Evolving Objects in Temporal Information Systems

Alessandro Artale [a] Christine Parent [b] Stefano Spaccapietra [c]

[a] *Faculty of Computer Science, Free University of Bolzano, I;*
E-mail: artale@inf.unibz.it
[b] *HEC/ISI, Université de Lausanne, CH;*
E-mail: christine.parent@unil.ch
[c] *Database Laboratory, Ecole Polytechnique Fédérale Lausanne, CH;*
E-mail: stefano.spaccapietra@epfl.ch

This paper presents a semantic foundation of temporal conceptual models used to design temporal information systems. We consider a modelling language able to express both timestamping and evolution constraints. We conduct a deeper investigation of evolution constraints, eventually devising a model-theoretic semantics for a full-fledged model with both timestamping and evolution constraints. The proposed formalization is meant both to clarify the meaning of the various temporal constructors that appeared in the literature and to give a rigorous definition, in the context of temporal information systems, to notions like satisfiability, subsumption and logical implication. Furthermore, we show how to express temporal constraints using a subset of first-order temporal logic, i.e. $\mathcal{DLR}_{\mathcal{US}}$, the description logic $\mathcal{DLR}$ extended with the temporal operators *Since* and *Until*. We show how $\mathcal{DLR}_{\mathcal{US}}$ is able to capture the various modelling constraints in a succinct way and to perform automated reasoning on temporal conceptual models.

Keywords: Temporal Data Models, Description Logics.

AMS Subject classification: computer science, knowledge representation, database theory

1. Introduction

Most of information modelling research and practice focus on a static view of the world, describing data as it should be and is day by day. Current data models and database systems are meant to capture snapshots of the world, i.e. the current state of the database, with the next snapshot replacing the previous one. Yet everybody is well aware that such an approach only gives a very partial view of the world, neglecting another essential component, its dynamics, i.e. how the world evolves as time passes. Recording the current, past, and possibly predicted future snapshots is the very first step towards capturing evolution. This functionality is supported by temporal information systems. Data warehousing systems, based on keeping aggregates of past snapshots, have extensively shown that keeping knowledge over time entails the possibility, for example, to analyze evolution trends and develop scenarios for the future. Such analysis and forecasting are fundamental components of most decision-making processes, which are critical for successfully facing the complexity of today's activities. A second step in

capturing evolution is enforcing the rules that govern the evolution of data. Rules play a fundamental role to maintain data consistency. In data modelling, evolution rules are expressed as evolution constraints, allowing to control requested changes to data and reject those changes that can be recognized as incorrect (i.e. leading to a new state that is inconsistent with the previous ones) or inappropriate (e.g. changes requested at a time they are not allowed) or detected as suspicious (e.g. resulting in an anomalous evolution that requires additional validation procedures). Further steps to enrich evolution management are possible, such as, for example, capturing the reasons for change (why the change happened), the actors of change (who prompted the change), its timing (when did it happen), as well as any other information related to the change (which at this point becomes like an object of interest from the data management viewpoint).

Knowledge of dynamics is intrinsically related to time awareness. Capturing dynamics is grounded in the ability to capture time, as proposed by research on temporal databases[1]. Abstracting from many details, the most popular time awareness mechanism is timestamping. From the evolution management viewpoint, timestamping supports the first step above, capturing evolution as a sequence of snapshots. Formal semantics approaches have extensively discussed timestamping [3,4,12,19,27,40]. Yet a clear formalization of evolution constraints (supporting the second step) is still missing, despite the fact that in the literature such constraints have largely been advocated as useful for modelling the behavior of temporal objects [4,21,22,29,31,32,37,35,39,40].

Our research aims at defining a conceptual temporal data model, the semantics of which is formally defined by grounding it on the results of temporal logics. We eventually devise a model-theoretic semantics for a full-fledged conceptual model with both timestamping and evolution constraints. This paper focuses on evolution constraints, more precisely on providing a formal semantics to describe how objects can evolve in their lifecycle and how they can be related throughout time. The formalization proposed here builds on previous efforts to formalize temporal conceptual models. Namely, we rely on a previous work to define the $\mathcal{ER}_{VT}$ model [4], a temporal Extended Entity-Relationship[2] (EER) model equipped with both a textual and a graphical syntax and based on a model-theoretic semantics. $\mathcal{ER}_{VT}$ captures timestamping constructors along with transition constraints. The work reported in this paper extends $\mathcal{ER}_{VT}$ with new functionality (hereinafter defined) for evolutionary modelling, namely *status classes*, *generation relationships* and *across-time relationships*. Another closely related work is the one of Finger and McBrien [16]. They propose a model-theoretic formalization for the ERT model, an EER model with timestamping and across-time relationships (called H-marked relationships by the authors and introduced in a previous paper by McBrien, Seltveit and Wrangler [31]). Our proposal modifies the semantics of across-time re-

[1] Outcomes in this domain include many proposals for modelling temporal data, and a large body of consensus on the fundamental underlying concepts. Readers interested in analyses of state of art in temporal modelling and of results achieved in the area are referred to two surveys [20,26] that still provide valuable information.

[2] EER denotes data models that enrich the standard ER modelling language with ISA links, disjoint and covering constraints, and full cardinality constraints—with cardinality (0,n) assumed by default.

lationships as presented in [16] to comply with a crucial modelling requirement, i.e. snapshot reducibility [27].

The advantage of associating a set-theoretic semantics to a language is not only to clarify the meaning of the language's constructors but also to give a semantic definition to relevant modelling notions. In particular, given an interpretation function to assign a set-theoretic semantics to the (temporal) modelling constructors, we are able to give a rigorous definition of the notions of: *schema satisfiability* when a schema admits a non empty interpretation which guarantees that the constraints expressed by the schema are not contradictory (similarly we define the notions of class and relationships satisfiability); *subsumption* between classes (relationships) when the interpretations of a class (relationships) is a subset of the interpretation of another class (relationships) which allows to check new ISA links; *logical implication* when a (temporal) constraint is implicitly true in the current schema thus deriving new constraints. In particular, in this paper we stress both the formalization of the constructors and the set of logical implications associated to such formalization. The obtained logical implications are generally in agreement with those mentioned in the literature on temporal conceptual models. Thus, each constructor's formalization (together with its associated logical implications) can be seen as a set of precise rules on the allowed behavior of objects, in particular regarding their evolution in time. Even if we do not address specific implementation issues, these rules can be turned into explicit integrity constraints in the form of trigger rules to be added to the schema specified by the database designer, thus enabling to check the validity of user actions involving object evolution. Since the rules are the result of a formal characterization we solve what is in our opinion a serious weakness of existing modelling approaches, i.e. without a rigorous foundation there is no guarantee that the proposed model leads to a sound system.

Finally, as a byproduct of the semantic formalization, we also show how (temporal) modelling constraints can be equivalently expressed by using a subset of first-order temporal logic, i.e. the temporal description logic $\mathcal{DLR}_{\mathcal{US}}$ [5]. $\mathcal{DLR}_{\mathcal{US}}$ is a combination of the expressive and decidable description logic $\mathcal{DLR}$ (a description logic with n-ary relationships) with the linear temporal logic with temporal operators *Since* ($\mathcal{S}$) and *Until* ($\mathcal{U}$) which can be used in front of both classes and relations. The choice of extending $\mathcal{DLR}$ is motivated by its ability to give a logical reconstruction and an extension of representational tools such as object-oriented and conceptual data models, frame-based and web ontology languages [8–11,24]. In this paper, we use $\mathcal{DLR}_{\mathcal{US}}$ both to capture the (temporal) modelling constructors in a succinct way, and to use reasoning techniques to check satisfiability, subsumption and logical implication. We show how $\mathcal{DLR}_{\mathcal{US}}$ axioms capture the above mentioned rules associated with each constructor's formal semantics while logical implications between $\mathcal{DLR}_{\mathcal{US}}$ axioms is a way to derive new rules. The mapping towards description logics presented in this paper builds on top of a mapping which has been proved correct in [3,4] while complexity results and algorithmic techniques can be found in [5,2,6]. Even if full $\mathcal{DLR}_{\mathcal{US}}$ is undecidable this paper addresses interesting modelling scenarios where subsets of the full $\mathcal{DLR}_{\mathcal{US}}$ logic is needed and where reasoning becomes a decidable problem.

The paper is organized as follows. Section 2 discusses in more details the main components for managing the dynamics of data, timestamping and evolution constraints, with a particular emphasis on object migration. Section 3 shows the modelling requirements that lead us in elaborating the rigorous definition of our evolution framework. Sections 4 and 5 recall the characteristics of the $\mathcal{DLR}_{\mathcal{US}}$ description logic and the $\mathcal{ER}_{VT}$ temporal data model on which we build our proposal. Section 6 presents the modelling of timestamping constraints as provided in $\mathcal{ER}_{VT}$. Section 7 discusses the evolution constraints we address and provides a formal characterization for them together with a set of logical implications and the correspondent $\mathcal{DLR}_{\mathcal{US}}$ axioms. Section 8 surveys the complexity results for reasoning over temporal models showing that reasoning on the full-fledged temporal setting is undecidable and providing useful scenarios where reasoning becomes decidable. Section 9 concludes the paper.

2. Recording and Controlling Evolution

As stated in the introduction, evolution management requires first to be able to record the different states of the database over time, second to be able to automatically check that each operation resulting in a change conforms to the rules that constrain permissible evolutions. In this Section we analyze the supporting techniques to achieve such functionality, i.e. timestamping and evolution constraints. The analysis is at the conceptual modelling level. Implementation aspects are not an issue in this paper.

2.1. Timestamping

Timestamping is a temporal marking mechanism that, according to some criterion (e.g. valid time or transaction time [25,41]), positions data relevance on a timescale. Hereinafter we only consider valid time (i.e. temporal references driven by the application view of evolution), which characterizes the vast majority of application requirements. Timestamping provides the following functionality:

- Attribute timestamping: Evolution of values.
  The most well known aspect of timestamping is its association with attribute values to keep the evolution of these values in time. For example, timestamping allows keeping the knowledge that the affiliation attribute for an employee *emp-123* has value "*University of Paris*" for the period from 10/1969 to 9/1983, then "*University of Dijon*" from 10/1983 to 9/1988, then "*EPFL*" from 10/1988 to 2/2010. Timestamped attributes are also called time-varying attributes. Research on temporal databases has extensively investigated how attribute timestamping can be defined and implemented using various data models (e.g. relational, entity-relationship).
- Object and relationship timestamping: Lifecycle.
  Similarly, temporal periods can characterize an object or relationship instance as a whole rather than through its attributes. Here, it is its membership in a class that is split into periods according to a given classification criterion. For example, existence

of an employee object in the `Employee` class can include periods where the object is an active member of the class (e.g. the employee is currently on payroll), periods where its membership is suspended (e.g. the employee is on temporary leave), and a period where its membership is disabled (e.g. the employee has left the company) [15]. These periods together form the *lifecycle* [37] of the object/relationship instance in a given class/relationship (more details are given in Section 7.1). The lifecycle of an object/relationship instance is a set of time instants corresponding to those instants where the instance belongs to the class or relationship. Instances with lifecycle are called temporal instances. It is worth stressing that, from a conceptual viewpoint, a real world object may simultaneously qualify in the database for membership into several classes, typically within the same is-a hierarchy (e.g. *Paul Carlton* can be seen as simultaneously belonging to three classes, the `Person` class, the `Employee` class, and the `Manager` class) and consequently hold a different lifecycle in each class. For example, the lifecycle of *Paul* as a manager obviously covers a lifespan included in the one of his lifecycle as an employee, which in turn is a subset of his lifespan as a person (the lifespan inclusion is due to the semantics of the is-a link between `Manager` and `Employee` and between `Employee` and `Person`).

Timestamping (both as time-varying attributes and lifecycles) is obviously optional, i.e. a data model should allow for both temporal and atemporal modelling constructors. Section 6 hereinafter shows how objects, attributes and relationships timestamping has been covered in $\mathcal{ER}_{VT}$. Coverage of object (and relationship) lifecycle is the $\mathcal{ER}_{VT}$ extension we introduce in this paper (see Section 7).

2.2. Evolution Constraints

Timestamping enriches the static view of data by allowing recording the states of the database over a period of time. A temporal database (i.e. a database equipped with timestamps) may indeed be seen as a sequence of snapshots, one per instant as defined by the smallest time granularity. Evolution constraints are imposed on a temporal database to control the mechanism that rules dynamic aspects, i.e. what are the permissible transitions from one state of the database to the next one. Integrity constraints in general can be as complex as needed to express rules on application data. Data models embed some predefined kinds of static integrity constraints (e.g. uniqueness specifications, cardinality constraints) but very rarely include constructs to express dynamic constraints. On the other hand, during modification operations, SQL triggers provide a construct to compare the new value replacing the existing value, thus enforcing an evolutionary constraint. For full expressiveness, integrity constraint definition languages are usually grounded on first order logic. In this paper we will show how evolution constraints can be expressed using a temporal description logic (here we propose the use of the temporal description logic $\mathcal{DLR}_{\mathcal{US}}$). In the following, we summarize the main features of evolution constraints as appeared in the literature.

- Applied to attributes, they are known as *dynamic integrity constraints* [13]. One

example, simply using arithmetic comparison operators, is the constraint that the value for the `Salary` attribute of an employee can only increase. A second example is the constraint that the number of values for a multivalued attribute, e.g. the `Diplomas` of a person, can only increase over time. In the latter case we say that the attribute is *expanding* meaning that the deletion of values is not allowed. Dynamic integrity constraints can be expressed in $\mathcal{ER}_{VT}$ thanks to the temporal constructs included in the model.

- Applied to the lifecycle of an object (relationship instance), evolution constraints are referred to as *status constraints*. They rule the permissible evolution of an instance's membership in a class/relationships along its lifespan. For example, an object that is an active member of a class may become a disabled member of the class, but not vice versa [15]. The different statuses (*scheduled, active, suspended, disabled*) entail different constraints. For example, in [15] if an object is in the suspended status the values of its attributes within the suspension period can be retrieved but cannot be modified. We further discuss status issues in Section 7.1.
- Applied to objects, evolution constraints are referred to as *transition constraints* and usually rule the evolution of an object from being member of a class to being member of another class [22]. For example, an object in the `Student` class may migrate to become an object of the `Faculty` class or evolve to also become an object of the `Alumnus` class. Conversely, an object now in the `FullProfessor` class cannot become later an object in the `AssistantProfessor` class. The next Section presents an extensive survey of related works on object migration while Section 7.2 shows a formalization with a temporal semantics and corresponding axioms in description logic.
- Finally, evolution constraints may be embedded in relationships. Evolution-related knowledge may indeed be conveyed through relationships associated to a specific evolutionary semantics. *Generation relationships* [21] between objects of class `A` and objects of class `B` (possibly equal to `A`) describe the fact that objects in `B` are generated by objects in `A`. For example, in a company database, the splitting of a department translates into the fact that the original department generates two (or more) new departments. Generation relationships allow backtracking the history of an object and its provenance. Genealogical search is an example of popular backtracking, supported by parenthood relationships holding generation semantics. Clearly, if `A` and `B` are temporal classes, a generation relationship with source `A` and target `B` entails that the lifecycle of a `B` object cannot start before the lifecycle of the related `A` object. This particular temporal framework (where related objects do not coexist in time) also applies to relationships other than generation relationships, for example when linking a historian to her favorite historical figure of the past. In our temporal approach, relationships are referred to as *across-time relationships* to emphasize the characteristic feature that they come with no implicit temporal constraints. This contrasts with most temporal data models, where relationships between temporal classes implicitly enforce an overlap synchronization constraint, as the rules of these models state that

the lifecycle of a relationship must be within the intersection of the lifecycle of the related objects. While this may indeed correspond to the requirements of a specific application, it cannot be stated as true in the general case. In our approach relationships may link simultaneously existing objects as well as objects whose existences are disjoint. They may also link a temporal object with an atemporal one. Across-time relationships may bear temporal constraints. Generation relationships can be understood as an across-time relationship with a particular temporal constraint (see Sections 7.3-7.4 for more details).

