Based Systems, Inc. and ER/1 from Embarcadero Technologies, or are integrated parts of larger CASE tools, e.g., Teamwork/IM SQL from Cayenne Software, Inc. and Visible Analyst Workbench from Visible Systems Corporation. Typical implementation platforms include those provided by major SQL-based database systems.

[1]The authors are with the Department of Mathematics and Computer Science, Aalborg University, Fr. Bajers Vej 7E, Dk–9220 Aalborg Øst, Denmark, {gregori,csj}@cs.auc.dk. This work is supported by grants 9400911 and 9502695 from the Danish Natural Science Research Council and the Danish Technical Research Council, respectively.

[2]In industry, ER diagrams are typically termed ER models. This is in contrast to common usage in the research community, and in this paper, where a data model is a modeling notation, and a diagram is a description using some notation.

In the research community as well as in industry, it has been recognized that temporal aspects of database schemas are both prominent and difficult to capture using the ER model. Put simply, when modeling fully the temporal aspects, the temporal aspects tend to obscure and clutter otherwise intuitive and easy-to-comprehend diagrams. As a result, some industrial users simply choose to ignore all temporal aspects in their ER diagrams and supplement the diagrams with phrases such as "full temporal support." The result is that the mapping of ER diagrams to relational tables must be performed by hand; and the ER diagrams do not document well the temporally extended relational database schemas used by the application programmers.

The research community's response has been to develop temporally enhanced ER models. Indeed, almost a dozen such models have been reported in the research literature. Their informative names include the Temporal Enhanced Entity Relationship model [12, 11], the Temporal Entity Relationship model [31], and the Relationship, Attribute, Keys, and Entities model [13], to name but a few.

Two general, orthogonal temporal aspects have received widespread attention, namely *valid time* and *transaction time* [15]. The valid time of a database fact is the time when the fact is true in the mini-world. (We use the term "mini-world" for the part of reality that the database under consideration stores information about.) Thus, all database facts have an associated valid time. Different time types may be used when modeling the valid-time aspect, e.g., single time instants, intervals, or sets of intervals.

Perhaps more importantly, the valid time may or may not be captured explicitly in the database—this is the choice of the database designer. In ER models, unlike in the relational model, a database is not structured as a collection of facts, but rather as a set of entities and relationships with attributes. Thus, the valid times are associated only indirectly with facts. As an example, consider an Employee entity "E1" with a Department attribute. A valid time of June 1996 associated with value "Shipping" does not say that "Shipping" is valid during June 1996, but rather that the fact "E1 is in Shipping" is valid during June 1996. Thus, when valid time is *captured* for an attribute such as Department, the database will record the varying Department values for the Employee entities. If it is not captured, the database will record only one department value for each Employee entity.

Orthogonal to valid time, the transaction time of a database fact is the time when the fact is current in the database and may be retrieved. Unlike valid time, transaction time may be associated with any structure stored in a database, not only with facts. Still, all structures stored in a database have a transaction-time aspect. And again, this aspect may or may not, at the designers discretion, be captured in the database. The transaction-time aspect has a duration: from insertion to (logical) deletion.

In addition to valid and transaction time, a data model may support arbitrary time attributes with no built-in semantics in the data model. For employee entities, such attributes could record birth dates, hiring dates, etc. A data model that supports such time attributes is said to support *user-defined* time.

In summary, facts stored in a database have a valid time and a transaction time, although those times may not be explicitly recorded [15]. We say that a data model *supports* a temporal aspect, i.e., valid or transaction time, if it provides built-in means for indicating where in an ER diagram this aspect should be captured.

The temporal ER models attempt to more naturally and elegantly model the temporal aspects, such as valid and transaction time, of information by changing the semantics of the ER model or by adding new constructs to the model. The models take quite different approaches to adding built-in temporal support to the ER model.

This paper is the first to survey all known (to the authors!) temporal ER models. In addition, the paper provides a comprehensive list of possible properties of temporal ER models, and it characterizes the models according to those properties. With about a dozen models having been proposed over the past 15 years, such a survey is in order. It consolidates in a single and easy-to-access source the central ideas, concepts, and insights achieved in temporal ER modeling. The survey makes it easier for future research and development to maximally build on, benefit from, and extend past results. Thus, the survey is aimed at researchers and practitioners interested in temporal data modeling and data model design.

Four studies are somewhat related to or complement the study reported here.

Theodoulidis and Loucopoulos [34] describe and compare nine approaches to specify and use time in conceptual modeling, here viewed as both semantic data modeling and requirement specification, of information systems. Their study includes only two of the ER models surveyed here. The comparison of the models fall in three parts and classifies the models in terms of time semantics, model semantics, and temporal functionalities. Our criteria also characterize the models in terms of user-friendliness. The primary focus of their paper is the examination of the ontology and properties of time in the context of information systems, whereas our focus is the examination of how the extensions of the ER model into temporal ER models are shaped.

McKenzie and Snodgrass [22] survey and evaluate twelve different temporal extensions of the relational algebra. They evaluate the algebras against 26 design criteria. These criteria are mainly concerned with the properties of the data objects—temporal relations and their components—that the algebras manipulate and with the properties of the algebraic operators themselves. While their survey concerns internal algebras, our survey concerns notations for conceptual modeling. In addition, our focus is on the properties of the structural aspects of the temporal ER models.

Without coauthors, Snodgrass has also conducted a critical comparison of temporal object-oriented data models [28]. While ER models do incorporate some structural object-oriented features, our study does not consider object-oriented models; for that, we instead refer the reader to Snodgrass' study. Also unlike our study that emphasizes structural aspects, Snodgrass' study focused on the models' query languages, i.e., on behavioral aspects.

Roddick and Patrick [24] survey the progress of incorporating time in various data models at the conceptual and, primarily, logical level of database modeling, and in artificial intelligence. The work describes nine different properties of temporal modeling systems, but unlike our survey they do not evaluate the models against the properties described. Their broad study briefly covers two of the temporal ER models in our study.

The descriptions in the literature of the different models use diverse and, at times, incompatible and conflicting terminology. In this survey, we adopt the coherent terminology of the temporal database glossary [15] when possible. In addition, the original definitions of the models are often informal and rely on the reader's intuition. In part also to achieve a homogeneous survey of a manageable size, we will give informal descriptions of some aspects. Further, we will emphasize the common core of features of the temporal ER models: the use of ER modeling to capture the structural aspects of a database schema. We will not cover behavioral aspects such as query and rule languages in detail.

The paper is structured as follows. Section 2 provides an overview of all temporal ER models known to the authors. Section 3 then identifies a set of 20 evaluation criteria and evaluates each model according to these criteria. Finally in Section 4, a conclusion and a discussion of future work is given.

2 Existing Models

This section describes each existing ER model separately and in turn. Initially, an overview is provided that explains the structuring of the descriptions and introduces a ("running") example that will be used for exemplification throughout.

2.1 Overview

This section describes all the temporal ER models that we are aware of. We will assume that the reader is familiar with Chen's standard ER model [2] and the various extensions of that model, e.g., subtyping (see, e.g., [10]).

We have chosen to present the models in chronological order of their first publication, with the exception that the model RAKE is presented before the model TERM because RAKE has a graphical notation while TERM does not.

The description of the models all have, with a few exceptions, the same basic layout. First, a short introduction of the model is given. Second, we describe how the model captures time. Third, we give an example diagram built using the model's notation. For each model, the sample diagram models the same mini-world, to be described shortly. In order to keep the diagrams simple and still be able to reach into the corners of the different models, we in some places deviate slightly from the description

below. This way, it is possible to more concisely present the special features of the models. When we do deviate, we will state this explicitly. Last, a short summary of the model is given.

Mappings of ER diagrams to implementation platforms for the models will only be explained if they are described in the papers and differ substantially from the typical mappings from the EER model to relational platforms.

The mini-world that we describe next concerns a company divided into different departments. Each department has a number and a name and is in charge of a number of projects. A department keeps track of the profits it makes on its projects. Because the company would like to be able to make statistics on its profits, each department must record the history of its profits.

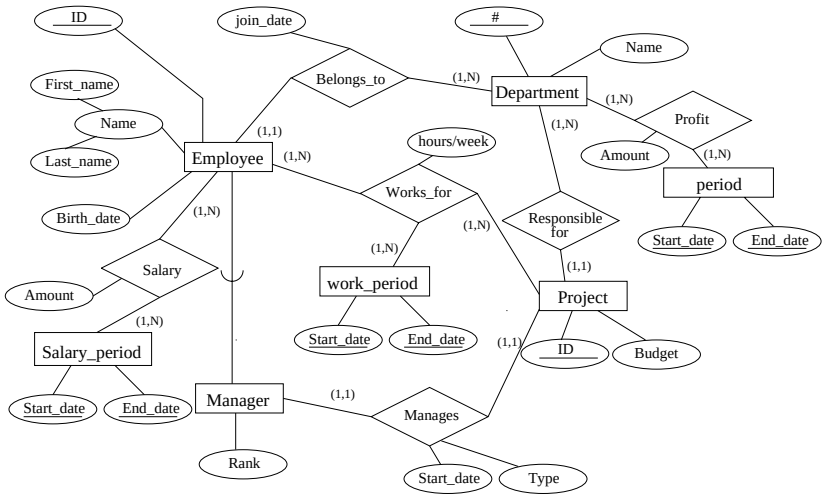

Figure 1: ER Diagram Describing the Running Example

Each project has a manager and some employees working on the project. Each project has an ID, and a budget. Each project is associated with a department which is responsible for the project. Employees belong to a single department. Once an employee is assigned to a department, the employee works for this department for as long as the employee is with the company. For each employee, the company registers the ID, the name, the date of birth, and the salary. The departments would like to keep records of the different employees' salary histories.

Employees work on one project at the time, but employees may be reassigned to other projects, e.g., due to the fact that a project may require employees with special skills. Therefore, it is important to keep track of who works for what project at a given time and what time they are supposed to finish working on their current project.

Some of the employees are project managers. Once a manager is assigned to a project, the manager will manage the project until it is completed or otherwise terminated. Figure 1 presents the ER diagram describing the database design corresponding to this mini-world.

Figure 2 provides an overview of the surveyed models, along with their main citations, the models on which they are based, and the identifiers we will be using in the rest of the paper.

It is important that the presentation (and definition!) of a model is precise and complete. The descriptions of the surveyed models range from very formal and detailed to vague and faulty.

The model we have found to be described the best is TERM, which in [17] is described in great detail. Models MOTAR, ERT, and TER are presented in articles dedicated to this single purpose, but their descriptions are not as detailed and comprehensive as that of TERM. Models RAKE, TEER, and STEER are also presented in articles only concerning the presentation of the models, but their descriptions are less comprehensive. The description of TempEER is further incomplete. For example, the description of the mapping algorithm supposed to translate TEER diagrams to relational schemas does not cover time-varying aspects. The description of TempRT is also incomplete, primarily because this model is not yet fully developed.

Name	Main citations	Based on model	Identifier
Enhanced Entity Relationship model	[10]	ER	EER
Relationships, Attributes, Keys, and Entities Model	[13]	ER	RAKE
Temporal Entity-relationship Model	[17, 18]	ER	TERM
Model for Objects with Temporal Attributes and Relationships	[23]	ER & OO	MOTAR
Temporal EER model	[12, 11]	EER	TEER
Semantic Temporal EER model	[8, 9]	ER	STEER
Entity-Relation-Time model	[33, 35, 21]	ER	ERT
Temporal ER model	[31]	ER	TER
Temporal EER model	[20]	EER	TempEER
Kraft's Model	[19]	ER	TempRT

Figure 2: Short Presentation of the Surveyed Models

2.2 The Relationships, Attributes, Keys, and Entities Model

The Relationships, Attributes, Keys, and Entities model, RAKE, [13] was developed in 1984 as part of a project at the U. S. Federal Reserve Board. One of the tasks in the project was to design a database to "store data on the history, attributes, and interrelationships of American and foreign financial institutions," and the model was developed to provide better support for this work than the ER model.

RAKE fundamentally adopts the ER model, but replaces some of the ER model's modeling constructs with new ones and adds entirely new constructs. Most prominently, RAKE introduces so-called key fields in diagrams: Key attributes of entity types are places in "key boxes" in the upper-left corners of the entity-type rectangles. This explicit representation of the entity-keys was unexpectedly found to also be useful when modeling time-varying data, to record multiple states of entities and relationships in the same application.

All new constructs are defined in terms of their mapping to relational tables and in terms of existing ER constructs. Following a discussion of the representation of the time domain in RAKE, we consider in turn the modeling of time-varying relationships and attributes.

2.2.1 The Representation of Time

The time type used in RAKE corresponds to the type `DATE` (or `TIMESTAMP`) supported by, e.g., various SQL implementations of relational DBMSs. This type is used for modeling of valid time and user-defined time, but it could also be used to capture transaction time.

It is noted that the history of entities and relationships consists of series of states succeeding one another in time. The series are punctuated by events that transform one state into another. The states have duration while the events do not. The valid times of states are thus modeled using a pair of time attributes, BEGINstamp and ENDstamp, and the valid times of events are modeled using an attribute Tstamp. Next, we shall see how these time attributes are used in RAKE diagrams.

2.2.2 The Model Components

As usual, entity types are represented by rectangles. The primary key of an entity type is placed, in a so-called keybox, in the upper-left corner of its rectangle. Weak entity types are also represented by rectangles. For these, the partial key is placed in the keybox, and the primary keys of the identifying relationships are stacked ontop of the keybox.

Non-primary-key attributes of entity types are represented by circles that, as usual, are linked to the entity types. If an attribute circle is enclosed by a square (also a rectangle), this means that the attribute may be treated as an entity type. As in the ER model, relationship types are represented by diamonds. As for attributes, if a relationship-type diamond is enclosed by a rectangle, this implies that the relationship type may also be treated as an entity type.

In non-temporal databases, only the current, or last-known, state of entities and relationships are stored. When recording multiple states, entities and relationships are identified differently. Entities are identified by non-reusable identifiers (e.g., serial numbers). In contrast, RAKE distinguishes between different relationships—that are instances of the same relationship type—solely by their timestamps. Below, we delve into these and other temporal aspects.

Modeling Time-varying Relationships

When changing a binary relationship type where only a single state is recorded, to record multiple states, the relationship type turns ternary. To see this, consider Figure 1. The Responsible_for relationship type consists of a set of pairs of Department and Project entities, with the entities being represented by their primary-key values. In contrast, because we want to record project assignments for different times, it is necessary for Works_for to be ternary: only with a third work_period entity is it possible to represent project assignments of the same employee to the same project, at different times.

Thus, the ternary relationship type in Figure 3(a) is the correct way to represent a temporal relationship between two entities in RAKE. To avoid cluttering the diagrams with time-period rectangles,

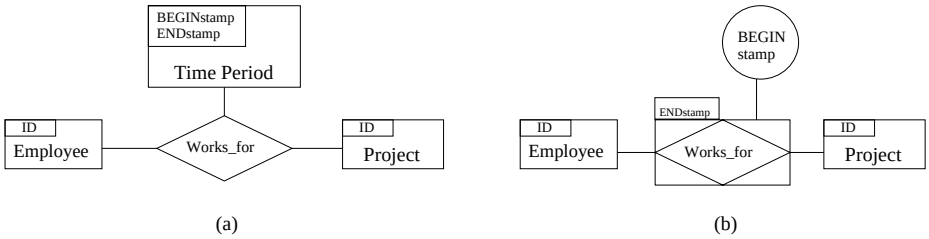

Figure 3: The Representation of Time-varying Relationship Types in RAKE

RAKE eliminates this notation and instead introduces the semantically equivalent notation in Figure 3(b). In this way, RAKE represents temporal relationship types as weak entity types owned by a time-period entity type that is not explicitly represented in the diagrams [13, pp. 282–283]. Together with the primary keys of the other entity types participating in the relationship type, the ENDstamp, which is part of the key of the owner entity type, is sufficient to uniquely identify instances of the relationship type. The BEGINstamp, also a part of the owner entity type, is therefore simply treated as an ordinary attribute.

