32

Conceptual Modeling of Time-Varying Information

H. Gregersen and C. S. Jensen

A wide range of database applications manage information that varies over time. Many of the underlying database schemas of these were designed using one of the several versions, with varying syntax and semantics, of the Entity-Relationship (ER) model. In the research community as well as in industry, it is common knowledge that the temporal aspects of the mini-world are pervasive and important, but are also difficult to capture using the ER model. Not surprisingly, several enhancements to the ER model have been proposed in an attempt to more naturally and elegantly support the modeling of temporal aspects of information. Common to the existing temporally extended ER models, few or no specific requirements to the models were given by their designers.

With the existing proposals, an ontological foundation, and novel requirements as its basis, this paper formally defines a graphical, temporally extended ER model. The ontological foundation serves to aid in ensuring a maximally orthogonal design, and the requirements aim, in part, at ensuring a design that naturally extends the syntax and semantics of the regular ER model. The result is a novel model that satisfies an array of properties not satisfied by any single previously proposed model.

Keywords: conceptual modeling, database design, entity-relationship models, temporal databases, temporal data models, temporal semantics.

1 Introduction

A wide range of prominent, existing database applications manage time-varying information. These include financial applications such as portfolio management, accounting, and banking; record-keeping applications, including personnel, medical-record, and inventory; and travel applications such as airline, train, and hotel reservations and schedule management.

Frequently, existing temporal-database applications such as these employ the Entity-Relationship (ER) model [1], in one of its different incarnations, for database design. The model is easy to understand and use, and an ER diagram provides a good overview of a database design. The focus of the model is on the structural aspects of the mini-world (we use the term "mini-world" for the part of reality that the database stores information about), as opposed to the behavioral aspects. This focus matches the levels of ambition for documentation adopted by many users.

In the research community as well as in industry, it has been recognized that although temporal aspects of mini-worlds are prevalent and important for most applications, they are also difficult to capture elegantly using the ER model. The temporal aspects have to be modeled explicitly in the ER diagrams, resulting in ER diagrams with entities and attributes that model the temporal aspects and that make otherwise intuitive and easy-to-comprehend diagrams difficult to understand. As a result, some industrial users simply ignore all temporal aspects in their ER diagrams and supplement the diagrams with textual phrases such as "full temporal support," indicating that some temporal aspect of data is somehow captured. The result is that the mapping of ER diagrams to the relational tables of the underlying DBMS must be performed by hand; and the ER diagrams do not document well the temporally extended relational database schemas used by the application programmers. An example, Figure 1 illustrates how temporal aspects may clutter an otherwise simple and easy-to-comprehend ER diagram. The example will be used throughout the paper.

Example 1 Figure 1 presents an ER diagram for a company divided into different departments. Each department has a number, a name, some locations, and is responsible for a number of projects. The company keeps track of when a department is inserted and deleted. It also keep track of the various locations of a department. 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 over periods of time.

Each project has a manager who manages the project and some employees who work for the project. Each project has an ID and a budget. The company registers the history of the budget of a project. Each project is associated with a department that is responsible for the project. Each employee belongs to a single

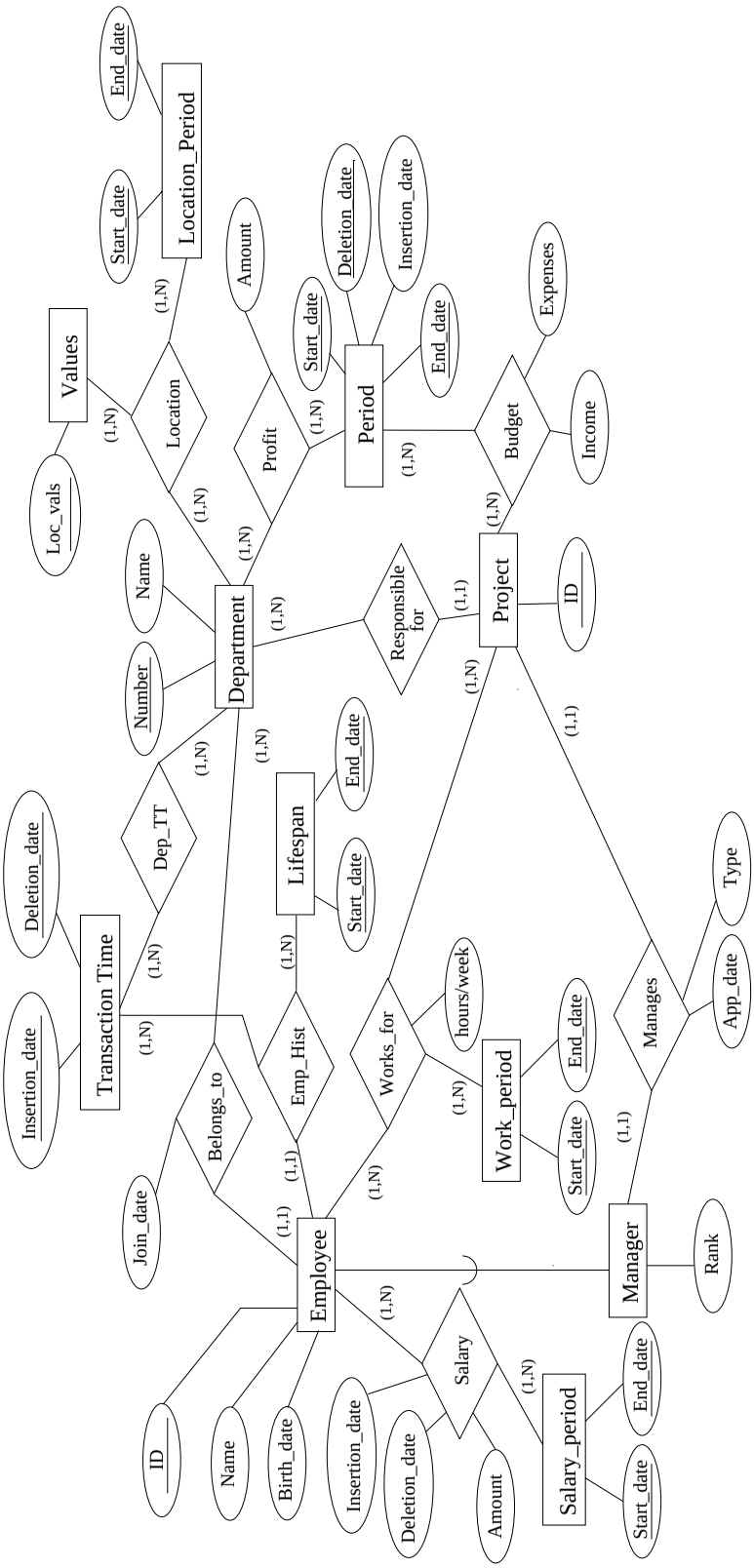


Figure 1: ER Diagram Modeling Temporal Aspects

department throughout his or her employment. For each employee, the company registers the ID, the name, the date of birth, and the salary. The company also records the history of employments. The departments would like to keep records of the different employees' salary histories. For reasons of accountability, it is important to be able to trace previous records of both profits and salaries.

Employees work on one project at a 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 which project at a given time and what time they are suppose to be finished 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.

If we ignored the temporal aspects, we could remove all the entity types in Figure 1 that model time periods; and the relationship types Profit, Budget, and Salary may be modeled simply with attributes. Figure 11 shows the same mini-world, but now captured using the TIMEER model proposed in this paper. □

The research community's response to the shortcomings of the regular ER model for the modeling of temporal aspects has been to develop temporally enhanced ER models, and a number of models have been reported in the research literature. These temporal ER models are developed in an attempt to provide modeling constructs that more naturally and elegantly permit the designer to capture temporal aspects, such as valid and transaction time, of information. For a detailed description of the existing models, see Gregersen and Jensen [15].

The approaches taken to add built-in temporal support into the ER model are quite different. The temporal ER models generally either change the semantics of the existing ER model constructs or introduce new constructs to the model. 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. 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.

While the existing temporal ER models represent a rich body of insight into the modeling of temporal data, an evaluation [16] of the models according to a dozen evaluation criteria indicate that no model is entirely satisfactory. For example, only one model supports the transaction-time aspect of data, which is important for many applications. A common characteristic of the existing temporal ER models is that few or no specific requirements to the models were given by their designers. Rather than being systematically founded on an analysis of general concepts and temporal aspects, their designs are often ad hoc. For example, the design of one model is the result of the need for the modeling of temporal aspects in a specific

application. The definitions of the existing models also generally lack comprehensiveness and precision and rely heavily on the reader's intuition. These conditions make it difficult to identify the ideas behind the designs of the models and to understand their semantics. Section 4 compares the proposed model to the existing models in more detail.

It is out contention that there is a need for a temporally extended ER model with an ontological foundation that analyzes and explicitly describes concepts fundamental to temporally enhanced data modeling. It is also essential that this model has explicitly formulated design goals and a comprehensive and precise definition.

We define a graphical, temporally extended ER model, called TIMEER, that extends the EER model as defined by Elmasri and Navathe [10] to provide built-in support for capturing temporal aspects of entities, relationships, superclasses and subclasses, and attributes. The design of the model is based on an ontology, which defines database objects, fundamental aspects of time, and indicates which aspects of time may be associated meaningfully with which database objects. Next, the model is designed to satisfy additional, explicitly formulated design goals for temporally extended ER models. Finally, a formal semantics, based on denotational semantics, for the TIMEER model is provided.

The paper is structured as follows. Section 2 first gives the ontological foundation of the TIMEER model, then formulates the design goals for the model. Section 3 proceeds to define the TIMEER model. An appendix gives the formal semantics of the model. Section 4 compares TIMEER with the most closely related and prominent previously proposed temporal ER models, pointing out the obtained improvements. Finally, Section 5 concludes and identifies promising research directions.

2 Ontological Foundations and Requirements

This section first relates the aspects of reality that may be captured by an ER model to the fundamental modeling constructs in ER modeling. Then follows an introduction of generic temporal aspects of information that are prime candidates for being given built-in support in an ER model. We proceed to introduce two fundamental distinctions; different decisions for these distinctions lead to fundamentally different ER models, and so these distinctions must be considered early in the process of designing an ER model. Finally, we present a set of requirements to a temporal ER model. We first relate the modeling constructs and temporal aspects, thus identifying exactly which combinations are meaningful. Next, we present design guidelines derived from a set of criteria for evaluating temporally extended ER models that we have previously developed [16].

2.1 Database Objects

Anything that exists in the mini-world and can be separated from other things in the mini-world is an *entity*; hence, a data model used for capturing a database representation of an entity should provide means of conveniently modeling the existence and unique identification of entities. The time during which an entity exists in the mini-world, we call the *existence time* of the entity.

Beyond having an independent existence, an entity is characterized by its properties, modeled by attributes. At any given point in time, an entity has a value for each of its attributes. The values of some attribute remain unchanged over time while others vary, that is, at different points in time, the values of an attribute for an entity may be different. We assume that it is meaningful for entities to have properties exactly when they exist (i.e., when they are entities)—it is meaningless for something that does not exist to have properties.

A database represents sets of entities that are similar, that is, have the same structure, or put differently, entities that have the same attributes. Entity types define sets of entities with the same attributes, and the entities of the same type is termed an entity set.

Entities may be interrelated via relationships. Such relationships can be seen from two very different points of view. We can either perceive relationships among entities as attributes of the participating entities, or we can perceive relationships as having existence in their own right. Both points of view have merit.

A relationship type among some entity types defines a set of relations among entities of these types. Each relationship relates exactly one entity from each of the entity types that the relationship type is defined over. The set of relationships defined by a relationship type is called a relationship set.

Another type of relationships exists, namely the superclass/subclass relationships that classifies entities of a superclass into different subclasses, e.g., employees may be divided into secretaries, engineers, and technicians. It is the same entities that occur in the subclasses and in the superclass; superclass/subclass relationships represent inheritance hierarchies rather than relate entities. For this reason, superclass/subclass relationships cannot exist in their own right, and nor can they be seen as attributes of the involved entity types. The entities of the subclasses inherit all the properties of entities of the superclass. It is not possible in subclasses to delete or modify the inherited properties, but it is possible to add new properties.

A data models should make it possible to conveniently and concisely capture all information about the mini-world that is meaningful to capture and is relevant for the application at hand. For example, since entities exist during some periods of time, it should be possible to capture this in the data model. In turn, this implies that the database designers should have the ability to indicate, using the conceptual data model, whether or not they want to register these periods of time for the entities.

2.2 Aspects of Time

In the database community, several types of temporal aspects of information have been discussed over the years. In this paper, we focus on five distinct types of temporal aspects that are candidates for being given built-in support in an ER model, namely *valid time*, *lifespan*, *transaction time*, *user-defined time* [17], and *decision time* [9, 2].

We use the term "fact" to denote any statement that can be assigned a truth value, i.e., true or false. The notion of *valid time* applies to facts: the valid time of a fact is time when that fact is true in the mini-world. Thus, any fact in the database may be associated with a valid time. However, the valid time may or may not be captured explicitly in the database.

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, with the database facts being implicit. 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 the 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 (at any time) record only one department value for each Employee entity.

The *lifespan* of an entity captures the existence time of the entity. 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. The lifespan of an entity e may be seen as the valid time of the related fact, "*e exists*." However, we choose to consider lifespans as separate aspects since the recording of lifespans of entities is important for many applications. If relationships are regarded as having existence in their own right, the concept of lifespan is also applicable to relationships, with the same meaning as for entities.

The *transaction time* of a database fact is the time when the fact is current in the database and may be retrieved. As is the case for lifespans, the transaction time of a fact F may be seen as the valid time of a related fact, namely the fact, "*F is current in the database*," but we have also chosen to record transaction time as a separate aspect. Unlike valid time, transaction time may be associated with any element stored in a database, not only with facts. Thus, all database elements have a transaction-time aspect.

Observe that all the above-mentioned temporal aspects have a duration.