2.3. Object Migration

As we stated above, evolution control includes expressing constraints on object migration. In this paper, starting from a temporal set-theoretic semantics, we aim at formally capturing such constraints thus characterizing the notion of object migration (see Section 7.2). With the generic term *transition constraints* we denote the set of constraints that deal with the various cases of migration. We also introduce a visual representation as a link from the source class of the migration to the corresponding target class.

Object migration has been addressed by many authors, in particular from the object-oriented database community, maybe as a natural follow up on its focus on capturing object behavior through methods. In classical object-oriented database approaches the generalization hierarchy is static (i.e., objects do not migrate) and supports only one instantiation per real world entity (i.e., the entity is instantiated in the most specific class it belongs to). To introduce more flexibility new modeling constructs and rules have been proposed. To model the dynamics of a migration and to support *multi-instantiation* two new kinds of classes, *base class* and *role class*, have been introduced. The first appearance of an object is created in the "base" class, while migration to "role" classes is captured by new instantiations of the same object in such "role" classes. Objects cannot be deleted in the base class as long as they are still represented in a role class. For example, once created as an instance of a class `Person`, more instances of this object may be created (in parallel or in some given sequence) as needed in related classes such as `Student`, `Employee`, `Manager`, `Sportsman`, etc. Foundation work on objects and their roles has been conducted in [36,18] where an object can also be instantiated several times as different instances of the same role. This allows representing, for example, a person who registers twice as a student (in two different institutions). In [29] an object can simultaneously migrate to several target classes. The work in [34] deviates from traditional static typing of classes and introduces a new approach where role classes can be created and deleted dynamically during application execution. In our work we consider all multi-instantiation classes as role classes. If a base class is needed by an application, it can be specified using the expressive power of the temporal description logic to enforce the temporal integrity constraints between a base and a role class.

An additional requirement for role classes is to accept in the same role class, instances coming from disjoint classes. For example, a `Car-Owner` role class may be

populated with instances coming from either of the two disjoint classes, `Person` and `Company`—both companies and persons may own cars. The *category concept* in [14] was proposed to cope with this situation. This requirement is easier to achieve in those proposals that do not require the existence of a base object class (see e.g. [28,30]). In these models the role class concept replaces the object class concept. Objects can enter the database through creation in any of the roles that accept creation operations, and then move around according to inter-role links (which can be bi-directional or not depending on application constraints). This is also our approach.

An alternative to the explicit specification of inter-role links is the definition of *membership predicates* associated to an object/role class. This allows for the automatic acquisition of new roles: when an object instance is modified, its new values are compared with the membership predicates and whenever a predicate is satisfied the instance is classified as a member of that population [33,34]. Predicates may also be checked on demand rather than automatically after a modification. Inference rules may be associated to each object/role class, specifying which other class may or may not be populated by an instance migrating or being generated from this class [36,34,30].

In short, our approach to supporting object migration does not make use of the notions of base and role classes. To deal with object migration we rely on a temporal semantics: objects are identified by OID's (Objects Identifiers) in a unique way trough their existence but the same OID can belong to different classes at different time points. Such a modelling assumption allows us to capture also migration cases where source and target classes do not belong to the same generalization hierarchy. Furthermore, the logic-based context of our work allows us to use reasoning services (in particular, the instance checking reasoning service in description logics) on static and dynamic membership predicates to classify an object in different classes at different times.

Another issue concerning object migration is to handle the typing conflicts that may arise when an object migrates from a class to another class with different typing constraints. This issue is beyond the scope of this paper and is mentioned here only to provide the reader with a broader view on object migration. In [32] an adaptation process that automatically solves typing conflicts is specified. In particular each inter-object reference is updated: whenever an object migrates, all references to it in the original class have to be changed to refer to the object in the new class. The update process may generate new migrations; therefore it needs to be controlled to avoid overflowing update propagation. Similarly, transformation functions associated to migration paths are proposed in [28] to compute both the new values and the new structure for the target instance starting from one or more source instances. On the other hand, preservation of information for non-conflicting situations (e.g. an attribute that is both in the source and in the target class) is analyzed in [29].

As another example, [39] focuses on characterizing object migration patterns under a given set of parameterized transactions rules governing the database state evolution as a consequence of some manipulation operations. A migration is said to be consistent if it respects a set of specified migration patterns. Three update languages (SL, CSL+ and CSL) are analyzed—each one supporting different hypotheses on their data manip-

ulation primitives—to formally specify the interrelationship between the languages and the migration patterns they support. Finally, an extended coverage of characteristics of object migration can be found in [29].

These last related works mainly deal with languages to manipulate migrating objects. On the other hand, this paper addresses the semantic foundation together the basic requirements and constraints for transition modelling. The reader should be aware that modelling and specification is just one aspect of object migration: issues related to manipulation languages are orthogonal to the modelling aspects we deal with and thus are beyond the scope of this paper.

3. Modelling Requirements

This Section briefly illustrates the requirements that are frequently advocated in the literature on temporal data models when dealing with temporal constraints.

- Orthogonality. Temporal constructors should be specified separately and independently for classes, relationships, and attributes. Depending on application requirements, the temporal support must be decided by the designer.
- Upward Compatibility. This term denotes the capability of preserving the nontemporal semantics of conventional (legacy) conceptual schemas when embedded into temporal schemas.
- Snapshot Reducibility. Snapshots of the database described by a temporal schema are the same as the database described by the same schema, where all temporal constructors are eliminated and the schema is interpreted atemporally. Indeed, this property specifies that we should be able to fully rebuild a temporal database by starting from the single snapshots.

These requirements are not so obvious when dealing with evolving objects. The formalization carried out in this paper provides a data model able to respect these requirements also in presence of evolving objects. In particular, orthogonality affects mainly timestamping [37] and our formalization satisfies this principle by introducing temporal marks that could be used to specify the temporal behavior of classes, relationships, and attributes in an independent way (see Section 6). Upward compatibility and snapshot reducibility [27] are strictly related. Considered together, they allow to preserve the meaning of atemporal constructors. In particular, the meaning of classical constructors must be preserved in such a way that a designer could either use them to model classical databases, or when used in a genuine temporal setting their meaning must be preserved at each instant of time. We enforce upward compatibility by using global timestamps over legacy constructors (see Section 6). Snapshot reducibility is hard to preserve when dealing with both generation and across-time relationships where involved object may not coexist. We enforce snapshot reducibility by a particular treatment of relationship typing (see Sections 7.4,7.3).

$$
\begin{aligned}
C \rightarrow\ & \top \mid \bot \mid CN \mid \neg C \mid C_1 \sqcap C_2 \mid \exists^{\leqslant k}[U_j]R \mid \\
& \Diamond^+ C \mid \Diamond^- C \mid \Box^+ C \mid \Box^- C \mid \oplus C \mid \ominus C \mid C_1 \,\mathcal{U}\, C_2 \mid C_1 \,\mathcal{S}\, C_2 \\
R \rightarrow\ & \top_n \mid RN \mid \neg R \mid R_1 \sqcap R_2 \mid U_i/n : C \mid \\
& \Diamond^+ R \mid \Diamond^- R \mid \Box^+ R \mid \Box^- R \mid \oplus R \mid \ominus R \mid R_1 \,\mathcal{U}\, R_2 \mid R_1 \,\mathcal{S}\, R_2
\end{aligned}
$$

$$
\begin{aligned}
\top^{\mathcal{I}(t)} &= \Delta^{\mathcal{I}} \\
\bot^{\mathcal{I}(t)} &= \emptyset \\
CN^{\mathcal{I}(t)} &\subseteq \top^{\mathcal{I}(t)} \\
(\neg C)^{\mathcal{I}(t)} &= \top^{\mathcal{I}(t)} \setminus C^{\mathcal{I}(t)} \\
(C_1 \sqcap C_2)^{\mathcal{I}(t)} &= C_1^{\mathcal{I}(t)} \cap C_2^{\mathcal{I}(t)} \\
(\exists^{\leqslant k}[U_j]R)^{\mathcal{I}(t)} &= \{\, d \in \top^{\mathcal{I}(t)} \mid \sharp\{\langle d_1, \ldots, d_n\rangle \in R^{\mathcal{I}(t)} \mid d_j = d\} \leqslant k\} \\
(C_1 \,\mathcal{U}\, C_2)^{\mathcal{I}(t)} &= \{\, d \in \top^{\mathcal{I}(t)} \mid \exists v > t.(d \in C_2^{\mathcal{I}(v)} \wedge \forall w \in (t,v).d \in C_1^{\mathcal{I}(w)})\} \\
(C_1 \,\mathcal{S}\, C_2)^{\mathcal{I}(t)} &= \{\, d \in \top^{\mathcal{I}(t)} \mid \exists v < t.(d \in C_2^{\mathcal{I}(v)} \wedge \forall w \in (v,t).d \in C_1^{\mathcal{I}(w)})\} \\
(\top_n)^{\mathcal{I}(t)} &\subseteq (\Delta^{\mathcal{I}})^n \\
RN^{\mathcal{I}(t)} &\subseteq (\top_n)^{\mathcal{I}(t)} \\
(\neg R)^{\mathcal{I}(t)} &= (\top_n)^{\mathcal{I}(t)} \setminus R^{\mathcal{I}(t)} \\
(R_1 \sqcap R_2)^{\mathcal{I}(t)} &= R_1^{\mathcal{I}(t)} \cap R_2^{\mathcal{I}(t)} \\
(U_i/n : C)^{\mathcal{I}(t)} &= \{\, \langle d_1, \ldots, d_n\rangle \in (\top_n)^{\mathcal{I}(t)} \mid d_i \in C^{\mathcal{I}(t)}\} \\
(R_1 \,\mathcal{U}\, R_2)^{\mathcal{I}(t)} &= \{\, \langle d_1, \ldots, d_n\rangle \in (\top_n)^{\mathcal{I}(t)} \mid \\
&\qquad \exists v > t.(\langle d_1, \ldots, d_n\rangle \in R_2^{\mathcal{I}(v)} \wedge \forall w \in (t,v).\langle d_1, \ldots, d_n\rangle \in R_1^{\mathcal{I}(w)})\} \\
(R_1 \,\mathcal{S}\, R_2)^{\mathcal{I}(t)} &= \{\, \langle d_1, \ldots, d_n\rangle \in (\top_n)^{\mathcal{I}(t)} \mid \\
&\qquad \exists v < t.(\langle d_1, \ldots, d_n\rangle \in R_2^{\mathcal{I}(v)} \wedge \forall w \in (v,t).\langle d_1, \ldots, d_n\rangle \in R_1^{\mathcal{I}(w)})\} \\
(\Diamond^+ R)^{\mathcal{I}(t)} &= \{\langle d_1, \ldots, d_n\rangle \in (\top_n)^{\mathcal{I}(t)} \mid \exists v > t.\, \langle d_1, \ldots, d_n\rangle \in R^{\mathcal{I}(v)}\} \\
(\oplus R)^{\mathcal{I}(t)} &= \{\langle d_1, \ldots, d_n\rangle \in (\top_n)^{\mathcal{I}(t)} \mid \langle d_1, \ldots, d_n\rangle \in R^{\mathcal{I}(t+1)}\} \\
(\Diamond^- R)^{\mathcal{I}(t)} &= \{\langle d_1, \ldots, d_n\rangle \in (\top_n)^{\mathcal{I}(t)} \mid \exists v < t.\, \langle d_1, \ldots, d_n\rangle \in R^{\mathcal{I}(v)}\} \\
(\ominus R)^{\mathcal{I}(t)} &= \{\langle d_1, \ldots, d_n\rangle \in (\top_n)^{\mathcal{I}(t)} \mid \langle d_1, \ldots, d_n\rangle \in R^{\mathcal{I}(t-1)}\}
\end{aligned}
$$

Figure 1. Syntax and semantics of $\mathcal{DLR}_{\mathcal{US}}$.

4. The Temporal Description Logic

The temporal description logic $\mathcal{DLR}_{\mathcal{US}}$ [5] combines the propositional temporal logic with *Since* and *Until* and the (non-temporal) description logic $\mathcal{DLR}$ [9,7]. $\mathcal{DLR}_{\mathcal{US}}$ can be regarded as a rather expressive fragment of the first-order temporal logic $L^{\{\textbf{since, until}\}}$ (cf. [12,23]).

The basic syntactical types of $\mathcal{DLR}_{\mathcal{US}}$ are *classes* (i.e. unary predicates, also known as *concepts*) and n-ary *relations* of arity ≥ 2. Starting from a set of *atomic classes* (denoted by CN), a set of *atomic relations* (denoted by RN), and a set of *role symbols* (denoted by U) we hereinafter define inductively (complex) class and relation expressions as is shown in the upper part of Figure 1, where the binary constructors ($\sqcap, \sqcup, \mathcal{U}, \mathcal{S}$) are applied to relations of the same arity, i, j, k, n are natural numbers, $i \leq n$, and j does not exceed the arity of R.

The non-temporal fragment of $\mathcal{DLR}_{\mathcal{US}}$ coincides with $\mathcal{DLR}$. For both class and relation expressions all the Boolean constructors are available. The selection expression

$U_i/n : C$ denotes an n-ary relation whose argument named U_i ($i \leq n$) is of type C; if it is clear from the context, we omit n and write $(U_i : C)$. The projection expression $\exists^{\leqslant k}[U_j]R$ is a generalisation with cardinalities of the projection operator over the argument named U_j of the relation R; the plain classical projection is $\exists^{\geq 1}[U_j]R$. It is also possible to use the pure argument position version of the language by replacing role symbols U_i with the corresponding position numbers i. To show the expressive power of $\mathcal{DLR}_{\mathcal{US}}$ we refer to the next Sections where $\mathcal{DLR}_{\mathcal{US}}$ is used to capture various forms of temporal constraints.