Modeling Time-varying Attributes

The use of a circle for representing an attribute may be seen as a shorthand for a relationship between a set of entities and a domain of attribute values. With this view, the domain of attribute values becomes an entity type, and the technique for modeling temporal relationship types may be used for modeling temporal attributes as well. Figure 4(a) illustrates this correspondence. When applying the

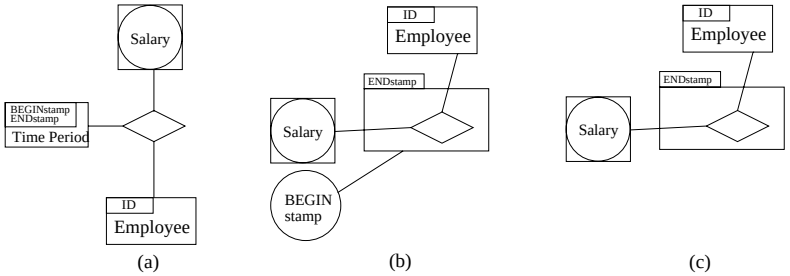

Figure 4: Modeling Time-varying Attributes in RAKE

transformation technique from relationship types, we arrive at Figure 4(b). Again, by having made the entity attribute relationship explicit, the relationship is treated as a weak entity with an implicit time period as owner. This, in turn, is abbreviated to Figure 4(c), where the BEGINstamp attribute is made implicit. This is how RAKE models temporal attributes.

Next, observe that the approach here is to use attribute-value timestamping. Each attribute is treated in isolation. RAKE also has special provisions for timestamping sets of attributes of an entity type. Assume that the Salary and Address of Employee are both temporal and that we want to timestamp them together. Figure 5 illustrates how this is accomplished. Figure 5(b) illustrates the

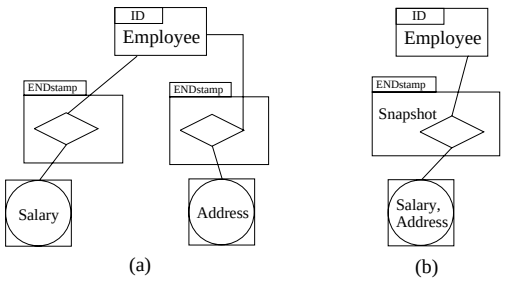

Figure 5: Modeling Time-varying Attributes Together in RAKE

new construct and Figure 5(a) shows the equivalent old construct. When mapping the two diagrams to a relational database schema, Figure 5(a) would be mapped to two relation schemas, while Figure 5(b) would only be mapped to a single table. These database schemas have different advantages. Indeed, this is the rationale for permitting both modeling constructs.

Finally, it is also possible to timestamp attributes and relationships with time points, to model temporal events. This is done simply be using an attribute Tstamp in place of ENDstamp and omitting BEGINstamp from temporal relationships.

2.2.3 Summary

RAKE retains most of the constructs of the ER model, with their usual semantics, but modifies the handling of primary keys by introducing special keyboxes on entity types and weak entity types. RAKE also introduces special constructs for modeling temporal relationship and attribute types. These are modelled as weak entity types owned by implicit time-period entity types. The new constructs of RAKE are defined in terms of their mapping to the relational model and of existing ER constructs.

2.3 The Temporal Entity-Relationship Model

TERM, the Temporal Entity-Relationship Model, was the first temporally extended ER model to be proposed [17, 18]. The main motivation for TERM was "to provide database designers with a model for data definition and data manipulation that allows a general and rigorous treatment of time" [17]. To accomplish this, TERM most notably introduces the notion of a history, which is a function from a time domain to some value domain. Histories are then used for the modeling of time-varying aspects. For example, the (time-varying) value of an attribute of an entity becomes a history, rather than a simple value.

Unlike all the other temporal ER models, TERM does not have a graphical syntax, but has a Pascal-like syntax.

2.3.1 The Representation of Time

In its outset, TERM makes a strict distinction between a real-world phenomenon and its ER-model representation. For example, TERM distinguishes between "time" and the representation of time—there is one "time," but many possible representations of time. This distinction extends to the other modeling constructs, e.g. values and histories. We focus on the representations.

Domains are termed *structures*. A time domain is thus a time structure. With TERM, the designer may define time structures, but TERM also includes a predefined time structure of Gregorian dates. These dates are equipped with a variety of predicates, termed structure relations, e.g., "before_date" and "is_in_leap" (is the argument date in a leap year?), and operators, e.g., "next-day" and "least-recent." Figure 6 illustrates two value structures, one for employee names and one for generic identifiers. It also provides a (partial) time structure, termed "date," with one relation.

```
Structure
   name= packed array [1..20] of char;

Structure
   st_ID = integer;

relations
   function is_in_leap( t: date): boolean;
   begin
      is_in_leap := t.y mod 4 = 0 and
                    (t.y mod 100 < > 0
                       or t.y mod 400 = 0);
   end
```

```
Structure
   date =
      record d,m,y: integer end
         where
            this.y   >= 1852   and
            this.m   >= 1 and this.m <= 12 and
            this.d   >= 1 and this.d <= 31 and
            (this.d < > 31
               or this.m in [1,3,5,7,8,10,12]) and
            (this.d < > 30 or this.m < > 2)     and
            ( (this.d < > 29or this.m < > 2)
               or (this.y mod 4 = 0 and
                   (this.y mod 100 < > 0
                      or this.y mod 400 = 0)));
```

Figure 6: Sample Value and Time Structures

A *history* is a mapping $h : T \rightarrow V$ where T is a time structure and V is a value structure. Histories are used for capturing the variability of time-varying aspects, as we shall see in the next section. Attributes of entities and roles of relationships have atomic histories while, e.g., entire entities have composite histories, i.e., histories composed of atomic and composite entities. All composite histories are sets of histories: An entity (relationship) history consists of an existence history and the set of all its attribute (role) histories.

The (atomic) history h is represented by the *history structure* $2^{T \times V}$, i.e., by a set of (time, value) pairs. To achieve a finite history structure in situations were time structure T represents continuous time, it is possible to introduce as part of the history structure a derivation function that uses the stored (time, value) pairs to compute values for additional times.

Figure 7 exemplifies histories. First, a generic existence history for entities and relationships is defined. The condition involving the three universally quantified variables "s1", "s2", and "s" disallows holes in existences. To the right, a salary history is defined that uses a step-wise constant derivation function, deriv_sal (least_recent_date, when applied to a pair of a set of dates and a date,"z" returns the largest date in the set that is not larger than "z").

```
Structure
   standard_existence=
      history
         t : date;
         v : kleenean end;
      where
         all s1, s2 from this where
         ((s1.v = true and s2.v = false and before_date(s1, s2)
         impl all s from this where
            before_date(s2, s) impl s.v = false)

         and
         ((s1.v = false and s2.v = true and before_date(s1, s2)
         impl all s from this where
            before_date(s, s1) impl s.v = false)

         and
         ((s1.v = true and s2.v =true and before_date(s1, s2)
         impl all s from this where
            before_date(s1, s) and before_date(s, s2)
               impl s.v = true);
```

```
Pattern
   sal_history=
      history
         t : date;
         v : real end;

   derivation
      function deriv_sal(h: sal_history;
                         z: date) : real;
      var x state of sal_history;
      begin
         x:= that s1 from h where
            s1.t=least_recent_date( those tx from date where
                                    exists s2 from h where
                                    s2.t=tx, z);
         if x <> nil
            deriv_sal:= x.v;
         else
            deriv_sal:= uncertain

      end;
```

Figure 7: TERM History Definitions [18]

All data items within a database will not change at the same time. Moreover, for some database items, only the current value is of interest, whereas for others, only some values in the past may be known, while still other items require a history of the entire past. For these reasons, histories are applied to individual database items instead of to the database as a whole.

2.3.2 The Model Components

The next step is to consider the association of histories with time-varying database items.

The basic modeling constructs of TERM are those of the ER model. *Entities* model the interesting objects from the mini-world; *values* model the properties of the mini-world objects. The values are associated with the entities via *attributes*.

If an attribute has no history, that is, if the value of an attribute never changes once it is assigned, it is referred to as a constant attribute; otherwise it is variable. Constant attributes are represented by a (attribute, value) pair, and variable attributes are represented by a (attribute, history) pair.

Entity types are declared by a name and a set of (constant and variable) attributes. The attribute named *existence* is mandatory and describes the existence of the entity type. If the existence attribute is specified as constant, the attribute has Boolean/Kleenean as its domain. This domain has values *false*, *true*, and *unknown*. A variable existence attribute has an associated Boolean/Kleenean-valued history.

Two or more entities can enter into a *relationship* in which each entity plays a *role*. Like attributes of entity types, roles of relationship types are represented by values, now entity references, or by histories, now entity-reference valued.

Relationship types are declared by a name, an existence description, a set of roles, and a set of attributes. Binary relations may be declared to express participation constraints such as 1:1, 1:N, and N:1, where the constraints are enforced for each database state in isolation. Writing a **one** after the role name restricts participation to at most one (at a time). By placing a **total** after a role name, total participation is indicated.

A TERM schema consists of a set of entity type definitions and a set of relationship type definitions. Figure 8 shows the two entity types, Project and Employee, and the relationship type, Works_for, between them.

```
Entity type
  Project
  existence
    variable
      standard_existence;
  attributes
    ID   constant st_ID;
    budget  constant  real;

Entity type
  Employee
  existence
    variable
      standard_existence;
  attributes
    ID   constant st_ID;
    First_name constant  name;
    Last_name constant  name;
    Salary  variable  sal_history;
    Birth_date   constant  date;

Relationship type
  Works_for
  existence
    constant
      kleenean;
  attributes
    no_of_hours/week  constant integer;
  roles
    emp  total  constant  Employee;
    proj  variable  work_history;
```

Figure 8: Sample TERM Entity and Relationship Types

A general bottom-up procedure for designing TERM schemas has been provided. There are four steps. The first step is to define all nonstandard component value sets. Figure 6 exemplifies this step. As illustrated by the date structure, it is possible to express constraints on the values of the value sets. A so-called relation is also shown that determines whether or not a given date is in a leap year. The next step is to define histories. As illustrated in part by Figure 7, histories have a name, a time structure, a value structure, an optional list of predicates for restricting the set of pairs forming a history, a list of relations, and a list of operations. The third step is to define patterns. A pattern is a value structure together with at least one assertion, at most one derivation function, and zero or more approximation functions, or it is a history structure together with at most one derivation function and zero or more approximation functions. The *sal_history* shown at Figure 7 is an example of the latter. The final step is to define entity and relationship types. These consist of a name and a list of components. The components are specified as either existence, attributes, or roles. Figure 8 give an example of this step.

2.3.3 Summary

TERM was the first temporal ER model and has a Pascal-like syntax. It allows database designers to model temporal aspects through the use of history structures as values of attributes and relationship-type roles. In addition, histories are employed to model the existence of entities and relationships.

2.4 The Model for Objects with Temporal Attributes and Relationships

The motivation for the development of the Model for Objects with Temporal Attributes and Relationships (MOTAR) [23] was to integrate database research in areas such as object-oriented databases, knowledge-based systems, and temporal databases. MOTAR database schemas, termed Data Model

Diagrams (DMDs), are graphical and extend the ER model with temporal relationships and attributes, and with rules. A tool for building DMDs is provided, as is a mapping of DMDs to relation schemas.

2.4.1 The Representation of Time

MOTAR provides built-in features for describing the temporal aspects of a database application, both at the conceptual and the logical level.

MOTAR concentrates on the modeling of the valid-time aspect of data. If the application at hand requires transaction-time support in the database, the approach is to simply add time columns (a single column, registration time, is suggested) to the appropriate relational schemas that result from mapping the DMD to the implementation schema.

At the conceptual level, explicit notation is added to describe the temporal aspects of a mini-world and database design. With this notation, valid-time timestamps become implicit.

The meaning of the new modeling constructs follows from their mapping to logical-level relational schemas. For every temporal aspect described at the conceptual level, corresponding timestamp attributes are added to the relational tables by the mapping algorithm. At the logical level, valid-time is modeled using SQL `DATE` columns; details will be given when the temporal constructs are discussed in the following.

2.4.2 The Model Components

MOTAR includes four kinds of data types: regular entity types, relationship types (non-procedural relationship types), attribute types, and rules (procedural relationship types). The model provides separate notations for temporal attribute types and temporal relationship types. When describing

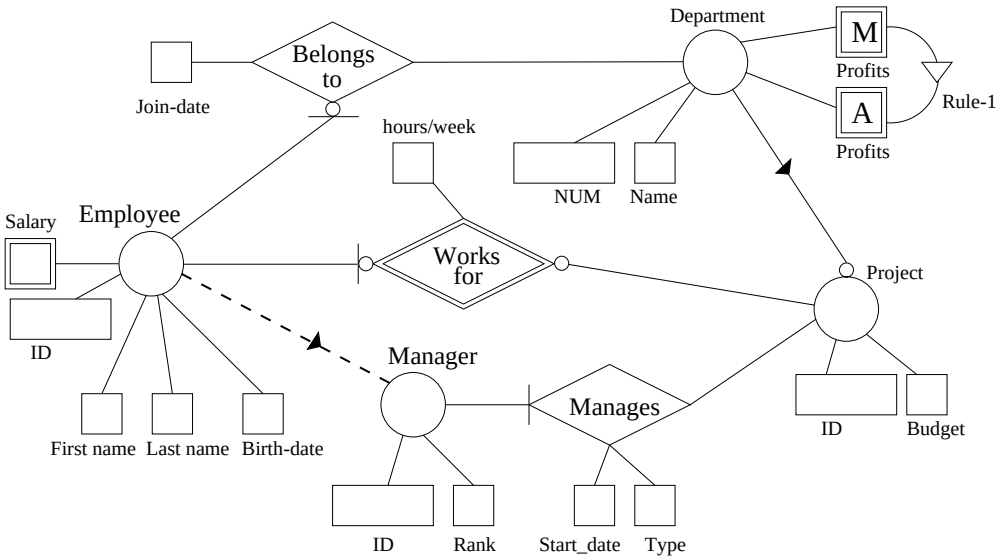

Figure 9: Describing the Running Example Using MOTAR

these constructs in the following, we will use the DMD in Figure 9 for exemplification.

Entity and Relationship Types

Entity types are represented by circles and may be primitive or composite. Composite entity types are built from primitive and composite entity types. In Figure 9, Employee is a primitive entity type. Entity type Department is, as we shall see next, related to Project by means of a Component-Composite relationship. Department entities thus contain Project entities, and Department is a composite entity type.

MOTAR proposes a wider definition of relationship types than do the usual ER models. This more general notion of relationship is introduced to make MOTAR general enough to support a wider variety of applications.

MOTAR relationship types are procedural (rules) or non-procedural. Briefly, the former operate on attribute values of entities or relationships, and they produce results that may update the attribute values of the same entity or relationship, or the attribute values of other sets of entities or relationships.

There are three kinds of non-procedural relationship types, each of which is illustrated in Figure 9 and explained next.

- **Superclass-Subclass (SS) Relationship Types.** These are represented by linking two entity types with a dashed line, with an arrow pointing from the superclass to the subclass. In SS relationship types, the instances of the subclass are of the same type as the instances of the superclass, but additional information is needed for instances of the subclass. In the figure, Manager is a subclass of Employee.

  Inheritance of attributes is supported. Thus instances of the subclass has the same attributes as instances of the superclass, in addition to the attributes specified for the subclass. This inheritance is built into the mapping of SS relationship types to relational tables. For example, the Employee-Manager relationship generates the following table.

  EMP_SUBCLASS_MGR(EMP_ID, MGR_ID)

  Each tuple in this table links information about a manager, in a Manager table, with information, stored in an Employee table, about the manager. Joining this table with an Employee table on EMP_ID and then with a Manager table on MGR_ID will retrieve all the attributes of a manager.

- **Component-Composite (CC) Relationship Types.** These are represented by linking two entity types with a solid line, with an arrow pointing from the composite to the component.

  The notation allows for specifying different constraints. Components being optional is indicated by using a double, solid line for linking the component and the composite. If the composite entities may contain multiple occurrences of the component entity type, the line linking the entity types is given a small circle at the component end. This is exemplified in Figure 9 by letting Project be a component of Department (this is a deviation from the running example). The CC relationship type between Department and Project results in the following relational table being generated.