User-defined time is supported when time-valued attributes are available in the data model [25]. These are then employed for giving temporal semantics—not captured in the data model, but only externally, in the application code and by

the database designer—to the ER diagrams. For employee entities, such attributes could record birth dates, hiring dates, etc.

The *decision time* of a fact is the time when the fact was decided. A fact can therefore have many decision times. Since the number and meaning of "the decision times of" a fact varies from application to application and because decision times, unlike transaction time, generally do not exhibit specialized properties, the desirability of building in decision time support into a temporal ER model appears to be somewhat limited.

2.3 Fundamental Design Decisions

Two questions must be answered initially—the answers to these fundamentally affect the nature and properties of a temporally extended ER model.

Temporal Support, How?

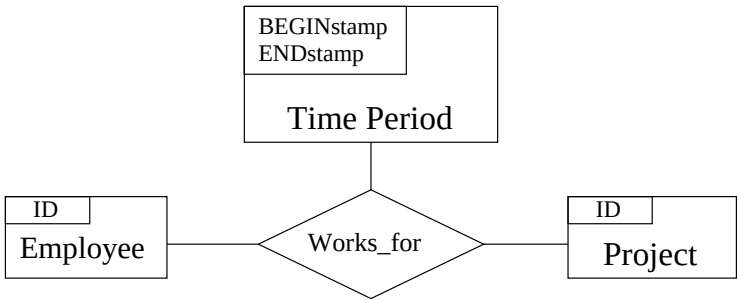
The first question is whether temporal support should be achieved by giving new temporal semantics to the existing constructs, or by introducing completely new temporal constructs.

The approach where all existing ER model constructs are given temporal semantics has been used in several of the existing temporal models [11, 22, 8] and has its strong points. Database designers are likely to be familiar with the existing ER constructs. So, after understanding the principle of 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. In its extreme, this approach rules out the possibility of designing non-temporal databases, i.e., databases that do not capture the temporal aspects of data. It is also not possible to design databases with non-temporal parts. Another problem is that old diagrams are no longer correct, i.e., while their syntax is legal, their semantics have changed, and they therefore no longer describe the underlying relational database schemas.

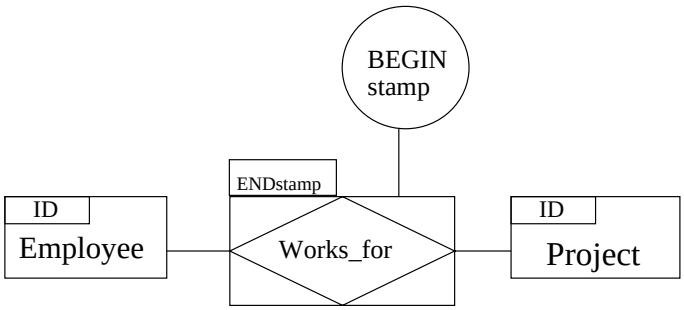
It is also possible to retain the existing ER constructs with their usual semantics while achieving temporal support. This is accomplished by adding new temporal constructs to the model that provide the support, and this approach is also widely used [20, 12, 24, 29, 23, 28, 21]. The extent of the changes made to the ER model may range from minor changes to a total redefinition of the model.

Two types of new temporal constructs may be distinguished. With *implicit* temporal support, the timestamp attributes used for capturing a temporal aspect are "hidden" in the new modeling constructs—explicit timestamps for capturing the temporal aspects are absent. In contrast, with *explicit* temporal support, timestamp attributes are explicit, and the semantics of the existing ER constructs are retained. Any new modeling constructs are notational shorthands for elements of regular ER

diagrams, introduced to make the modeling of temporal aspects more convenient. Figure 2 exemplifies this approach [12, 15].



(a)



(b)

Figure 2: Modeling Time-Varying Attributes in RAKE [12]

The models that retain the existing constructs with their old semantics and introduce new temporal constructs also have problems. If their extensions are comprehensive, they are likely to be more difficult for the database designers to learn and understand. The larger initial investment in training that this induces may prevent a model from being accepted in industry. On the other hand, this approach avoid the problem of legacy diagrams not describing the underlying database, since the semantics of the existing ER constructs are retained.

Design Model or Implementation Model?

The second question is whether the temporal ER model should have a query language, or whether algorithms that map ER diagrams to implementation platforms should be provided.

A mapping algorithm translates a temporal ER diagram into a corresponding database schema in another data model. The most obvious possible target data models are the relational model, some temporal relational model, and the conventional ER model.

The algorithm may map temporal ER diagrams directly to relational database schemas [12, 20, 24, 29, 23, 22], or a two-phase approach may be adopted

where temporal ER diagrams are first mapped to conventional ER diagrams and then mapped to relational database schemas, reusing mappings from the conventional ER model to the relational model [12, 28]. For minor extensions of the ER model, the reuse in the two-phase approach may be attractive. However, the two-phase translation yields less control over what relational schemas result from the combined mapping. With this approach, which is the one assumed in most existing temporal ER models, the ER model is a design model that is used solely for database design and has no directly associated database instance. In industry, the ER model is generally used as a design model, with variations of the relational model (as supported by the various specific relational products) being the most popular target models.

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 [11, 8, 29, 23, 22]. With this approach, a mapping to an implementation platform is not required. Instead, a query language should be available for querying ER databases. If this approach is taken, one faces the challenges of devising a query language for a temporal ER model.

2.4 Requirements for Capturing Temporal Aspects

Valid and transaction time are general—rather than application specific—aspects of all database facts [25]. Lifespan and transaction time are general aspect of entities. As such, these aspects are prime candidates for being built into a temporal ER model. In this section we describe what aspects of time we believe that a temporal ER model should provide built-in support for, and for which database objects this support should be provided.

Lifespan Since any entity has existence and thus an existence time, and since this aspect is important for many applications, it must be possible for database designers using a temporal ER model to conveniently register the existence time of entities. Lifespans are used for capturing existence time in the database, so a temporal ER model should offer built-in support for the registration of lifespans of entity types. Lifespans may or may not, at the designer's discretion, be captured in the database.

We have argued that a relationship can be seen as having independent existence; in that case, it also has an associated existence time, and it should be possible to register lifespans for relationships in the same way as for entities. Superclass/subclass relationships are closely tied to the superclasses and subclasses they relate, and their existence is dependent on these. We therefore do not find it useful to capture lifespans for this type of relationships.

When a model provides a built-in notion of relationship lifespans, it may also enforce certain constraints that involve these lifespans. The obvious constraint on

a relationship lifespan is that the lifespan of an relationship instance should be a non-empty subset of the union of the lifespans of the participating entities.

Built-in support for capturing lifespans of entities and relationships is important because lifespans are important in many applications and because entities and relationships may exist beyond the times when their attributes have (non-null) values—it is thus not possible to infer lifespans of entities or relationships from the valid times of the attribute values associated with the entities or relationships.

Valid Time Because facts have valid time and attributes are the modeling constructs used to capture facts at the conceptual level, a temporal ER model should support the possibility to register valid time for attributes. Built-in support for valid time is important because it is fundamentally important in a large class of applications to know at what times the facts recorded in the database are true.

Three different cases arise in connection with the recording (or non-recording) of the valid time of an attribute. First, if we record the valid time, this implies that we obtain the ability to capture all the values that have ever been valid for the attribute. Second, if we do not register the valid time of the attribute, this may be because the value of the attribute either never changes or because we are only interested in the current value of the attribute. Third, it could be that we do not know the valid time of the attribute—we know the valid value, but not the time when it is valid.

An inherent constraint applies to valid time and lifespans. Specifically, at any time during the database's evolution, the valid time of any attribute value of any entity must be a subset of the lifespan of the entity. If a relationship is viewed as an attribute of the participating entities, the data model should also provide built-in support for capturing the valid times of relationships. Superclass/subclass relationships are excluded because these are not considered attributes of the involved entities.

Transaction Time Transaction time is similar to valid time, but there are also some differences. Anything, not just facts, that may be stored in a database has a transaction time. With transaction time captured, past states of a database are retained, which is essential in applications with accountability or trace-ability requirements, of which there are many. The need for recording transaction time is thus widespread. It should be noted that separate support for transaction time of superclass/subclass relationships is not needed, due to the semantics of these.

It is an inherent constraint that the transaction time of an entity (or relationship with identity) be a subset of its lifespan: something that does not exist cannot be captured in the database.

User-Defined Time User-defined time attributes, i.e., time-valued attributes with no special support, are already available in the ER model and should also be available in a temporally extended ER model.

Relating Temporal Aspects and Modeling Constructs Figure 3 summarizes the temporal support we believe a temporal ER model must offer. The "Yes"'es in parenthesis indicate that a temporal ER model should offer support for both valid time and lifespans for relationship types. The users of the model must then decide whether or not relationships are to have independent existence and thus whether to capture lifespans or valid times for them.

	Entity types	Relationship types	Superclass/subclass Relationships	Attributes
Lifespan	Yes	(Yes)	No	No
Valid time	No	(Yes)	No	Yes
Transaction time	Yes	Yes	No	Yes

Figure 3: Modeling Constructs and Their Supported Aspects of Time

Maximally Meaningful and Flexible Support So far, we have argued that the different temporal aspects should be supported for exactly the modeling constructs where the aspects make sense. This provides maximum meaningful temporal support.

The different temporal aspects may or may not, depending on the application requirements, be captured in the database. Therefore, the support for these aspects should be user-specifiable and maximally flexible. This is achieved if the temporal ER model permits the database designer to decide which temporal aspects to capture of the different database elements. It must be possible to make these decisions independently for independent database elements. Following this principle, the granules of temporal support in an ER model are the following.

- Entity types.
- Relationship types.
- Attributes.

This means that the ER model should allow the designer to, e.g., specify the temporal support of an attribute and the attribute's entity independently. For example, the designer may capture lifespans for the Employee entity type while capturing both transaction time and valid time for some of the attributes of Employee.

Time Data Type Support Different time data types may be used for capturing the temporal aspects of database objects, including instants, time intervals, and temporal elements (temporal elements are finite unions of time intervals [13]).

For example, one option is to associate valid-time intervals with attribute values of entities, and another option is to timestamp attribute values with valid-time elements. An attribute value may also be defined as a function from a time domain to a value domain. This is similar to element timestamping in that an attribute may associate a value with some subset of the time domain. A data type of sets of time intervals is somewhat similar to the time element data type: with a discrete and bounded time domain, the two types are equivalent, but this does not hold in general.

A temporal ER model may provide the database designer with a choice of data types, thereby increasing the utility of the model. Instants, intervals, and elements may all be used for encoding durations. When instants are used for this purpose, they have associated interpolation functions. The instant data type may also encode the occurrence of instantaneous events. The importance of the availability of time data types 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.

Support for Interpolation Temporal interpolation functions derive information about times for which no data is explicitly stored in the database (see, e.g., [20] and [18]). 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.

Support for Granularities and Temporal (Im-) Precision It may be that the temporal variability of different objects in the mini-world is captured using times of different granularities [30, 4]. It should then also be possible to capture the variability of the different objects in the database using these different granularities. To exemplify, 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 assignment of employees to projects.

The temporal variability of different objects in the mini-world may be known with different precisions [20, 7, 5, 3], and although some imprecision may be captured using multiple granularities, granularities do not provide 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 time. This phenomenon may be prevalent and important to capture in scientific and monitoring applications that store measurements made by instruments. Thus the usability of a temporal ER model would be increased if support for temporal precision is provided.

Upward Compatibility To increase the usability of a new ER model, it is very important that legacy ER diagrams remain correct in the new model. This property, briefly mentioned earlier, is called *upward compatibility*. A temporal ER model is upward compatible with respect to a conventional ER model if any legal conventional ER diagram is also a legal ER diagram in the temporal model and if the semantics of the diagrams in the two models are the same. Upward compatibility protects investments in legacy systems and provides the basis for a smooth transition from a conventional ER model to a temporally enhanced ER model [26]. We thus require that a temporal ER model be upward compatible with respect to the conventional ER model it extends.

Snapshot Reducible Temporal Support The next property of a temporal extension is that of snapshot reducibility [27], which may be explained as follows. A temporal ER model that adds temporal support implicitly may provide temporal counterparts of, e.g., the ordinary attribute types, meaning that it provides temporal single-valued, temporal multi-valued, temporal composite, and temporal derived attribute types.

These temporal attribute types may be snapshot reducible with respect to their corresponding snapshot attribute types. In general, this occurs if snapshots of the databases described by a temporal ER diagram are the same as the 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 it 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.

Beyond attributes, snapshot reducibility also applies to the various constraints that may be defined on relationship types, including specialized relationship types such as superclass/subclass (ISA) 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 the 1–N snapshot constraint applies to any state, valid at any single point in time, of any possible instances of the temporal relationship type.

3 The Time extended EER (TIMEER) Model

In this section, the Time-Extended-EER model, TIMEER, is presented. First, the model on which to base the new model and positions regarding the fundamental design decisions from Section 2.3 are chosen. Second, the constructs of the new model are described.

3.1 The Basic Model of TIMEER

Since its publication, the ER model [1] has had various notations and semantics. It has been extended in order to capture superclass/subclass relationships and complex entity types, to name but a few extensions, and is then known as the EER model. Because no EER model has become a standard, the EER model presented by Elmasri and Navathe [10] is chosen as the basic model of TIMEER. The reader is assumed to be familiar with this model.

With respect to the fundamental design decisions presented in Section 2, the following choices are made. We have chosen to introduce new temporal constructs and provide *implicit* temporal support for the TIMEER model. This choice makes it possible to achieve a temporal ER model that is upward compatible with the ER model it extends. This means that existing legacy ER diagrams are valid temporal ER diagrams and retain their legacy meaning in the new model. This is important for industrial users with many legacy diagrams. Another important reason for this choice is the desire to retain the ability to design non-temporal databases as well as databases where some parts are non-temporal and others are temporal. We have chosen to provide mapping algorithms for the TIMEER model (described elsewhere). This decision is consistent with most temporal ER models being considered design models and with current practice in industry. Upon designing an ER diagram, the diagram is mapped to a schema of an available DBMS, i.e., is mapped to an implementation platform. A description of the mapping algorithms is beyond the context of this paper.