The model-theoretic semantics of $\mathcal{DLR}_{\mathcal{US}}$ assumes a flow of time $\mathcal{T} = \langle \mathcal{T}_p, < \rangle$, where $\mathcal{T}_p$ is a set of time points (or chronons) and $<$ a binary precedence relation on $\mathcal{T}_p$, is assumed to be isomorphic to $\langle \mathbb{Z}, < \rangle$. The language of $\mathcal{DLR}_{\mathcal{US}}$ is interpreted in *temporal models* over $\mathcal{T}$, which are triples of the form $\mathcal{I} \doteq \langle \mathcal{T}, \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}(t)} \rangle$, where $\Delta^{\mathcal{I}}$ is non-empty set of objects (the *domain* of $\mathcal{I}$) and $\cdot^{\mathcal{I}(t)}$ an *interpretation function* such that, for every $t \in \mathcal{T}$ (in the following the notation $t \in \mathcal{T}$ is used as a shortcut for $t \in \mathcal{T}_p$), every class C, and every n-ary relation R, we have $C^{\mathcal{I}(t)} \subseteq \Delta^{\mathcal{I}}$ and $R^{\mathcal{I}(t)} \subseteq (\Delta^{\mathcal{I}})^n$. The semantics of class and relation expressions is defined in the lower part of Figure 1, where $(u, v) = \{w \in \mathcal{T} \mid u < w < v\}$. For classes, the temporal operators $\Diamond^+$ (some time in the future), $\oplus$ (at the next moment), and their past counterparts can be defined via $\mathcal{U}$ and $\mathcal{S}$: $\Diamond^+ C \equiv \top \,\mathcal{U}\, C$, $\oplus C \equiv \bot \,\mathcal{U}\, C$, etc. The operators $\Box^+$ (always in the future) and $\Box^-$ (always in the past) are the duals of $\Diamond^+$ (some time in the future) and $\Diamond^-$ (some time in the past), respectively, i.e. $\Box^+ C \equiv \neg\Diamond^+\neg C$ and $\Box^- C \equiv \neg\Diamond^-\neg C$, for both classes and relations. The operators $\Diamond^*$ (at some moment) and its dual $\Box^*$ (at all moments) can be defined for both classes and relations as $\Diamond^* C \equiv C \sqcup \Diamond^+ C \sqcup \Diamond^- C$ and $\Box^* C \equiv C \sqcap \Box^+ C \sqcap \Box^- C$, respectively.

A *knowledge base* is a finite set Σ of $\mathcal{DLR}_{\mathcal{US}}$ axioms of the form $C_1 \sqsubseteq C_2$ and $R_1 \sqsubseteq R_2$, with R_1 and R_2 being relations of the same arity. An interpretation $\mathcal{I}$ satisfies $C_1 \sqsubseteq C_2$ ($R_1 \sqsubseteq R_2$) if and only if the interpretation of C_1 (R_1) is included in the interpretation of C_2 (R_2) at all time, i.e. $C_1^{\mathcal{I}(t)} \subseteq C_2^{\mathcal{I}(t)}$ ($R_1^{\mathcal{I}(t)} \subseteq R_2^{\mathcal{I}(t)}$), for all $t \in \mathcal{T}$. Various *reasoning services* can be defined in $\mathcal{DLR}_{\mathcal{US}}$. A knowledge base, Σ, is *satisfiable* if there is an interpretation that satisfies all the axioms in Σ (in symbols, $\mathcal{I} \models \Sigma$). A knowledge base, Σ, *logically implies* an axiom, $C_1 \sqsubseteq C_2$ ($R_1 \sqsubseteq R_2$), and write $\Sigma \models C_1 \sqsubseteq C_2$ ($\Sigma \models R_1 \sqsubseteq R_2$), if we have $\mathcal{I} \models C_1 \sqsubseteq C_2$ ($\mathcal{I} \models R_1 \sqsubseteq R_2$) whenever $\mathcal{I} \models \Sigma$. In this latter case, the class C_1 (relation R_1) is said to be *subsumed* by the class C_2 (relation R_2) in the knowledge base Σ. A class C is satisfiable, given a knowledge base Σ, if there exists a model $\mathcal{I}$ of Σ such that $C^{\mathcal{I}(t)} \neq \emptyset$ for some $t \in \mathcal{T}$, i.e. $\Sigma \not\models C \sqsubseteq \bot$. A relation R is satisfiable, given a knowledge base Σ, if there exists a model $\mathcal{I}$ of Σ such that $R^{\mathcal{I}(t)} \neq \emptyset$ for some $t \in \mathcal{T}$, i.e. $\Sigma \not\models R \sqsubseteq \bot$. Finally, knowledge base satisfiability, class subsumption and relation satisfiability can be reduced to class satisfiability in the following way: $\Sigma \not\models \top \sqsubseteq \bot$, $\Sigma \models C_1 \sqcap \neg C_2 \sqsubseteq \bot$, $\Sigma \not\models \exists^{\geq 1}[U_j]R \sqsubseteq \bot$ for some $j \leq n$ where n is the arity of R, respectively.

While $\mathcal{DLR}$ knowledge bases are fully able to capture atemporal EER schemas [8–10]—i.e. given an EER schema there is an equi-satisfiable $\mathcal{DLR}$ knowledge base—in

the following Sections we show how $\mathcal{DLR}_{\mathcal{US}}$ knowledge bases can capture temporal EER schemas with both timestamping and evolution constraints.

5. The Temporal Conceptual Model $\mathcal{ER}_{VT}$

In this Section, the temporal EER model $\mathcal{ER}_{VT}$—which will be the basis to present our proposal—is briefly introduced (see [3,4] for full details). $\mathcal{ER}_{VT}$ supports timestamping for classes, attributes, and relationships. $\mathcal{ER}_{VT}$ is equipped with both a textual and a graphical syntax along with a model-theoretic semantics as a temporal extension of the EER semantics [11]. The formal foundations of $\mathcal{ER}_{VT}$ allowed also to prove a correct encoding of $\mathcal{ER}_{VT}$ schemas as knowledge base in $\mathcal{DLR}_{\mathcal{US}}$ [5,4].

An $\mathcal{ER}_{VT}$ schema is a tuple: $\Sigma = (\mathcal{L}, \text{REL}, \text{ATT}, \text{CARD}, \text{ISA}, \text{DISJ}, \text{COVER}, \text{S}, \text{T}, \text{KEY})$, such that: $\mathcal{L}$ is a finite alphabet partitioned into the sets: $\mathcal{C}$ (*class* symbols), $\mathcal{A}$ (*attribute* symbols), $\mathcal{R}$ (*relationship* symbols), $\mathcal{U}$ (*role* symbols), and $\mathcal{D}$ (*domain* symbols). ATT is a function that maps a class symbol in $\mathcal{C}$ to an $\mathcal{A}$-labeled tuple over $\mathcal{D}$, $\text{ATT}(C) = \langle A_1 : D_1, \ldots, A_h : D_h \rangle$. REL is a function that maps a relationship symbol in $\mathcal{R}$ to an $\mathcal{U}$-labeled tuple over $\mathcal{C}$, $\text{REL}(R) = \langle U_1 : C_1, \ldots, U_k : C_k \rangle$, and k is the *arity* of R. CARD is a function $\mathcal{C} \times \mathcal{R} \times \mathcal{U} \mapsto \mathbb{N} \times (\mathbb{N} \cup \{\infty\})$ denoting cardinality constraints. We denote with $\text{CMIN}(C, R, U)$ and $\text{CMAX}(C, R, U)$ the first and second component of CARD. In Figure 2, $\text{CARD}(\texttt{TopManager}, \texttt{Manages}, \texttt{man}) = (1, 1)$. ISA is a binary relationship $\text{ISA} \subseteq (\mathcal{C} \times \mathcal{C}) \cup (\mathcal{R} \times \mathcal{R})$. ISA between relationships is restricted to relationships with the same arity. ISA is visualized with a directed arrow, e.g. `Manager` ISA `Employee` in Figure 2. DISJ, COVER are binary relations over $2^{\mathcal{C}} \times \mathcal{C}$, describing disjointness and covering partitions, respectively, over a group of ISA that share the same superclass. DISJ is visualized with a circled "d" and COVER with a double directed arrow, e.g. `Department`, `InterestGroup` are both disjoint and they cover `OrganizationalUnit`. The set $\mathcal{C}$ is partitioned into: a set $\mathcal{C}^S$ of S*napshot classes* (the S-*marked* classes in Figure 2), a set $\mathcal{C}^M$ of M*ixed classes* (the *unmarked* classes in Figure 2), and a set $\mathcal{C}^T$ of T*emporary classes* (the T-*marked* classes in Figure 2). A similar partition applies to the set $\mathcal{R}$. S, T are binary relations over $\mathcal{C} \times \mathcal{A}$ containing, respectively, the snapshot and temporary attributes of a class (see S, T marked attributes in Figure 2). KEY is a function, $\text{KEY} : \mathcal{C} \rightarrow \mathcal{A}$, that maps a class symbol in $\mathcal{C}$ to its key attribute. Keys are visualized as underlined attributes.

The model-theoretic semantics associated with the $\mathcal{ER}_{VT}$ modelling language adopts the *snapshot*[3] representation of abstract temporal databases and temporal conceptual models [12]. Following the snapshot paradigm, the flow of time $\mathcal{T} = \langle \mathcal{T}_p, < \rangle$, where $\mathcal{T}_p$ is a set of time points (or chronons) and $<$ is a binary precedence relation on $\mathcal{T}_p$, is assumed to be isomorphic to either $\langle \mathbb{Z}, < \rangle$ or $\langle \mathbb{N}, < \rangle$. Thus, standard relational databases can be regarded as the result of mapping a temporal database from time points in $\mathcal{T}$ to atemporal constructors, with the same interpretation of constants and the same

[3] The snapshot model represents the same class of temporal databases as the so called *timestamp* model [26, 27] which adds a temporal attribute to each relation [12].

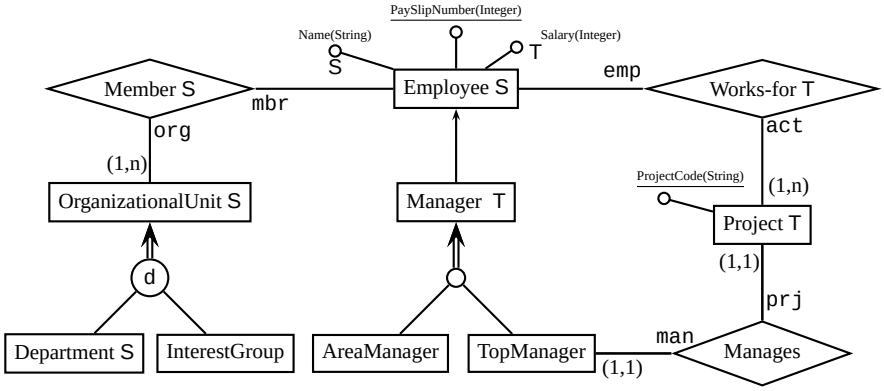

Figure 2. The company $\mathcal{ER}_{VT}$ diagram

domain.

Definition 5.1. ($\mathcal{ER}_{VT}$ Semantics) Let Σ be an $\mathcal{ER}_{VT}$ schema. A *temporal database state* for the schema Σ is a tuple $\mathcal{B} = (\mathcal{T}, \Delta^{\mathcal{B}} \cup \Delta^{\mathcal{B}}_D, \cdot^{\mathcal{B}(t)})$, such that: $\Delta^{\mathcal{B}}$ is a nonempty set of abstract objects disjoint from $\Delta^{\mathcal{B}}_D$; $\Delta^{\mathcal{B}}_D = \bigcup_{D_i \in \mathcal{D}} \Delta^{\mathcal{B}}_{D_i}$ is the set of basic domain values used in the schema Σ; and $\cdot^{\mathcal{B}(t)}$ is a function that for each $t \in \mathcal{T}$ maps:

- Every basic domain symbol D_i into a set $D_i^{\mathcal{B}(t)} = \Delta^{\mathcal{B}}_{D_i}$.
- Every class C to a set $C^{\mathcal{B}(t)} \subseteq \Delta^{\mathcal{B}}$—thus *objects* are instances of classes.
- Every relationship R to a set $R^{\mathcal{B}(t)}$ of $\mathcal{U}$-labeled tuples over $\Delta^{\mathcal{B}}$—i.e. let R be an n-ary relationship connecting the classes $C_1, \ldots, C_n$, $\text{REL}(R) = \langle U_1 : C_1, \ldots, U_n : C_n \rangle$, then, $r \in R^{\mathcal{B}(t)} \rightarrow (r = \langle U_1 : o_1, \ldots, U_n : o_n \rangle \wedge \forall i \in \{1, \ldots, n\}.o_i \in C_i^{\mathcal{B}(t)})$. We adopt the convention: $\langle U_1 : o_1, \ldots, U_n : o_n \rangle \equiv \langle o_1, \ldots, o_n \rangle$, when $\mathcal{U}$-labels are clear from the context.
- Every attribute A to a set $A^{\mathcal{B}(t)} \subseteq \Delta^{\mathcal{B}} \times \Delta^{\mathcal{B}}_D$, such that, for each $C \in \mathcal{C}$, if $\text{ATT}(C) = \langle A_1 : D_1, \ldots, A_h : D_h \rangle$, then, $o \in C^{\mathcal{B}(t)} \rightarrow (\forall i \in \{1, \ldots, h\}, \exists a_i.\ \langle o, a_i \rangle \in A_i^{\mathcal{B}(t)} \wedge \forall a_i.\langle o, a_i \rangle \in A_i^{\mathcal{B}(t)} \rightarrow a_i \in \Delta^{\mathcal{B}}_{D_i})$.

$\mathcal{B}$ is said a *legal temporal database state* if it satisfies all of the constraints expressed in the schema (we don't report here the semantics for temporal constraints since they will be discussed in details in the next Sections):

- For each $C_1, C_2 \in \mathcal{C}$, if C_1 ISA C_2, then, $C_1^{\mathcal{B}(t)} \subseteq C_2^{\mathcal{B}(t)}$.
- For each $R_1, R_2 \in \mathcal{R}$, if R_1 ISA R_2, then, $R_1^{\mathcal{B}(t)} \subseteq R_2^{\mathcal{B}(t)}$.
- For each cardinality constraint $\text{CARD}(C, R, U)$, then:
  $o \in C^{\mathcal{B}(t)} \rightarrow \text{CMIN}(C, R, U) \leq \#\{r \in R^{\mathcal{B}(t)} \mid r[U] = o\} \leq \text{CMAX}(C, R, U)$.
- For $C, C_1, \ldots, C_n \in \mathcal{C}$, if $\{C_1, \ldots, C_n\}$ DISJ C, then,
  $\forall i \in \{1, \ldots, n\}.C_i$ ISA $C \wedge \forall j \in \{1, \ldots, n\}, j \neq i.C_i^{\mathcal{B}(t)} \cap C_j^{\mathcal{B}(t)} = \emptyset$.

- For $C, C_1, \ldots, C_n \in \mathcal{C}$, if $\{C_1, \ldots, C_n\}$ COVER C, then, $\forall i \in \{1, \ldots, n\}.C_i$ ISA $C \wedge C^{\mathcal{B}(t)} = \bigcup_{i=1}^{n} C_i^{\mathcal{B}(t)}$.
- For each $C \in \mathcal{C}, A \in \mathcal{A}$ such that KEY$(C) = A$, then, A is a snapshot attribute–i.e. $\langle C, A_i \rangle \in$ S— and $\forall a \in \Delta_D^{\mathcal{B}}.\#\{o \in C^{\mathcal{B}(t)} \mid \langle o, a \rangle \in A^{\mathcal{B}(t)}\} \leq 1$.

Given such a set-theoretic semantics we are able to rigorously define some relevant modelling notions such as satisfiability, subsumption and derivation of new constraints by means of logical implication.