  DEP_COMPONENT_PROJ(DEP_NUM, PROJ_ID)

  If the composite only contains at most one occurrence of the component, the key of the above relational table will be reduced to the composite identifier only. Whether the component object is optional or not does not matter to the mapping algorithm.

- **General Relationship (GR) Types.** These are relationships between entity types that are neither of type SS nor type CC. They are represented by linking the involved entity types to a diamond with solid lines. N-ary GR types are allowed.

  Each entity type that participates in a GR type has a cardinality ratio that can be either 1 or N. A cardinality ratio of 1 for an entity type means that *the same* instances of the other participating entity types are related to at most one instance on the entity type. A cardinality ratio of 1 is represented by linking the entity type to the diamond with a solid line, as mentioned before. A cardinality ratio of N for an entity type means that *the same* instance of the other participating entity types may be related to more than one instance of the relationship type. A cardinality ratio of N is represented using a solid line ending with a small circle at the diamond side. The DMD in the figure indicates that a department may have more than one employee, but that one employee belongs to at most one department. The meaning of cardinality ratios for time-varying GRs is not given.

  All GR types have one reference entity type that indicates to which entity type the attributes of the GR type refer. The reference entity type is determined from the semantics of the GR type. Figure 9 exemplifies this: because hours/week is meant to describe how many hours per week an employee is working on a project, Employee is the reference entity type of the relationship type Works_for. A reference entity type of relationship is indicated with a small line perpendicular to the line connecting the entity type to the diamond; see the figure.

Using special time-varying GR types, it is possible to describe relations that vary over time, such as project assignments of employees and marriages. Time-varying GRs are represented by double diamonds. In Figure 9, the relationship type Works_for is time varying, stating that employees may be reassigned to other projects. The meaning of time-varying GRs is revealed by their mapping to relational tables in which `DATE` type attributes Start_date and End_date are introduced. Specifically, the Works_for relationship type will be mapped to the following two tables.

REL_WORKS_FOR(EMP_ID, PROJ_ID)

WORKS_FOR(EMP_ID, PROJ_ID, EMP_Hours/week, EMP_Start_date, EMP_End_date)

From the second table, it can be seen that employees may only work for the same project once because the key of the relation WORKS_FOR only consists of EMP_ID and PROJ_ID. Almost all the attribute names are prefixed with EMP because Employee is the reference entity type of Works_for.

Attributes

There are four types of attributes in the model. They are initially divided into identifiers and simple attributes; and simple attributes are either regular, aperiodic, or periodic. Identifiers are represented by rectangles and are considered time-invariant. For example, ID is the identifier of, e.g., the entity type Project. Regular attributes do not change over time and are thus non-temporal. They are represented by squares. For example, as departments' names are not expected to change, Name of Department in Figure 9 is modeled as a simple, regular attribute.

Aperiodic attributes are expected to change over time, at irregular intervals. A double square without a letter inside represents an aperiodic attribute. Attribute Salary of Employee is an example of an aperiodic attribute; it is mapped to the following table.

EMP_Salary(EMP_ID, Salary_date, Salary)

This mapping, with only one time attribute, results in several interpretations of the meaning of aperiodic attributes. For example, aperiodic attributes may be assumed to be step-wise constant. For example, the value of a salary remains constant between updates. The Salary_date value of a tuple then indicates when the tuple's Salary value takes effect. Another interpretation is that aperiodic attributes are assumed to be discrete. For the Salary attribute, this means that a tuple's Salary value is valid only at the time indicated by the value of its Salary_date attribute. The intended meaning is not clear from the description of the model.

Periodic attributes are expected to change over time within specific intervals, e.g., monthly or weekly. A double square with a letter inside represents a periodic attribute. The letter indicates the intervals with which the attribute is monitored. Two periodic attributes, Profits, are used for recording departments' profits. One is sampled monthly, and the other is sampled annually. Rule-1 computes the annual profits, taking the monthly profits as input. Entity type Department is mapped to the following tables.

DEP(DEP_NUM, Name)

DEP_Annual_Profit(DEP_NUM, Profit_Year, Annual_Profit)

DEP_Monthly_Profit(DEP_NUM, Profit_Month, Profit_Year, Monthly_Profit)

From this it can be seen that it is possible to specify a granularity for periodic attributes.

Rules

The notion of rules as known from knowledge-based systems is used for the modeling of procedural relationships. Reference [23] provides argumentation for why rules are thought of as data in MOTAR. Rules are represented using an arrow head that points from the condition of the rule to its conclusion. In Figure 9, Rule-1 exemplifies this; for further details, see [23].

2.4.3 Summary

MOTAR provides the database designer with new modeling constructs for describing time-varying attributes, both periodic and aperiodic, and for describing time-varying relationships. These constructs "hide" the time attributes that would otherwise be necessary.

2.5 The Temporal EER Model

The motivation for developing the Temporal EER (TEER) Model [12, 11] was that its authors believe that it would be more natural to specify temporal data and temporal queries in a conceptual, entity-oriented model than in a tuple-oriented relational data model. TEER does not add new syntactical constructs to the EER model; instead, it gives new meaning to the existing EER modeling constructs making them temporal.

2.5.1 The Representation of Time

The time representation is similar to that proposed by Gadia and Yeung [14] for the relational model, but is adapted to the requirements of the ER model. A *time interval*, denoted by $[t_1, t_2]$, is defined to be a set of consecutive equidistant time instants, where t_1 is the starting instant and t_2 the ending instant. The distance between two consecutive time instants can be adjusted based on the granularity of the application to be equal to months, days, or other suitable time units. A *temporal element* is a finite union of time intervals denoted by, $\{I_1, I_2, \ldots, I_n\}$ where I_i is an interval in $[0, now]$. A *temporal database* stores historical information for a time interval $[0, now]$ where 0 represents the starting time, of the database mini-world application, and now represent the current time which is continuously expanding.

The authors state that the TEER model has no limitations regarding support of time dimensions, but due to space limitations, the articles consider only valid time.

2.5.2 The Model Components

The TEER model extends the EER model [10] to include temporal information on entities, relationships, superclass/subclasses, and attribute. Since the graphical representation of TEER model components is similar to that of the EER model presented by Elmasri and Navathe [10], we will not explain it in detail. Instead, we will concentrate our attention on the new meaning given to the syntactical constructs of the EER model.

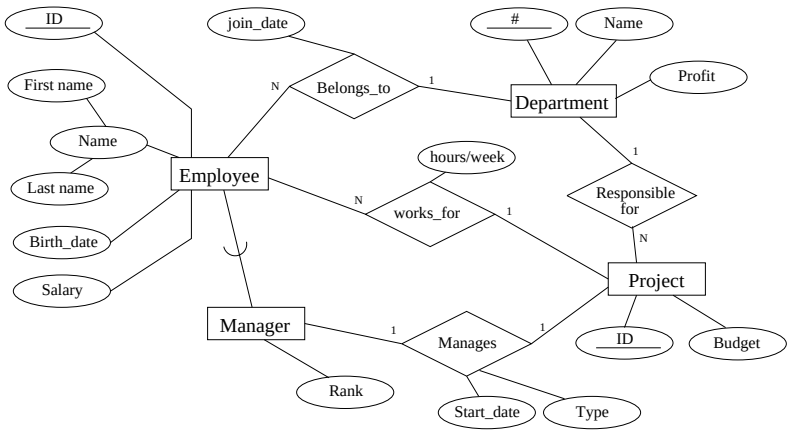

Figure 10: A TEER Schema Modeling the Running Example

Entities and Entity types

In the TEER model, each entity e of entity type E is associated with a temporal element $T(e) \subseteq [0, now]$ that gives the *lifespan* of the entity. The lifespan of an entity can be a continuous time interval, or it can be the union of a number of disjoint time intervals. In TEER, each entity type has

a system-defined SURROGATE attribute whose value is unique for every entity in the database. The value of this attribute is hidden from the user and does not change throughout the lifespan of the entity. The temporal element of the SURROGATE attribute of entity e defines the lifespan $T(e)$ of the entity.

The temporal properties of weak entities are similar to those of regular entities, except that the temporal element $T(e)$ of each weak entity must be a subset of the temporal element of its owner entity.

Attributes and Keys

The attribute types of the TEER model are the same as those of the EER model, although they are all temporal. The *temporal value* of each attribute A_i of e, denoted by $A_i(e)$, is a partial function $A_i(e) : T(e) \rightarrow dom(A_i)$. This is also referred to as a *temporal assignment*. The subset of $T(e)$ in which $A_i(e)$ is defined and denoted by $T(A_i(e))$ is called the *temporal element of the temporal assignment*. It is assumed that A_i has the value NULL or UNKNOWN during the time intervals $T(e) - T(A_i(e))$.

To give an example of the above, consider the database described by Figure 10, and assume that the chosen granularity of time is a day. A particular EMPLOYEE entity e with lifespan $T(e) = [7/1/90, now]$ may have the temporal attribute values given in Figure 11.

$$
\begin{array}{lcl}
\text{SURROGATE}(e) & = & \{[7/1/90, now] \rightarrow \text{ surrogate\_id}\} \text{ (system generated)} \\
\text{ID}(e) & = & \{[7/1/90, now] \rightarrow 98765\} \\
\text{First name}(e) & = & \{[7/1/90, now] \rightarrow \text{Chris}\} \\
\text{Last name}(e) & = & \{[7/1/90, now] \rightarrow \text{Johnson}\} \\
\text{Birth\_date}(e) & = & \{[7/1/90, now] \rightarrow 8/23/46\} \\
\text{Salary}(e) & = & \{[7/1/90, 6/30/92] \rightarrow \$\ 20K, \\
 & & \ [7/1/92, now] \rightarrow \$\ 30K\}
\end{array}
$$

Figure 11: Example of a Lifespan of an Entity

The following constraint apply to attributes and keys in the TEER model. Simple single-valued attributes have at most one atomic value for each entity at each time instant $[t]$. Multivalued attributes can have more that one value for an entity at a given time instant $[t]$. For a given time instant $[t]$, the value of a composite attribute of an entity is the concatenation of the values of its components. The temporal element of a temporal assignment of a composite attribute is the union of the temporal elements of the temporal assignments of its components. A key attribute is an attribute of an entity type with the constraint that at any time instant $[t]$ in $[0, now]$, no two entities will have the same value for this attribute. TEER allows updates of key attributes since each entity is uniquely identified by its system-defined SURROGATE.

Relationship Types

Like entities of entity types, each relationship instance r is associated with a temporal element $T(r)$ that defines the lifespan of the relationship instance. A constraint states that $T(r)$ must be a subset of the intersection of the temporal element of the participating entities. That is, $T(r) \subseteq (T(e_1) \cap T(e_2) \cap \ldots \cap T(e_n))$ where $T(e_i)$ is the lifespan of the i'th entity participating in r. Relationship attributes are treated similarly to entity attributes; the temporal value $A_i(r)$ of each simple attribute A_i is a partial function $A_i(r) : T(r) \rightarrow dom(A_i)$ and its temporal element $T(A_i(r))$ must be a subset of $T(r)$. The cardinality ratios of the participating entity types have not been given any new meaning.

The TEER model also offers user-defined and predicate-defined superclass/subclass relationships. An entity e of a superclass E will belong to a predicate-defined subclass C throughout all time intervals where the defining predicate evaluates to true for that entity. For a user-defined subclass, the user specifies when the entity is to be a member of the subclass. In either case, the entity will have a temporal element $T(e/C)$ that specifies the time intervals during which it is a member of the subclass C. The constraint $T(e/C) \subseteq T(e)$ on temporal elements must hold. Attributes of a subclass are treated similarly to other attributes; the temporal elements of their temporal assignments must be subsets of $T(e/C)$.

2.5.3 Summary

TEER does not add any new syntactical constructs to the EER model, but changes the semantics of all the standard EER constructs, making them temporal. TEER do not provide any mapping from TEER diagrams to any implementation model.

2.6 The Semantic Temporal EER Model

The Semantic Temporal EER model (STEER) [8, 9] was developed in order to compensate for a lack of consideration of the semantics associated with time in previous research that had concentrated on temporal data models and query languages in the context of the relational model and not so much in the context of conceptual data models. STEER introduces a new classification concept for temporal and conceptual objects and provides guidelines for identifying objects as conceptual or temporal.

2.6.1 The Representation of Time

The representation of time in STEER is very similar to the representation of time in the TEER model just surveyed. Actually, the only difference is that the time domain T of the database application is expanded from $T = \{t_0, t_1, t_2, \ldots, t_{now}\}$ to $T = \{t_0, t_1, t_2, \ldots, t_{now}, t_{now+1}, \ldots\}$. That is, it is now possible to reference future time points. NULL is used to represent the unknown time point, and t_{now} is used to represent the current time point. STEER only supports valid time.

2.6.2 The Model Components

The STEER model distinguishes between conceptual and temporal entities. A conceptual entity is treated as an object with permanent existence. That is, once an entity is created in the database, it can be referenced at any future point in time. A temporal entity—also called an entity role because it models one of the several roles that a conceptual entity can participate in over time—on the other hand, has a specific lifespan describing its existence. STEER distinguishes between temporal and non-temporal attributes, and it differentiates between temporal and conceptual relationships as well. It also defines temporal constraints among entity roles and conceptual and temporal relationships.

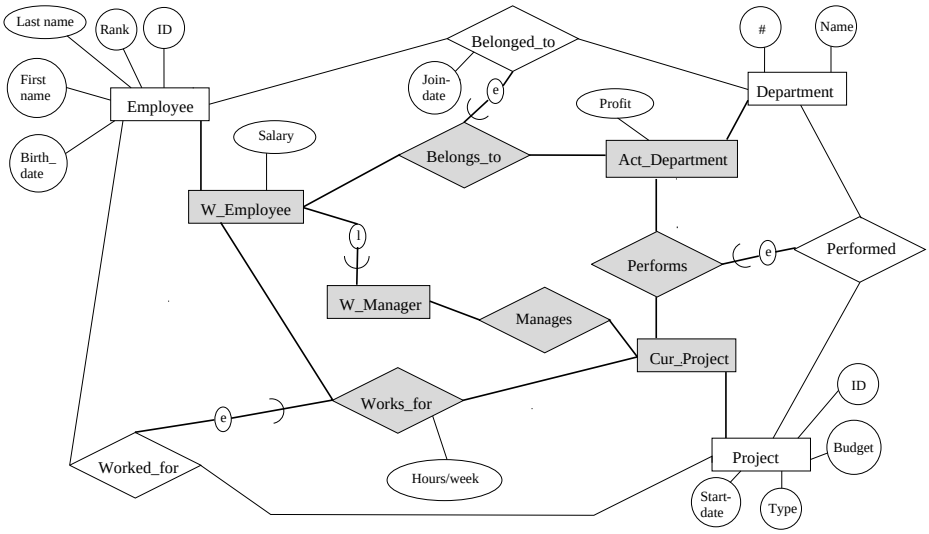

Figure 12: The Running Example Modeled Using the STEER Model

Conceptual Entities and Their Entity Roles

To understand the idea behind the distinction between conceptual entities and entity roles, consider an example. Initially, note that entities from the modeled mini-world need to be represented in the database when they become of interest. For example, students exist in the mini-world as persons.

However, they do not become of interest to a university before they have been accepted at the university. At that point, the university might want to record previous information about the students. Then, when students leave the university, they often remain of interest to the university for some time. So the conceptual existence of an entity does not directly correspond to the birth, dead, or change of the entity. In this example, persons are modeled as conceptual entities, and (persons in their roles as) students are modeled as entity roles.

Conceptual entities describe the conceptual aspects of the real world. A conceptual entity type is a set of conceptual entities of the same type. Conceptual entity types are represented by rectangles in STEER diagrams; in Figure 12, Employee is an example.

The temporal aspects of the real world are described by temporal entities which are also called *entity roles* because they represent the active roles a conceptual entity can participate in. A *role type* is a set of entity roles of the same type. Each role type is associated with a single entity type called its *owner entity*. A role type is represented by a filled rectangle and connected to its owner entity type. W_Employee in Figure 12 is an example. W_Employee models all the employees currently employed by the company.

Each conceptual entity e is associated with an *existence time*, ET. The *start time point* ST of the existence time refers to the time when the entity was recorded in the database. The *end time point* of an existence time is infinity because an entity once created never ceases to exist. Hence, $ET = [ST, \infty[$.