TIMEER supports the time data types “instant” and “temporal element.”

3.2 TIMEER Modeling Constructs

We proceed to present the modeling constructs of TIMEER model. The TIMEER model has implicit temporal support, and the existing EER constructs and their semantics are retained, meaning that new notation with implicit temporal support is added to the EER model to reach the TIMEER model. More specifically, TIMEER extends the EER model to include, where indicated, built-in temporal support for entities, relationships, superclasses/subclasses, and attributes.

Regular Entity Types

A regular entity type is represented by a rectangle. Since all entities represented by an entity type have existence time, modeled by lifespans in the database, and a transaction time aspect, the TIMEER model offers support for lifespans and transaction time for entity types. Thus, for each entity type in a TIMEER diagram, the database designer must decide whether or not to capture these temporal aspects of the entities in the database.

If the lifespan or the transaction time of an entity type is to be captured, this is indicated by placing an LS (LifeSpan) or a TT (Transaction Time) in the upper right corner of the rectangle, respectively. If both lifespan and the transaction time are captured, an LT (Lifespan and Transaction time) is placed as before. Entity types that capture at least one temporal aspect are termed temporal entity types; otherwise, they are termed non-temporal.

In Figure 1 in Example 1, we model that we want to capture both the lifespan and the transaction time of the entity type Employee, by associating it with two different time period entity types, Lifespan and Transaction Time. In the TIMEER this is modeled as shown in Figure 4.

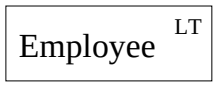


Figure 4: The Temporal Entity Type *Employee*

Weak Entity Types

Weak entity types are represented by double rectangles and are used to represent entities that are existence dependent on specific entities of another entity type and that cannot by themselves be lexically uniquely identified. A weak entity type must therefore be related via an (or a chain of) identifying relationship type (represented by a double diamond) to at least one regular entity type that is then the owner of the weak entity type. Weak entity types can be specified to capture the same temporal aspects as regular entity types, and this specification is independent of the temporal support specified for the owner(s) of the weak entity type. It is an inherent constraint that the existence time of a weak entity must be included in the existence time of the owner entity, due to the existence dependency.

Attributes

Entities are characterized by their attributes. A single-valued attribute is represented by an oval, a multi-valued attribute is represented by a double oval, and a composite

attribute is represented by an oval connected directly to other ovals representing the component attributes of the composite attribute.

All facts, modeled by attributes, have a valid time and a transaction time aspect, and the TIMEER model offers support for valid time and transaction time for all attribute types. If the database designer decides to capture the valid time of an attribute is captured, a VT is placed to the right in the oval; if transaction time is captured, a TT is placed as before. If both the valid time and the transaction time is captured, a BT (BiTemporal) is used. The components of a temporal composite attribute inherit the temporal specification for the composite attribute because we assume that all the components change synchronously, that is, the composite is considered to be one element; therefore, temporal support cannot be added separately to the components of a composite attribute. If no temporal aspects of an attribute are captured, we call the attribute non-temporal; otherwise, it is temporal.

It is meaningful for both temporal and non-temporal entity types to have temporal and non-temporal attributes. Temporal entity types may have non-temporal attributes; for example, it could be that the application at hand does not require the capture of any temporal aspects of the attributes of a temporal entity type; it could also be that some attributes are temporal. Similarly, for non-temporal entity types, it is possible that temporal aspects of some attributes are to be captured, even if no information about the entity is of interest after the deletion of the entity from the database.

In Figure 1 in Example 1, we model that we want to capture the valid time and the transaction time of the Salary of an Employee. To be able to capture the valid time, we convert the attribute Salary into a relationship type, Salary, between Employee and Salary_period, with an single-valued attribute Amount to actually record the salary. The transaction time is captured by associating the attributes Insertion_date and Deletion_date with the relationship. In TIMEER this may be modeled as shown in Figure 5.

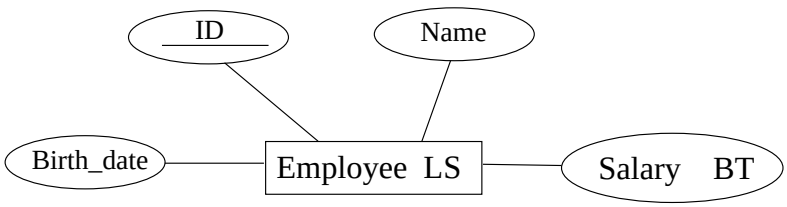


Figure 5: The Temporal, Single-Valued Attribute, Salary

Key Attributes

To indicate that a set of attributes represent the key of an entity type, the attribute names of the involved attributes are underlined. Key attributes of an entity type can

be specified as temporal or non-temporal. Simple and composite attributes may be specified as key attributes.

We allow key attributes to be specified as temporal and define these in terms of conventional keys and snapshot reducibility. Snapshot reducibility ensures, for example, that a single-valued attribute capturing valid time, at any point in the valid-time domain, is single-valued. Thus, combining snapshot reducibility of attribute types with the application of the conventional key constraint, we have that any key attribute at any point in time uniquely identifies an entity.

Relationship Types

A relationship type is represented by a diamond. The model offers support for lifespans and valid and transaction time for relationship types. The reason for offering support for both lifespans and valid time is that relationships can be perceived as either attributes of the participating entities, or as things that exist in their own right. For each relationship type, it has to be decided by the database designer whether or not to capture the temporal aspects for the relationship type. If some temporal aspect is captured for a relationship type, we call it temporal; otherwise, it is non-temporal.

If the relationship is perceived as an attribute, the possible temporal aspects are as for attributes, and the indication is placed in the lower corner of the diamond. When relationships are perceived as things that exist in their own right, the temporal aspects supported are lifespans and transaction time.

In Figure 1 in Example 1, we model that we want to capture the valid time of the relationship Works_for between Employee and Project. We therefore have to make the relationship type ternary by associating an entity type Work_period with the attributes Start_date and End_date to model this. A corresponding TIMEER diagram is shown in Figure 6.

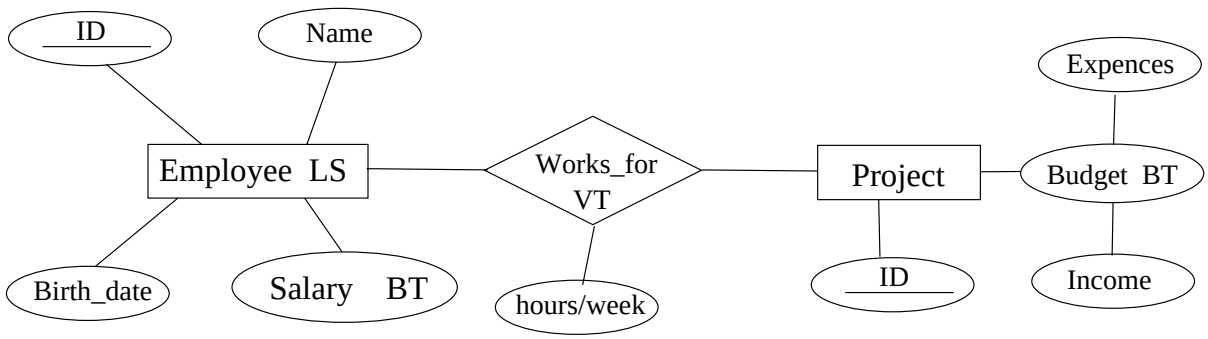


Figure 6: The Temporal Relationship Type Works_for

The temporal support of a relationship type can be specified independently of the temporal support for the participating entity types. An example is seen in Fig-

ure 6 where the relationship type Works_for, capturing valid time, relates the entity type Employee, capturing lifespans, and the non-temporal entity type Project. This means that during the valid time of a relationship instance, the entities participating in the relationship must exist, i.e., for a sample relationship, the participating Project entity has to be (current) in the database during the valid time of the relationship, and the participating Employee entity must have a lifespan that includes the valid time of the relationship.

Snapshot Participation Constraints

The snapshot participation constraint of an entity type E with respect to a relationship type R is represented by placing min and max values in parentheses by the line connecting entity type E with relationship type R. If $min = 0$ then the participation of the entities of E is optional; if $min \geq 1$ then the participation is total (mandatory). If $max = 1$, this means that the entities of E cannot participate in more than one relationship at a time, whereas a $max = n$, with $n > 1$ means that E entities can participate in n relationships at a time.

The intuitive meaning of this is: at any point in time, each instance e of the entity type E will participate in at least min and at most max instances r of R. That is, the the snapshot participation constraint is snapshot reducible with respects to the conventional participation constraints.

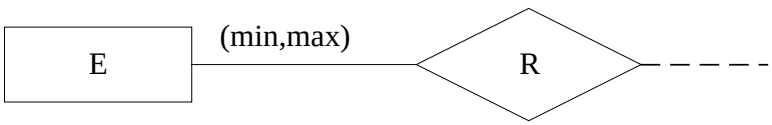


Figure 7: Representation of Snapshot Participation Constraints in TIMEER

We can now describe that an Employee works on one project at a time by adding the participation constraint (1, 1) next to the line connecting the relationship type Works_for to the entity type Employee. In Figure 8, we have also added a participation constraint stating that a project must have at at least one employee assigned to it at any point in time.

3.3 Advanced Features

The previous section described the fundamental design of the TIMEER model. This section proceeds to present additional features of the model.

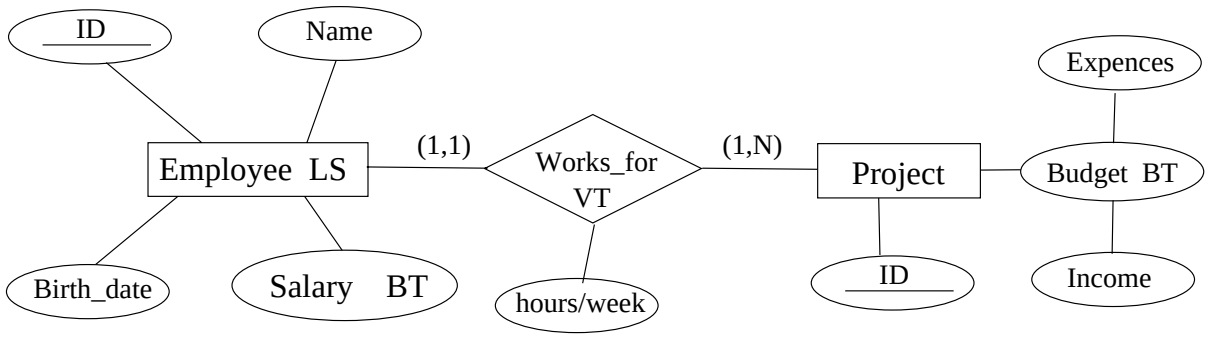


Figure 8: The Temporal Relationship Type *Works_for* with Participation Constraints

Lifespan Participation Constraints

The snapshot participation constraints already described constrain the participation of the entities at each isolated point in time. It is also useful to be able to describe the participation of an entity in a relationship over the entire existence time of the entity. This is useful if, for example, we want to state that an employee only can be assigned to at most one project at a time, but can be assigned to any number of projects and must be on at least one during the entire employment.

The participation constraint in Figure 8 ensures that an employee participates in exactly one relationship at any point in time, but it says nothing about the entire employment period. If we change the participation constraint from $(1, 1)$ to $(1, N)$, this means that an employee at any single point in time is now allowed to appear in Works_for N times, which is not intended. Another type of participation constraint, called the lifespan participation constraint, must instead be added to the model, making it possible to express participation constraints throughout the existence times of the entities.

The lifespan participation constraint of entity type E with respect to relationship type R is represented by placing min and max values in square brackets by the line connecting entity type E with relationship type R.

The intuitive meaning of the lifespan participation constraint is: over all of time, any instance e of the entity type E must participate in at least min and at most max instances r of R.

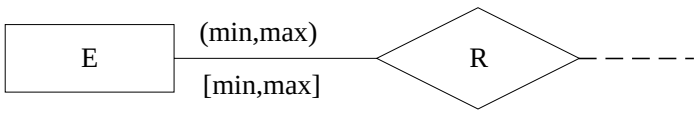


Figure 9: Representation of Lifespan Participation Constraint in TIMEER

The lifespan participation constraint specified for the participation of an entity type with respect to a non-temporal relationship type must be the same as the

specified snapshot participation constraint, for which reason they can be omitted from the diagrams.

There are combinations of snapshot and lifespan participation constraints that are contradictory. For constraints (a, b) and $[c, d]$, this occurs when $a > d$, which is the case for the combination of (M, N) and [1, 1].

Other cases exist where lifespan participation constraints do not add to pre-existing snapshot participation constraints. For example, a lifespan participation constraint [1, N] does not add to the snapshot participation constraint (1, 1).

Generally, we expect the *min* of the lifespan participation constraint to be equal to or larger than the *min* of the snapshot participation constraint; and the *max* of the lifespan participation constraint is expected to be equal to or larger than the *max* of the snapshot participation constraint.

Using both participation constraints, we can state that any employee must be assigned to at most one project at a time, but must be assigned to at least one project during the employment period. This is shown in Figure 10.

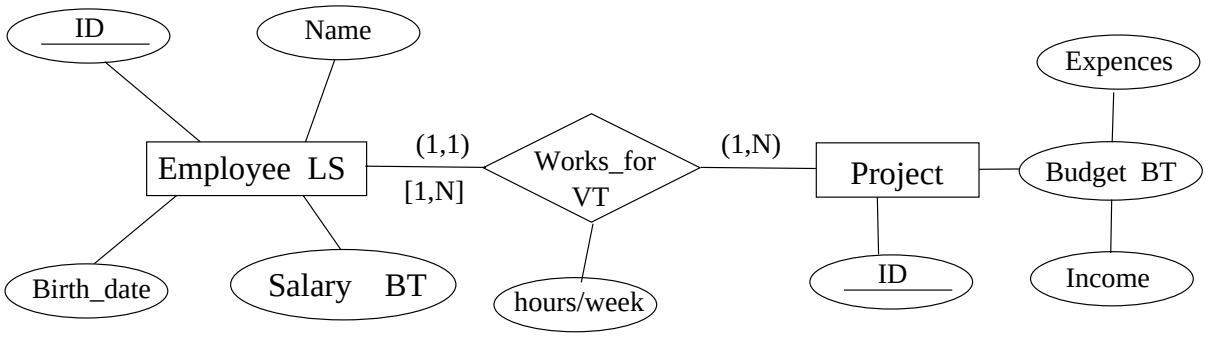


Figure 10: The Temporal Relationship Type Works_for with Participation Constraints

Superclasses and Subclasses

We offer support for specifying superclass/subclass relationships. The syntax is as in the EER model.