Definition 5.2. Let Σ be a schema, $C \in \mathcal{C}$ a class, and $R \in \mathcal{R}$ a relationship. The following modelling notions can be defined:

1. C (R) is *satisfiable* if there exists a legal temporal database state $\mathcal{B}$ for Σ such that $C^{\mathcal{B}(t)} \neq \emptyset$ ($R^{\mathcal{B}(t)} \neq \emptyset$), for some $t \in \mathcal{T}$;

2. Σ is *satisfiable* if there exists a legal temporal database state $\mathcal{B}$ for Σ ($\mathcal{B}$ is also said a *model* for Σ);

3. C_1 (R_1) is *subsumed* by C_2 (R_2) in Σ if every legal temporal database state for Σ is also a legal temporal database state for C_1 ISA C_2 (R_1 ISA R_2);

4. A schema Σ' is *logically implied* by a schema Σ over the same signature if every legal temporal database state for Σ is also a legal temporal database state for Σ'.

In the following Sections we will show how temporal database states, $\mathcal{B}$, support defining the semantics of timestamping and then how to extend both $\mathcal{ER}_{VT}$ and $\mathcal{B}$ to capture evolution constraints.

6. Timestamping

$\mathcal{ER}_{VT}$ is able to distinguish between *snapshot* constructors—i.e. constructors which bear no explicit specification of a given lifespan [25], which we convey by assuming a global lifespan (see Section 7.1) associated to each of their instances—*temporary* constructors—i.e. each of their instances has a limited lifespan—or *mixed* constructors—i.e. their instances can have either a global or a temporary existence. In the following, a class, relationship or attribute is called temporal if it is either temporary or mixed. The two temporal marks, S (snapshot) and T (temporary), introduced at the conceptual level, together with unmarked constructors capture the temporal distinction between snapshot, temporary and mixed constructors. The semantics of timestamping can now be defined as follows:

$o \in C^{\mathcal{B}(t)} \rightarrow \forall t' \in \mathcal{T}.o \in C^{\mathcal{B}(t')}$ `Snapshot Class`
$o \in C^{\mathcal{B}(t)} \rightarrow \exists t' \neq t.o \notin C^{\mathcal{B}(t')}$ `Temporary Class`
$r \in R^{\mathcal{B}(t)} \rightarrow \forall t' \in \mathcal{T}.r \in R^{\mathcal{B}(t')}$ `Snapshot Relationship`
$r \in R^{\mathcal{B}(t)} \rightarrow \exists t' \neq t.r \notin R^{\mathcal{B}(t')}$ `Temporary Relationship`
$(o \in C^{\mathcal{B}(t)} \wedge \langle o, a_i \rangle \in A_i^{\mathcal{B}(t)}) \rightarrow \forall t' \in \mathcal{T}.\langle o, a_i \rangle \in A_i^{\mathcal{B}(t')}$ `Snapshot Attribute`
$(o \in C^{\mathcal{B}(t)} \wedge \langle o, a_i \rangle \in A_i^{\mathcal{B}(t)}) \rightarrow \exists t' \neq t.\langle o, a_i \rangle \notin A_i^{\mathcal{B}(t')}$ `Temporary Attribute`

Since mixed constructors do not specify any temporal constraint there is no need to add a semantic equation. The semantics for attribute timestamping respects the $\mathcal{ER}_{VT}$ syntax where attributes are defined snapshot or temporary only locally, i.e. w.r.t. the classes they are attached with. Timestamps for both classes and relationships are captured by the following $\mathcal{DLR}_{\mathcal{US}}$ axioms (remember that $\Box^*$ is the "at all time" operator while $\Diamond^*$ is the "at some time" operator):

$C \sqsubseteq (\Box^* C)$ `Snapshot Class`
$C \sqsubseteq (\Diamond^* \neg C)$ `Temporary Class`
$R \sqsubseteq (\Box^* R)$ `Snapshot Relationship`
$R \sqsubseteq (\Diamond^* \neg R)$ `Temporary Relationship`

Considering attributes we first remember that they are captured in $\mathcal{DLR}$ as binary relationships [8]. For each attribute, $A \in \mathcal{A}$, the following $\mathcal{DLR}$ axiom holds: $A \sqsubseteq \texttt{From} : \top \sqcap \texttt{To} : \top$. Thus, if $\langle A, C \rangle \in \texttt{S}$ or $\langle A, C \rangle \in \texttt{T}$ then the following $\mathcal{DLR}_{\mathcal{US}}$ axioms hold, respectively:

$C \sqsubseteq \neg\exists[\texttt{From}](A \sqcap \Diamond^* \neg A)$ `Snapshot Attribute`
$C \sqsubseteq \neg\exists[\texttt{From}](\Box^* A)$ `Temporary Attribute`

The distinction between snapshot, temporary and mixed constructors has been adopted in $\mathcal{ER}_{VT}$ to avoid *overloading* the meaning of un-marked constructors. Indeed, the classical distinction between temporal (using a temporal mark) and atemporal (leaving the constructor un-marked) constructors may be ambiguous in the meaning of un-marked constructors. In this classical setting, un-marking is used to model both truly atemporal constructors (i.e. snapshot classes whose instances lifespan is always equal to the whole database lifespan), as well as legacy constructors (for *upward compatibility*) where the constructor is not marked as temporal because the original data model did not support the temporal dimension. The problem is that, due to the interaction between the various components of a temporal model, un-marked constructors can even purposely represent temporary constructors. As an example, think of an ISA involving a temporary entity (as superclass) and an un-marked entity (as a subclass). Since a designer cannot forecast all the possible interactions between the (temporal) constraints of a given conceptual schema, this ultimately means that in the classical approach *atemporality cannot be guaranteed* and this is true even for the upward compatibility.

$\mathcal{ER}_{VT}$ explicitly introduces a snapshot mark to force both atemporality and upward compatibility. As logical implication is formally defined in $\mathcal{ER}_{VT}$ (see Defini-

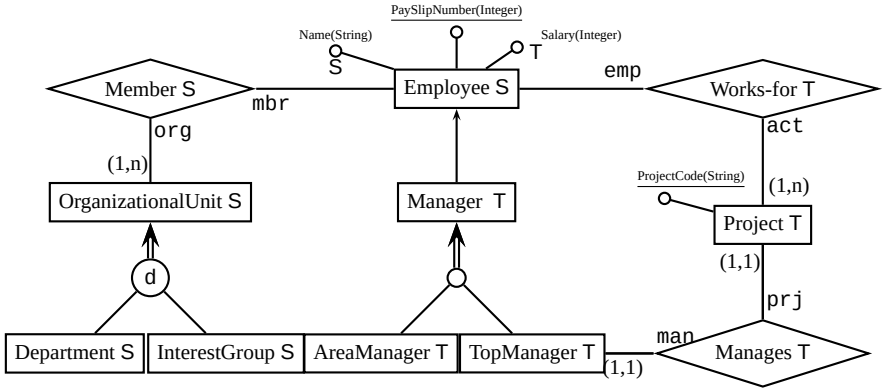

Figure 3. The company diagram with deductions on timestamps

tion 5.2), missing specifications can be inferred and in particular a set of logical implications hold in the case of timestamping. For instance, in Figure 2, as `Manager` is temporary both `AreaManager` and `TopManager` are temporary, too. Because `OrganizationalUnit` is snapshot and partitioned into two sub-classes, `Department` which is snapshot and `InterestGroup`, the latter should be snapshot, too. As the temporary class `TopManager` participates in the relationship `Manages`, then the latter must be temporary, too. The result of these deductions is given in Figure 3 (see [4] for an exhaustive list of deductions involving timestamps). Note that, when mapping $\mathcal{ER}_{VT}$ into a relational schema both temporary and un-marked constructors are mapped into a relation with added timestamp attributes, while snapshot constructors do not need any additional time attribute (for full details on the $\mathcal{ER}_{VT}$ relational mapping see [1]).

7. Formalizing Evolving Objects

Evolution constraints contribute in modelling the temporal behavior of an object. This Section discusses in details the aspects of evolutionary modelling that we take into account in our work. We first recall the basic concepts that have been proposed in the literature to deal with evolution, and their impact on the resulting conceptual language. Then we propose a formalization of the basic temporal concepts that are at the root of advanced conceptual temporal models: *lifecycle* with four statuses (scheduled, active, suspended, disabled); *transitions* of objects between different classes along their whole lifecycle; *generation* and *across-time* relationships asserting evolution constraints on objects linked by temporal relationships. These are genuine extensions to the $\mathcal{ER}_{VT}$ model that need to be taken into account in proposing a formalization based on a model-theoretic semantics and a corresponding set of axioms expressed using the temporal description logic $\mathcal{DLR}_{\mathcal{US}}$.

We aim both at presenting a formal characterization of the temporal conceptual modelling constructors for timestamping and evolution, and using the reasoning capabilities of $\mathcal{DLR}_{\mathcal{US}}$ to check satisfiability, subsumption and logical implications over

temporal schemas. The model-theoretic semantics we illustrate here for the various evolution constraints and the corresponding set of $\mathcal{DLR}_{\mathcal{US}}$ axioms are an extension of the one developed for the model $\mathcal{ER}_{VT}$, introduced in Section 5. The validity of the proposed formalization is justified by providing a set of logical implications which are in agreement with the derivations mentioned in the literature on temporal data modelling.

7.1. Status Classes

Status [37,15] is a conceptual notion associated to temporal classes as a component of the description of the lifecycle of their objects. It records the evolving state of membership of each object in the class. Following [37], status modelling includes up to four different statuses, and the allowed transitions between them:

- Scheduled. An object is scheduled if the planning of its existence within the class has to be recorded while its membership in the class will only become effective (active) some time later. For example, if a new project is approved but will not start until a later date the given project can be created as a new object in the `Project` class, with status scheduled for the valid time interval starting at the date of the approval decision and ending at the expected launching date. Each scheduled object will eventually become an active object. A scheduled object bears its identity (has an oid), but its attribute values do not need to be present. Supporting a scheduled status avoids the introduction of a new time type, the decision time [15], and smoothes the processing of lifecycle queries.
- Active. The status of an object is active if the object is a full member of the class (and therefore conforms to its type). For example, a currently ongoing project is an active member, at time now, of the `Project` class. Being active entails that the object can undergo any operation (retrieval, update, deletion, etc.), unless otherwise specified by the application.
- Suspended. This status qualifies objects that exist as members of the class, but are to be seen as temporarily inactive members of the class. Being inactive means that the object cannot undergo some operations. For example, in [15] no change to the values of the attributes of an object is allowed in the periods the object is suspended. An employee taking a temporary leave of absence is an example of what can be considered as a suspended employee. Only active objects can be suspended. A suspended object was in the past an active one.
- Disabled. This status is used to specify that the object's membership in the class has expired, meaning that the object is no more accessible in a normal mode of operation. While logically deleted, disabled objects are kept for some specific application purposes, e.g. statistical analyses. When the object becomes definitely irrelevant for the application, it is killed, rather than disabled, and disappears from the class. A disabled object was in the past an active member of the class (an object cannot be created in the disabled status). It can never again become a non-disabled member of that class (e.g. an expired project cannot be reactivated).

These four statuses intuitively correspond to a behavior we are familiar with in the real world. They are application-independent notions. Their choice has been driven by the abstract view of what an object behavior may be in terms of membership into a class. For specific applications specific classes may be equipped with a simplified form of lifecycle. For example, the lifecycle of a given class may be defined as not including the scheduled status, or not including the suspended status. As already mentioned, the simplest lifecycle consists of a single period with active status (which would be the case for atemporal objects, should they be given a lifecycle).

A critical issue is deciding the operational semantics of the statuses. Following the modification control approach in [15], statuses differ in terms of the operations that are allowed on objects in each status. Obviously active objects are fully operational, i.e. they can undergo any operation. But should modification of suspended objects be inhibited, as proposed in [15]? What if, for example, while an employee is suspended the categorization scheme of the company changes and the suspended employee now qualifies for a different category? Should the change be performed for the suspended employee as for all other employees, or should the change be stored in some log of changes for this employee and activated only at the moment the employee recovers its active status? The latter policy suggests an analogy between suspended objects and site failure in a network system. But in network systems objects in a failing site cannot be retrieved, while in [15] suspended objects can be retrieved. In summary, we could always find an example where the application requirements include the possibility to update an object whatever its status is. Consequently, a generic approach is to leave up to the designer to decide which restrictions to a full operational semantics, if any, should characterize the non-active statuses. The manipulation language, in its turn, should include the necessary operations to perform a change in the status of an object (e.g. create an object in a given status, suspend or disable an active object, activate a scheduled or a suspended object) and allow predicates on status of objects in the formulation of a query. Using such predicates the user can, for example, retrieve the active employees who have been suspended at some time in the last three years.

A similar difficult issue is to decide to what extent, if any, the status of objects constrains the relationships holding between those objects. Most data models only allow creation of relationships between objects in the active status at the time the relationship is created. Our discussion on across-time relationships shows however that applications may require the capability to involve also suspended/scheduled/disabled objects in the creation of a new relationship (see Sections 7.4,7.3). Thus, unless explicitly inhibited, objects can get involved in the creation of new relationships whatever their status is.

To conclude this discussion on statuses, it is worth noticing that application-oriented lifecycles are frequently found and may be organized using the same mechanism as for the application-independent lifecycle. For example, in a supplying company objects in a class `Order` can be categorized as standing-order, registered-order, order-in-process, billed-order, paid-order, order-in-delivery, delivered-order. The designer could then specify the transition between these ”statuses”, together with the corresponding transition rules, and let the system enforce the consistency of orders’ evolution with the

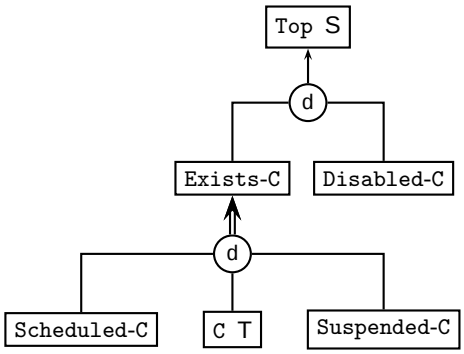

Figure 4. Status classes.

stated constraints. It would be worth investigating the possibility to devise a formalism that handles the definition and management of application lifecycles in a similar way as it handles the predefined lifecycle we discussed above. But this is beyond the scope of this paper. Notice that, conversely, having application-independent lifecycles associated to objects and monitored by the system has a definite advantage it allows relying on standard operators (e.g. activate, suspend, reactivate, disable) for status manipulation.

Formalization. Let C be a temporal (i.e. temporary or mixed) class. We capture status transition of membership in C by associating to C the following *status classes*: `Scheduled-C`, `Suspended-C`, `Disabled-C`. In particular, status classes are represented by the hierarchy of Figure 4 (where C may also be mixed) that classifies C instances according to their actual status. To preserve upward compatibility we do not explicitly introduce an active class, but assume by default that the name of the class itself denotes the set of active objects, i.e. `Active-C` $\equiv$ `C`. We can assume that the status classes are created automatically by the system each time a class is declared temporal. Thus, designers and users are forced neither to introduce nor to manipulate status classes. They only have to be aware of the different statuses in the lifecycle of an object. Note that, since membership of objects into snapshot classes is global, i.e. objects are always active, the notion of status classes does not apply to snapshot classes.