Each entity role ro of a role type RO is associated with a temporal element $T(ro) \subset [t_0, \infty[$ that gives the lifespan of the entity role. The lower bound (start time) t_l of a lifespan $[t_l, t_u]$ of an entity role must be closed; t_l cannot be NULL because the start time of an entity role cannot be unknown; nor can it be t_{now}, since the current time is a dynamic concept. The upper bound (end time) t_u can either be closed or open; t_u can be t_{now} if $t_l \leq t_{now}$ or NULL if $t_l > t_{now}$.

The association between a conceptual entity and its entity roles can be viewed as some sort of superclass/subclass relationship with mutual inheritance of attributes and relationship instances. The following set of rules clarify this relationship.

1. A role type has exactly one entity type as owner.
2. The start time of the lifespan of en entity role must be greater than or equal to the start time of the owner entity.
3. A role type can only have temporal attributes.
4. Attributes of a role type are "public" to the owner entity type, and attributes (temporal and non-temporal) of the owner entity type are "public" to all the associated role types.
5. An entity role can access all relationship instances of relationship types in which the owner entity participates, and, reversely, an entity can access all relationship instances of relationship types in which the associated entity roles participates.

Non-temporal and Temporal Attributes

Non-temporal attributes can only be properties of conceptual entity types. The value of a non-temporal attribute of an entity holds over the entire existence time of the entity. Non-temporal attributes are diagrammically represented with circles. An example is the non-temporal attribute ID of Employee in Figure 12.

Each entity is provided with a system-defined non-temporal SURROGATE attribute whose value is unique for every entity in the database. The value is not visible to the user and is never altered.

Each entity type E or role type RO may have a set of temporal attributes $TA_1, TA_2, \ldots, TA_n$, and each temporal attribute TA_i is associated with a domain of values, $dom(TA_i)$. In STEER diagrams, temporal attributes are represented by ellipses; an example is the temporal attribute profit of Act_Department in Figure 12.

The next definitions are very similar to those presented in Section 2.5. For entity roles, the temporal value of each attribute TA_i of ro, referred to as $TA_i(ro)$ is a partial function from $T(ro)$ to $dom(TA_i)$. The subset of $T(ro)$ in which $TA_i(ro)$ is defined is denoted by $T(TA_i(ro))$. It is assumed that TA_i has NULL or UNKNOWN as its value during the intervals $T(ro) - T(TA_i(ro))$. The similar definitions

apply to entities, the only difference being that $T(ro)$ is replaced by $ET(e)$ (i.e., the lifespan of entity e).

The partial function that describes the value of a temporal attribute is also called a temporal assignment. The subset of time points during which a temporal attribute is defined is called the temporal element of the temporal assignment. The different types of temporal attributes are similar to those of the TEER model. For non-temporal attributes of an entity, the temporal element of the temporal assignment is equal to the existence time of the entity.

For an example of the above, consider the database described in Figure 12 and assume that the chosen granularity of time is a day. A particular Employee entity e with existence time $ET(e) = [7/1/90, \infty[$ may have the following temporal attribute values:

SURROGATE(e)	=	$\{[7/1/90, \infty[\rightarrow$ surrogate_id$\}$ (system generated and non-temporal)
ID(e)	=	$\{[7/1/90, \infty[\rightarrow 98765\}$ (non-temporal)
First name(e)	=	$\{[7/1/90, \infty[\rightarrow$ Chris$\}$ (non-temporal)
Rank(e)	=	$\{[7/1/90, \infty[\rightarrow$ Senior manager$\}$ (non-temporal)
Last name(e)	=	$\{[7/1/90, now] \rightarrow$ Johnson$\}$
Birth_date(e)	=	$\{[7/1/90, \infty[\rightarrow 8/23/46\}$ (non-temporal)
Salary(e)	=	$\{[7/1/90, 6/30/92] \rightarrow \$\ 20K$,
		$[7/1/92, now] \rightarrow \$\ 30K\}$

Conceptual and Temporal Relationships

A conceptual relationship type R of degree n has n participating entity types $E_1, E_2, \ldots, E_n$. Conceptual relationship types cannot have role types as participants. Each relationship instance r in R is an n-tuple $\langle e_1, e_2, \ldots, e_n \rangle$ with $e_i \in E_i$. Each relationship instance r in R has an existence time ET. The start time must be greater or equal to the start time of the existence time of each of the n participating entities, i.e., $ST(r) \geq ST(e_i)$ for all e_i. Conceptual relationships are represented by diamonds in STEER diagrams. Worked_for in Figure 12 is an example.

A temporal relationship type TR of degree n has n participating entity types or role types $O_1, O_2, \ldots, O_n$ where O_i is either an entity type or a role type. Thus, each *temporal* relationship instance tr in TR is a n-tuple $\langle o_1, o_2, \ldots, o_n \rangle$ with $o_i \in O_i$. Temporal relationships are represented by filled diamonds, and an example in Figure 12 is Belongs_to. Each temporal relationship instance tr is associated with a temporal element $T(tr)$ that give the lifespan of the temporal relationship instance. This lifespan must be a subset of the intersection of the lifespans of the involved entity roles and entities.

As for entities and entity roles, the association between a conceptual relationship type and a temporal relationship type can be seen as some sort of superclass/subclass relationship. Two constraints are enforced on temporal and conceptual relationships.

First there is the *R-existence Constraint*. This constraint, denoted by R/TR, holds between a conceptual relationship type R and temporal relationship type TR where all the participating object types are role types if for each $tr_i = \langle ro_1, ro_2, \ldots, ro_n \rangle$ in TR, the following two conditions hold.

- There exists a corresponding conceptual relationship $r_i = \langle e_1, e_2, \ldots, e_n \rangle$ in R such that $owner(ro_j) = e_j$ for each ro_j in tr_i.
- The start time of the lifespan of tr_i must be greater than or equal to the start time of the existence time of the corresponding conceptual relationship r_i.

Second, there is the *R-lifespan constraint*, denoted by TR/R. This constraint holds between a temporal relationship type TR where all the participating objects are role types and a conceptual relationship type R if for each $r_i = \langle e_1, e_2, \ldots, e_n \rangle$ in R, the following two conditions hold.

- There exists a corresponding temporal relationship $tr_i = \langle ro_1, ro_2, \ldots, ro_n \rangle$ in TR such that $e_j = owner(ro_j)$ for each e_j in r_i.
- The start time of the existence time of the conceptual relationship r_i must be greater than or equal to the start time of the lifespan of the corresponding temporal relationship tr_i.

The R-lifespan constraint is used to model the cases where a conceptual relationship cannot exist until after a temporal relationship has started. For, example students cannot get transcript entries for courses until *after* they have enrolled. R-existence and R-lifespan constraints are represented in STEER diagrams by placing an oval with an e an a l, respectively, on the line connecting the involved relationship types.

Superclass/Subclass Relationships

Like the EER model, STEER supports the concepts of sub and superclasses and the related concepts of specialisation and generalization. A class is any set of entities; hence, an entity type is also a class.

A member entity of a conceptual subclass represents the same real-world entity as some member entity in its conceptual superclass. Thus, an entity cannot exists in the database as a member of a subclass without also being a member of the superclass. This implies that an entity that is a member of a subclass will have the same existence time as the corresponding entity in its superclass.

Attributes of a superclass are inherited by its subclasses. A subclass entity also inherits all relationship instances in which its corresponding entity in the superclass participates. The graphical notation for superclass/subclass relationships is similar to that of the EER model [10]. However, one should notice that when converting a non-temporal EER diagram into an STEER diagram, many or most of the subclasses are likely to become role types. An example of this is given in Figure 13 where the non-temporal EER schema to the left is converted to the STEER diagram to the right. This is also the reason why no conceptual entity type Manager exists in Figure 12 and why the non-temporal attribute Rank has to be moved to Employee.

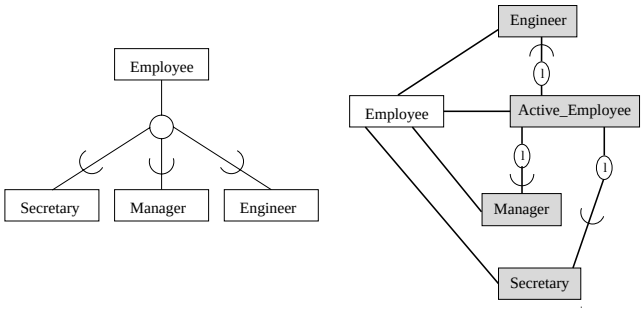

Figure 13: Mapping Non-temporal Superclass/subclass Relationships to the TEER Model

When role types participate in superclass/subclass relationships, two temporal constraints may be indicated. An existence constraint holds between two role types RO_i (superclass) and RO_j (subclass) if for all roles ro_{jk} in RO_j, there exists a role ro_{il} in RO_i such that $ro_{jk} \equiv ro_{il}$. Next, a lifespan constraint holds if the lifespan of any entity role ro_{jk} in RO_j is a subset of the lifespan of the entity role ro_{il} in RO_i with $ro_{jk} \equiv ro_{il}$. Notice that the lifespan constraint implies the existence constraint, but not vice versa. In STEER diagrams existence and lifespan constraints are represented the same way as R-existence and R-lifespan constraints. Figure 12 contains an example of a lifespan constraint between W_Employee and W_Manager is shown. The l in the oval is replaced by an e if an existence constraint is to be indicated.

2.6.3 Summary

STEER is a semantic temporal model where conceptual entities are considered to exist forever (or more precisely, from when they become of interest to the application), whereas the roles they participate in, i.e., the temporal entities, have lifespans to determine their existence. The same distinction holds for relationships. A general set of constraints for preserving temporal consistency is presented.

2.7 The Entity-Relation-Time Model

The Entity-Relation-Time (ERT) model exists in two versions, the original version [33, 35] and a recent refinement [21]. We survey first the original model and then discuss the refinements at the end.

The motivation for the development of the original ERT model was to meet the need for conceptual models of enhanced system functionality. In ERT, this need is addressed through the use of a conceptual modeling formalism that caters for the modeling of *business rules, time*, and *complex objects*. This formalism is supported at the database level by an extension of the relational model with temporal semantics and an execution mechanism that provides active-database functionality.

In the description of ERT, the term class is used instead of the term type. We will follow the description. The basic structures of ERT are those of the binary entity-relationship model, with the exception that it regards any association between objects as a relationship. Specifically, the distinction between "attributeships" and relationships is avoided. The ERT model extends the ER model both in its semantics and graphical notation in two directions: the modeling of time-varying information; and the modeling of complex objects.

In the ERT model, the term *time-varying* information refers to pieces of information where the modeler wants to keep track of their evolution, i.e., wants to record their variation over time.

2.7.1 The Representation of Time

Time is introduced in the ERT model via a distinguished entity class, the *time period* class, and the time period is considered the most primitive temporal notion in the model. A time period starts and ends in a *tick* and also has a duration expressed in ticks, i.e., a tick is defined as the smallest unit of time permitted in ERT. Each time-varying entity class and relationship class is timestamped with a time period class. That is, a time period with a specified granularity is assigned to every time-varying piece of information that exists in an ERT schema.

When a time period class is associated with an entity class, it models the lifespans of the entities in the class. The lifespan of an entity is also referred to as its *existence period*. When a time period class is associated with a relationship class, it models the time period during which a relationship is valid. This is referred to as the *validity period* of a relationship instance and models the period in time that the relationships holds. This latter time notion thus corresponds to valid time.

A number of assumptions were made in order to increase the feasibility and practicality of the proposed approach, including the following.

1. System-generated surrogates are used for unique identification of entities.

2. Reincarnation of entities is permitted, i.e., if an entity no longer is in the database, meaning that the existence period of the entity ends in a tick less than the current time, it can return using the same surrogate. This implies that entities keep their identity through time.

3. Existence and validity periods should always be mapped onto the calendar axis, i.e., they should be specified in absolute terms. That is,

   - if the existence period of a timestamped entity is not specified explicitly as an absolute value, then the current time is taken as start point of the existence period, and
   - if the validity period of a timestamped relationship is not specified explicitly as an absolute value, then the most recent starting point of the existence times of the involved entities is taken as start point of the validity period of the relationship.

4. Non-timestamped entities and relationships are assumed to always exist, i.e., they exist from the system start-up time until the current time.

2.7.2 The Model Components

The most central concept of the ERT model is that of a class, defined in the usual way. This means that the most primitive data abstraction is classification of individual objects. Thus, in ERT schemas, entity classes, value classes, complex entity classes, complex value classes, and relationship classes are specified.

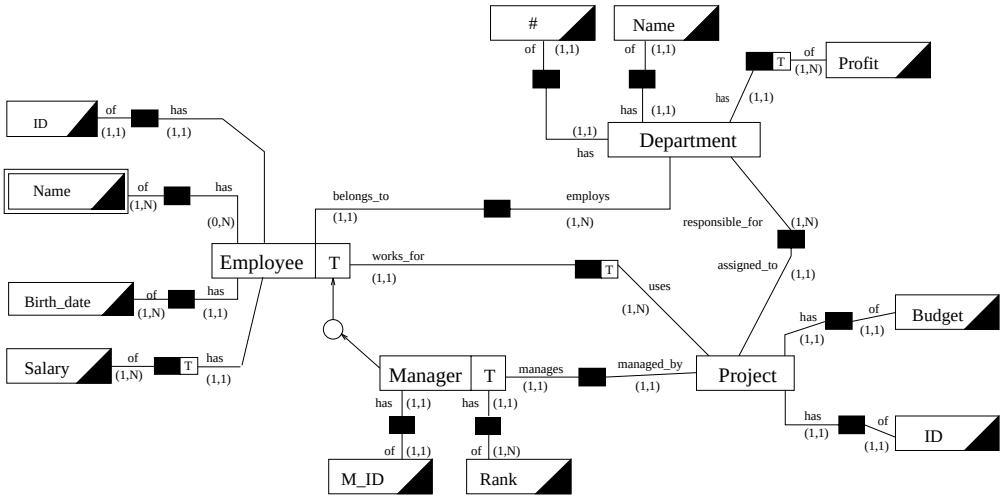

Figure 14: ERT Schema Description of the Running Example

Simple Entity Classes and Simple Value Classes

A simple object cannot be decomposed into other objects and hence has independent existence—it is irreducible. The simple objects classes of the ERT model are divided into two groups: simple entity classes and simple value classes.

A simple entity class is represented by a rectangle, and if the entity class is time-varying, the rectangle is expanded with a "time box." An example of a time-varying, simple entity class is Employee (shown in Figure 14), and an example of non-timestamped entity class is Project.

A simple entity class can be derived. This implies that its instances are not stored by default, but can be obtained dynamically when needed, by using *derivation formulas*. A derived entity class is represented by a dashed rectangle. Derived entity classes can be time-varying as well. For each derived entity class, there is exactly one derivation formula that gives the members of that entity class at any time. If the derived entity class is not timestamped, the corresponding derivation formula instantiates this entity class at all times; whereas if the entity class is timestamped then the derivation formula obtains instances of this class together with their existence periods.

A simple value class is represented by a rectangle with a black triangle placed in the bottom right corner. Simple value classes cannot be time-varying. An example of a simple value class is Name in Figure 14. A simple value class can, like a simple entity class, be derived and is then represented by a dashed rectangle with a black triangle placed as before.

Complex Entity Classes and Complex Value Classes

A complex object is an object that can be decomposed into other objects, and thus its existence depends of the existence of its component objects. The relationship between the complex object and its component objects is modeled using IS_PART_OF relationships. The complex object classes, like the simple objects classes, are divided into two groups, complex entity classes and complex value classes.

Complex value classes are represented by a double rectangle with the black triangle placed (as usual) in the inner rectangle. Complex value classes can only have complex value classes or simple value classes as components, and hence a complex value class cannot be time-varying. An example of a complex value class is Name in Figure 14. The IS_PART_OF relationship cannot be seen at this level of abstraction; an example of unfolding a complex class will be given later.

A complex entity class is represented by a double rectangle, and if it is time-varying, the "time box" is added to the inner rectangle. The components of a complex entity class can be both simple and complex, and they can be of value class and entity class type. The time semantics of timestamped complex objects will be explained in detail after the explanation of the IS_PART_OF relationship.