All subclasses inherit the attributes of the their superclasses, and just as inherited attributes cannot be given new data types, it is not possible to change the temporal support given in the superclasses to the inherited attributes. But it is possible to add temporal and non-temporal attributes in the subclasses.

It must also be decided whether a subclass inherits the temporal specification of its superclass, or whether this is to be specified for each individual class participating in a superclass/subclass relationship.

We have chosen that subclasses inherit the temporal aspects of their superclasses and that the inherited time specification is expandable, e.g., if we decide

to capture lifespans for Employee entities and let Secretary be a subclass of entity type Employee, we can decide to capture both lifespans and transaction time for Secretary entities. It is not possible to delete the inherited temporal support. This choice is consistent with the fact that subclasses inherit all properties, and thereby also the temporal support, of their superclasses and that it is not possible to delete or modify inherited properties, but only to add properties.

Temporal Interpolation Functions

As described earlier, temporal interpolation functions derive information about times for which no data is explicitly stored in the database. Support for interpolation is perhaps particularly important in applications where processes are monitored and variables are sampled.

We provide the designer with the possibility to define not only temporal interpolation, but also derivation functions for derived attributes, and we extend the model with temporal (and non-temporal) derived attributes. These are represented by dotted ovals with the same possibilities for specifying temporal support as for the stored attributes. The interpolation functions must be specified in the query language of the intended target platform, since we do not provide a query language with the TIMEER model. The tool implementing the model must provide means for linking the derived attribute with its defining query-language statement.

Example 2 Figure 11 gives a TIMEER diagram that corresponds to the the EER diagram given in Figure 1. □

In Appendix A we develop the formal semantics of the TIMEER model based on denotational semantics. To ease the development of the formal semantics, we initially transform the graphical notation of the model into an equivalent textual notation. The appendix uses part of the running example to exemplify the translation and explain the semantics.

3.4 Properties of The TIMEER Model

In Section 2 we listed a set of design goals. Having introduced TIMEER, we now examine its design with respect to the goals.

Temporal Aspects Supported We provide built-in support for capturing lifespans and transaction time for entities and relationships. This is achieved by making it possible for the database designer to specify for every entity type and relationship type whether or not to capture the temporal aspects of the modeling constructs. Similarly, built-in support for capturing valid time and transaction time for attributes and relationships is provided. Finally, user-defined time attributes are available.

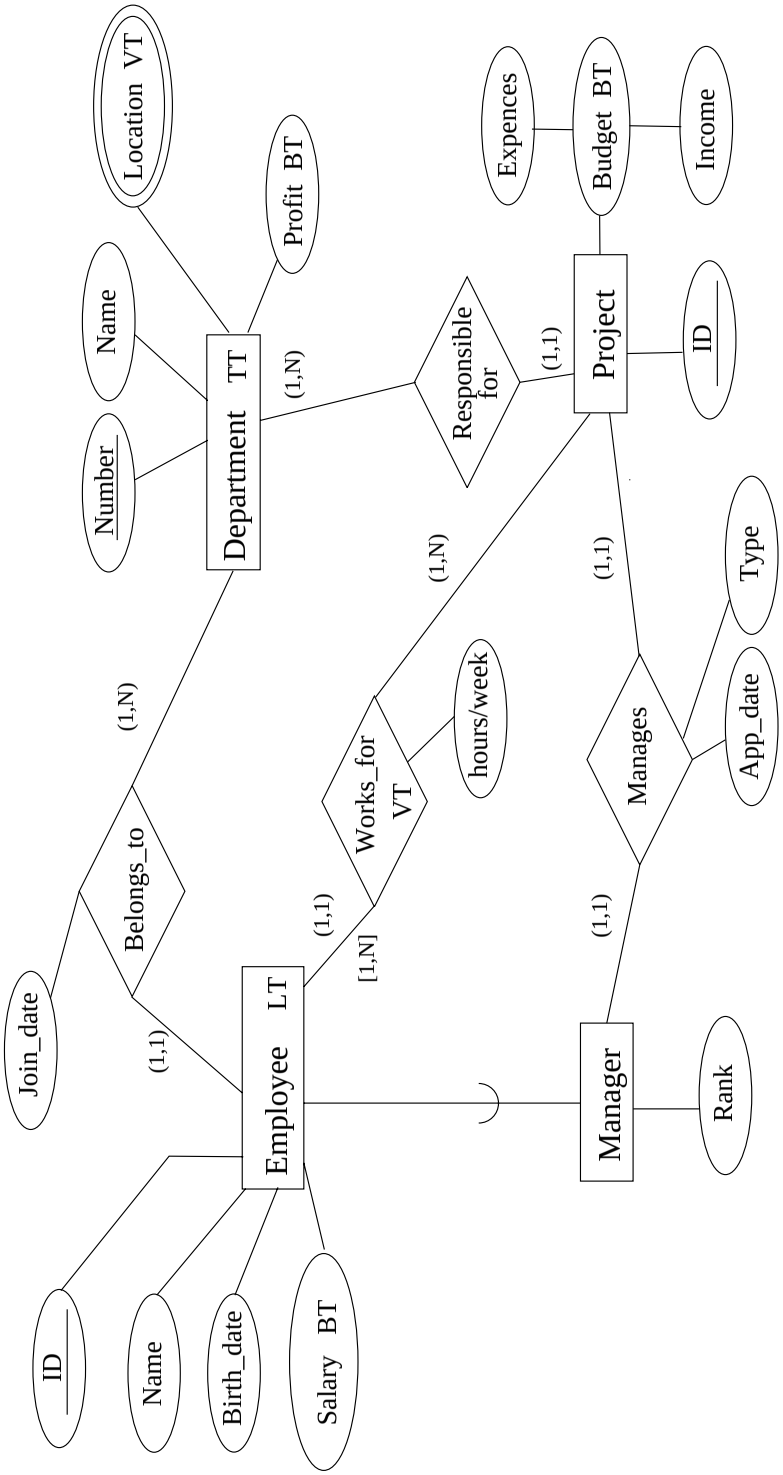


Figure 11: TIMEER Diagram of the Example

Maximally Meaningful and Flexible Support Because the database designer is able for each modeling construct to specify whether or not to capture each meaningful temporal aspect of the construct, TIMEER provides maximally meaningful and flexible temporal support. The model has optional use of the temporal constructs, providing the database designer with the possibility of mixing temporal and non-temporal constructs in the same diagram.

Time Data Type Support TIMEER supports time data types for the modeling of both instantaneous events and phenomena that persist in time, namely the "instant" and "temporal element" types. For simplicity, we have omitted the"interval" (or "period") data type. This type may be introduced in the model or may be introduced in the actual representation of modeled database instances.

Support for Interpolation The model provides support for defining temporal interpolation functions and derivation functions for derived attributes. The interpolation functions must be specified in the query language of the intended target platform—a separate language for this is not provided. A tool supporting the development of TIMEER diagrams must offer means for associating the definition of the interpolation functions with the attributes they compute.

Support for Granularities and Temporal (Im-) Precision The time granularities supported by TIMEER are second, minute, hour, day, week, month, and year. The model does not, at present, support temporal imprecision.

Upward Compatibility The designed model is upward compatible with respect to the EER model [10] because we *extend* this model with new temporal constructs while retaining all original EER constructs, with their original syntax and semantics.

Snapshot Reducible Temporal Support The temporal ER model presented in this paper has implicit temporal support and includes snapshot reducible temporal counterparts of the ordinary attribute types, i.e., provides temporal single valued, temporal multi-valued, temporal composite, and temporal derived attribute types. To achieve this, the semantics of the model given in Appendix A define the temporal attributes as functions from the time domain specified for the attributes into a domain of values.

Next, the snapshot participation constraints are also snapshot reducible, while lifespan participation constraints have no non-temporal counterparts. Finally, the constraints associated with superclass/subclass relationships are snapshot reducible.

For example, the temporal participation constraint (*disjoint, total*) for a superclass/-subclass relationship is snapshot reducible, so that for any snapshot of the underlying database, any entity of the superclass is present in exactly one subclass.

4 Related Research

We have previously conducted a comprehensive survey [15] of all previously proposed temporally extended ER models that we found in the research literature. The study of these models pointed to varying limitations in the existing models, motivating the development of a new temporal ER model that attempted to build maximally on the insights accumulated in the existing models.

More specifically, the existing temporal ER models represent quite diverse approaches to capturing temporal aspects of data at the conceptual level, and it is our contention that the models, to varying degrees, have succeeded in more elegantly capturing the temporal aspects of data than does the ER model. However, evaluating the existing models against a list of desirable properties [16] reveals that no single model satisfies all properties, but that the models collectively cover the design space well.

As mentioned in the introduction, a common characteristic for the existing temporally extended ER models is that few or no specific requirements to the models are given by their designers. In contrast, we have based the design of the TIMEER model on the design goals presented in Section 2, some of which are based on ontological considerations, and some of which are derived from previously presented properties [16].

One approach to developing a temporal extension is to give the existing ER constructs new temporal semantics. This approach has been followed in several models [11, 22, 8], and it has its strong points. But there are also weaknesses. The main weakness is the lack of upward compatibility, and for this reason we have not chosen this approach for TIMEER.

Another approach is to retain the existing ER constructs with their usual semantics and introduce new temporal constructs that provide temporal support. This can be done by offering new modeling constructs with either implicit temporal support [20, 24, 29, 23, 28, 21] or explicit temporal support [12]. Since the latter type of support still leads to cluttered diagrams, although to a lesser degree than in the ER model, we have chosen to add new temporal constructs with implicit temporal support.

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 databases to be temporal, but rather permits the designer to mix temporal and non-temporal parts. We believe that the TIMEER model has these properties.

The concept of snapshot reducibility applies to attributes as well as the various constraints that may be defined on relationship types, including those on superclass/subclass hierarchies. Satisfying reducibility is very important because this provides a uniform and natural generalization of standard, snapshot ER modeling constructs to temporal counterparts.

Although we have seen that this requirement never previously has been applied explicitly to an ER model, aspects of existing temporal ER models turn out to be snapshot reducible. Only two temporal ER models have snapshot reducible relationship constraints [28, 21], while most models have snapshot reducible attributes [12, 20, 24, 11, 8, 22, 21], This latter property of the various models follows implicitly from how the temporal attributes are defined as shorthands for patterns made up of conventional constructs, from the properties of the models' mapping algorithms, from explicitly formulated semantics for the attributes, or from the attributes being defined in terms of snapshot reducible temporal relationships types.

The TIMEER model provides snapshot reducible attribute types as well as relationship constraints. Lifespan participation constraints do not have non-temporal counterparts to reduce to.

All but one of the existing temporal ER models support only valid time. We believe that the support for transaction time is just as important, and TIMEER supports both time aspects. Support for lifespans is also included, which is only provided by a subset of the existing temporal ER models [20, 11, 8, 29, 22].

5 Conclusions and Research Directions

Temporal aspects are prevalent in most real-world database applications, but they are also difficult to capture elegantly using the ER model. In an attempt to alleviate this problem, this paper presents a temporally extended ER model capable of more elegantly and naturally capturing temporal aspects of data.

The TIMEER model systematically extends the EER model [10] with new, enhanced modeling constructs with implicit temporal support. The new constructs provide built-in support for capturing lifespans of entities and relationships and provides built-in support for capturing valid times for attributes and relationships. And the model provides built-in support for capturing the transaction times for all modeling constructs. The temporal aspects of the modeling constructs are captured using either instants or temporal elements, and support for multiple granularities is included. The database designer may, or may not, use the new temporal constructs, and the resulting model is upward compatible with respect to the EER model.

We are currently developing algorithms that provide well-behaved mappings from TIMEER diagrams to various implementation platforms, including non-temp-

oral (e.g., SQL–92) and temporal (e.g., TSQL2) platforms. Next, it may be desirable to extend the model to include *graphical* notation for describing more temporal aspects of data, including the update and observation patterns for temporal attributes [19], as well as other advanced temporal constraints (allowing the designer to specify that, e.g., the values of an attribute must increase over time). The extent to which this is feasible is still unclear. Also, we are currently conducting new (internal) case studies in order to gain insight into the strengths and potential weaknesses of the TIMEER model. Finally, the presence of a graphical editor that supports the development of TIMEER is highly desirable. The availability of such a tool will facilitate the evaluations of the model involving real users.

References

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

[2] S. Chakravarthy and S. K. Kim. Semantics of Time-Varying Information and Resolution of Time Concepts in Temporal Databases. In R. T. Snodgrass, editor, *Proceedings of the International Workshop on an Infrastructure for Temporal Databases*, June 1993.

[3] 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.

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

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

[6] C. E. Dyreson and R. T. Snodgrass. The Baseline Clock. In R. T. Snodgrass, editor, *The TSQL2 Temporal Query Language*, Chapter 5, pp. 77–96. Kluwer Academic Publishers, 1995.

[7] C. E. Dyreson and R. T. Snodgrass. Supporting Valid-time Indeterminacy. *ACM Transaction on Database Systems*, 23(1):?–?, March 1998.

[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*, pp. 239–254, October 1990.

[9] O. Etzion, A. Gal, and A. Segev. A Temporal Active Database Model. Technical Report LBL 32587, Lawrence Berkeley Laboratory, 1992.