To capture the intended meaning of status classes, we define ad-hoc constraints and then prove that such constraints capture the evolving behavior of status classes as described in the literature [37,15]. First of all, disjointness and ISA constraints between statuses of a class C can be described in $\mathcal{ER}_{VT}$ as illustrated in Figure 4, where `Top` is supposed to be a snapshot class which represents the universe of abstract objects (i.e. $\texttt{Top}^{\mathcal{B}(t)} \equiv \Delta^{\mathcal{B}}$). Other than hierarchical constraints, the intended semantics of status classes induces the following rules that are related to their temporal behavior:

(EXISTS**)** *Existence persists until Disabled.*
$o \in \texttt{Exists-C}^{\mathcal{B}(t)} \rightarrow \forall t' > t.(o \in \texttt{Exists-C}^{\mathcal{B}(t')} \vee o \in \texttt{Disabled-C}^{\mathcal{B}(t')})$
(DISAB1**)** *Disabled persists.*
$o \in \texttt{Disabled-C}^{\mathcal{B}(t)} \rightarrow \forall t' > t.o \in \texttt{Disabled-C}^{\mathcal{B}(t')}$

(DISAB2**)** *Disabled was Active in the past.*
$o \in \texttt{Disabled-C}^{\mathcal{B}(t)} \rightarrow \exists t' < t.o \in \texttt{C}^{\mathcal{B}(t')}$
(SUSP**)** *Suspended was Active in the past.*
$o \in \texttt{Suspended-C}^{\mathcal{B}(t)} \rightarrow \exists t' < t.o \in \texttt{C}^{\mathcal{B}(t')}$
(SCH1**)** *Scheduled will eventually become Active.*
$o \in \texttt{Scheduled-C}^{\mathcal{B}(t)} \rightarrow \exists t' > t.o \in \texttt{C}^{\mathcal{B}(t')}$
(SCH2**)** *Scheduled can never follow Active.*
$o \in \texttt{C}^{\mathcal{B}(t)} \rightarrow \forall t' > t.o \notin \texttt{Scheduled-C}^{\mathcal{B}(t')}$

As an example of a rule expressing the semantics associated to the different statuses, the following rule formally expresses the suggestion from [15] to make unchangeable the attributes of suspended objects, the unchangeability starting at the time instant the object becomes suspended. Let A an attribute of a class C then:

(FREEZ**)** *Freezing attributes of suspended classes.*
$o \in \texttt{Suspended-C}^{\mathcal{B}(t)} \wedge \langle o, a\rangle \in A^{\mathcal{B}(t)} \rightarrow \langle o, a\rangle \in A^{\mathcal{B}(t-1)}$

$\mathcal{DLR}_{\mathcal{US}}$ axioms are able to fully capture the hierarchical constraints of Figure 4 (see [4] for more details). Moreover, the above semantic equations are captured by the following set of $\mathcal{DLR}_{\mathcal{US}}$ axioms:

(EXISTS) $\texttt{Exists-C} \sqsubseteq \Box^{+}(\texttt{Exists-C} \sqcup \texttt{Disabled-C})$
(DISAB1) $\texttt{Disabled-C} \sqsubseteq \Box^{+}\texttt{Disabled-C}$
(DISAB2) $\texttt{Disabled-C} \sqsubseteq \Diamond^{-}\texttt{C}$
(SUSP) $\texttt{Suspended-C} \sqsubseteq \Diamond^{-}\texttt{C}$
(SCH1) $\texttt{Scheduled-C} \sqsubseteq \Diamond^{+}\texttt{C}$
(SCH2) $\texttt{C} \sqsubseteq \Box^{+}\neg\texttt{Scheduled-C}$

We denote with Σ_{st} the above set of axioms together with the $\mathcal{DLR}_{\mathcal{US}}$ axioms that capture the hierarchy of Figure 4. On the other hand, the axiom that captures the inhibition to update suspended objects is:

(FREEZ) $\texttt{Suspended-C} \sqsubseteq \neg\exists[\texttt{From}](\texttt{A} \sqcap \ominus \neg\texttt{A})$

As a consequence of the above formalization, scheduled and disabled status classes can be true only for a single interval, while active and suspended classes can hold for set of intervals (i.e. an object can move many times back and forth from active to suspended status and vice versa). In particular, the following set of new rules can be derived.

Proposition 7.1. (Status Classes: Logical Implications) Given the set of axioms Σ_{st} that formalize status classes, the following logical implications hold (each implication is described by a natural language sentence and the corresponding $\mathcal{DLR}_{\mathcal{US}}$ logical implication):

(DISAB3**)** *Disabled will never become active anymore.*
$\Sigma_{st} \models \texttt{Disabled-C} \sqsubseteq \Box^{+}\neg\texttt{C}$

(SCH3) *Scheduled persists until active.*
$\Sigma_{st} \models \texttt{Scheduled-C} \sqsubseteq \texttt{Scheduled-C}\,\mathcal{U}\,\texttt{C}$
(SCH4) *Scheduled cannot evolve directly to Disabled.*
$\Sigma_{st} \models \texttt{Scheduled-C} \sqsubseteq \oplus \neg\texttt{Disabled-C}$

Proof

(DISAB3) $\Sigma_{st} \models \texttt{Disabled-C} \sqsubseteq \Box^{+}\neg\texttt{C}$
Let $o \in \texttt{Disabled-C}^{\mathcal{B}(\mathtt{t}_0)}$, then, by (DISAB1), $\forall t' > t_0.o \in \texttt{Disabled-C}^{\mathcal{B}(\mathtt{t}')}$. Since Disabled-C is disjoint from C, then $\forall t' > t_0.o \notin \texttt{C}^{\mathcal{B}(\mathtt{t}')}$.
(SCH3) $\Sigma_{st} \models \texttt{Scheduled-C} \sqsubseteq \texttt{Scheduled-C}\,\mathcal{U}\,\texttt{C}$
Let $o \in \texttt{Scheduled-C}^{\mathcal{B}(\mathtt{t}_0)}$, then $o \in \texttt{Exists-C}^{\mathcal{B}(\mathtt{t}_0)}$ and, by (SCH1), $\exists t_1 > t_0.o \in C^{\mathcal{B}(t_1)}$. Let assume that $t_1 = min\{t \in \mathcal{T} \mid t > t_0$ *and* $o \in C^{\mathcal{B}(t)}\}$. Now, by (EXISTS), $\forall t'.t_0 < t' < t_1.o \in (\texttt{Exists-C} \sqcup \texttt{Disabled-C})^{\mathcal{B}(t')}$. On the other hand, by (DISAB3), $o \notin \texttt{Disabled-C}^{\mathcal{B}(t')}$. By the "min" choice of t_1, $o \notin C^{\mathcal{B}(t')}$ and also $o \notin \texttt{Suspended-C}^{\mathcal{B}(t')}$. Thus, $\forall t'.t_0 < t' < t_1.o \in \texttt{Scheduled-C}^{\mathcal{B}(\mathtt{t}')}$.
Together with axiom (SCH2), we can also conclude that Scheduled-C is true just on a single interval.
(SCH4) $\Sigma_{st} \models \texttt{Scheduled-C} \sqsubseteq \oplus \neg\texttt{Disabled-C}$
Let $o \in \texttt{Scheduled-C}^{\mathcal{B}(\mathtt{t}_0)}$, then by (SCH1), $\exists t_1 > t_0.o \in C^{\mathcal{B}(t_1)}$. Thus, by (DISAB3), $o \notin \texttt{Disabled-C}^{\mathcal{B}(t_0+1)}$.

❑

Status classes are central in describing the evolutionary behavior of objects. In the following we show the adequacy of the semantics associated to status classes to describe: a) the behavior of temporal classes involved in ISA relationships; b) the notion of *lifespan* of an object. In the next Section status classes will be used to model: c) the object migration between classes; d) the relationships that involve objects existing at different times (both generation and across-time relationships).

Isa vs. status. When an ISA relationship is specified between two temporal classes, say B ISA A, some rules have to be obeyed to guarantee consistency between the status of the object in the subclass, B, and its status in the superclass, A. These common sense rules follow from the perception that being in a subclass represents a "sub-activity" the object pursues while continuing its "activity" in the superclass. Thus, the level of activity in the superclass must be higher than or equal to the level of activity in the subclass. Being *active* is a higher level of activity as being *suspended*, which in turn is a higher level of activity than being *disabled*. This is expressed by constraints ISA1-3 and ISA5 below between the respective status classes. Similarly, the *scheduled* status expresses a planned *existence*, i.e. a lower level of existence than being currently in existence: hence constraint ISA4.

(ISA1) *Objects active in B must be active in A.*
$\texttt{B} \sqsubseteq \texttt{A}$

(ISA2**)** *Objects suspended in B must be either suspended or active in A.*
$\texttt{Suspended-B} \sqsubseteq \texttt{Suspended-A} \sqcup \texttt{A}$

(ISA3**)** *Objects disabled in B must be either disabled, suspended or active in A.*
$\texttt{Disabled-B} \sqsubseteq \texttt{Disabled-A} \sqcup \texttt{Suspended-A} \sqcup \texttt{A}$

(ISA4**)** *Objects scheduled in B must exist in A.*
$\texttt{Scheduled-B} \sqsubseteq \texttt{Exists-A}$

(ISA5**)** *Objects disabled in A, and active in B in the past, must be disabled in B.*
$\texttt{Disabled-A} \sqcap \Diamond^{-}\texttt{B} \sqsubseteq \texttt{Disabled-B}$

The formalization of status classes provided above is not sufficient to guarantee properties (ISA1-5)[4]. We need to further assume that the system behaves under the *temporal* ISA *assumption*: Each time an ISA between two temporal classes holds (B ISA A), then an ISA between the respective existence status classes (Exists-B ISA Exists-A) is automatically added by the system. The temporal ISA assumption is thus captured by the following set of axioms: $\Sigma_{\text{ISA}} = \{B \sqsubseteq A, \texttt{Exists-B} \sqsubseteq \texttt{Exists-A}\}$. Now, we are able to prove that points (ISA1-5) above are entailed by the semantics associated to status classes under the temporal ISA assumption.

Proposition 7.2. (Status Classes Vs. ISA**: Logical Implications)** Let A, B be two temporal classes such that BISAA, then properties (ISA1-5) are valid logical implications under the *temporal* ISA *assumption*.

(ISA1**)** *Objects active in B must be active in A.*
$\Sigma_{st} \cup \Sigma_{\text{ISA}} \models \texttt{B} \sqsubseteq \texttt{A}$

(ISA2**)** *Objects suspended in B must be either suspended or active in A.*
$\Sigma_{st} \cup \Sigma_{\text{ISA}} \models \texttt{Suspended-B} \sqsubseteq \texttt{Suspended-A} \sqcup \texttt{A}$

(ISA3**)** *Objects disabled in B must be either disabled, suspended or active in A.*
$\Sigma_{st} \cup \Sigma_{\text{ISA}} \models \texttt{Disabled-B} \sqsubseteq \texttt{Disabled-A} \sqcup \texttt{Suspended-A} \sqcup \texttt{A}$

(ISA4**)** *Objects scheduled in B must exist in A.*
$\Sigma_{st} \cup \Sigma_{\text{ISA}} \models \texttt{Scheduled-B} \sqsubseteq \texttt{Exists-A}$

(ISA5**)** *Objects disabled in A, and active in B in the past, must be disabled in B.*
$\Sigma_{st} \cup \Sigma_{\text{ISA}} \models \texttt{Disabled-A} \sqcap \Diamond^{-}\texttt{B} \sqsubseteq \texttt{Disabled-B}$

Proof

(ISA1**)** Obviously true since BISAA holds in Σ_{ISA}, and both A, B are considered active.

(ISA2**)** Let $o \in \texttt{Suspended-B}^{\mathcal{B}(t_0)}$, since Suspended-B ISA Exists-B, and (by temporal ISA assumption) Exists-B ISA Exists-A, then, $o \in \texttt{Exists-A}^{\mathcal{B}(t_0)}$. On the other hand, by (SUSP), $\exists t_1 < t_0 . o \in B^{\mathcal{B}(t_1)}$, and then, $o \in A^{\mathcal{B}(t_1)}$. Then, by (SCH2), $o \notin \texttt{Scheduled-A}^{\mathcal{B}(t_0)}$. Thus, due to the disjoint covering constraint between active, scheduled and suspended classes, either $o \in A^{\mathcal{B}(t_0)}$ or $o \in \texttt{Suspended-A}^{\mathcal{B}(t_0)}$.

(ISA3**)** Let $o \in \texttt{Disabled-B}^{\mathcal{B}(t_0)}$, then, by (DISAB2), $\exists t_1 < t_0 . o \in B^{\mathcal{B}(t_1)}$. By B ISA A and A ISA Exists-A, then, $o \in \texttt{Exists-A}^{\mathcal{B}(t_1)}$. By (EXISTS) and the disjointness

[4] We let the reader check that points 2,4 and 5 are not necessarily true.

between existing and disabled classes, there are only two possibilities at point in time $t_0 > t_1$:

1. $o \in \texttt{Exists-A}^{\mathcal{B}(t_0)}$, and thus, by (SCH2), $o \in A^{\mathcal{B}(t_0)}$ or $o \in \texttt{Suspended-A}^{\mathcal{B}(t_0)}$; or

2. $o \in \texttt{Disabled-A}^{\mathcal{B}(t_0)}$.

(ISA4**)** Let $o \in \texttt{Scheduled-B}^{\mathcal{B}(t_0)}$, then, $o \in \texttt{Exists-B}^{\mathcal{B}(t_0)}$. Thus, by the temporal ISA assumption, $o \in \texttt{Exists-A}^{\mathcal{B}(t_0)}$. As a further logical implication, it also follows that objects scheduled in B cannot be disabled in A.

(ISA5**)** Let $o \in \texttt{Disabled-A}^{\mathcal{B}(t_0)}$ and $o \in B^{\mathcal{B}(t_1)}$ for some $t_1 < t_0$, then, $o \in \texttt{Exists-B}^{\mathcal{B}(t_1)}$. By (EXISTS) and the disjointness between existing and disabled classes, there are only two possibilities at time $t_0 > t_1$: either $o \in \texttt{Exists-B}^{\mathcal{B}(t_0)}$ or $o \in \texttt{Disabled-B}^{\mathcal{B}(t_0)}$. By absurd, let $o \in \texttt{Exists-B}^{\mathcal{B}(t_0)}$, then by temporal ISA assumption, $o \in \texttt{Exists-A}^{\mathcal{B}(t_0)}$, which contradicts the assumption that $o \in \texttt{Disabled-A}^{\mathcal{B}(t_0)}$.

❑

Temporal applications often use concepts that are derived from the notion of object statuses, e.g. the *lifespan* of a temporal object or its *birth* and *death* instants. These concepts are supported through methods in object-oriented DBMS. Hereinafter we provide formal definitions for these concepts.

Lifespan and related notions. The lifespan of an object w.r.t. a class describes the temporal instants[5] where the object can be considered a member of the class. The lifespan concept together with the notion of status classes support the definition of temporal constraints between objects (see Section 7.3). With the introduction of status classes we can distinguish between the following notions: EXISTENCESPAN_C, LIFESPAN_C, ACTIVESPAN_C, BEGIN_C, BIRTH_C and DEATH_C. They are functions which depend on the object membership to the status classes associated to a temporal class C.