In the presentation of MOTAR, Project was described as a component of the composite entity type Department. This could also have been done in ERT by making Department a complex entity

class, but then it would not have been possible to describe the relationship between Project and, e.g., Employee.

Relationships Classes

In ERT there are four kinds of relationship classes. There are the user-defined relationship classes, the IS_PART_OF relationships between complex objects and their composite objects, the ISA relationships between subclasses and their superclasses, and the objectified relationships. We explain each in turn.

User-defined relationship classes are binary and are represented by small filled rectangles; if they are time-varying, a "time box" is added. There are two constraints on the validity periods of a relationship class' instances. First, the intersection of the existence periods of the participating entities must be non-empty. Second, the validity period of the relationship instance must be a sub-period of the intersection of the existence periods of the involved entities.

An instance of a user-defined relationship class is viewed as a named set of two (entity or value, role) pairs, where each role expresses the way that a specific entity or value is involved in the relationship. These two named roles are called *relationship involvements* and are required in a ERT schema for completeness reasons. In addition to the relationship involvements, a *cardinality constraint* is required to be specified for each entity class participating in the relationship class. Each cardinality constraint is a pair (α, β) where α indicates the minimum and β the maximum number of times that an entity or value can participate in a relationship. The cardinality constraints are also used to specify whether the involvement is *optional* or *mandatory*. If the involvement is mandatory then α=1, whereas if α=0, the involvement is optional.

As an example, see the relationship class between Employee and Department shown in Figure 14. The two relationship involvements are "belongs_to" and "employs." The two corresponding cardinality constraints state that each Employee instance is related to (i.e., belongs_to) precisely one Department instance, yielding a *uniqueness constraint* on Employee; and each Department instance is related to (i.e., employs) from one to N Employee instances. If both the cardinality constraints of a relationship class between a entity class and a value class are (1,1), this corresponds to the notion of a *key* in database theory.

User-defined relationship classes can, like simple entity classes, be derived and are then represented by dashed, non-filled rectangles; and they can be time varying. As for a derived timestamped entity class, the derivation formula of a derived timestamped relationship class specifies a validity period for each instances of the class.

ISA relationship classes are first divided into two groups, partial and total, that are further sub-divided into overlapping and disjoint, yielding four types of ISA relationship classes. The partial ISA relationship class is represented by a non-filled circle with arrows flowing from the subclass to the circle and an arrow flowing from the circle to the superclass. The total ISA relationship class is represented by a filled circle. If there is more than one subclass and more than one arrow is pointing into the circle, the relationship class is disjoint; otherwise the relationship class is overlapping. The existence time of a specialized entity should be a sub-period of the existence time of the corresponding parent entity.

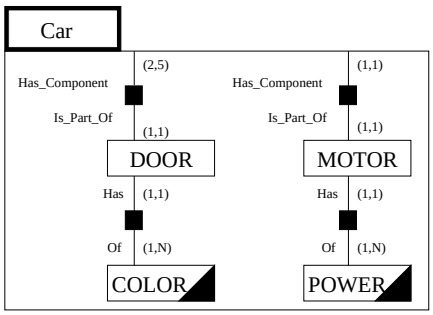

Figure 15: Unfolding a Complex Entity Class

IS_PART_OF relationship classes are used to specify the relationships between the components of a complex object and the complex object itself. Each directly subordinate object class is IS_PART_OF-

related to the complex object class, which in turn is HAS_COMPONENT-related to the composite object class. This composition mechanism does not make any distinction between aggregation and grouping, but is rather general. Whether the HAS_COMPONENT involvement is one of aggregation or grouping can be indicated using cardinality constrains. That is, if the cardinality is one of (1,1) or (0,1), the component is an aggregate, whereas if it is (0,N) or (1,N) the component is a set. Figure 15 gives an example.

In ERT, complex objects can be used to model both *logical part hierarchies*, where the same component can be part of more that one complex object, and *physical part hierarchies*, in which an object cannot be part of more than one complex object at the same time. To achieve this, four different IS_PART_OF relationship classes are defined using combinations of two orthogonal types of constraints, namely *dependency* and *exclusiveness*. The dependency constraint *dependent* states that when a complex object ceases to exist, all its components also cease to exist (dependent composite reference), and the dependency constraint *independent* states that if a complex object ceases to exist, this does not imply that the components cease to exist (independent composite reference). The exclusiveness constraint *exclusive* states that a component object can be part of at most one complex object (exclusive composite reference) at a time, and the exclusiveness constraint *shared* states that it can be part of more than one complex object at a time (shared composite reference).

No specific notation is introduced for these constraints. Rather, they are given by the cardinality constrains of the IS_PART_OF relationship. That is, assume that the cardinality constraint of the IS_PART_OF relationship is (α, β). Then $\alpha = 0$ implies independent dependency, $\alpha \neq 0$ implies dependent dependency, $\beta = 1$ implies exclusive exclusiveness, and $\beta \neq 1$ implies shared exclusiveness.

Timestamping in a time-varying IS_PART_OF relationship of a complex object is subject to different time constraints depending on whether it has dependent or independent dependency semantics and exclusive or shared exclusiveness semantics. The dependency constraint dependent in time-varying IS_PART_OF relationships implies that the existence time of the complex object and the component object should end at the same time as does the validity period of the IS_PART_OF relationship. The exclusiveness constraint exclusive implies that if an object A is part of objects B and C, then the period during which A is part of B should have empty intersection with the period during which A is part of C.

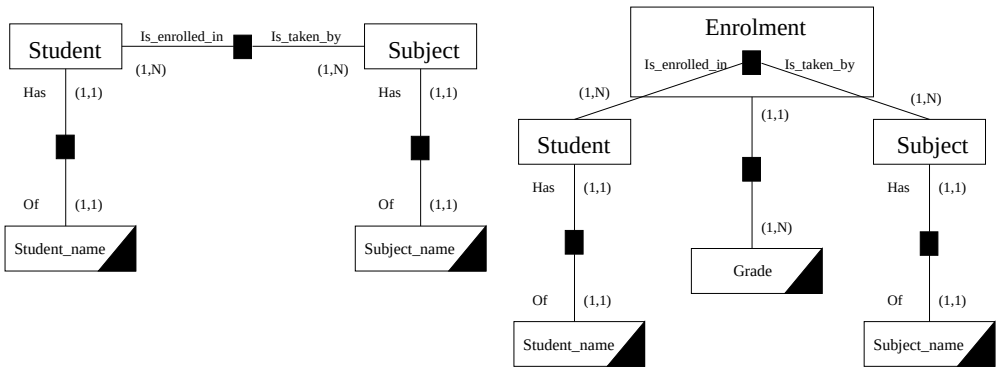

Figure 16: Objectified Relationship

In ERT, only binary relationship classes can be specified. Thus attributes cannot be attached to relationship classes since this would make the relationship class "ternary." As illustrated in Figure 16(a) this may yield problems. If we want to add the class GRADE to this schema, we will face the problem of where to add it. Specifically, GRADE has to be attached to either STUDENT or SUBJECT, both of which are problematic. There is thus a need to model ternary relationships. To achieve this, ERT permits relationship classes to participate in relationships. This is called *nominalisation*, and the particular construct in which a relationship class is viewed as an entity class is called an *objectified relationship*. An objectified relationship must include the two corresponding involvements. The relationship class that is objectified should always be many to many (the cardinality constraints on both of the involvements must be (1,N)). The status of an objectified relationship is that of an

entity class. As such, it may participate in any relationship *except* that of an ISA relationship. Also, the existence periods of the objectified timestamped relationship class' instances are the same as the validity periods of the corresponding nominalised relationship class instances. The graphical notation of objectified relationships is depicted in Figure 16(b).

2.7.3 Refining the Original ERT Model

The original ERT model has recently been refined [21] in two respects. First, the definitions of temporal objects (entities or relationships) are given mathematically, by specifying what constraints are placed on the existence or validity periods of an object when a *temporal marking* is applied to it. Second, temporal markings are used to represent temporal variation of object with respect to each other. In particular, the period in which a relationship involvement can exist is related to the period in which the associated entities exist, and the periods in which entity subclasses exist are related to the period in which their superclass exists. Two distinct aspects of the temporal nature of relationship involvements, called *historical perspective* and *temporal variation*, are identified. As a precursor to delving further into this, we consider a refinement of ERT's time periods.

A notation for describing the ticks when an instance of a temporal entity class exists or a temporal relationship class holds is introduced. The period over which an instance of a temporal entity class or temporal relationship class x exists/holds is a set $I_x = \{t_a, t_b, \ldots, t_z\}$ where $t_a, t_b, \ldots, , t_z$ are the ticks at which x exists/holds. Since the series of ticks usually is continuous, I_x is called an *interval* although what actually has been defined is a set of intervals [21]. This definition of "intervals" allows for the use of the usual set operators. To ensure a discrete bounded model, the possible ticks of an interval are limited to the finite set of $\aleph = \{0 \ldots \tau\}$, and for all x, the interval I_x will satisfy $I_x \subseteq \aleph$.

In the original ERT model, a relationship class could only be marked with a T-mark indicating that the relationship was time-varying. The temporal marking is refined in [21] to include H-marks and TH-marks. In the following, interval I_E ranges over all intervals associated with entity class E; and the properties of intervals that we give must hold for all instances of the entity class. Thus stating $I_E \subseteq \aleph$ means that for all entities e in E, $I_e \subseteq \aleph$.

If a relationship involvement exists for a subset of the ticks for which both the entities it associates exist, and only associate entities which exist at the same time tick, then the relationship is said to undergo *temporal variation* with respect to the entities it associates, and the relationship is T-marked.

If a relationship involvement exists at certain ticks between entity E_1, which exists at those ticks, and a entity E_2, which exists at *other* ticks, then the relationship is said to have *historical perspective*, and the relationship is H-marked. Note that such relationships are asymmetric, since at any tick only E_1 is required to exist; the inverse relationship (from E_2 to E_1) may not hold at the same tick.

The above-mentioned terms can be combined. Saying that a historical perspective has temporal variation means that that one of the entities involved does not have the perspective for its entire existence.

Four constraints on the validity period of an instance of a relationship class results. Initially, let I_{E_1} and I_{E_2} be the intervals for which entity classes E_1 and E_2 exist. First, if I_R is the interval over which the instances of E_1 and E_2 are involved in an unmarked relationship, then $I_R = I_{E_1} \cap I_{E_2}$. Second, if I_R is the interval over which the instances of E_1 and E_2 are involved in a T-marked relationship class, then $I_R \subseteq I_{E_1} \cap I_{E_2}$. Third, assume that instances of entity classes E_1 and E_2 are related by R. If the instances of E_1 and E_2 are involved in R over period I_{E_1} and I_{E_2}, respectively, and the relationship class R is H-marked, then $I_{E_1} \neq I_{E_2}$ is allowed. To exemplify an H-marked relationship class, consider the grandparent/grandchild relationship between persons. Here, related persons need not exist simultaneously for any tick; a grandparent may die before the birth of a grandchild. As we shall see next, the historical perspective of a relationship has a *temporal direction*. An H-marked relationship class R relating E_1 and E_2 is described as

- *past* if E_2 holds at ticks before the ticks at which E_1 holds,
- *current* if E_2 holds at the same ticks for which E_1 holds, and
- *future* if E_2 holds at ticks after the ticks at which E_1 holds.

Finally, Boolean combinations of the above are possible. It follows that an unmarked relationship class is merely a *current* historical perspective relationship class. In the above example the temporal direction could be *past* and *current* (depending on what is E_1 and E_2). The characteristics of H-marked relationships can be described using derived entity classes, for details see [21]. Fourth, assume that instances of E_1 are involved in R over period $I_{E_1R} \subseteq I_{E1}$ and instances of E_2 are involved in the same relationship instance for I_{E2} and relationship R is TH-marked. Then $I_{E_1R} \neq I_{E2}$ is allowed. This TH-mark can be used to model that we do not want the grandparent to be related to the grandchild before the grandchild is actually born.

2.7.4 Summary

The data model ERT extends a binary entity-relationship model with complex entity classes and complex value classes. ERT provide the users with temporal markings of time-varying entity and relationship classes, and instances of these classes are timestamped with time periods. The temporal markings of classes have later been refined.

2.8 The Temporal ER Model

The Temporal ER model (TER) [31] has it origin in the ER model. Most centrally, TER replaces the ordinary cardinality constraints with snapshot and lifetime cardinality constraints. This permits a refinement of the classification of relationship types, thereby obtaining a total of six different classes of relationship types; and it leads to a refinement of the optionality of relationship participation.

Designing a database using the TER model includes three steps. First, a TER diagram is constructed that uses the two new types of constraints for describing the time-varying aspect of relationship types. No time attributes are included. Then, based on how often historic data is expected to be accessed, a particular algorithm that translates TER diagrams into traditional ER diagrams is applied, leading to a diagram with only regular cardinality constraints and with explicit time attributes. Third, the ER diagrams is translated into relational tables using a standard mapping.

2.8.1 The Model Components

The key differences between TER and the (binary) ER model are the inclusion of snapshot and lifetime cardinality constraints, and the intermediate step of transforming TER diagrams to ER diagrams. Time is thus implicit in TER diagrams. The TER diagram describing the running example is shown in Figure 17. In the remainder of this subsection, we consider the cardinalities; the next subsection considers the intermediate step.

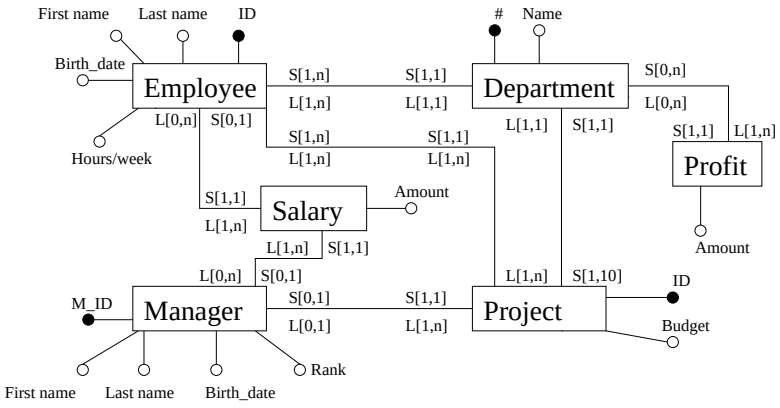

Figure 17: A TER Diagram of The Running Example

The modeling of time-varying information is improved in TER by replacing the traditional cardinality constraints by two new types of constraints, the lifetime cardinality, denoted by L[*minL,maxL*], and the snapshot cardinality, denoted by S[*minS,maxS*]. For an example, consider the relationship type between entity types Department and Project in Figure 17. Relationship types have two directions,

with each direction having a source and a target. In TER diagrams, the cardinality constraints are with respect to a direction of a relationship type, and they are placed next to the target entity type, by the relationship type.

A *lifetime cardinality* constraint L[*minL,maxL*] states that the minimum and maximum number of instances of the target entity related to one instance of the source entity over all of time is *minL* and *maxL*, respectively. Similarly, a *snapshot cardinality* constraint S[*minS,maxS*] states that the minimum and maximum number of instances of the target entity related to one instance of the source entity at any single point in time is *minS* and *maxS*, respectively. Below, the conditions that define any valid combination of cardinalities for any given relationship direction in TER are defined.

$$0 < maxS \text{ and } 0 < maxL$$
$$0 \leq minS \leq maxS \leq maxL$$
$$0 \leq minS \leq minL \leq maxL$$

In the relationship type between Department and Project, a Department instance (a "department") may have from 1 to n associated Project instances ("projects") during its lifetime, but it may have at most 10 associated projects at any single point in time. A project is associated with precisely one department at any single point in time; and a project is associated with precisely one department throughout its lifetime. Thus, projects cannot be reassigned from one department to another.