[10] R. Elmasri and S. B. Navathe. *Fundamentals of Database Systems.* The Benjamin/Cummings Publishing Company, 2. edition, 1994. ISBN 0-8053-1753-8.

[11] 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*, pp. 76–83, 1990.

[12] S. Ferg. Modeling the Time Dimension in an Entity-Relationship Diagram. In *4th International Conference on the Entity-Relationship Approach*, pp. 280–286, 1985.

[13] S. K. Gadia. A Homogeneous Relational Model and Query Languages for Temporal Databases. *Transactions on Database Systems*, 13(4):418–448, December 1988.

[14] M. Gogolla and U. Hohenstein. Towards a Semantic View of an Extended Entity-Relationship Model. *ACM Transaction on Database Systems*, 16(3):369–416, September 1991.

[15] H. Gregersen and C. S. Jensen. Temporal Entity-Relationship Models—a Survey. *IEEE Transactions on Knowledge an Data Engineering*. To appear.

[16] H. Gregersen, C. S. Jensen, and L. Mark. Evaluating Temporally Extended ER Models. In K. Siau, Y. Wand, and J. Parsons, editors, *Proceedings of the Second CAiSE/IFIP8.1 International Workshop on Evaluation of Modeling Methods in Systems Analysis and Design*, 12 pages, June 1997.

[17] C. S. Jensen and C. E. Dyreson, editors. The Consensus Glossary of Temporal Database Concepts - February 1998 Version. In O. Etzion, S. Jajodia, and S. Sripada, editors, *Temporal Databases: Research and Practice*, volume 1399 of *Lecture Notes in Computer Science*, pp. 367–405. Springer-Verlag, 1998.

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

[19] C. S. Jensen and R. T. Snodgrass. Temporally Enhanced Database Design. In M. P. Papazoglou, S. Spaccapietra, and Z. Tari, editors, *Object-Oriented Data Modeling*, MIT Press, 1998. To appear.

[20] 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*, pp. 477–512, October 1981.

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

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

[23] 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*, pp. 244–260, July 1992.

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

[25] R. Snodgrass and I. Ahn. A Taxonomy of Time in Databases. In S. Navathe, editor, *Proceedings of ACM-SIGMOD 1985 International Conference on Management of Data*, pp. 236–246, May 1985.

[26] 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*, page 40, December 1995. ANSI Expert's Contribution.

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

[28] B. Tauzovich. Toward Temporal Extensions to the Entity-Relationship Model. In *The 10th International Conference on the Entity Relationship Approach*, pp. 163–179, October 1991.

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

[30] G. Wiederhold, 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*, pp. 124–140, May 1991.

A Formal Semantics of TIMEER

This section defines the formal semantics of TIMEER. As a first step, we we translate the graphical TIMEER diagrams into equivalent textual representations. The semantics of a TIMEER diagram is then defined as the semantics of the equivalent textual variant of the diagram.

In Section A.1, we present the textual representation of the model and exemplify the transformation of the graphical representation of a diagram into an equivalent textual representation. We also present the axiomatic conventions that define the notation used, followed by definitions of the predefined atomic data types supported by the TIMEER model.

In section A.2, we proceed to define the semantics of the basic data types supported by TIMEER, then define the semantic domains of the timestamps data types supported, followed by the semantics of the textual representation of the TIMEER model. Part of the running example is used to illustrate the main ideas behind the semantics.

A.1 Textual Representation of TIMEER Diagrams

The translation from TIMEER diagrams to the equivalent textual representations is straightforward; given the TIMEER diagram in Figure 11, we will transform a part of this diagram in order to explain the transformation.

Before we present the full syntax of the textual representation of the TIMEER model, we describe the notation as well as conventions used in the abstract syntax of the textual representation and in the defintion of the semantics.

Axiomatic Conventions

We let *SET* denote the class of sets, *FSET* the class of finite sets, *FUN* the class of total functions, and *REL* the class of relations. The following inclusions hold $FSET \subseteq SET$ and $FUN \subseteq REL \subseteq SET$.

Next, assume that sets $S, S_1, \ldots, S_n \in SET$ are given. We let $F(S)$ denote the restriction of the power set 2^S to finites sets, S^* denote the set of finite lists over S, S^+ the set of non-empty finite lists over S, and $S \times S_1 \times \cdots \times S_n$ denote the Cartesian product over the sets $S, S_1, \ldots, S_n$. The set of finite multisets over S is given by $M(S)$. A multiset can be considered a finite set S together with a counting function $occ : S \to N$, giving for each element the number of occurrences in the multiset. We let $S_1 \uplus S_2$ denote the disjoint union of sets, that is, the result of $S_1 \uplus S_2$ is $\{S_1, S_2\}$.

We write finite sets as $\{c_1, c_2, \ldots, c_n\}$, lists as $\langle c_1, c_2, \ldots, c_n \rangle$, elements of Cartesian products as $(c_1, c_2, \ldots, c_n)$, and multisets as $\{\!\{c_1, c_2, \ldots, c_n\}\!\}$. For a set

$\{c_1, c_2, \ldots, c_n\}$, $i \neq j$ implies $c_i \neq c_j$. This is not necessarily true for multisets. Given multiset $\{\{c_1, c_2, \ldots, c_n\}\}$ with $occ(c) = k$, there are k indices $i_1, \ldots, i_k \in 1, \ldots, k$ with $c_{i_j} = c$ for $j \in 1, \ldots, k$. For any set, we use $\perp$ to denote the undefined value of the set.

Predefined Data Types

A data signature describes the predefined data types, operations, and predicates. We assume the data types *int, real,* and *string*; adding additional data types is straightforward. The data types *int, real,* and $sting$ and the operations and predicates on these have the usual semantics, and this interpretation is fixed, that is, defined once and for all. This definition follows the approach of Gogolla and Hohenstein [14].

Let the syntax of a data signature DS be given as follows.

- the sets *DATA, OPNS, PRED* $\in$ *FSET*
- a function *input* $\in$ *FUN* such that *input* : *OPNS* $\rightarrow$ *DATA**
- a function *output* $\in$ *FUN* such that *output* : *OPNS* $\rightarrow$ *DATA*
- a function *args* $\in$ *FUN* such that *args* : *PRED* $\rightarrow$ *DATA*$^+$

If $\sigma \in$ *OPNS*, *input*$(\sigma) = \langle d_1, \ldots, d_n \rangle$, and *output*$(\sigma) = d$, this is denoted as $\sigma : \langle d_1, \ldots, d_n \rangle \rightarrow d$. If $\pi \in$ *PRED* with *args*$(\pi) = \langle d_1, \ldots, d_n \rangle$, this is denoted as $\pi : \langle d_1, \ldots, d_n \rangle$

Example 3 The predefined data types and some operators and predicates working on the data types are given below.

DATA $\supseteq$ { *int*, *real*, *string* }

OPNS $\supseteq$	$\{ +_i, -_i, *_i :$	*int* $\times$ *int*	$\rightarrow$ *int*,
	$+_r, -_r, *_r :$	*real* $\times$ *real*	$\rightarrow$ *real*,
	$/_i :$	*int* $\times$ *int*	$\rightarrow$ *real*,
	$/_r :$	*real* $\times$ *real*	$\rightarrow$ *real*,
	$\uparrow_i :$	*int* $\times$ *int*	$\rightarrow$ *int*,
	$\uparrow_r :$	*real* $\times$ *int*	$\rightarrow$ *real*,
	$square_i :$	*int*	$\rightarrow$ *int*,
	$square_r :$	*real*	$\rightarrow$ *real*,
	$abs_i :$	*int*	$\rightarrow$ *int*,
	$abs_r :$	*real*	$\rightarrow$ *real*,
	$trc, rnd :$	*real*	$\rightarrow$ *int*,
	$cat :$	*string* $\times$ *string*	$\rightarrow$ *string*$\}$

PRED $\supseteq$	$\{ <_i, >_i, \leq_i, \geq_i, \neq_i :$	*int* $\times$ *int*,	
	$<_r, >_r, \leq_r, \geq_r, \neq_r :$	*real* $\times$ *real*,	
	$<_s, >_s, \leq_s, \geq_s, \neq_s :$	*string* $\times$ *string* $\}$	$\square$

Example 4 As a precursor to giving the textual representation of TIMEER diagrams, we transform the entity types Employee and Department, the relationship type Belongs_to, and the constraints related to these three modeling constructs into their textual representations.

For the entity type Employee, it is specified that both the lifespan and the transaction time of the instances must be captured. In the diagram in Figure 11, the data types of the timestamps are implicit; in the textual representation they are specified explicitly. The data type is temporal elements, and the granularity of the timestamps is hour for both temporal aspects to be captured. This results in the textual description below. Words in boldface are keywords.

Entity Type *Employee* **with** (*LS*, *temporal element*, *hour*), (*TT*, *temporal element*, *hour*)

We now have to add the attributes of the entity type Employee. It has the attributes ID, Name, Birth_date, and Salary. The only attribute where the temporal aspect is captured is Salary, and the time dimensions captured are valid time and transaction time. For all attributes, we have to specify the data type of the attribute values. For the temporal attributes, as for temporal entity types, the data type and the granularity of the timestamps capturing the temporal aspects are implicit in the diagrams, but have to be specified explicitly in the textual representation of the temporal attributes. The attributes of the Employee entity type are given as next.

Attribute *ID* **is of type** *int*;
Attribute *Name* **is of type** *string*;
Attribute *Bith_date* **is of type** *string*;
Attribute *Sal* **is of type** *real* **with** (*VT*, *temporal element*, *day*),
(*TT*, *temporal element*, *day*);

The translation of the other modeling constructs follow the same procedure and the textual representations of entity type Department and relationship type Belongs_to are as follows.

Entity Type *Department* **with** (*TT*, *temporal element*, *day*) **has**
Attribute *Number* **is of type** *int*;
Attribute *Name* **is of type** *string*;
Attribute *Location* **is Multivalued of type** *string* **with** (*VT*, *temporal element*, *day*);
Attribute *Profit* **is of type** *real* **with** (*VT*, *temporal element*, *month*),
(*TT*, *temporal element*, *month*);

Relationship Type *Belongs_to* **has**
Attribute *Join_date* **is of type** *string*;
involves *Employee*; *Department*;

We now add key constraints to the entity types and snapshot participation constrains to the relationship type. ID is the key of Employee and Number is the key of Department; the snapshot participation constraint on Employee is (1,1), and the

snapshot participation constraint on Department is (1,N). This gives us the following textual representation of the constraints.

ID **is key of** *Employee*;
Number **is key of** *Department*;
participation of *Employee* **in** *Belongs_to* **is** (1,1);
participation of *Department* **in** *Belongs_to* **is** (1,N); □

The full syntax of the textual representation of the TIMEER model is given next.

Meta variables

$Sc_D \in$ *Schemadecls* — TIMEER schema declarations
$E_D \in$ *Enttypedecls* — Entity type declarations
$R_D \in$ *Reltypedecls* — Relationship type declarations
$A_D \in$ *Attributedecls* — Attribute declarations
$IC_D \in$ *ICdecls* — Integrity constraints declarations
$E \in$ *E_TYPE* — The set of entity type names
$R \in$ *R_TYPE* — The set of relationship type names
$A \in$ *ATT* — The set of attribute names
$B \in 2^{ATT}$ — The set of subsets of attribute names
$T_S \in$ *T_SPEC* — The set of specifications for temporal support
$I_S \in$ *I_SPEC* — The set of involvement specifications
$d \in$ *DATA* — The set of basic data types supported by TIMEER
$max, min \in$ *Integer contants* — The set of integer constants
$dim \in \{LS, VT, TT\}$ — The set of time dimensions supported by TIMEER
$ts \in \{instant, temporal\ element\}$ — The set of data types for timestamps supported by TIMEER
$g \in \{sec, min, hour, day, week, month, year\}$ — The set of granules supported by TIMEER
$p_1, p_2 \in \{disjoint, overlapping, total, partiel\}$ — The set superclass/subclass participation constraints

Abstract Syntax

$Sc_D ::= E_D; R_D; IC_D$
$E_D ::= E_{D_1}; E_{D_2}$
| **Entity Type** E **has** A_D
| **Entity Type** E **with** T_S **has** A_D
| **Weak Entity Type** E **has** A_D
| **Weak Entity Type** E **with** T_S **has** A_D
| **Subclass** E_1 **of** E_2 **has** A_D
| **Subclass** E_1 **of** E_2 **with** T_S **has** A_D

$$
\begin{aligned}
R_D ::= \;& R_{D_1}; R_{D_2} \\
& \mid \textbf{Relationship Type } R \textbf{ has } A_D \textbf{ involves } I_S \\
& \mid \textbf{Relationship Type } R \textbf{ with } T_S \textbf{ has } A_D \textbf{ involves } I_S \\
IC_D ::= \;& IC_{D_1}; IC_{D_2} \\
& \mid B \textbf{ is primary key of } E \\
& \mid B \textbf{ is partial key of } E \\
& \mid \textbf{Snapshot participation of } E \textbf{ in } R \textbf{ is } (min, max) \\
& \mid \textbf{Lifespan participation of } E \textbf{ in } R \textbf{ is } [min, max] \\
& \mid \textbf{Participation of } I_S \textbf{ with respect to } E \textbf{ is } p_1, p_2 \\
A_D ::= \;& A_{D_1}; A_{D_2} \\
& \mid \textbf{Attribute } A \textbf{ is } A'_D \\
A'_D ::= \;& \textbf{of type } d \\
& \mid \textbf{of type } d \textbf{ with } T_S \\
& \mid \textbf{composite}(A_D) \\
& \mid \textbf{composite}(A_D) \textbf{ with } T_S \\
& \mid \textbf{Multivalued of type } d \\
& \mid \textbf{Multivalued of type } d \textbf{ with } T_S \\
T_S ::= \;& T_{S_1}; T_{S_2} \\
& \mid (dim, ts, g) \\
I_S ::= \;& I_{S_1}; I_{S_2} \\
& \mid E \\
& \mid E\textbf{(identifies)} \\
dim ::= \;& LS \mid VT \mid TT \\
ts ::= \;& \textit{instant} \mid \textit{temporal element} \\
g ::= \;& \textit{sec} \mid \textit{min} \mid \textit{hour} \mid \textit{day} \mid \textit{week} \mid \textit{month} \mid \textit{year} \\
d ::= \;& \textit{int} \mid \textit{real} \mid \textit{string} \\
p_1 ::= \;& \textit{disjont} \mid \textit{overlapping} \\
p_2 ::= \;& \textit{total} \mid \textit{partial}
\end{aligned}
$$

A.2 Semantics of TIMEER

We are now able to define the semantics of the TIMEER model. First, we define the semantics of the predefined data types and define the model of time used in the semantics. Next, we explain the ideas behind the semantics, followed be the full semantics of the TIMEER model.