The *existencespan* of an object describes the temporal instants where the object is either a scheduled, active or suspended member of a given class. More formally, $\text{EXISTENCESPAN}_C : \Delta^{\mathcal{B}} \rightarrow 2^{\mathcal{T}}$, such that:

$$\text{EXISTENCESPAN}_C(o) = \{t \in \mathcal{T} \mid o \in \texttt{Exists-C}^{\mathcal{B}(t)}\}$$

The *lifespan* of an object describes the temporal instants where the object is an active or suspended member of a given class (thus, $\text{LIFESPAN}_C(o) \subseteq \text{EXISTENCESPAN}_C(o)$). More formally, $\text{LIFESPAN}_C : \Delta^{\mathcal{B}} \rightarrow 2^{\mathcal{T}}$, such that:

$$\text{LIFESPAN}_C(o) = \{t \in \mathcal{T} \mid o \in \texttt{C}^{\mathcal{B}(t)} \cup \texttt{Suspended-C}^{\mathcal{B}(t)}\}$$

[5] Note that the semantics of $\mathcal{ER}_{VT}$ allows for set of intervals—usually mentioned as temporal elements in the literature—as generic lifespan of an object.

The *activespan* of an object describes the temporal instants where the object is an active member of a given class (thus, $\text{ACTIVESPAN}_C(o) \subseteq \text{LIFESPAN}_C(o)$). More formally, $\text{ACTIVESPAN}_C : \Delta^{\mathcal{B}} \rightarrow 2^{\mathcal{T}}$, such that:

$$\text{ACTIVESPAN}_C(o) = \{t \in \mathcal{T} \mid o \in \mathtt{C}^{\mathcal{B}(t)}\}$$

The functions BEGIN_C and DEATH_C associate to an object the first and the last appearance, respectively, of the object as a member of a given class, while BIRTH_C denotes the first appearance as an active object of that class. More formally, BEGIN_C, BIRTH_C, $\text{DEATH}_C : \Delta^{\mathcal{B}} \rightarrow \mathcal{T}$, such that:

$$\text{BEGIN}_C(o) = \mathtt{min}(\text{EXISTENCESPAN}_C(o))$$
$$\text{BIRTH}_C(o) = \mathtt{min}(\text{ACTIVESPAN}_C(o)) \equiv \mathtt{min}(\text{LIFESPAN}_C(o))$$
$$\text{DEATH}_C(o) = \mathtt{max}(\text{LIFESPAN}_C(o))$$

We could still speak of existencespan, lifespan or activespan for snapshot classes, but in this case $\text{EXISTENCESPAN}_C(o) \equiv \text{LIFESPAN}_C(o) \equiv \text{ACTIVESPAN}_C(o) \equiv \mathcal{T}$. Furthermore, $\text{BEGIN}_C(o) = \text{BIRTH}_C(o) = -\infty$, and $\text{DEATH}_C(o) = +\infty$ either when C is a snapshot class or in cases of instances existing since ever and/or living forever.

7.2. Transition

A database object represents a real world object seen as a member of the class the object belongs to. As the real world evolves, the same real world object may lose its quality as a member of the class and may acquire other or additional memberships in other classes defined in the database. For example, there are formalisms allowing *John* to be simultaneously represented as a member of the class `Employee` and as a member of the class `TennisPlayer`, and later to become a member of the class `Manager`. In other words, objects can dynamically show up and move around through the classes defined in a schema.

Transition constraints [22,37] bear specific transition semantics. They have been introduced to model the phenomenon called *object migration*. A transition records objects migrating from a *source* class to a *target* class. At the schema level, it expresses that the instances of the source class may *migrate* into the target class. Two types of transitions have been considered: *dynamic evolution*, when objects cease to be instances of the source class to become instances of the target class, and *dynamic extension*, when the creation of the target instance does not force the removal of the source instance. For example, considering the company schema (Figure 3), if we want to record data about the promotion of area managers into top managers we can specify a dynamic evolution from the class `AreaManager` to the class `TopManager`. We can also record the fact that a mere employee becomes a manager by defining a dynamic extension from the class `Employee` to the class `Manager` (see Figure 5).

Finally, transitions are particularly relevant in the case of *not directly instantiable* classes. A class is said not directly instantiable if creation of objects with a new oid is not allowed in the class. In this case, transition constraints define the trajectory of

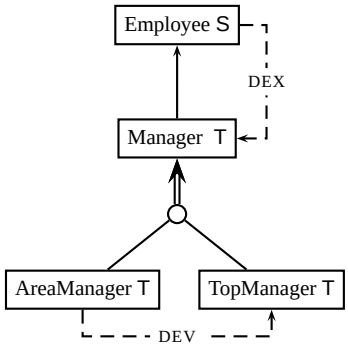

Figure 5. Transitions employee-to-manager and area-to-top manager

objects in time and consequently rule the object's evolution as far as existence in a not directly instantiable class is concerned.

Regarding the graphical representation, as illustrated in Figure 5, we use a dashed arrow pointing to the target class and labeled with either DEX or DEV denoting dynamic extension and evolution, respectively.

Formalization. Specifying a transition between two classes means that: *a)* We want to keep track of such migration; *b)* Not necessarily all the objects in the source or in the target participate in the migration; *c)* When the source class is a temporal class, migration only involves active or suspended objects—thus, neither disabled nor scheduled objects can take part in a transition.

In the following, we present a formalization that satisfies the above requirements. Notice that transitions are constrained by the fact that they consider single objects. Formalizing dynamic transitions as relationships would result in binary relationships linking the same migrating object twice, once as an instance in the source class and once as an instance in the target class. Rather than defining a relationship type with an equality constraint on the identity of the linked instances, we represent transitions by introducing a new class denoted by either DEX_{C_1,C_2} or DEV_{C_1,C_2} for dynamic extension and evolution, respectively. More formally, in case of a *dynamic extension* between classes C_1, C_2 the following semantic equation holds:

$$o \in \text{DEX}_{C_1,C_2}^{\mathcal{B}(t)} \rightarrow (o \in (\texttt{Suspended-C}_1{}^{\mathcal{B}(t)} \cup \texttt{C}_1{}^{\mathcal{B}(t)}) \wedge o \notin \texttt{C}_2{}^{\mathcal{B}(t)} \wedge o \in C_2^{\mathcal{B}(t+1)})$$

And the equivalent $\mathcal{DLR}_{\mathcal{US}}$ axiom is:

$$(\text{DEX}) \quad \text{DEX}_{C_1,C_2} \sqsubseteq (\texttt{Suspended-C}_1 \sqcup \texttt{C}_1) \sqcap \neg \texttt{C}_2 \sqcap \oplus C_2$$

In case of a *dynamic evolution* between classes C_1, C_2 the source object cannot remain active in the source class. Thus, the following semantic equation holds:

$$o \in \text{DEV}_{C_1,C_2}^{\mathcal{B}(t)} \rightarrow (o \in (\texttt{Suspended-C}_1{}^{\mathcal{B}(t)} \cup \texttt{C}_1{}^{\mathcal{B}(t)}) \wedge o \notin \texttt{C}_2{}^{\mathcal{B}(t)} \wedge o \in C_2^{\mathcal{B}(t+1)} \wedge o \notin C_1^{\mathcal{B}(t+1)})$$

And the equivalent $\mathcal{DLR}_{\mathcal{US}}$ axiom is:

(DEV) $\text{DEV}_{C_1,C_2} \sqsubseteq (\texttt{Suspended-C}_1 \sqcup \texttt{C}_1) \sqcap \neg\texttt{C}_2 \sqcap \oplus (C_2 \sqcap \neg C_1)$

Finally, we formalize the case where the source (C_1) and/or the target (C_2) totally participate in a dynamic extension/evolution (at schema level we add mandatory cardinality constraints on DEX/DEV links):

$o \in C_1^{\mathcal{B}(t)} \rightarrow \exists t' > t . o \in \text{DEX}_{C_1,C_2}^{\mathcal{B}(t')}$ `Source Total Transition`
$o \in C_2^{\mathcal{B}(t)} \rightarrow \exists t' < t . o \in \text{DEX}_{C_1,C_2}^{\mathcal{B}(t')}$ `Target Total Transition`
$o \in C_1^{\mathcal{B}(t)} \rightarrow \exists t' > t . o \in \text{DEV}_{C_1,C_2}^{\mathcal{B}(t')}$ `Source Total Evolution`
$o \in C_2^{\mathcal{B}(t)} \rightarrow \exists t' < t . o \in \text{DEV}_{C_1,C_2}^{\mathcal{B}(t')}$ `Target Total Evolution`

The above cases are captured by the following $\mathcal{DLR}_{\mathcal{US}}$ axioms, respectively:

(STT) $C_1 \sqsubseteq \Diamond^+ \text{DEX}_{C_1,C_2}$ `Source Total Transition`
(TTT) $C_2 \sqsubseteq \Diamond^- \text{DEX}_{C_1,C_2}$ `Target Total Transition`
(STE) $C_1 \sqsubseteq \Diamond^+ \text{DEV}_{C_1,C_2}$ `Source Total Evolution`
(TTE) $C_2 \sqsubseteq \Diamond^- \text{DEV}_{C_1,C_2}$ `Target Total Evolution`

Note that, either (TTT) or (TTE) are appropriate constraints to describe the behavior of not directly instantiable classes. An interesting set of consequences of the above proposed modelling of dynamic transitions are shown in the following proposition.

Proposition 7.3. (Transition: Logical Implications) Let $\Sigma_{tr} = \{(\text{DEV}),(\text{DEX})\}$, then the following logical implications hold:

1. *The classes* DEX_{C_1,C_2} *and* DEV_{C_1,C_2} *are temporary classes; actually, they hold at single time points.*
   $\Sigma_{st} \cup \Sigma_{tr} \models \text{DEX}_{C_1,C_2} \sqsubseteq \oplus \neg\, \text{DEX}_{C_1,C_2} \sqcap \ominus \neg\text{DEX}_{C_1,C_2}$
   $\Sigma_{st} \cup \Sigma_{tr} \models \text{DEV}_{C_1,C_2} \sqsubseteq \oplus \neg\, \text{DEV}_{C_1,C_2} \sqcap \ominus \neg\text{DEV}_{C_1,C_2}$
2. *Objects in the classes* DEX_{C_1,C_2} *and* DEV_{C_1,C_2} *cannot be disabled as* C_2.
   $\Sigma_{st} \cup \Sigma_{tr} \models \text{DEX}_{C_1,C_2} \sqsubseteq \neg\texttt{Disabled-C}_2$
   $\Sigma_{st} \cup \Sigma_{tr} \models \text{DEV}_{C_1,C_2} \sqsubseteq \neg\texttt{Disabled-C}_2$
3. *The target class* C_2 *cannot be snapshot (it becomes temporary in case of both* TTT *and* TTE *constraints).*
   $\Sigma_{st} \cup \Sigma_{tr} \models \text{DEX}_{C_1,C_2} \sqsubseteq \Diamond^* [C_2 \sqcap (\Diamond^+ \neg C_2 \sqcup \Diamond^- \neg C_2)]$
4. *As a consequence of dynamic evolution, the source class,* C_1*, cannot be snapshot (and it becomes temporary in case of* STE *constraints).*
   $\Sigma_{st} \cup \Sigma_{tr} \models \text{DEV}_{C_1,C_2} \sqsubseteq \Diamond^* [C_1 \sqcap (\Diamond^+ \neg C_1 \sqcup \Diamond^- \neg C_1)]$
5. *Dynamic evolution cannot be specified between a class and one of its sub-classes.*
   $\Sigma_{st} \cup \Sigma_{tr} \cup \{C_2 \sqsubseteq C_1\} \models \text{DEV}_{C_1,C_2} \sqsubseteq \bot$
6. *Dynamic extension between disjoint classes logically implies Dynamic evolution.*
   $\Sigma_{st} \cup \Sigma_{tr} \cup \{C_1 \sqsubseteq \neg C_2\} \models \text{DEX}_{C_1,C_2} \sqsubseteq \text{DEV}_{C_1,C_2}$

Proof

1. $\Sigma_{st} \cup \Sigma_{tr} \models \text{DEX}_{C_1,C_2} \sqsubseteq \oplus \neg\, \text{DEX}_{C_1,C_2} \sqcap \ominus \neg\text{DEX}_{C_1,C_2}$ (similar for DEV_{C_1,C_2})
   Indeed, let $o \in \text{DEX}_{C_1,C_2}^{\mathcal{B}(t)}$, then, from (DEX), $o \notin C_2^{\mathcal{B}(t)}$ and $o \in C_2^{\mathcal{B}(t+1)}$, thus $o \notin \text{DEX}_{C_1,C_2}^{\mathcal{B}(t+1)}$ and $o \notin \text{DEX}_{C_1,C_2}^{\mathcal{B}(t-1)}$. Note that, the time t such that $o \in \text{DEX}_{C_1,C_2}^{\mathcal{B}(t)}$ records when the transition event happens. Similar considerations apply for DEV_{C_1,C_2}.
2. $\Sigma_{st} \cup \Sigma_{tr} \models \text{DEX}_{C_1,C_2} \sqsubseteq \neg\texttt{Disabled-C}_2$ (similar for DEV_{C_1,C_2})
   Indeed, from (DEX), $\text{DEX}_{C_1,C_2} \sqsubseteq \oplus C_2$, i.e. objects in DEX_{C_1,C_2} are active in C_2 starting from the next point in time, then by property (DISAB3), $\text{DEX}_{C_1,C_2} \sqsubseteq \neg\texttt{Disabled-C}_2$. The same holds for DEV_{C_1,C_2}.
3. $\Sigma_{st} \cup \Sigma_{tr} \models \text{DEX}_{C_1,C_2} \sqsubseteq \Diamond^*[C_2 \sqcap (\Diamond^+\neg C_2 \sqcup \Diamond^-\neg C_2)]$
   Indeed, from (DEX), $\text{DEX}_{C_1,C_2} \sqsubseteq \neg\texttt{C}_2 \sqcap \oplus \texttt{C}_2$ (the same holds for DEV_{C_1,C_2}).
4. $\Sigma_{st} \cup \Sigma_{tr} \models \text{DEV}_{C_1,C_2} \sqsubseteq \Diamond^*[C_1 \sqcap (\Diamond^+\neg C_1 \sqcup \Diamond^-\neg C_1)]$
   Indeed, from (DEV), an object evolving from C_1 to C_2 ceases to be a member of C_1.
5. $\Sigma_{st} \cup \Sigma_{tr} \cup \{C_2 \sqsubseteq C_1\} \models \text{DEV}_{C_1,C_2} \sqsubseteq \bot$
   Indeed, from (DEV), $\text{DEV}_{C_1,C_2} \sqsubseteq \oplus (C_2 \sqcap \neg C_1)$ which contradicts $C_2 \sqsubseteq C_1$.
6. $\Sigma_{st} \cup \Sigma_{tr} \cup \{C_1 \sqsubseteq \neg C_2\} \models \text{DEX}_{C_1,C_2} \sqsubseteq \text{DEV}_{C_1,C_2}$
   Let $o \in \text{DEX}_{C_1,C_2}^{\mathcal{B}(t)}$, then, from (DEX), $o \in C_2^{\mathcal{B}(t+1)}$, and, since $C_1 \sqsubseteq \neg C_2$, $o \notin C_1^{\mathcal{B}(t+1)}$. Thus, $o \in \text{DEV}_{C_1,C_2}^{\mathcal{B}(t)}$.

❑

7.3. Across-Time Relationships

Across-Time relationships [40,31,37] describe relationships between objects that may not coexist at the same time and possibly not at the time the relationship is asserted. The conceptual model MADS [37,38] allows for *synchronization* relationships to specify temporal constraints (Allen temporal relations) between the lifespan of linked objects. *Historical marks* are used in the ERT model [31] to express a relationship between objects not existing at the same time (both past and future historical marks are introduced).