As it is the case for cardinality constraints in the ER model, cardinality constraints in the TER model can also express *connectivity*. Thus, a set of connective values of relationship type directions are defined as follows.

one	for ($maxS =$) $maxL = 1$
oneT	for $maxS = 1$ and $maxL > 1$
many	for $maxS > 1$

The introduction of the new connective value *oneT* ("one at a time") leads to a refined classification of relationship types. Traditionally, there are three distinct and exhaustive classes of relationship types: *one-to-one*, *one-to-many*, and *many-to-many*. While still disjoint, these classes are no longer exhaustive when snapshot and lifetime cardinality constraints are used, as the classes no longer cover all valid combinations of values for *minS* and *minL* in both directions. Therefore, three new relationship classes are added to the before mentioned three, namely *one-to-oneT*, *oneT-to-oneT*, and *oneT-to-many*.

Up until now, the *optionality* of a relationship-type direction has been implicit. It has been assumed that if $minS = 0$ in a direction, this implies that participation is optional in that direction. But given the definitions of snapshot and lifetime cardinalities, the notion of optionality can be refined. A relationship-type direction is *snapshot optional* (*optS*) if $minS = 0$; otherwise, it is *snapshot mandatory* (*mandS*). A relationship-type direction is *lifetime optional* (*optL*) if $minL = 0$; otherwise, it is *lifetime mandatory* (*mandL*). The refinement implies that columns in the relational tables, that result from snapshot mandatory directions of relationship types are not allowed to have null values. The following holds for the refined optionalities:

optL	implies	*optS*
mandS	implies	*mandL*
mandS and *optL*	are incompatible	

In TER diagrams such as that in Figure 17, the entity types do not include attributes that make is possible to distinguish different states of entities. For example, there are no means of recording different states of Employee entities. These means are implicit, and they are brought out by the mapping of TER diagrams to ER diagrams, as described next.

2.8.2 Mapping TER Diagrams to ER Diagrams

One consequence of introducing the temporal aspects of relationships into TER is that there now exists a basis for the semi-automatic incorporation of time-varying data in relational tables. How applications

are to deal with time-varying data largely depends on the volume of such data, the frequency of access to it, etc. TER provides three general approaches of dealing with time-varying data. They are based on the frequency of access to non-current data.

Never If there is no interest in the non-current data, there is no reason for storing it. No provisions for retaining non-current data are needed; old data is simply overwritten by new.

Occasional If the non-current data is accessed infrequently, it would be rather inefficient to store it together with the much more frequently accessed current data. Thus, separating the current data from the non-current data at the conceptual level simplifies the design process.

Frequent If the non-current data is anticipated to be accessed almost as frequently as the current data, it is most efficient to store them together.

TER then provides three different algorithms for translating TER diagrams into ER diagrams, one for each category. Figure 18 shows the result of using the algorithms on a fraction of the running example.

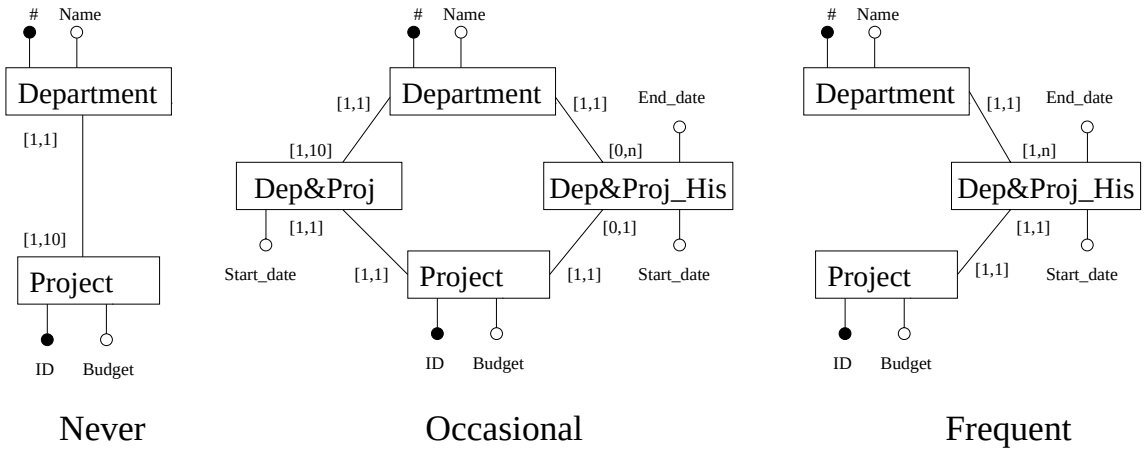

Figure 18: Mappings of TER Diagrams to ER Diagrams

The mappings only provide means of recording multiple states of time-varying TER relationship types; while not documented, it should be straightforward to extend them to also provide means of recording multiple states of time-varying entities and attributes. Note how lifetime and snapshot constraints are replaced with appropriate regular cardinality constraints.

2.8.3 Summary

The TER model provides means for better time-varying data modeling. Specifically, ordinary cardinality constraints are replaced with snapshot and lifetime cardinality constraints. Using these, TER redefines the classification of relationships and the notion optionality. Specifically, a new, *one*T cardinality is introduced. Time is implicit in TER diagrams, but the temporal aspects are made explicit through the mapping of TER diagrams to ER diagrams.

2.9 The TempEER model

The motivation behind TempEER[3] [20] is to be able to capture temporal information in a conceptual model (specifically, the EER model) and then, via an appropriate mapping, in the relational data model.

In achieving this, TempEER does not add new syntactical constructs to the EER model, but assumes a temporal dimension to the existing EER constructs. The mapping to the relational-model

[3]The authors gave their model the same name as the TEER already proposed by Elmasri et al. We adopt the name "TempEER."

level, adds two attributes, ValidTime and TransTime, to all the relation schemas that a conventional mapping algorithm yields. It is an underlying assumption that the TempEER model is a design model only and that the implementation platform is relational.

2.9.1 The Representation of Time

TempEER captures both valid and transaction time, both of which are assumed to have discrete domains, and different granularities may be specified for both of these domains. Time intervals are used as valid-time values, and time instants are used as transaction-time values.

Valid-time intervals are a subset of $[0, \mathit{UNTIL}]$, with *UNTIL* being a time value greater than or equal to the current time. Thus, the time domain for valid times extends beyond that used in the TEER model (Section 2.5). Transaction times never exceed the current time.

2.9.2 The Model Components

The TempEER model does not add any new syntactical constructs to the EER model; rather, the temporal aspects are implicit in TempEER diagrams. The TempEER diagram of the running example is therefore identical to that of Figure 10.

Entities and Entity Types

In TempEER diagrams, each entity of an entity type is associated with a *lifespan* capturing the valid time of the entity. The lifespan can be a time interval or a temporal element.

When mapped to a relational platform, an entity is represented by a set of tuples where each tuple describes one state of the entity over time. An entity type is mapped to a relation schema with the attributes dictated by a standard mapping and with an interval-valued ValidTime attribute. Thus, any change to an attribute of an entity results in the creation of new tuple capturing the new state of the entity. The lifespan of an entity is then the union of the ValidTime intervals in the set of tuples that represent the entity. In addition to the ValidTime attribute, each tuple has a TransTime attribute that records the insertion time of the tuple, making it possible to capture the transaction time of each tuple.

To exemplify, let us reconsider the entity described by the example given in Figure 11. This entity has lifespan $T = [7/1/90, \mathit{UNTIL}]$ and is represented by the following two tuples at the relational level.

VT	ID	First name	Last name	Birth_date	Salary	TT
7/1/90,6/30/92	98765	Chris	Johnson	8/23/46	$ 20K	6/15/90
7/1/92,¡em UNTIL	98765	Chris	Johnson	8/23/46	$ 30K	6/30/92

Figure 19: The Relational Representation of an Employee Entity

The temporal information of weak entity types is stored exactly as for ordinary entity types. The constraint that the lifespan of a weak entity must be a subset of the lifespan of its owner entity is enforced (the interaction with transaction time is not considered).

Attributes

The attribute types of TempEER are those of the EER model, with the exception that their changing values over time are retained.

A single-valued attribute has one atomic value for any point in time; multivalued attributes can have more that one value at a given point in time; and the value of a composite attribute at a given point in time is the concatenation of the values of its components at that point in time.

The valid time associated with an attribute value can be deduced from the tuples at the relational level representing the entity. For example, the temporal element associated with the attribute value Johnson of the above entity is $[7/1/90, \mathit{UNTIL}]$, whereas the temporal element associated with the value 20K is $[7/1/90, 6/30/92]$. The temporal element of an attribute value of an entity must be a subset of the lifespan of the entity.

Relationships Types

Each relationship instance of a relationship type is associated with a lifespan defined in the same way as for entities. The lifespan of a relationship instance must be a subset of the intersection of the lifespans of the participating entities.

Finally, TempEER also has superclass/subclass relationships. The lifespan of a subclass entity must be a subset of the lifespan of its superclass entity.

2.9.3 Summary

The sparsely documented TempEER model does not add any new syntactical constructs to the EER model, but instead changes the meaning of the existing constructs. TempEER diagrams are mapped to tuple-timestamped bitemporal relation schemas. Temporal constraints are introduced.

2.10 The TempRT Model

In a working paper, Kraft [19] proposes TempRT[4] that incorporates valid time support into a binary ER model. To motivate his approach, he first considers capturing valid time using explicit timestamp attributes, which is unattractive.

In his approach valid time is captured through temporal relationships, temporal entities, and temporal attributes. The basic temporal construct is the temporal relationship type. While ER diagrams are usually translated to relational schemas, in this model there is an extensional level with is own graphical notation associated with the ER diagrams. In this notation, nodes represent the instances of entities and the edges represent relations between instances.

The valid time domain employed is discrete, but is not otherwise described.

2.10.1 The Model Components

The model is based on a binary ER model, and Figure 20 exemplifies the notation. In this model only entity types, described by rectangles, and relationship types, described by "crows' feet," may be specified. The attributes in Figure 20 are actually shorthand for one-to-many relationship types between an entity type with all possible values of some value domain as instances and the entity type having the attribute. Two diagonal lines are used to indicate that a construct is temporal. For example, the relationship type between Employee and Emp_Proj is marked as temporal. The temporal markings of Employee and Emp_Proj are deviations from the running example.

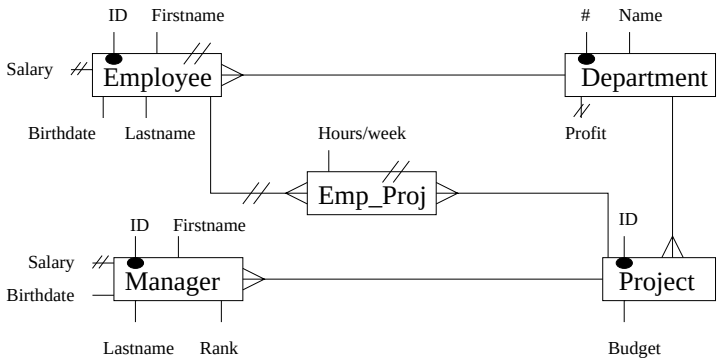

Figure 20: TempRT Diagram Modeling the Running Example

Temporal Relationship Types

The basic temporal structure is the temporal relationship. The semantics of a temporal relationship is an extension of the semantics of an ordinary relationship.

[4]The author has not given the model a name. To clearly identify the model, we adopt the name "TempRT."

In Figure 21(a), on the left hand side, the non-temporal relationship between Employee and Department is repeated, and on the right hand side, some instances are shown. The meaning of the relationship is that every instance of Employee must at any point in time be related to one and only one instance of Department, and every instance of Department may be related to zero or more instances of Employee. Only one (the current) department assignment of an employee is recorded. Thus, if an employee is reassigned to a new department, the previous assignment is lost.

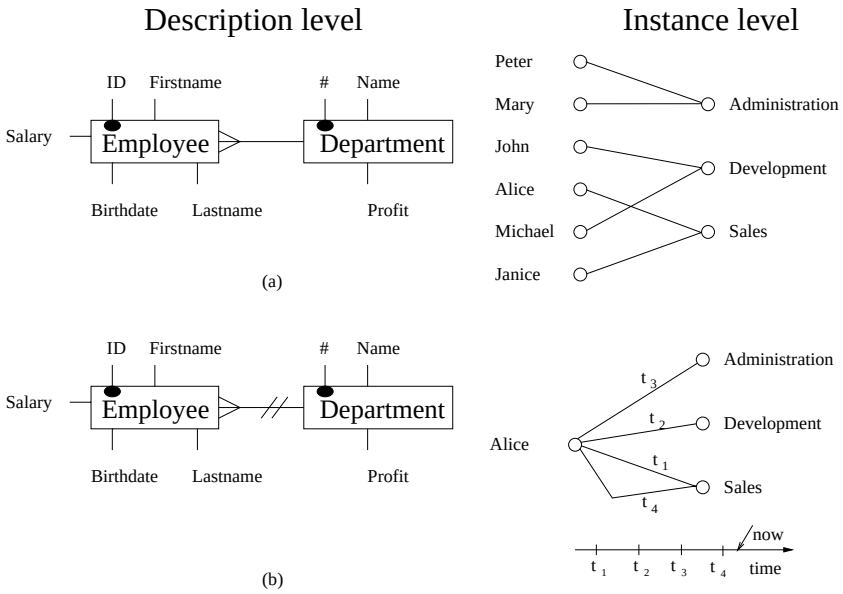

Figure 21: Temporal Relationships

In Figure 21(b), the relationship type is considered to be temporal. The semantics of the temporal relationship type is almost the same as for the non-temporal relationship type. Every Employee instance still has to be related to one and only one instance of Department at any point in time. The difference is that temporal relationships are timestamped and retained. As an illustration of this, the right hand side of Figure 21(b) gives the employment history of Alice. At time t_1, Alice becomes associated with Sales, and at time t_2 she is associated with Development. Then at t_3 she is attached to Administration, and lastly, at time t_4 she returns to Sales. The union of all the timestamps of a temporal relationship between two instances describes the lifespan of the relationship.

Temporal Entity Types and Attributes

Entity types do not have to be temporal. A non-temporal entity that participates in a temporal relationship cannot ever be changed or deleted. If this consequence is unwanted, the concept of lifespans has to be added to the instances, making them temporal.

The lifespan of an instance is modeled through temporal relationships. Specifically, a universal entity type U with only one instance is introduced. This entity type is connected, using a temporal relationship type, with the entity type we want to be temporal. Figure 22(a) illustrates this. The time in which an Employee instance references the U instance gives the lifespan of the Employee instance. Figure 22(b) shows the shorthand used in the model.

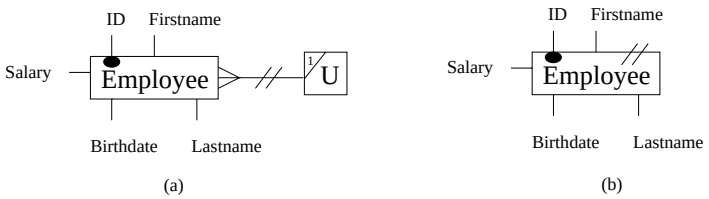

Figure 22: Temporal Entity Types

Temporal attributes are also defined using temporal relationships. As mentioned, a (non-time-varying) attribute is a shorthand for a regular many-to-one relationship between an entity with all possible values of some value domain as instances and the entity having the attribute. In order to make an attribute temporal, the ordinary relationship between the entity having the attribute and the entity modeling the value domain is replaced by a temporal relationship.

2.10.2 Summary

The TempRT model makes it possible to specify temporal relationships, temporal entities, and temporal attributes. The temporal entities and attributes are defined by temporal relationships.

3 Design Criteria and Evaluation of the Models

In Sections 2.2 to 2.10, we described all temporal ER models known to us. It is a common characteristic that few or no specific requirements to the models were given by their designers. To compare and better understand the models, this section defines a comprehensive set of design criteria for temporal ER models and evaluates the models against these criteria. We have chosen to also evaluate the EER model against the criteria. When doing so, the model will be treated as a temporal model, capturing time through timestamp attributes.

We have identified a total of 19 design criteria covering time semantics, model semantics, temporal functionality, and user-friendliness. The criteria are numbered C1 through C19. With each criterion defined, we indicate its source, if possible. We have attempted to only include criteria that have an objective basis for being evaluated. Together, the criteria identify important aspects of designing a temporal ER model. The possible outcomes of an evaluation of a model with respect to each criterion will be stated explicit together with the definition of each criterion, unless the possible outcomes are **N.A.**, **Yes**, and **No**.