The semantics of a data signature DS is given by three functions.

- A function $\mathcal{D}[\![DATA]\!] \in FUN$ such that $\mathcal{D}[\![DATA]\!] : DATA \rightarrow SET$ and $\perp \in \mathcal{D}[\![DATA]\!](d)$ for every $d \in DATA$. The membership of $\perp \in \mathcal{D}[\![DATA]\!](d)$ is required because it is necessary to have an undefined value as a result of an incorrect application of a function.

- A function $\mathcal{D}[\![OPNS]\!] \in FUN$ such that $\mathcal{D}[\![OPNS]\!] : OPNS \rightarrow FUN$ and $\sigma : d_1 \times \cdots \times d_n \rightarrow d$ implies $\mathcal{D}[\![OPNS]\!](\sigma) : \mathcal{D}[\![DATA]\!](d_1) \times \cdots \times \mathcal{D}[\![DATA]\!](d_n) \rightarrow \mathcal{D}[\![DATA]\!](d)$ for every $d \in DATA$.
- A function $\mathcal{D}[\![PRED]\!] \in FUN$ such that $\mathcal{D}[\![PRED]\!] : PRED \rightarrow REL$ and $\pi : d_1 \times \cdots \times d_n$ implies $\mathcal{D}[\![PRED]\!](\pi) \subseteq \mathcal{D}[\![DATA]\!](d_1) \times \cdots \times \mathcal{D}[\![DATA]\!](d_n)$ for every $d \in DATA$.

Example 5 The semantics of the predefined data types in Example 3 and some of the associated operations are defined as follows.

$$
\begin{array}{lcl}
\mathcal{D}[\![DATA]\!](int) & = & \mathbb{Z} \cup \{\bot\} \\
\mathcal{D}[\![DATA]\!](real) & = & \mathbb{R} \cup \{\bot\} \\
\mathcal{D}[\![DATA]\!](string) & = & \mathbb{A}^* \cup \{\bot\} \\
\mathcal{D}[\![OPNS]\!](+_i) & : & \mathcal{D}[\![DATA]\!](int) \times \mathcal{D}[\![DATA]\!](int) \rightarrow \mathcal{D}[\![DATA]\!](int) \\
 & = & \begin{cases} i_1 \times i_2 \rightarrow i_1 + i_2 & \text{if } i_1, i_2 \in \mathbb{Z} \\ \bot & \text{otherwise} \end{cases} \\
\mathcal{D}[\![OPNS]\!](square_r) & : & \mathcal{D}[\![DATA]\!](real) \rightarrow \mathcal{D}[\![DATA]\!](real) \\
 & = & \begin{cases} r \rightarrow r * r & \text{if } r \in \mathbb{R} \\ \bot & \text{otherwise} \end{cases}
\end{array}
$$

□

The Time Model

We assume that the real time line is bounded in both ends, so that time begins at the "Big Bang" and ends at the "Big Crunch." A point t on the real time line is called an instant. The real time line is represented in the database by a so-called baseline clock [6]. In accord with the general consensus in the database community that a discrete model of time is adequate, the base-line clock, and thus our time domains, is discrete. Our time domains are then ordered, finite sets of elements isomorphic to finite subsets of the natural numbers. The elements are termed chronons. This may be seen as dividing the real time line into indivisible equal-size segments (the chronons). Real-world time instants are represented in the model by the chronons during which they occur. We will use c, possibly indexed, to denote chronons. The size of a chronon, called the granularity of the chronon, can be specified explicitly.

We introduce a domain for each combination of the temporal aspects and granularities supported. These domains are given by D^g_{dim}. The different valid-time domains are given as $D^g_{VT} = \{c^v_1, c^v_2, \ldots, c^v_k\}$. The domain of all valid times is given as $\mathcal{D}_{VT} = \cup_g D^g_{VT}$. The transaction-time domains are given as $\mathcal{D}^g_{TT} = \{c^t_1, c^t_2, \ldots, c^t_{now}\} \cup \{UC\}$ where UC ("until changed") is a special transaction-time marker. The domain of all transaction times is then $\mathcal{D}_{TT} = \cup_g D^g_{TT}$. The different lifespan domains are given as $\mathcal{D}^g_{LS} = \{c^l_1, c^l_2, \ldots, c^l_{now}\}$, and the domain

of all lifespan times is given as $\mathcal{D}_{LS} = \cup_g D^g_{LS}$. Some chronons are expected to be in the future and some are expected to be in the past. The chronon c_{now} denotes the chronon representing the current time.

A time interval is defined as the time between two instants, a starting instant and a terminating instant. A time interval is thus represented by the sequence of consecutive chronons where each chronon represents the instants that occurred during the chronon. We may represent a sequence of chronons by the starting and the ending chronon. We define intervals $[c_i, c_j]^g$ where c_i is the starting chronon, c_j is the terminating chronon, and the size of the chronon is g. We let $[c_i, c_j]^g_{vt}$, $[c_i, c_j]^g_{tt}$, $[c_i, c_j]^g_{ls}$ denote intervals over the valid-time, transaction-time, and lifespan domains, respectively.

We also define temporal elements over time domains. A temporal element is a union of intervals and is represented by $I^g = [c_i, c_j]^g \cup \cdots \cup [c_l, c_k]^g$. Since our time domains are discrete and finite, we can define a temporal element as an element of the set $2^{D^g_{dim}}$. We let I^g_{vt}, I^g_{tt}, I^g_{ls} denote temporal elements over the valid-time, transaction-time, and the lifespan domains, respectively.

Semantics of Example 4

In order to better understand the ideas behind the semantics of TIMEER, we will explain in detail the semantics of the entity type Employee, the relationship type Belongs_to, the key constraint on Employee, and the snapshot participation constraint of Employee in Belongs_to.

An entity type in a TIMEER diagram defines an entity set. The attributes of an entity characterize the entity, and each attribute of an entity has a value domain. The association between a set of attributes $X = \{A_1, A_2, \ldots, A_n\}$ and the set of value domains D is given by a function $dom : X \rightarrow D$. An entity together with its attributes can be regarded as a tuple. A tuple t over a set of attributes X is actually a function that associates each attribute $A_i \in X$ with a value from the value domain $dom(A_i)$. For attribute A, we denote this value $t[A]$. In TIMEER, we use surrogates to identify the entities, and so extend the tuples with a surrogate attribute.

The semantics of the entity type Employee is therefore a set of functions (tuples), termed *Employee*. The domain of each function t is the set of attribute names connected to the entity type and the surrogate attribute, s. The value domain of the attributes connected to the entity type Employee is determined by the semantics of the attribute declarations, while the value domain of the surrogate attribute is the set of surrogate values assigned to *Employee*. The mathematical description of the above is presented next.

$$\begin{aligned}
&\mathcal{E}[\![\textbf{Entity Type } \textit{Employee} \ldots]\!] = \\
&\quad \{\textit{Employee}\} \times \{t \mid t \in \textit{FUN} \wedge \textit{dom}(t) = \\
&\qquad\qquad \{s, \textit{ID}, \textit{Name}, \textit{Bith_date}, \textit{Sal}\} \wedge t[s] \in D_S^{\textit{Employee}} \wedge \\
&\qquad\qquad t[\textit{ID}] \in \mathcal{A}[\![\textbf{Attribute } \textit{ID} \textbf{ is } A'_D]\!] \wedge \\
&\qquad\qquad t[\textit{Name}] \in \mathcal{A}[\![\textbf{Attribute } \textit{Name} \textbf{ is } A'_D]\!] \wedge \\
&\qquad\qquad t[\textit{Bith_date}] \in \mathcal{A}[\![\textbf{Attribute } \textit{Bith_date} \textbf{ is } A'_D]\!] \wedge \\
&\qquad\qquad t[\textit{Sal}] \in \mathcal{A}[\![\textbf{Attribute } \textit{Sal} \textbf{ is } A'_D]\!]\}
\end{aligned}$$

In the above description, we have not yet determined the value domains of the attributes. The attribute *ID* is specified as non-temporal with data type *int*. This means that we do not want to capture the changes of this attribute over time. The semantics is therefore modeled as an constant belonging to the set of integers, i.e., the value domain of this attribute is the set of integers, including the undefined value.

$$\mathcal{A}[\![\textbf{Attribute } \textit{ID} \textbf{ is of type } \textit{int}]\!] = \mathcal{D}[\![\textit{DATA}]\!](\textit{int}) = \mathbb{Z} \cup \{\perp\}$$

The *Birth_date* of an employee never changes, so this attribute is also described as non-temporal, but here we use the data type *string*. The value is therefore modeled as a constant sentence defined over some alphabet, i.e., the value domain is some alphabet, again including the undefined value.

$$\mathcal{A}[\![\textbf{Attribute } \textit{Bith\_date} \textbf{ is of type } \textit{string}]\!] = \mathcal{D}[\![\textit{DATA}]\!](\textit{string}) = \mathbb{A}^* \cup \{\perp\}$$

We will not explain in detail the semantics of the attribute *Name*, but proceed to the attribute *Sal*. This attribute is a temporal attribute with data type *real*, that is, we want to record how the values of this attribute change over time. This means that the value domain of this attribute must be a function from some time domain to a value domain. The temporal aspects to be captured for this attribute are valid time and transaction time.

$$\begin{aligned}
&\mathcal{A}[\![\textbf{Attribute } \textit{Sal} \textbf{ is of type } \textit{real} \textbf{ with } (\textit{VT}, \textit{temporal element}, \textit{day}), \\
&\qquad\qquad (\textit{TT}, \textit{temporal element}, \textit{day})]\!] = \\
&\quad \mathcal{T}[\![(\textit{VT}, \textit{temporal element}, \textit{day})]\!] \times \mathcal{T}[\![(\textit{TT}, \textit{temporal element}, \textit{day})]\!] \rightarrow \\
&\quad \mathcal{D}[\![\textit{DATA}]\!](\textit{real}) = D_{VT}^{day} \times D_{TT}^{day} \rightarrow \mathbb{R}^* \cup \{\perp\}
\end{aligned}$$

The resulting, full semantics of the entity type *Employee* is presented next.

$$\begin{aligned}
&\mathcal{E}[\![\textbf{Entity Type } \textit{Employee} \ldots]\!] = \\
&\quad \{\textit{Employee}\} \times \{t \mid t \in \textit{FUN} \wedge \textit{dom}(t) = \{s, \textit{ID}, \textit{Name}, \textit{Bith_date}, \textit{Sal}\} \wedge \\
&\qquad\qquad t[s] \in D_S^{\textit{Employee}} \wedge t[\textit{ID}] \in \mathbb{Z} \cup \{\perp\} \wedge t[\textit{Name}] \in \mathbb{A}^* \cup \{\perp\} \wedge \\
&\qquad\qquad t[\textit{Bith_date}] \in \mathbb{A}^* \cup \{\perp\} \wedge t[\textit{Sal}] \in D_{VT}^{day} \times D_{TT}^{day} \rightarrow \mathbb{R} \cup \{\perp\}\}
\end{aligned}$$

The key constraint of an entity set is a set of predicates, the entity set has to satisfy. In the textual representation, all constraints are separate constructs, so we

first have to check if the entity type mentioned in the constraint construct exists at all. We also have to check that the set of attributes mentioned in the constraint really are attributes of the entity type. Next, we define the predicate that ensures that the values of the key attributes are unique for the entity set. The key constraint on Employee is described next.

$$\begin{aligned}&\mathcal{C}[\![\mathit{ID}\ \textbf{is key of}\ \mathit{Employee}]\!] = \\ &\quad \mathit{inSch}(\mathit{Employee}, \mathit{Sc}_D) \wedge \mathit{ID} \in \mathit{attOf}(\textbf{Entity Type}\ \mathit{Employee}\ \textbf{has}\ A_D) \wedge \\ &\quad \forall t_i, t_j \in \mathcal{E}[\![\textbf{Entity Type}\ \mathit{Employee}\ \textbf{has}\ A_D]\!](t_i[\mathit{ID}] = t_j[\mathit{ID}] \Rightarrow t_i[s] = t_j[s])\end{aligned}$$

A relationship type in a TIMEER diagram defines a relationship set. Its semantics is therefore a set of relationships. The relationship type Belongs_to describes relationships among entities from the entity types Employee and Department. We use the surrogates of the participating entities to identify which entities participate in which relationship(s). As for entities, we can regard each relationship in a set of relationships as an element of a Cartesian product over a set of attributes. The attributes of a relationship are the attributes of the relationship type and a surrogate attribute for each participating entity type. To identify the participating entity types, we use the auxiliary function $\mathit{parOf}(I_S)$ that takes an involvement specification as input and returns a set (or if the relationship type involves the same entity type more than once, a multiset) of entity type names. The semantics of the relationship type Belongs_to is given next.