There are many examples of these relationships (see Figure 6). Consider, for example, a relationship `Biography` between an author and a famous person already dead, or the relationship `Grandparent` that holds even if the grandparent passed away before the grandchild was born or the grandchild is not yet born. Considering the company schema (Figure 3), the relationship `Works-for` is an across-time relationship if company rules allow assigning an employee to a project before its official launching, or if employees may keep on working on a project after its official closure.

This Section formalizes across-time relationships with the aim of preserving the snapshot reducibility of the resulting model. Let us consider a concrete example. Let

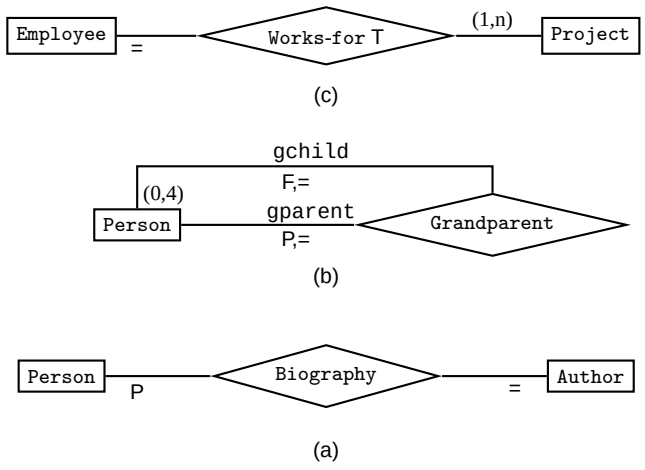

Figure 6. Across-Time Relationships

Biography be an across-time relationship linking the author of a biography with a famous person no more in existence. Snapshot reducibility says that if there is an instance (say, bio $= \langle$Tulard, Napoleon$\rangle$) of the Biography relationship at time t_0 (Tulard wrote a bio on Napoleon in 1984), then, the projection of Biography at time t_0 (1984 in our example) must contain the pair $\langle$Tulard, Napoleon$\rangle$. Now, while Tulard is a member of the class Author in 1984, we cannot say that Napoleon is an active member of the class Person in 1984. Our formalization of across-time relationships proposes the use of status classes to preserve snapshot reducibility. The biography example can be solved by asserting that Napoleon is a member of the Disabled-Person class in 1984—i.e. the disabled status associated to the class Person.

At the conceptual level, we mark relationship roles with P,=,S,F (standing for Past, Now, Suspended and Future, respectively). The role's mark expresses that the class typing the role participates in the relationship as disabled, active, suspend or scheduled. Furthermore, we allow to freely compose the marks, e.g. $\langle$P,=$\rangle$ denotes a role to a past or current object—i.e the class participates as either disabled or active—while $\langle$F,=$\rangle$ stands for a role to a future or current object (see Figure 6).

Remark 7.4. Note that, across-time relationships represent a generalization of the classical notion of relationships. They do not impose any temporal constraint on the involved objects allowing to capture a simplified version of the synchronization relationships introduced in MADS [38]. In particular, if no mark is explicitly stated on a relationship's role (as in the case of the role restricted to Project in Figure 6.c) we implicitly assume the compound mark $\langle$P,=,S,F$\rangle$—said *full-cross* mark. This assumption changes the semantics for relationships as given in Section 5. We assume, by default, that for each relationship's role the full-cross semantics holds (see the formal definition below). This new semantics for relationships maintains the compositionality of the language. In particular, to force a relationship to hold on an active class we need to add the $\langle$=$\rangle$ mark

(as in the case of the role restricted to `Employee` in Figure 6.c). Furthermore, cardinality constraints apply to a participating class in every status as specified by the corresponding role mark. For example, from Figure 6.b, each active or scheduled person can have at most four grandparents.

Formalization. Let R be a relationship, then, the semantics of marking the U_1-labeled role of the relationship is (we report the semantics for the single marks and the full-cross compound mark, the other compound marks are just the disjunction of the single ones):

$\langle o_1, o_2\rangle \in R^{\mathcal{B}(t)} \rightarrow o_1 \in \mathtt{C_1}^{\mathcal{B}(t)}$	Now $\langle$=$\rangle$
$\langle o_1, o_2\rangle \in R^{\mathcal{B}(t)} \rightarrow o_1 \in \mathtt{Disabled\text{-}C_1}^{\mathcal{B}(t)}$	Past $\langle$P$\rangle$
$\langle o_1, o_2\rangle \in R^{\mathcal{B}(t)} \rightarrow o_1 \in \mathtt{Scheduled\text{-}C_1}^{\mathcal{B}(t)}$	Future $\langle$F$\rangle$
$\langle o_1, o_2\rangle \in R^{\mathcal{B}(t)} \rightarrow o_1 \in \mathtt{Suspended\text{-}C_1}^{\mathcal{B}(t)}$	Suspended $\langle$S$\rangle$
$\langle o_1, o_2\rangle \in R^{\mathcal{B}(t)} \rightarrow o_1 \in (\mathtt{Exists\text{-}C_1} \sqcup \mathtt{Disabled\text{-}C_1})^{\mathcal{B}(t)}$	Full-Cross $\langle$P,=,S,F$\rangle$

The corresponding $\mathcal{DLR}_{\mathcal{US}}$ axioms are:

$R \sqsubseteq U_1 : \mathtt{C_1}$	Now $\langle$=$\rangle$
$R \sqsubseteq U_1 : \mathtt{Disabled\text{-}C_1}$	Past $\langle$P$\rangle$
$R \sqsubseteq U_1 : \mathtt{Scheduled\text{-}C_1}$	Future $\langle$F$\rangle$
$R \sqsubseteq U_1 : \mathtt{Suspended\text{-}C_1}$	Suspended $\langle$S$\rangle$
$R \sqsubseteq U_1 : (\mathtt{Exists\text{-}C_1} \sqcup \mathtt{Disabled\text{-}C_1})$	Full-Cross $\langle$P,=,S,F$\rangle$

We say that a role in an across-time relationship is *strict historical* if its mark does not contain the $\langle$=$\rangle$ mark (e.g. $\langle$P,F$\rangle$ is strict historical while $\langle$P,=$\rangle$ is not). The following Proposition shows how timestamping interacts with across-time relationships.

Proposition 7.5. (Across-Time: Logical Implications) The following logical implications hold as a consequence of the across-time semantics.

1. *If a relationship, R, has at least one strict historical role, then it is a temporary relationship.*
   $\Sigma_{st} \cup \{R \sqsubseteq U_i : (\mathtt{Disabled\text{-}C_i} \sqcup \mathtt{Scheduled\text{-}C_i} \sqcup \mathtt{Suspended\text{-}C_i})\} \models R \sqsubseteq \Diamond^+\neg R \sqcup \Diamond^-\neg R$
2. *If a relationship, R, is snapshot then all the historical marks must contain the $\langle$=$\rangle$ mark (i.e. R does not have strict historical roles).*
3. *If a relationship, R, has a strict historical role to a class C_1, then the C_1 class cannot be snapshot. Moreover, if the participation for C_1 is total, the class is temporary.*
   $\Sigma_{st} \cup \{R \sqsubseteq U_1 : (\mathtt{Disabled\text{-}C_1} \sqcup \mathtt{Scheduled\text{-}C_1} \sqcup \mathtt{Suspended\text{-}C_1})\} \models R \sqsubseteq U_1 : \Diamond^*[C_1 \sqcap (\Diamond^+\neg C_1 \sqcup \Diamond^-\neg C_1)]$

Proof

1. $\Sigma_{st} \cup \{R \sqsubseteq U_i : (\mathtt{Disabled\text{-}C_i} \sqcup \mathtt{Scheduled\text{-}C_i} \sqcup \mathtt{Suspended\text{-}C_i})\} \models R \sqsubseteq \Diamond^+\neg R \sqcup \Diamond^-\neg R$

Let $r = \langle o_1, \ldots, o_i, \ldots, o_n \rangle \in R^{\mathcal{B}(t)}$. Then, one of the following cases hold: $i)$ $o_i \in \texttt{Disabled-C}_\texttt{i}^{\mathcal{B}(\texttt{t})}$ and, by (DISAB2), $\exists t_1 < t$ s.t. $o_i \in C_i^{\mathcal{B}(t_1)}$; $ii)$ $o_i \in \texttt{Scheduled-C}_\texttt{i}^{\mathcal{B}(\texttt{t})}$ and, by (SCH1), $\exists t_1 > t$ s.t. $o_i \in C_i^{\mathcal{B}(t_1)}$; $iii)$ $o_i \in \texttt{Suspended-C}_\texttt{i}^{\mathcal{B}(\texttt{t})}$ and, by (SUSP), $\exists t_1 < t$ s.t. $o_i \in C_i^{\mathcal{B}(t_1)}$. Then, $\exists t_1 \neq t$ s.t. $o_i \in C_i^{\mathcal{B}(t_1)}$, thus $\langle o_1, \ldots, o_i, \ldots, o_n \rangle \notin R^{\mathcal{B}(t_1)}$.

2. Direct consequence of the above point.

3. $\Sigma_{st} \cup \{R \sqsubseteq U_1 : (\texttt{Disabled-C}_1 \sqcup \texttt{Scheduled-C}_1 \sqcup \texttt{Suspended-C}_1)\} \models R \sqsubseteq U_1 : \Diamond^*[C_1 \sqcap (\Diamond^+ \neg C_1 \sqcup \Diamond^- \neg C_1)]$
Let $r = \langle o_1, \ldots, o_i, \ldots, o_n \rangle \in R^{\mathcal{B}(t)}$. Then, $o_1 \in (\texttt{Disabled-C}_1 \sqcup \texttt{Scheduled-C}_1 \sqcup \texttt{Suspended-C}_1)^{\mathcal{B}(t)}$, and, by the disjointness constraints in Σ_{st}, $o_1 \notin C_1^{\mathcal{B}(t)}$. On the other hand, similarly to point 1, $\exists t_1 \neq t$ s.t. $o_1 \in C_1^{\mathcal{B}(t_1)}$.

❑

7.4. Generation Relationships

Generation relationships [37,21,35] represent processes that lead to the emergence of *new objects* starting from a set of existing objects. In their most generic form, a generation relationship can have a collection of objects as source and a collection of objects as target. For example (see Figure 7), assuming an organization remodels its departments, it may be that an existing department is split into two new departments, while two existing departments are merged into a single new department and three existing departments are reorganized as two new departments. Cardinality constraints can be added to specify the cardinality of sets involved in a generation. For example, if we want to record the fact that a group of managers proposes at most one new project at a time a generation relationship from `Manager` to `Project` can be defined with the cardinality "*at most one*" on the manager side.

Depending whether the source objects are preserved (as member of the source class) or disabled by the generation process, we distinguish between *production* and *transformation* relationships, respectively. Managers creating projects is an example of the former, while departmental reorganization is an example of the latter. At the conceptual level we introduce two marks for generation relationships: GP for production and GT for transformation relationships, and an arrow pointing to the target class (see Figure 7).

Formalization. We model generation as binary relationships connecting a source class to a target one: $\text{REL}(R) = \langle \texttt{source} : C_1, \texttt{target} : C_2 \rangle$. The semantics of *production relationships*, R, is described by the following equation:

$$\langle o_1, o_2 \rangle \in R^{\mathcal{B}(t)} \rightarrow (o_1 \in C_1^{\mathcal{B}(t)} \wedge o_2 \in \texttt{Scheduled-C}_2{}^{\mathcal{B}(t)} \wedge o_2 \in C_2^{\mathcal{B}(t+1)})$$

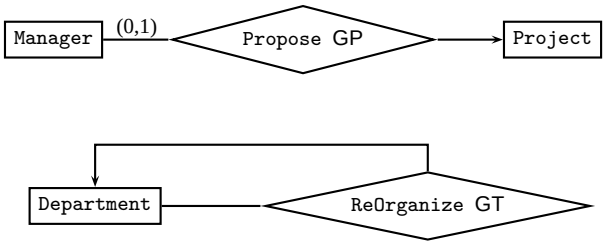

Figure 7. Production and transformation generation relationships.

Thus, objects active in the source class produce objects active in the target class at the next point in time. A production relationship is a special case of across-time relationship with an $\langle = \rangle$ mark on the source role and an $\langle \mathsf{F} \rangle$ mark on the target role. As for across-time relationships, the use of status classes allows us to preserve snapshot reducibility. Indeed, for each pair of objects, $\langle o_1, o_2 \rangle$, belonging to a generation relationships o_1 is active in the source while o_2 is scheduled in the target. The $\mathcal{DLR}_{\mathcal{US}}$ axiom capturing the production semantics is:

(PROD) $R \sqsubseteq \texttt{source} : C_1 \sqcap \texttt{target} : (\texttt{Scheduled-C}_2 \sqcap \oplus C_2)$

The case of *transformation* is captured by the following semantic equation:

$$\langle o_1, o_2 \rangle \in R^{\mathcal{B}(t)} \rightarrow (o_1 \in C_1^{\mathcal{B}(t)} \wedge o_1 \in \texttt{Disabled-C}_1{}^{\mathcal{B}(t+1)} \wedge o_2 \in \texttt{Scheduled-C}_2{}^{\mathcal{B}(t)} \wedge o_2 \in C_2^{\mathcal{B}(t+1)})$$

Thus, objects active in the source generate objects active in the target at the next point in time while the source objects cease to exist as member of the source. As for production relationships, transformations are special cases of across-time relationships. The $\mathcal{DLR}_{\mathcal{US}}$ axiom capturing the transformation semantics is:

(TRANS) $R \sqsubseteq \texttt{source}:(C_1 \sqcap \oplus \texttt{Disabled-C}_1) \sqcap \texttt{target}:(\texttt{Scheduled-C}_2 \sqcap \oplus C_2)$

Proposition 7.6 (Generation: Logical Implications).The following logical implications hold as a consequence of the generation semantics:

1. *A generation relationship, R, is temporary; actually, it is instantaneous.*
   $\Sigma_{st} \cup \{(\text{PROD})\} \models R \sqsubseteq \Box^+ \neg R \sqcap \Box^- \neg R$
   Indeed, let $\langle o_1, o_2 \rangle \in R^{\mathcal{B}(t)}$, then, since $o_2 \notin \texttt{Scheduled-C}_2^{\mathcal{B}(\texttt{t+1})}$, then $\langle o_1, o_2 \rangle \notin R^{\mathcal{B}(t+1)}$. Since, $o_2 \notin C_2^{\mathcal{B}(t)}$, then $\langle o_1, o_2 \rangle \notin R^{\mathcal{B}(t-1)}$.

2. *The target class, C_2, cannot be snapshot. Moreover, if the participation for C_2 is total, the class is temporary.*
   $\Sigma_{st} \cup \{(\text{PROD})\} \models R \sqsubseteq \texttt{target}:\Diamond^*[C_2 \sqcap (\Diamond^+ \neg C_2 \sqcup \Diamond^- \neg C_2)]$
   Indeed, let $\langle o_1, o_2 \rangle \in R^{\mathcal{B}(t)}$, then, $o_2 \notin C_2^{\mathcal{B}(t)}$ and $o_2 \in C_2^{\mathcal{B}(t+1)}$.