Figures 23, 24, and 25 present the results of the evaluations of evaluating the models with respect to criteria C1–C3, C4–C13, and C14–C19, respectively.

	C1	C2	C3
EER	UDT	none	N.A.
RAKE	UDT, VT	temporal relationships types and attributes	Optional
TERM	UDT, VT	history structures and patterns	Optional
MOTAR	UDT, VT	temporal relationship types and attributes	Optional
TEER	UDT, VT	temporal entity types, attributes, and relationship types	Mandatory
STEER	UDT, VT	conceptual entity types, entity roles, temporal attributes, and conceptual and temporal relationship types	Mandatory
ERT	UDT, VT	timestamped entity classes, relationship types, complex entity classes, and temporal constraints	Optional
TER	UDT, VT	snapshot cardinalities, lifetime cardinalities, and OneT connective	Mandatory
TempEER	UDT, VT, TT	temporal entity types, attributes, and relationship types	Mandatory
TempRT	UDT, VT	temporal relationship types, entity types, and attributes	Optional

Figure 23: Evaluation of Criteria C1, C2, and C3

C1—Time Dimensions with built-in Support Valid and transaction time are general—rather than application specific—aspects of all database facts. As such, they are prime candidates for being built into a temporal ER model that is to be used for both analysis and database design. Being orthogonal and independent aspects of database facts, it is possible to support the two times independently. Support for these times may take different shapes and may be more or less elaborate. Another kind of time exists, namely the so-called *user-defined time* (UDT). This refers to "support" for temporal aspects with no built-in support in the model. User-defined times are supported when time-valued attributes are available. These are then employed for giving temporal semantics—not captured in data model, but only externally, by the database designer—to the the ER diagrams. We will say that a

time is supported simply if some support has been documented. The possible outcomes of evaluating a model against this criterion are **UDT**, **VT**, **TT**, and **N.A.** (and combinations of **VT** and **TT**).

For a model to be considered temporal, at least one time dimension must be supported. Almost all the models support *valid time*. The only model that does not is the EER model that only supports user-defined time. All the models support user-defined time. *Transaction time* is supported by only TempEER. That is, the valid-time aspect of a database application seems to be regarded as the most interesting aspect to support, thereby aiming at high-fidelity modeling of the mini-world.

C2—New Temporal Constructs Two general approaches to providing temporal support exist. With *implicit* temporal support, explicit timestamp attributes are "hidden" in the temporal semantics of the modeling constructs. For example, no timestamp attributes are necessary on a temporal relationship type to indicate that the instances of the type record their variation over time. With this approach, it is possible to obtain a temporal ER model without adding any new syntactical constructs to the ER model. Rather, the existing ER constructs are simply made temporal by changing their semantics. For example, ordinary relationship types are given temporal semantics, making their instances record variation over time, rather than just single states. It is also possible with this approach to retain the existing ER constructs and their semantics and add new temporal constructs to capture the time-varying information. The new notation for a temporal relationship type in MOTAR is an example. The extent of the changes made to the ER model may range from minor changes to a total redefinition of the model.

With *explicit* temporal support, the semantics of the existing ER constructs are retained. With this approach, timestamp attributes are explicit. Any new modeling constructs are notational shorthands introduced to make the modeling of temporal aspects more convenient.

Nearly all the models have added new temporal constructs. Some of the models have changed the semantics of the ordinary ER model constructs entirely. These models are TEER and TempEER. Other models have retained the old ER constructs and have added new temporal constructs. These models include TERM, MOTAR, ERT, TER and TempRT. RAKE does not add any new constructs to the ER model; instead, it introduces notational shorthands for certain patterns made up of ordinary ER constructs. However, we will consider these notational shorthands to be temporal constructs. One model has both changed the semantics of the ER constructs and added new temporal constructs, namely STEER. The specific names of the added constructs can be seen in the third column of Figure 23 (they are mentioned in the order in which they are introduced in this paper). The EER model has not added any new constructs—it captures time solely through timestamp attributes.

C3—Mandatory vs. Optional Use of Temporal Constructs The extent of changes made to the notation of the ER model determines whether the use of the temporal constructs added to the model are mandatory or optional. If all the original ER modeling constructs have simply been made temporal, the original constructs are no longer available. Mandatory use of the temporal constructs means that the designer cannot use non-temporal constructs in diagrams. Optional use of the temporal constructs provides the designer with the possibility of mixing temporal and non-temporal constructs in the same diagram. The possible outcomes of evaluating the models against this criterion are **N.A.**, **Mandatory** and **Optional**.

The models with mandatory use of the temporal constructs are TEER, STEER, TempEER and TER. TEER and TempEER have changed the semantics of all the original ER model constructs to be temporal. STEER has—besides making the original ER constructs temporal—added new temporal constructs to the model. Since TER has replaced the ordinary cardinality constraints with two new ones, and it is mandatory to specify the constraints, it becomes mandatory to use the temporal constructs, even if the users later decide to only record a single state of data.

The models that have retained the ordinary ER constructs and have added new temporal construct have optional use of temporal constructs. Thus, it is possible to mix temporal and non-temporal constructs in these models that include MOTAR, ERT, and TempRT. TERM has optional use of history structures and history patterns since all attributes (inclusive the existence and roles attributes) can be declared as constant. RAKE also has optional use of the temporal constructs since these are

notational shorthands for patterns made up of ordinary ER constructs. Since the EER model has not added new constructs, N.A. is the result of evaluating the model against this criterion.

	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13
EER	N.A.	N.A.	No	No	N.A.	No	No	No	N.A.	Yes
RAKE	instant, interval	N.A.	No	No	No	No	No	No	(Yes)	Yes
TERM	instant, interval	N.A.	Yes	Yes	Yes	Yes	Yes	Yes	No	No
MOTAR	instant	N.A.	Yes	No	Yes	No	No	No	(Yes)	Yes
TEER	temporal element	N.A	No	No	No	Yes	Yes	Yes	No	No
STEER	temporal element	N.A.	No	No	No	(Yes)	(Yes)	Yes	No	No
ERT	interval	N.A.	No	No	Yes	Yes	Yes	Yes	(Yes)	Yes
TER	instant	N.A.	No	No	No	No	No	No	(Yes)	No
TempEER	interval	instant	No	No	No	Yes	Yes	Yes	No	No
TempRT	interval	N.A.	No	No	No	(Yes)	(Yes)	No	(Yes)	Yes

Figure 24: Evaluation of Criteria C4—C13

C4—Data Types Supported for Valid Time Different data types for implicit or explicit timestamps may be used for indicating the valid time of an object, e.g., an attribute value or a relationship. Possible time data types include instants, intervals, and temporal elements. For example, one option is to associate valid-time intervals with attribute values of entities. Another option is to timestamp attribute values with valid-time elements, finite unions of intervals. An attribute value may also be defined as a function from a time domain to a value domain. In this way, an attribute may associate a value with a set of time instants. We will consider this element timestamping, and we will also consider the timestamping with sets of instants and intervals as being element timestamping. Since all models surveyed adopt a discrete model of time, we will not distinguish between support for closed versus open or half-open intervals.

A data model may provide the database designer with a choice of data types. This may increase the utility of the model. Possible outcomes include **N.A.**, **instant**, **interval**, and **temporal element**. All three data types mentioned may encode validity for durations, and the instant data type may also encode validity for single instants of time. In the former case, instants have associated interpolation functions (see Criterion C8). The impact of which data types are available is dependent on whether the model under consideration is used solely as a design model or is also used as an implementation model, complete with database instances and a query language.

The models that timestamp with instants include RAKE (events), TERM, MOTAR, and TER. The models RAKE, TERM, ERT, TempEER, and TempRT timestamp with intervals. The models TEER and STEER timestamp with temporal elements. Finally, this criterion is not applicable to the EER model, since it does not support valid time.

C5—Data Types Supported for Transaction Time As valid and transaction time have different semantics, the timestamp types available for the two times may differ. The possible outcomes are as for valid time. TempEER is the only model that supports transaction time. The timestamp used for transaction time in TempEER is instants (that encode durations). N.A. is indicated in the figure for all the other models.

C6—Support for Multiple Granularities It may be that the temporal variability of different objects in the mini-world are captured using times of different granularities [36, 37, 6]. They should then also be captured in the database using these different granularities. For example, the granularity of a minute may be used when recording the actual working hours of employees, while the granularity of a day may be used when recording the department assignments of employees. Notice that this criterion relates to valid time.

There are two models in which it is possible, at the conceptual level, to specify the granularity of the timestamps. In MOTAR, the user is allowed to specify the frequency of the recording of periodic attributes. In TERM, atomic histories can have different time domains. The rest of the models only

briefly state that the granularity of the timestamps should be suitable for specific applications and hence postpone the choice of granularity to the logical design phase.

C7—Support for Temporal (Im-) Precision The temporal variability of different objects in the mini-world may be known with different precisions [17, 4, 7, 5]. Although some imprecision may be captured using multiple granularities, granularities are not a general solution. For example, the variability of an attribute may be recorded using timestamps with the granularity of a second, but the varying values may only be known to the precision of ± 5 seconds of the recorded value. This phenomenon may be prevalent and important to capture in scientific and monitoring applications that store measurements made by instruments.

The only model which support temporal precision is TERM, where it is possible to specify precision on the timestamps (and also the values of attributes).

C8—Temporal Interpolation Functions Temporal interpolation functions derive information about times for which no data is explicitly stored in the database (see, e.g., [17] and [16, pp. 35–40]). For example, it is possible to record times when new salaries of employees take effect and then define an interpolation function (using so-called step-wise constant interpolation) that gives the salaries of employees at any time during their employment. In the scientific domain, interpolation is particularly important, e.g., when variables are sampled at different rates.

User-definable temporal interpolation functions are supported by TERM, MOTAR, and ERT. In TERM, functions handle both incomplete and not-explicitly-stored data, while the derivation functions in ERT only handle data not explicitly stored. In MOTAR, rules can be considered as some sort of derivation functions. The other models do not consider how to handle incomplete and not-explicitly-stored data.

C9—Lifespans of Entities The lifespan of an entity is the time over which the entity exists in the mini-world. Entities may exist beyond the times when their attributes have (non-null) values, making it impossible to infer lifespans from the assignments of timestamps to attribute values. If the concept of lifespan of entities is supported, this means that the model has built-in support for capturing the times when entities exist.

Four models support the concept of lifespan for all entity types, namely TERM, TEER, ERT, and TempEER. The lifespans for the entity types with constant existence in TERM and the lifespans for non-timestamped entity types in ERT are given implicitly as the lifespan of the database. Some models support lifespans for a subset of the entity types—for these models, we enclose the **Yes** in parentheses. These models are STEER, which supports lifespans for entity roles, and TempRT, which supports lifespans for temporal entity types. The models that do not support lifespans of entity types include RAKE, MOTAR, TER, and EER.

C10—Lifespans of Relationships The concept of lifespan is also applicable to relationships, with the same meaning as for entities. When a model provides a built-in notion of relationship lifespans, it may also enforce certain temporal constraints that involve these lifespans. For example, it does not make sense for an entity to have an attribute value at a time when the entity does not exist.

The models that support lifespans for all relationship types include TERM, TEER, ERT, and TempEER. STEER and TempRT support the concept of lifespan for a subset of the relationship types, namely the temporal relationship types—for these, we enclose the **Yes** in parentheses. The models that do not support lifespans of Relationship types include RAKE, MOTAR, TER, and EER.

C11—Temporal Constraints A temporal data model may include built-in temporal constraints and facilities for user-definable temporal constraints. If built-in temporal constraints are not present, then the possibility of having illegal data is present. For example, a (binary) relationship between two entities can usually not exist if the entities do not exist. The presence of an appropriate set of (built-in) constraints on the built-in temporal constructs is thus of essence. Next, it should be possible for the database designer to specify additional temporal constraints. For example, we have seen that

the TER model (Section 2.8) supports two types of temporal constraints on relationship types, namely snapshot and lifetime cardinality constraints.

Temporal constraints are supported by TERM, TEER, ERT, STEER, TER, and TempEER, while the models RAKE, MOTAR, TempRT, and EER do not support temporal constraints.

C12—User-specifiable Temporal Support A temporal ER model offers user-specifiable temporal support if it is up to the database designer to decide which temporal aspects of data to capture. For example, a temporal ER model may provide built-in support for both valid and transaction time, but a specific application may only require support for transaction time. It should then be possible to omit support for valid time.

The models RAKE, MOTAR, ERT, and TempRT partly satisfy this criterion. In RAKE, MOTAR, ERT, and TempRT the temporal support is valid time, but only if the database designer uses the temporal constructs of the models. So does TER, but not at the conceptual level—only when translating the TER diagram to ER diagrams. If the designer wants to record the variations of data, the temporal dimension supported is valid time; and if no access is wanted, no temporal support is given. This criterion is not applicable to the EER model which supports only user-defined-time. The remaining models have enforced temporal support.

C13—Upward Compatibility A temporal ER model is upward compatible with respect to the conventional ER model if any legal conventional ER diagram is also a legal ER diagram in the temporal model and if the meanings of the diagram in the two models is the same. Upward compatibility is very important because it enables legacy ER diagrams to be used immediately in the new temporal model. Investments in legacy systems are protected, and a basis for a smooth transition to a temporally enhanced ER model is provided [30].

When evaluating a model against this criterion, we will evaluate whether the model is upward compatible with respect to the ER model that it extends, if specified; otherwise, we will use Chen's ER model [2] for models without superclass/subclass relationships and the EER model [10] for models with superclass/subclass relationships.

Five models—RAKE, MOTAR, ERT, and TempRT—are upward compatible with respect to their basic models. In these models, no syntactical constructs from the basic models have been given new semantics. The EER is also upward compatible with itself; this holds trivially true. TERM is not upward compatible since its existence attributes have to be specified for all entity and relationship types. TEER and TempEER are not upward compatible with respect to the EER model because these models have changed the semantics of the existing EER modeling constructs. STEER has both changed the semantics of the original model and added new syntactical constructs. Due to the change of semantics of the original model, STEER is not upward compatible with the EER model. TER is not upward compatible with the ER model since it has replaced the ordinary cardinality constraints with the snapshot and lifetime cardinality constraints.

	C14	C15	C16	C17	C18	C19
ERR	N.A.	Yes	Relational model	Yes	Yes	Yes
RAKE	Yes	?	ER model Relational model	No	Yes	No
TERM	Yes	?	Relational model	No	No	No
MOTAR	Yes	?	Relational model	No	Yes	No
TEER	Yes	?	None	Yes	Yes	(Yes)
STEER	Yes	N.A.	None	Yes	Yes	No
ERT	N.A	?	Relational model	Yes	Yes	Yes
TER	N.A	Yes	ER model	No	Yes	Yes
TempEER	Yes	?	Relational model	Yes	Yes	(Yes)
TempRT	Yes	Yes	None	No	Yes	No

Figure 25: Evaluation of Criteria C14–C20

C14—Snapshot Reducibility of Attribute Types Temporal ER models that add temporal support implicitly may include temporal counterparts of the ordinary attribute types, i.e., provide temporal single valued, temporal multi-valued, temporal composite, and temporal derived attribute types. These temporal attribute types may be snapshot reducible [26] with respect to their corresponding snapshot attribute types. This occurs if snapshots of the databases described using the temporal ER diagram constructs are the same as databases described by the corresponding snapshot ER diagram where all temporal constructs are replaced by their snapshot counterparts. For example, a temporal single-valued attribute is snapshot reducible to a snapshot single-valued attribute if the temporal single-valued attribute is single valued at each point in time.

Generalizing snapshot constructs this way yields a natural temporal model that is easily understood in terms of the conventional ER model.

The models that have snapshot reducible attribute types are RAKE, TERM, MOTAR, TEER, STEER, and TempEER. RAKE has only single-valued attribute types. These are snapshot reducible since the temporal attributes are modeled through relationships treated as weak entity types owned by time-period entity types, thereby having ENDstamp as part of the key. This structure cannot have more than one value at any point in time. TERM has only single-valued attributes, and all variable attributes have a atomic history structure to ensure that the attribute only have one value at a time. The temporal attributes of MOTAR are also snapshot reducible since the mapping algorithm ensures that timestamps are made part of the key in the relations representing the attributes. TEER, STEER, TempEER all have temporal single-valued, multivalued, and composite attribute types. The models also have the mutually same semantics for these attribute types, and the semantics state that they are snapshot reducible. TempRT only has single-valued attribute types, and since the temporal attributes of this model are defined using the temporal relationship that is snapshot reducible, see the next criterion, the temporal attributes must be snapshot reducible. Because ERT, TER, and EER do not have temporal attributes, this concept is inapplicable to these models.