$$\begin{aligned}&\mathcal{R}[\![\textbf{Relationship Type}\ \mathit{Belongs\_to}\ \textbf{has}\ A_D\ \textbf{involves}\ I_S]\!] = \\ &\quad \{\mathit{Belongs\_to}\} \times \{t \mid t \in \mathit{FUN} \wedge \mathit{dom}(t) = \textstyle\bigcup_{E_i \in \mathit{parOf}(I_S)} s_{E_i} \cup \{\mathit{Join\_date}\} \\ &\qquad\qquad \textstyle\bigwedge_{E_i \in \mathit{parOf}(I_S)} t[s_{E_i}] \in \mathcal{I}[\![E_i]\!] \wedge \\ &\qquad\qquad t[\mathit{Join\_date}] \in \mathcal{A}[\![\textbf{Attribute}\ \mathit{Join\_date}\ \textbf{is}\ A'_D]\!]\} = \\ &\quad \{\mathit{Belongs\_to}\} \times \{t \mid t \in \mathit{FUN} \wedge \mathit{dom}(t) = \{s_{\mathit{Employee}}, s_{\mathit{Department}}, \mathit{Join\_date}\} \wedge \\ &\qquad\qquad t[s_{\mathit{Employee}}] \in D_S^{\mathit{Employee}} \wedge t[s_{\mathit{Department}}] \in D_S^{\mathit{Department}} \wedge \\ &\qquad\qquad t[\mathit{Join\_date}] \in \mathbb{A}^* \cup \{\perp\}\}\end{aligned}$$

Snapshot participation constraints are like key constraints and define predicates that relationship sets have to satisfy. Again, since the participation constraints are separate constructs in the textual representation, we have to ensure that the entity type and the relationship type mentioned in each constraint exist. To count the number of relationships, an entity participate in, we use the auxiliary function $\mathit{cnt}(e, E, \mathcal{R}[\![R_D]\!])$ that takes an entity, an entity type, and a relationship set as input and returns the number of relations in the relationship set, the entity e participates in.

$$\begin{aligned}
&\mathcal{C}[\![\textbf{Snapshot participation of } \textit{Employee} \textbf{ in } \textit{Belongs_to} \textbf{ is } (1,1)]\!] = \\
&\quad \textit{inSch}(Employee, Sc_D) \wedge \textit{inSch}(Belongs_to, Sc_D) \wedge \\
&\quad \textit{Employee} \in \textit{parOf}(\textbf{Relationship Type } \textit{Belongs_to} \textbf{ has } A_D \textbf{ involves } I_S) \wedge \\
&\quad \forall e_j \in D_S^{Employee}(min \leq cnt(e_j, Employee, \\
&\qquad\qquad \mathcal{R}[\![\textbf{Relationship Type } \textit{Belongs_to} \textbf{ has } A_D \textbf{ involves } I_S]\!]) \leq max)
\end{aligned}$$

The full semantics of the TIMEER model follow. First, we define the semantic domains. Second, we define the auxiliary functions to be used in the semantic functions. Finally, we define the semantic functions.

Semantic Domains

$D_S \cup \{\perp\}$ — The set of surrogates
$D_S^E \subseteq D_S$ — The set of surrogates assigned to $E \in [\![E\_TYPE]\!]$
$D_S^R \subseteq D_S$ — The set of surrogates assigned to $R \in [\![R\_TYPE]\!]$
$D_{LS} = (\cup_g D_{LS}^g) \cup \{\perp\}$ — The set of lifespan domains
$D_{VT} = (\cup_g D_{VT}^g) \cup \{\perp\}$ — The set of valid time domains
$D_{TT} = (\cup_g D_{TT}^g) \cup \{\perp\}$ — The set of transaction time domains
$\mathcal{D}[\![DATA]\!]$ — The set of basic domains

Auxiliary Functions

The function *attOf* takes as input an entity type declarations and returns the list of attributes names of the entity type.

$$\begin{aligned}
&\textit{attOf}(\textbf{Entity Type } E \textbf{ has } A_D) = \textit{attOf}(\textbf{Entity Type } E \textbf{ with } T_S \textbf{ has } A_D) = \\
&\quad \textit{attOf}(\textbf{Subclass } E_1 \textbf{ of } E_2 \textbf{ has } A_D) = \\
&\quad \textit{attOf}(\textbf{Subclass } E_1 \textbf{ of } E_2 \textbf{ with } T_S \textbf{ has } A_D) = \textit{attOf}(A_D) \\
&\textit{attOf}(A_{D_1}; A_{D_2}) = \textit{attOf}(A_{D_1}) \cup \textit{attOf}(A_{D_2}) \\
&\textit{attOf}(\textbf{Attribute } A \textbf{ is } A'_D) = A
\end{aligned}$$

The function *parAtt* takes the name of an entity type as argument and returns the names of attributes of the entity type and its ancestor(s), if the entity type is declared as a subclass.

$$\textit{parAtt}(E) = \begin{cases} \textit{attOf}(\textbf{Entity Type } E \ldots) & \text{if } \textbf{Entity Type } E \ldots \in E_D \\ \textit{attOf}(\textbf{Subclass } E_1 \textbf{ of } E_2 \ldots) \cup \textit{parAtt}(E_2) & \text{if } \textbf{Subclass } E_1 \textbf{ of } E_2 \ldots \in E_D \\ \perp & \text{otherwise} \end{cases}$$

The function *parOf* takes a relationship type declaration as argument and returns the entity types that participate in the relationship type.

$$
\begin{aligned}
&\mathit{parOf}(\textbf{Relationship Type } R \textbf{ has } A_D \textbf{ involves } I_S) = \\
&\quad \mathit{parOf}(\textbf{Relationship Type } R \textbf{ with } T_S \textbf{ has } A_D \textbf{ involves } I_S) = \mathit{parOf}(I_S) \\
&\mathit{parOf}(I_{S_1}; I_{S_2}) = \mathit{parOf}(I_{S_1}) \cup \mathit{parOf}(I_{S_2}) \\
&\mathit{parOf}(E) = \{\!\{E\}\!\} \\
&\mathit{parOf}(E\textbf{(identifies)}) = \{\!\{E\}\!\}
\end{aligned}
$$

The function *tempSpec* takes either an entity type declaration or a relationship type declaration as argument. If the declaration is non-temporal, it returns the empty set; and if the declaration is temporal, the specification of the required temporal support is returned.

$$
\mathit{tempSpec}(E) = \begin{cases}
T_S & \text{if } \textbf{Entity Type } E \textbf{ with } T_S \textbf{ has } A_D \in E_D \\
\emptyset & \text{if } \textbf{Entity Type } E \textbf{ has } A_D \in E_D \\
T_S \times \mathit{tempSpec}(E_2) & \text{if } \textbf{Subclass } E_1 \textbf{ of } E_2 \textbf{ with } T_S \textbf{ has } A_D \in E_D \\
\emptyset \times \mathit{tempSpec}(E_2) & \text{if } \textbf{Subclass } E_1 \textbf{ of } \textbf{ has } A_D \in E_D \\
\perp & \text{otherwise}
\end{cases}
$$

$$
\mathit{tempSpec}(R) = \begin{cases}
T_S & \text{if } \textbf{Relationship Type } R \textbf{ with } T_S \textbf{ has } A_D \textbf{ involves } I_S \in R_D \\
\emptyset & \text{if } \textbf{Relationship Type } R \textbf{ has } A_D \textbf{ involves } I_S \in R_D \\
\perp & \text{otherwise}
\end{cases}
$$

The function *ownerOf* takes as arguments the name of a weak entity type and an identifying relationship type declaration and returns the list of entity type names of the owners of the weak entity type.

$$
\begin{aligned}
&\mathit{ownerOf}(E, \textbf{Relationship Type } R \textbf{ with } T_S \textbf{ has } A_D \textbf{ involves } I_S) = \\
&\mathit{ownerOf}(E, \textbf{Relationship Type } R \textbf{ has } A_D \textbf{ involves } I_S) = \\
&\quad \begin{cases}
\mathit{parOf}(I_S) - E & \text{if } E\textbf{(identifies)} \in I_S \\
\emptyset & \text{otherwise}
\end{cases}
\end{aligned}
$$

The predicate *inSch* takes as arguments either an entity type name or a relationship type name, as well as a schema declaration. The predicate returns *true* if the entity type or the relationship type is declared in the schema and is *false* otherwise.

$$
\begin{aligned}
&\mathit{inSch}(E, Sc_D) = \mathit{inSch}(E, E_D; R_D; IC_D) = \\
&\mathit{inSch}(E, E_D) = \begin{cases}
\mathit{true} & \text{if } \textbf{Entity Type } E \textbf{ with } T_S \textbf{ has } A_D \in E_D \\
\mathit{true} & \text{if } \textbf{Entity Type } E \textbf{ has } A_D \in E_D \\
\mathit{false} & \text{otherwise}
\end{cases}
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{inSch}(R, Sc_D) = \mathit{inSch}(R, E_D; R_D; IC_C) = \\
&\mathit{inSch}(R, R_D) = \begin{cases}
\mathit{true} & \text{if } \textbf{Relationship Type } R \textbf{ with } T_S \textbf{ has } A_D \textbf{ involves } I_S \in R_D \\
\mathit{true} & \text{if } \textbf{Relationship Type } R \textbf{ has } A_D \textbf{ involves } I_S \in R_D \\
\mathit{false} & \text{otherwise}
\end{cases}
\end{aligned}
$$

The function *cnt* takes an entity, an entity type, and a relationship set as inputs and returns the number of relations in the relationship set, the entity e participates in.

$$cnt(e, E, \{t_1, \ldots, t_n\}) = \begin{cases} 0 & \text{if } n = 0 \\ cnt(e, E, \{t_1, \ldots, t_{n-1}\}) & \text{if } n \geq 1 \wedge t_n[s_E] \neq e \\ cnt(e, E, \{t_1, \ldots, t_{n-1}\}) + 1 & \text{if } n \geq 1 \wedge t_n[s_E] = e \end{cases}$$

Semantic Functions

$\mathcal{I} : E\_TYPE \rightarrow D_S^E$
$\mathcal{T} : T\_SPEC \rightarrow D_{VT} \cup D_{TT} \cup D_{LS}$
$\mathcal{A} : ATT \times DATA \times T\_SPEC \rightarrow \mathcal{D}[\![DATA]\!] \cup F(2^{\mathcal{D}[\![DATA]\!]}) \cup (\mathcal{T}[\![T_S]\!] \rightarrow \mathcal{D}[\![DATA]\!]) \cup (\mathcal{T}[\![T_S]\!] \rightarrow F(2^{\mathcal{D}[\![DATA]\!]}))$
$\mathcal{E} : E\_TYPE \times T\_SPEC \times A\_DECL \rightarrow \mathbb{A}^+ \times D_S \times \mathcal{A}[\![A_D]\!] \cup \mathbb{A}^+ \times (D_S \times \mathcal{T}[\![T_S]\!]) \times \mathcal{A}[\![A_D]\!]$
$\mathcal{R} : R\_TYPE \times I\_SPEC \times T\_SPEC \times A\_DECL \rightarrow \mathbb{A}^+ \times (D_S \times \mathcal{I}[\![I_S]\!] \times \mathcal{T}[\![T_S]\!]) \times \mathcal{A}[\![A_D]\!] \cup \mathbb{A}^+ \times (\mathcal{I}[\![I_S]\!] \times \mathcal{T}[\![T_S]\!]) \times \mathcal{A}[\![A_D]\!] \cup \mathbb{A}^+ \times \mathcal{I}[\![I_S]\!] \times \mathcal{A}[\![A_D]\!]$
$\mathcal{C} : IC_D \rightarrow PRED$
$\mathcal{S} : Sc_D \rightarrow \mathcal{S}[\![Sc_D]\!]$

$\mathcal{I}[\![I_{S_1}; I_{S_2}]\!] = \mathcal{I}[\![I_{S_1}]\!] \times \mathcal{I}[\![I_{S_2}]\!]$
$$\mathcal{I}[\![E]\!] = \begin{cases} D_S^E & \text{if } E \in E\_TYPE \\ \perp & \text{otherwise} \end{cases}$$

$\mathcal{T}[\![\textbf{with } T_{S_1}; T_{S_2}]\!] = \mathcal{T}[\![T_{S_1}]\!] \times \mathcal{T}[\![T_{S_2}]\!]$
$\mathcal{T}[\![(dim, instant, g)]\!] = D_{dim}^g$
$\mathcal{T}[\![(dim, temporal\ element, g)]\!] = 2^{D_{dim}^g}$

$\mathcal{A}[\![A_{D_1}; A_{D_2}]\!] = \mathcal{A}[\![A_{D_1}]\!] \times \mathcal{A}[\![A_{D_2}]\!]$
$\mathcal{A}[\![\textbf{Attribute } A \textbf{ is } A'_D]\!] = \mathcal{A}[\![A'_D]\!]$
$\mathcal{A}[\![\textbf{ of type } d]\!] = \mathcal{D}[\![DATA]\!](d)$
$\mathcal{A}[\![\textbf{ of type } d \textbf{ with } T_S]\!] = \mathcal{T}[\![T_S]\!] \rightarrow \mathcal{D}[\![DATA]\!](d)$
$\mathcal{A}[\![\textbf{ composite}(A_D)]\!] = \mathcal{A}[\![A_D]\!]$
$\mathcal{A}[\![\textbf{ composite}(A_D) \textbf{ with } T_S]\!] = \mathcal{T}[\![T_S]\!] \rightarrow \mathcal{A}[\![A_D]\!]$
$\mathcal{A}[\![\textbf{Multivalued of type } d]\!] = F(2^{[\![DATA]\!](d)})$
$\mathcal{A}[\![\textbf{Multivalued of type } d \textbf{ with } T_S]\!] = \mathcal{T}[\![T_S]\!] \rightarrow F(2^{[\![DATA]\!](d)})$

$\mathcal{E}[\![E_{D_1}; E_{D_2}]\!] = \mathcal{E}[\![E_{D_1}]\!] \uplus \mathcal{E}[\![E_{D_2}]\!]$

$\mathcal{E}[\![\textbf{Entity Type } E \textbf{ has } A_D]\!] =$
$\{E\} \times \{t \mid t \in FUN \wedge dom(t) = \{s, attOf(A_D)\} \wedge t[s] \in D_S \bigwedge_{A_i \in attOf(A_D)} t[A_i] \in \mathcal{A}[\![\textbf{Attribute } A_i \textbf{ is } A'_D]\!]\}$