C15—Snapshot Reducibility of Relationship Constraints Snapshot reducibility also applies to the various constraints that may be defined on relationship types, including specialized relationship types such as ISA (superclass/subclass) and PART-OF (composite/component) relationships. For example, the temporal cardinality constraint 1–N on a binary temporal relationship type is snapshot reducible to the snapshot cardinality constraint 1–N on a binary snapshot relationship type if at any single point in time, the 1–N snapshot constraint applies to the possible instances of the temporal relationship type.

Only three models have snapshot reducible relationship constraints: TER, by virtue of the semantics of the snapshot cardinality constraint; TempRT, due to the semantics given to its temporal relationships (these semantics explicit states that the cardinality constraints given by the relationship should hold at any point in time); and EER, trivially, because it does not propose any additional types of relationships and constraints. The models RAKE, TERM, MOTAR, TEER, ERT, and TempEER do not describe what the meaning of the different relationship constraints that can be specified are in a temporal database. This is the reason for the question marks in Column C15 of Figure 25. The STEER model does nor include cardinality constraints on relationship types, making this criterion inapplicable.

C16—Mapping Algorithms Available A mapping algorithm translates an ER diagram in a temporal ER model into a corresponding database schema in another data model. The temporal ER models are typically considered to be design models. Upon designing an ER diagram, the diagram is mapped to a schema of an available DBMS, i.e., is mapped to an implementation platform. The most popular mappings are to the relational model (or, the platform of a specific relational product). It is also possible to map temporal ER diagrams to conventional ER diagrams. Then mappings from the conventional ER model may be exploited.

Most of the models provide mapping algorithms into regular ER diagrams or the relational model. The only model that can be mapped into both *ER diagrams* and the *relational model* is RAKE. The TER model provide an algorithm that transforms TER diagrams into ER diagrams, which can be transformed into a relational schema by a standard algorithm. The models that provide a algorithm for

translation into a relational schema include TERM, MOTAR, ERT, and TempEER. Models TEER, STEER, and TempRT do not specify any mappings of their diagrams. One good reason for an absence of mappings is that the models themselves may be considered implementation models, see the discussion of the next criterion.

C17—Query Language Provided As an alternative to mapping ER diagrams to the schema of a separate implementation platform, another approach is to assume a system that implements the ER model directly. With this approach a mapping to an implementation platform is not required. Instead, a query language should be available for querying ER databases.

No query languages is provided for the following models: RAKE, TERM, MOTAR, TER, and TempRT. A temporal extension of the query language GORDAS is provided as query language for the models TEER and STEER. The ERT model is provides with a query language called the External Rule Language. As query language for the TempEER model, a temporal extension of SQL is proposed.

C18—Graphical Notation Provided While it is usually assumed that a graphical notation is available for describing ER diagrams, this needs not be so. It is also possible to provide only a textual notation for describing ER diagrams. It is generally believed that ER models with a graphical notation have an advantage over ER models with a programming language-like notation. Graphical notations tend to be easier to learn, and we believe that the simplicity of the graphical ER notation is one of the main reasons for its success.

The only model that does not have a graphical notation is the TERM model, which has a Pascal-like syntax.

C19—Graphical Editor Available If the notation of a model is graphical, then the presence of an editor supporting the model is very important if the model is to be used widely. Potential users should have the opportunity to try and use at least some prototype of an editor supporting the model.

Two models, namely TER and ERT, come with an editor to support their use. The editor for TER is called MODELLER [31] and is a commercially available product; and the editor for ERT is called the ERT-TOOL. Models TEER and TempEER can use editors that supports the EER model for schema design, but not for mapping to their implementation models. Thus, a **Yes** in parentheses has been indicated for these models. Editors for EER exist in the public domain. No other model is accompanied by an editor.

4 Conclusion and Future Work

This paper has surveyed nine proposals for extending the ER model to better capture the temporal aspects of data. Although the detailed motivation for the development of each proposal varies, it is a general observation that while temporal aspects of data are prevalent, the ER (and EER) model does not provide adequate support for elegantly and concisely capturing these aspects.

The survey has emphasized the common aspect of the temporal ER models, namely their use as design models that are employed to capture, in a conceptual model, a database design that is implemented in a separate data model, typically the relational model. This yields a focus on structural aspects, rather than on ER query languages.

The proposed extensions are based on quite different approaches. One approach is to devise new notational shorthands that replace some of the patterns that occur frequently in ER diagrams when temporal aspects are being modeled. One example is the pattern that occurs when modeling a time-varying attribute in the ER model. Another approach is to change the semantics of the existing ER model constructs, making them temporal. In its extreme form, this approach does not result in any new syntactical constructs—all the original constructs have simply become temporal. With this approach, it is also possible to add new constructs. Yet another approach is to retain the existing ER constructs.

Many of the models assume that their schemas are mapped to schemas in the relational model that serves as the implementation platform. The algorithms that map the schemas of these models to the relational model are constructed to add appropriate time-valued attributes to the relation schemas.

This corresponds to how the ER model is (currently) used in industry. In contrast, three of the models we have examined are themselves implementation models and provide a query language for the model.

We have identifies a total of 19 design properties that are relevant for the evaluation of temporal ER models and should be taken into consideration when designing a temporal ER model. None of the 19 design properties we have presented this paper are incompatible—they can all be fulfilled at the same time. We have evaluated the nine models against the design properties discovered. While no single model satisfies all the properties, the models seem to collectively cover the design space. The models illustrate different ways in which the temporal aspects of data can be conveniently captured at the conceptual level, and it is our contention that all the temporal ER models, to varying degrees, have succeeded in more naturally capturing the temporal aspects of data than does the ER model.

In our work with the models, we have come to the conclusion that the approach where all existing ER model constructs are given a temporal semantics is attractive. The database designers are likely to be familiar with the existing ER constructs. So, upon understanding the principle at work when making these constructs temporal, the designers are ready to work with, and benefit from using, the temporal ER model.

However, this approach is not totally without problems. This approach rules out the possibility of designing non-temporal databases or databases where some part of a database is non-temporal and the rest is temporal. Another problem is that old diagrams are no longer are valid, i.e., while their syntax is valid, their semantics have changed, and they therefore no longer describe the existing relational database schemas.

The models that retain the existing constructs with their old semantics and introduce new temporal constructs have the opposite problems and advantages. It is likely to be more difficult for the database designers to learn and understand the new temporal model. The larger initial investment in training that this induces may prevent a model from being accepted in industry. On the other hand, the models taking this approach may not face the problem of the diagrams not describing the underlying relational database, since the semantics of the existing ER constructs are retained. This is important for industrial users with many legacy diagrams. it is also possible to design non-temporal databases as well as databases where some parts are non-temporal and others are temporal.

The ideal temporal ER model is easy to understand in terms of the ER model; does not invalidate legacy diagrams and database applications; and does not restrict database to be temporal, but rather permits the designer to mix temporal and non-temporal parts.

As stated, most of the models rely on another data model as an implementation model: their schemas are mapped to schemas in these other models, and it is these other schemas that are subsequently populated and queried.

The relational model is the implementation model of choice. Either, temporal ER diagrams are mapped to relation schemas directly, or they are mapped to regular ER diagrams which are then mapped to relation schemas. The time-valued attributes that result from mapping ER diagrams to relation schemas are not interpreted by the relational model,i.e., they have no built-in semantics in the relational model. As a result, queries on time-varying data are often hard to formulate in SQL [29].

None of the models have one of the many time-extended relational models proposed [27] as their implementation model. The temporal relational models have data-definition and query-language capabilities that better support the management of temporal data and would thus constitute natural candidate implementation platforms.

To summarize, since most DBMSs used in industry are relational, temporal ER models should ideally includes mappings to several implementation platforms: the relational model (as fleshed out in the different DBMS products), temporal relational models, and emerging models (e.g., SQL3). It is a challenge to design mappings that maximally exploit these and other candidate implementation platforms.

References

[1] S. Ceri, S. B. Navathe, and C. Batini. *Conceptual Database Design. An Entity-Relationship Approach.* The Benjamin/Cummings Publishing Company, Inc., California, 1992.

[2] P. P-S. Chen. The Entity-Relationship Model – Toward a Unified View of Data. *ACM Transaction on Database Systems*, 1(1):9–36, March 1976.

[3] C. J Date. *An Introduction to Database Systems*, Volume I of *Addison-Wesley Systems Programming Series*. Addison-Wesley Publishing Company, 5th edition, 1990.

[4] C. E. Dyreson and R. T. Snodgrass. Valid-time Indeterminacy. In *Proceedings of the 9th International Conference on Data Engineering*, pages 335–343, Vienna, Austria, 1993.

[5] C. E. Dyreson and R. T. Snodgrass. Temporal Granularity and Indeterminacy: Two Sides of the Same Coin. Technical Report TR 94-06, University of Arizona, Department of Computer Science, February 1994.

[6] C. E. Dyreson and R. T. Snodgrass. Temporal Granularity. In R. T. Snodgrass, editor, *The TSQL2 Temporal Query Language*, Chapter 19, pages 347–383. Kluwer Academic Press, 1995.

[7] C. E. Dyreson and R. T. Snodgrass. Temporal Indeterminacy. In R. T. Snodgrass, editor, *The TSQL2 Temporal Query Language*, Chapter 18, pages 327–346. Kluwer Academic Publishers, 1995.

[8] R. Elmasri, I. El-Assal, and V. Kouramajian. Semantics of Temporal Data in an Extended ER Model. In *9th International Conference on the Entity-Relationship Approach*, pages 239–254, Lausanne, Switzerland, October 1990.

[9] R. Elmasri and V. Kouramajian. A Temporal Query Language for a Conceptual Model. In N. R. Adam and B. K. Bhargava, editors, *Advanced database systems*, volume 759 of *Lecture Notes in Computer Science*, Chapter 9, pages 175–195. Berlin, Springer-verlag, 1993.

[10] R. Elmasri. and S. B. Navathe *Fundamentals of Database Systems*. The Benjamin/Cummings Publishing Company, INC, 2nd edition, 1994.

[11] R. Elmasri, G. Wuu, and V. Kouramajian. A Temporal Model and Query Language for EER Databases. In A. Tansel et al., editor, *Temporal Databases: Theory, Design, and Implementation*, Chapter 9, pages 212–229. Benjamin/Cummings Publishers, Database Systems and Applications series, 1993.

[12] R. Elmasri and G. T. J. Wuu. A Temporal Model and Query Language for ER databases. In *Proceedings of the Sixth International Conference on Data Engineering*, pages 76–83, 1990.

[13] S. Ferg. Modeling the Time Dimension in an Entity-Relationship Diagram. In *4th International Conference on the Entity-Relationship Approach*, pages 280–286, Silver Spring, MD, 1985. Computer Society Press.

[14] S. K. Gadia and C. S. Yeung. A Generalized Model for a Relational Temporal Database. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 251–259, Chicago, IL, June 1988.

[15] C. S. Jensen, J. Clifford, R. Elmasri, S. K. Gadia, P. Hayes, and S. Jajodia (eds). A Glossary of Temporal Database Concepts. *ACM SIGMOD Record*, 23(1):52–64, March 1994.

[16] C. S. Jensen and R. T. Snodgrass. Semantics of Time-varying Information. *Information Systems*, 21(4):311–352, 1996.

[17] M. R. Klopprogge. TERM: An Approach to Include the Time Dimension in the Entity-Relationship Model. In *Proceedings of the Second International Conference on the Entity Relationship Approach*, pages 477–512, Washington, DC, October 1981.

[18] M. R. Klopprogge and P. C. Lockeman. Modeling Information Preserving Databases: Consequences of the Concept of Time. In *Proceedings from the Ninth International Conference on Very Large Data Bases*, pages 399–416, October 1983.

[19] P. Kraft. Temporale Kvaliteter i ER Modeller. Hvordan? Working paper 93, The Aarhus School of Business, Department of Information Science, January 1996.

[20] V. S. Lai, J-P. Kuilboer, and J. L. Guynes. Temporal Databases: Model Design and Commercialization Prospects. *DATA BASE*, 25(3):6–18, 1994.

[21] P. McBrien, A. H. Seltveit, and B. Wangler. An Entity-Relationship Model Extended to Describe Historical Information. In *International Conference on Information Systems and Management of Data*, pages 244–260, Bangalore, India, July 1992.

[22] E. McKenzie and R. Snodgrass. An Evaluation of Relational Algebras Incorporating the Time Dimension in Databases. *ACM Computing Survey*, 23(4):501–543, December 1991.

[23] A. Narasimhalu. A Data Model for Object-Oriented Databases With Temporal Attributes and Relationships. Technical report, National University of Singapore, 1988.

[24] J. F. Roddick and J. D. Patrick. Temporal Semantics in Information Systems – a Survey. *Information Systems*, 17(3):249–267, October 1992.

[25] A. Silberschatz and H. F. Korth. *Database System Concepts*. McGraw-Hill International Editions, 2nd edition, 1991.

[26] R. T. Snodgrass. The Temporal Query Language TQUEL. *ACM Transaction on Database Systems*, 12(2):247–298, June 1987.

[27] R. T. Snodgrass. Temporal Databases. In A. U. Frank, I. Campari, and U. Formanti, editors, *Theories and Methods of Spatio-Temporal Reasoning in Geographic Space*, volume 639 of *Lecture Notes in Computer Science*, pages 22–64. Springer-Verlag, 1992.

[28] R. T. Snodgrass. Temporal object oriented databases: A critical comparison. In W. Kim, editor, *Modern Database Systems: The Object Model, Interoperability and Beyond*, chapter 19, pages 386–408. Addison-Wesley/ACM Press, 1995.

[29] R. T. Snodgrass, I. Ahn, G. Ariav, D. S. Batory, J. Clifford, C. E. Dyreson, R. Elmasri, F. Grandi, C. S. Jensen, W. Kafer, N. Kline, K. Kulkanri, T. Y. C. Leung, N. Lorentzos, J.F. Roddick, A. Segev, M. D. Soo, and S. M. Sripada. *The TSQL2 Temporal Query language*. Kluwer academic Publishers, 1995.

[30] R. T. Snodgrass, Böhlen M., C. S. Jensen, and A. Steiner. Change Proposal to SQL/Temporal: Adding Valid Time—Part A. *International Organization for Standardization*, 40 pages, December 1995. ANSI Expert's Contribution.

[31] B. Tauzovich. Toward Temporal Extensions to the Entity-Relationship Model. In *The 10th International Conference on the Entity Relationship Approach*, pages 163–179, San Mateo, California, October 1991. E/R Institute.

[32] T. J Teorey. *Database Modeling and Design*. Morgan Kaufmann Series in Data Management Systems. Morgan Kaufmann Publishers, Inc., San Mateo, Calefornia, 1990.

[33] C. Theodoulidis, B. Wangler, and P. Loucopoulos. The Entity Relationship Time Model. In *Conceptual Modelling, Databases, and CASE: An Integrated View of Information Systems Development*, Chapter 4, pages 87–115. John Wiley, 1992.

[34] C. I. Theodoulidis and P. Loucopoulos. The Time Dimension in Conceptual Modelling. *Information Systems*, 16(3):273–300, 1991.

[35] C. I. Theodoulidis, P. Loucopoulos, and B. Wangler. A Conceptual Modelling Formalism for Temporal Database Applications. *Information Systems*, 16(4):401–416, 1991.

[36] X. Wang, S. Jajodia, and W. Litwin. Dealing with Granularity of Time in Temporal Databases. In R. Anderson et al., editors, *Proceedings of the 3rd International Conference on Advanced Information Systems Engineering*. Lecture Notes in Computer Science, Vol. 498, Springer Verlag, 1991.

[37] G. Wiederhold, S. Jajodia, and W. Litwin. Dealing With Granularity of Time in Temporal Databases. In *Proc. 3rd Nordic Conf. on Advanced Information Systems Engineering*, Trondheim, Norway, May 1991.