$\mathcal{E}[\![\textbf{Entity Type } E \textbf{ with } T_S \textbf{ has } A_D]\!] =$
$\{E\} \times \{t \mid t \in FUN \wedge dom(t) = \{s, attOf(A_D)\} \wedge t[s] \in \mathcal{T}[\![T_S]\!] \rightarrow D_S \bigwedge_{A_i \in attOf(A_D)} t[A_i] \in \mathcal{A}[\![\textbf{Attribute } A_i \textbf{ is } A'_D]\!]\}$

$$\mathcal{E}[\![\textbf{Weak Entity Type } E \textbf{ has } A_D]\!] =$$
$$\{E\} \times \{t \mid t \in FUN \wedge dom(t) = \{\bigcup_{E_i \in ownerOf(E,R_D)} s_{E_i}, attOf(A_D)\}$$
$$\bigwedge_{E_i \in ownerOf(E,R_D)} t[s_{E_i}] \in \mathcal{I}[\![E_i]\!]$$
$$\bigwedge_{A_i \in attOf(A_D)} t[A_i] \in \mathcal{A}[\![\textbf{Attribute } A_i \textbf{ is } A'_D]\!]\}$$

$$\mathcal{E}[\![\textbf{Weak Entity Type } E \textbf{ with } T_S \textbf{ has } A_D]\!] =$$
$$\{E\} \times \{t \mid t \in FUN \wedge dom(t) = \{\bigcup_{E_i \in ownerOf(E,R_D)} s_{E_i}, attOf(A_D)\}$$
$$\bigwedge_{E_i \in ownerOf(E,R_D)} t[s_{E_i}] \in \mathcal{T}[\![T_S]\!] \rightarrow \mathcal{I}[\![E_i]\!]$$
$$\bigwedge_{A_i \in attOf(A_D)} t[A_i] \in \mathcal{A}[\![\textbf{Attribute } A_i \textbf{ is } A'_D]\!]\}$$

$$\mathcal{E}[\![\textbf{Subclass } E_1 \textbf{ of } E_2 \textbf{ has } A_D]\!] =$$
$$\{E_1\} \times \{t \mid t \in FUN \wedge dom(t) = \{s_{E_2}, parAtt(E_2), attOf(A_D)\}$$
$$\wedge t[s_{E_2}] \in \mathcal{T}[\![tempSpec(E_2)]\!] \rightarrow D_S^{E_2}$$
$$\bigwedge_{A_i \in parAtt(E_2)} t[A_i] \in \mathcal{A}[\![\textbf{Attribute } A_i \textbf{ is } A'_D]\!]$$
$$\bigwedge_{A_i \in attOf(A_D)} t[A_i] \in \mathcal{A}[\![\textbf{Attribute } A_i \textbf{ is } A'_D]\!]\}$$

$$\mathcal{E}[\![\textbf{Subclass } E_1 \textbf{ of } E_2 \textbf{ with } T_S \textbf{ has } A_D]\!] =$$
$$\{E_1\} \times \{t \mid t \in FUN \wedge dom(t) = \{s_{E_2}, parAtt(E_2), attOf(A_D)\} \wedge$$
$$t[s_{E_2}] \in \mathcal{T}[\![tempSpec(E_2)]\!] \times \mathcal{T}[\![T_S]\!] \rightarrow D_S^{E_2}$$
$$\bigwedge_{A_i \in parAtt(E_2)} t[A_i] \in \mathcal{A}[\![\textbf{Attribute } A_i \textbf{ is } A'_D]\!]$$
$$\bigwedge_{A_i \in attOf(A_D)} t[A_i] \in \mathcal{A}[\![\textbf{Attribute } A_i \textbf{ is } A'_D]\!]\}$$

$$\mathcal{R}[\![R_{D_1}; R_{D_2}]\!] = [\![R_{D_1}]\!] \uplus [\![R_{D_2}]\!]$$
$$\mathcal{R}[\![\textbf{Relationship Type } R \textbf{ has } A_D \textbf{ involves } I_S]\!] =$$
$$\{R\} \times \{t \mid t \in FUN \wedge dom(t) = \{\bigcup_{E_i \in parOf(I_S)} s_{E_i}, attOf(A_D)\}$$
$$\bigwedge_{E_i \in parOf(I_S)} t[s_{E_i}] \in \mathcal{I}[\![E_i]\!]$$
$$\bigwedge_{A_i \in attOf(A_D)} t[A_i] \in \mathcal{A}[\![\textbf{Attribute } A_i \textbf{ is } A'_D]\!]\}$$
$$\mathcal{R}[\![\textbf{Relationship Type } R \textbf{ with } T_S \textbf{ has } A_D \textbf{ involves } I_S]\!] =$$
$$\{R\} \times \begin{cases} \{t \mid t \in FUN \wedge dom(t) = \{\bigcup_{E_i \in parOf(I_S)} s_{E_i}, attOf(A_D)\} \\ \quad \bigwedge_{E_i \in parOf(I_S)} t[s_{E_i}] \in \mathcal{T}[\![T_S]\!] \rightarrow \mathcal{I}[\![E_i]\!] \\ \quad \bigwedge_{A_i \in attOf(A_D)} t[A_i] \in \mathcal{A}[\![\textbf{Attribute } A_i \textbf{ is } A'_D]\!]\} \text{ if } \mathcal{T}[\![T_S]\!] \notin D_{LS}^g \\ \{t \mid t \in FUN \wedge dom(t) = \{s_R, \bigcup_{E_i \in parOf(I_S)} s_{E_i}, attOf(A_D)\} \wedge \\ \quad t[s_R] \in \mathcal{T}[\![T_S]\!] \rightarrow D_S^R \bigwedge_{E_i \in parOf(I_S)} t[s_{E_i}] \in \mathcal{T}[\![T_S]\!] \rightarrow \mathcal{I}[\![E_i]\!] \\ \quad \bigwedge_{A_i \in attOf(A_D)} t[A_i] \in \mathcal{A}[\![\textbf{Attribute } A_i \textbf{ is } A'_D]\!]\} \quad \text{otherwise} \end{cases}$$

$$\mathcal{C}[\![IC_{D_1}; IC_{D_2}]\!] = \mathcal{C}[\![IC_{D_1}]\!] \wedge \mathcal{C}[\![IC_{D_2}]\!]$$

$$\mathcal{C}[\![B \textbf{ is key of } E]\!] = inSch(E, Sc_D) \wedge ((B \subseteq attOf(\textbf{Entity Type } E \textbf{ has } A_D) \wedge$$
$$\forall t_i, t_j \in \mathcal{E}[\![\textbf{Entity Type } E \textbf{ has } A_D]\!](t_i[B] = t_j[B] \Rightarrow t_i[s] = t_j[s])) \vee$$
$$(B \subseteq attOf(\textbf{Entity Type } E \textbf{ with } T_S \textbf{ has } A_D) \wedge$$
$$\forall t_i, t_j \in \mathcal{E}[\![\textbf{Entity Type } E \textbf{ with } T_S \textbf{ has } A_D]\!]$$
$$(\mathcal{T}[\![T_S]\!] \rightarrow t_i[B] = \mathcal{T}[\![T_S]\!] \rightarrow t_j[B] \Rightarrow \mathcal{T}[\![T_S]\!] \rightarrow t_i[s] = \mathcal{T}[\![T_S]\!] \rightarrow t_j[s])))$$

$\mathcal{C}[\![B$ **is partial key of** $E]\!] = inSch(E, Sc_D) \wedge$
$((B \subseteq attOf($**Weak Entity Type** E **has** $A_D) \wedge$
$\forall t_i, t_j \in \mathcal{E}[\![$**Weak Entity Type** E **has** $A_D]\!]$
$(\bigcup_{E_l \in ownerOf(E)} t_i[s_{E_l}] = \bigcup_{E_l \in ownerOf(E)} t_j[s_{E_l}] \wedge$
$t_i[B] = t_j[B] \Rightarrow t_i = t_j)) \vee$
$(B \subseteq attOf($**Weak Entity Type** E **with** T_S **has** $A_D) \wedge$
$\forall t_i, t_j \in \mathcal{E}[\![$**Weak Entity Type** E **with** T_S **has** $A_D]\!]$
$(\mathcal{T}[\![T_S]\!] \rightarrow \bigcup_{E_l \in ownerOf(E)} t_i[s_{E_l}] = \mathcal{T}[\![T_S]\!] \rightarrow \bigcup_{E_l \in ownerOf(E)} t_j[s_{E_l}] \wedge$
$\mathcal{T}[\![T_S]\!] \rightarrow t_i[B] = \mathcal{T}[\![T_S]\!] \rightarrow t_j[B] \Rightarrow \mathcal{T}[\![T_S]\!] \rightarrow t_i = \mathcal{T}[\![T_S]\!] \rightarrow t_j)))$

$\mathcal{C}[\![$**Participation of** I_S **with respect to** E **is** *disjoint*, *total*$]\!] = inSch(E)$
$\bigwedge_{E_i \in I_S} inSch(E_i) \wedge$
$\forall c^l \in \mathcal{D}_{LS}\ \forall c^t \in \mathcal{D}_{TT}(\forall t \in \bigcup_{E_i \in I_S} \mathcal{E}[\![$**Subclass** E_i **of** $E \ldots]\!]$
$\exists t' \in \mathcal{E}[\![\ldots E \ldots]\!](t[s_{E_i}] = t'[s_E]) \wedge$
$\forall t \in \mathcal{E}[\![\ldots E \ldots]\!] \exists t' \in \mathcal{E}[\![$**Subclass** E_i **of** $E \ldots]\!](t[s_E] = t'[s_{E_i}]) \wedge$
$\forall E_i, E_j \in I_S \nexists t_1 \in \mathcal{E}[\![$**Subclass** E_i **of** $E \ldots]\!]$
$t_2 \in \mathcal{E}[\![$**Subclass** E_j **of** $E \ldots]\!](i \neq j \wedge t_1[s_{E_i}] = t_2[s_{E_j}]))$

$\mathcal{C}[\![$**Participation of** I_S **with respect to** E **is** *disjoint*, *partial*$]\!] = inSch(E)$
$\bigwedge_{E_i \in I_S} inSch(E_i) \wedge$
$\forall c^l \in \mathcal{D}_{LS}\ \forall c^t \in \mathcal{D}_{TT}(\forall t \in \bigcup_{E_i \in I_S} \mathcal{E}[\![$**Subclass** E_i **of** $E \ldots]\!]$
$\exists t' \in \mathcal{E}[\![\ldots E \ldots]\!](t[s_{E_i}] = t'[s_E]) \wedge$
$\forall E_i, E_j \in I_S \nexists t_1 \in \mathcal{E}[\![$**Subclass** E_i **of** $E \ldots]\!]$
$t_2 \in \mathcal{E}[\![$**Subclass** E_j **of** $E \ldots]\!](i \neq j \wedge t_1[s_{E_i}] = t_2[s_{E_j}]))$

$\mathcal{C}[\![$**Participation of** I_S **with respect to** E **is** *overlapping*, *total*$]\!] = inSch(E)$
$\bigwedge_{E_i \in I_S} inSch(E_i) \wedge$
$\forall c^l \in \mathcal{D}_{LS}\ \forall c^t \in \mathcal{D}_{TT}(\forall t \in \bigcup_{E_i \in I_S} \mathcal{E}[\![$**Subclass** E_i **of** $E \ldots]\!]$
$\exists t' \in \mathcal{E}[\![\ldots E \ldots]\!](t[s_{E_i}] = t'[s_E]) \wedge$
$\forall t \in \mathcal{E}[\![\ldots E \ldots]\!] \exists t' \in \mathcal{E}[\![$**Subclass** E_i **of** $E \ldots]\!](t[s_E] = t'[s_{E_i}]))$

$\mathcal{C}[\![$**Participation of** I_S **with respect to** E **is** *overlapping*, *partial*$]\!] = inSch(E)$
$\bigwedge_{E_i \in I_S} inSch(E_i) \wedge$
$\forall c^l \in \mathcal{D}_{LS}\ \forall c^t \in \mathcal{D}_{TT}(\forall t \in \bigcup_{E_i \in I_S} \mathcal{E}[\![$**Subclass** E_i **of** $E \ldots]\!]$
$\exists t' \in \mathcal{E}[\![\ldots E \ldots]\!](t[s_{E_i}] = t'[s_E]))$

$\mathcal{C}[\![$**Snapshot participation of** E **in** R **is** $(min, max)]\!] = inSch(E, Sc_D) \wedge inSch(R, Sc_D) \wedge$
$(E \in parOf($**Relationship Type** R **with** T_S **has** A_D **involves** $I_S) \vee$
$\in parOf($**Relationship Type** R **has** A_D **involves** $I_S)) \wedge$
$(\forall e_j \in D_S^E(min \leq cnt(e_j, E,$
$\mathcal{R}[\![$**Relationship Type** R **has** A_D **involves** $I_S]\!]) \leq max)) \vee$
$(\forall c \in \mathcal{T}[\![tempSpec(R)]\!]\ \forall e_j \in D_S^E$
$(min \leq cnt(e_j, E,$
$\mathcal{R}[\![$**Relationship Type** R **with** T_S **has** A_D **involves** $I_S]\!]) \leq max))$

$\mathcal{C}[\![$**Lifespan participation of** E **in** R **is** $[min, max]]\!] = inSch(E, Sc_D) \wedge inSch(R, Sc_D) \wedge$
$E \in (parOf($**Relationship Type** R **with** T_S **has** A_D **involves** $I_S) \wedge$
$\forall e_j \in D_S^E(min \leq cnt(e_j, E,$
$\mathcal{R}[\![$**Relationship Type** R **with** T_S **has** A_D **involves** $I_S]\!]) \leq max))$

$$\mathcal{S}[\![Sc_D]\!] = \mathcal{S}[\![E_D; R_D; IC_D]\!]$$
$$\mathcal{S}[\![E_D; R_D; IC_D]\!] = \mathcal{E}[\![E_D]\!] \uplus \mathcal{R}[\![R_D]\!] \uplus \mathcal{C}[\![IC_D]\!]$$