3. *Objects participating as target cannot be disabled.*

$\Sigma_{st} \cup \{(\text{PROD})\} \models R \sqsubseteq \texttt{target}{:}\neg\texttt{Disabled-C}_2$
Indeed, let $\langle o_1, o_2\rangle \in R^{\mathcal{B}(t)}$, then, $o_2 \in C_2^{\mathcal{B}(t+1)}$. Thus $o_2 \notin \texttt{Disabled-C}_2^{\mathcal{B}(\texttt{t})}$.

4. *If R is a transformation relationship, then, C_1 cannot be snapshot. Moreover, if the participation for C_1 is total, the class is temporary.*
$\Sigma_{st} \cup \{(\text{TRANS})\} \models R \sqsubseteq \texttt{source}{:}\Diamond^*[C_1 \sqcap (\Diamond^+\neg C_1 \sqcup \Diamond^-\neg C_1)]$
Indeed, objects in C_1 that participate in R will be disabled at the next point in time.

Proof

1. $\Sigma_{st} \cup \{(\text{PROD})\} \models R \sqsubseteq \Box^+\neg R \sqcap \Box^-\neg R$
Indeed, let $\langle o_1, o_2\rangle \in R^{\mathcal{B}(t)}$, then, since $o_2 \notin \texttt{Scheduled-C}_2^{\mathcal{B}(\texttt{t+1})}$, then $\langle o_1, o_2\rangle \notin R^{\mathcal{B}(t+1)}$. Since, $o_2 \notin C_2^{\mathcal{B}(t)}$, then $\langle o_1, o_2\rangle \notin R^{\mathcal{B}(t-1)}$.
2. $\Sigma_{st} \cup \{(\text{PROD})\} \models R \sqsubseteq \texttt{target}{:}\Diamond^*[C_2 \sqcap (\Diamond^+\neg C_2 \sqcup \Diamond^-\neg C_2)]$
Indeed, let $\langle o_1, o_2\rangle \in R^{\mathcal{B}(t)}$, then, $o_2 \notin C_2^{\mathcal{B}(t)}$ and $o_2 \in C_2^{\mathcal{B}(t+1)}$.
3. $\Sigma_{st} \cup \{(\text{PROD})\} \models R \sqsubseteq \texttt{target}{:}\neg\texttt{Disabled-C}_2$
Indeed, let $\langle o_1, o_2\rangle \in R^{\mathcal{B}(t)}$, then, $o_2 \in C_2^{\mathcal{B}(t+1)}$. Thus $o_2 \notin \texttt{Disabled-C}_2^{\mathcal{B}(\texttt{t})}$.
4. $\Sigma_{st} \cup \{(\text{TRANS})\} \models R \sqsubseteq \texttt{source}{:}\Diamond^*[C_1 \sqcap (\Diamond^+\neg C_1 \sqcup \Diamond^-\neg C_1)]$
Indeed, objects in C_1 that participate in R will be disabled at the next point in time.

❑

Note that the `Department` class that is both the source and target of a transformation relationship (Figure 7) can no longer be snapshot (as it was in Figure 3) and must be changed to temporary. This is an example of inconsistency checking that an automated reasoner could perform to avoid inconsistent classes in a temporal schema. Furthermore, as a consequence of this new timestamp for the `Department` class, `InterestGroup` is now a genuine mixed class (compare Figure 2 with Figure 3).

8. Complexity of Reasoning on Temporal Models

As this paper shows, the temporal description logic $\mathcal{DLR}_{\mathcal{US}}$ is able to fully capture temporal schemas with both timestamping and evolution constraints. Reasoning over $\mathcal{DLR}_{\mathcal{US}}$ knowledge bases, i.e checking satisfiability, subsumption and logical implications, turns out to be undecidable [5]. The main reason for this is the possibility to postulate that a binary relation does not vary in time. Note that, showing that temporal schemas can be mapped into $\mathcal{DLR}_{\mathcal{US}}$ axioms does not necessarily imply that reasoning over temporal schemas is an undecidable problem. Unfortunately, [2] shows that the undecidable Halting Problem can be encoded as the problem of class satisfiability w.r.t.

a temporal schema with, among the others, the following constructs: disjoint and covering constraints, sub-relationships, timestamping on both classes and relationships, and evolution constraints.

On the other hand, the fragment, $\mathcal{DLR}^{-}_{\mathcal{US}}$, of $\mathcal{DLR}_{\mathcal{US}}$ deprived of the ability to talk about temporal persistence of n-ary relations, for $n \geq 2$, is decidable. Indeed, reasoning in $\mathcal{DLR}^{-}_{\mathcal{US}}$ is an EXPTIME-complete problem [5]. This result gives us an useful scenario where reasoning over temporal schemas becomes decidable. In particular, if we forbid timestamping for relationships (i.e. relationships are just unmarked) reasoning on temporal models with just class timestamping but full evolution constraints can be reduced to reasoning over $\mathcal{DLR}^{-}_{\mathcal{US}}$. The problem of reasoning in this setting is complete for EXPTIME since the EXPTIME-complete problem of reasoning with $\mathcal{ALC}$ knowledge bases can be captured by such schemas [8].

We maintain decidability also by allowing full timestamping (i.e. timestamping for relationships, attributes and classes) but dropping evolution constraints. This is the basic temporal conceptual modelling scenario where temporal marks allow to distinguish between temporary and global constructs. A complete and decidable reasoning procedure for reasoning over timestamping is provided in [6]. This scenario is decidable since it is possible to encode temporal schemas without evolution constraints by using a combination between the description logic $\mathcal{DLR}$ and the epistemic modal logic S5 (see [6] for the exact mapping). Reasoning over $\mathcal{DLR}_{S5}$ has been recently proved to be decidable and 2-EXPTIME-complete [6] by extending a previous result on the logic $\mathcal{ALC}_{S5}$ [17].

Other interesting scenarios under investigation are the cases where the temporal expressivity is maintained in its full capability (i.e. both full timestamping and evolution constraints) but some of the classical EER constructs are dropped. In particular, we claim that by dropping isa between relationships and/or partitioning constraints we could regain decidability in the full temporal scenario.

9. Conclusions

In this paper we proposed a formalization of the various modelling constructors that support the design of temporal DBMS with particular attention to evolution constraints. The formalization, based on a model-theoretic semantics, has been developed with the aim to preserve three fundamental modelling requirements: Orthogonality, Upward Compatibility and Snapshot Reducibility. The introduction of status classes, which describe the evolution in the membership of an object in a temporal class, allowed us to maintain snapshot reducibility when characterizing both generations and across-time relationships. The formal semantics clarified the meaning of the language's constructors and also gave a rigorous definition to relevant modelling notions like: satisfiability of schemas, classes and relationships; subsumption for both classes and relationships; logical implication. Furthermore, for each constructor we presented its formalization together with the associated set of logical implications.

Finally, we have been able to show how temporal schemas can be equivalently

expressed using a subset of first-order temporal logic, i.e. $\mathcal{DLR}_{\mathcal{US}}$, the description logic $\mathcal{DLR}$ extended with the temporal operators *Since* and *Until*. Overall, we obtained a temporal conceptual model that preserves well established modelling requirements, equipped with a model-theoretic semantics where each constructor can be seen as a set of precise rules, and with the possibility to perform automated reasoning by mapping temporal schemas into temporal description logic knowledge bases.

References

[1] B. Ahmad. Modeling bi-temporal databases. Master's thesis, UMIST Department of Computation, UK, 2003.

[2] A. Artale. Reasoning on temporal conceptual schemas with dynamic constraints. In *11th Int. Symposium on Temporal Representation and Reasoning (TIME04)*. IEEE Computer Society, 2004. Also in Proc. of the 2004 Int. Workshop on Description Logics (DL'04).

[3] A. Artale and E. Franconi. Temporal ER modeling with description logics. In *Proc. of the Int. Conf. on Conceptual Modeling (ER'99)*. Springer-Verlag, November 1999.

[4] A. Artale, E. Franconi, and F. Mandreoli. Description logics for modelling dynamic information. In Jan Chomicki, Ron van der Meyden, and Gunter Saake, editors, *Logics for Emerging Applications of Databases*. Lecture Notes in Computer Science, Springer-Verlag, 2003.

[5] A. Artale, E. Franconi, F. Wolter, and M. Zakharyaschev. A temporal description logic for reasoning about conceptual schemas and queries. In S. Flesca, S. Greco, N. Leone, and G. Ianni, editors, *Proceedings of the 8th Joint European Conference on Logics in Artificial Intelligence (JELIA-02)*, volume 2424 of *LNAI*, pages 98–110. Springer, 2002.

[6] Alessandro Artale, Carsten Lutz, and David Toman. A description logic of change. In *Int. Joint Conference on Artificial Intelligence (IJCAI-07)*, Hyderabad, India, Jan 2007.

[7] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, 2002.

[8] Daniela Berardi, Diego Calvanese, and Giuseppe De Giacomo. Reasoning on UML class diagrams. *Artificial Intelligence*, 168(1–2):70–118, 2005.

[9] D. Calvanese, G. De Giacomo, and M. Lenzerini. On the decidability of query containment under constraints. In *Proc. of the 17th ACM SIGACT SIGMOD SIGART Sym. on Principles of Database Systems (PODS'98)*, pages 149–158, 1998.

[10] D. Calvanese, M. Lenzerini, and D. Nardi. Description logics for conceptual data modeling. In J. Chomicki and G. Saake, editors, *Logics for Databases and Information Systems*. Kluwer, 1998.

[11] D. Calvanese, M. Lenzerini, and D. Nardi. Unifying class-based representation formalisms. *J. of Artificial Intelligence Research*, 11:199–240, 1999.

[12] J. Chomicki and D. Toman. Temporal logic in information systems. In J. Chomicki and G. Saake, editors, *Logics for Databases and Information Systems*, chapter 1. Kluwer, 1998.

[13] M. A. Pacheco e Silva. Dynamic integrity constraints definition and enforcement in databases: A classification framework. In *Proc. of the IFIP TC11 Working Group 11.5, First Working Conf. on Integrity and Internal Control in Information Systems*, pages 65–87, London, UK, 1997. Chapman & Hall, Ltd.

[14] Ramez Elmasri, James A. Weeldreyer, and Alan R. Hevner. The category concept: An extension to the entity-relationship model. *Data & Knowledge Engeneering*, 1(1):75–116, 1985.

[15] O. Etzion, A. Gal, and A. Segev. Extended update functionality in temporal databases. In O. Etzion, S. Jajodia, and S. Sripada, editors, *Temporal Databases - Research and Practice*, Lecture Notes in Computer Science, pages 56–95. Springer-Verlag, 1998.

[16] M. Finger and P. McBrien. Temporal conceptual-level databases. In D. Gabbay, M. Reynolds, and

M. Finger, editors, *Temporal Logics – Mathematical Foundations and Computational Aspects*, pages 409–435. Oxford University Press, 2000.

[17] D. Gabbay, A.Kurucz, F. Wolter, and M. Zakharyaschev. *Many-dimensional modal logics: theory and applications*. Studies in Logic. Elsevier, 2003.

[18] Georg Gottlob, Michael Schrefl, and Brigitte Röck. Extending object-oriented systems with roles. *ACM Transaction on Information Systems*, 14(3):268–296, 1996.

[19] H. Gregersen and J.S. Jensen. Conceptual modeling of time-varying information. Technical Report TimeCenter TR-35, Aalborg University, Denmark, 1998.

[20] H. Gregersen and J.S. Jensen. Temporal Entity-Relationship models – a survey. *IEEE Transactions on Knowledge and Data Engineering*, 11(3):464–497, 1999.

[21] R. Gupta and G. Hall. An abstraction mechanism for modeling generation. In *Proc. of ICDE'92*, pages 650–658, 1992.

[22] G. Hall and R. Gupta. Modeling transition. In *Proc. of ICDE'91*, pages 540–549, 1991.

[23] I. Hodkinson, F. Wolter, and M. Zakharyaschev. Decidable fragments of first-order temporal logics. *Annals of Pure and Applied Logic*, 106:85–134, 2000.

[24] I. Horrocks, P.F. Patel-Schneider, and F. van Harmelen. From SHIQ and RDF to OWL: The making of a web ontology language. *Journal of Web Semantics*, 1(1):7–26, 2003.

[25] C. S. Jensen, J. Clifford, S. K. Gadia, P. Hayes, and S. Jajodia et al. The Consensus Glossary of Temporal Database Concepts. In O. Etzion, S. Jajodia, and S. Sripada, editors, *Temporal Databases - Research and Practice*, pages 367–405. Springer-Verlag, 1998.

[26] C. S. Jensen and R. T. Snodgrass. Temporal data management. *IEEE Transactions on Knowledge and Data Engineering*, 111(1):36–44, 1999.

[27] C. S. Jensen, M. Soo, and R. T. Snodgrass. Unifying temporal data models via a conceptual model. *Information Systems*, 9(7):513–547, 1994.

[28] Yahiko Kambayashi and Zhiyong Peng. Object deputy model and its applications. In *Proc. of the 4th Int. Conf. on Database Systems for Advanced Applications (DASFAA)*, pages 1–15. World Scientific Press, 1995.

[29] Qing Li and Guozhu Dong. A framework for object migration in object-oriented databases. *Data & Knowledge Engeneering*, 13(3):221–242, 1994.

[30] Qing Li and Frederick H. Lochovsky. Adome: An advanced object modeling environment. *IEEE Transactions on Knowledge and Data Engineering*, 10(2):255–276, 1998.

[31] P. McBrien, A.H. Seltveit, and B. Wangler. An Entity-Relationship model extended to describe historical information. In *Proc. of CISMOD'92*, pages 244–260, Bangalore, India, 1992.

[32] Alberto O. Mendelzon, Tova Milo, and Emmanuel Waller. Object migration. In *Proc. of the 13th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS94)*, pages 232–242, New York, NY, USA, 1994. ACM Press.

[33] Erik Odberg. Category classes: flexible classification and evolution in object-oriented databases. In *Proc. of the 6th Int. Conf. on Advanced information systems engineering (CAiSE94)*, LNCS 881, pages 406–420, Secaucus, NJ, USA, 1994. Springer-Verlag New York, Inc.

[34] Mike P. Papazoglou, Bernd J. Kramer, and Athman Bouguettaya. On the representation of objects with polymorphic shape and behaviour. In *Int. Conf. on Conceptual Modeling / the Entity Relationship Approach*, LNCS 881, pages 223–240, 1994.

[35] C. Parent, S. Spaccapietra, and E. Zimanyi. The MurMur project: Modeling and querying multi-representation spatio-temporal databases. *Information Systems*, 31(8):733–769, 2006.

[36] B. Pernici. Objects with roles. In *Proc. of the ACM SIGOIS and IEEE CS TC-OA conference on Office information systems*, pages 205–215, New York, NY, USA, 1990. ACM Press.

[37] S. Spaccapietra, C. Parent, and E. Zimanyi. Modeling time from a conceptual perspective. In *Int. Conf. on Information and Knowledge Management (CIKM98)*, 1998.

[38] S. Spaccapietra, C. Parent, and E. Zimanyi. *Conceptual Modeling for Traditional and Spatio-Temporal Applications—The MADS Approach*. Springer, 2006.

[39] Jianwen Su. Dynamic constraints and object migration. *Theoretical Computer Science*, 184(1-2):195–236, 1997.

[40] C. Theodoulidis, P. Loucopoulos, and B. Wangler. A conceptual modelling formalism for temporal database applications. *Information Systems*, 16(3):401–416, 1991.

[41] Wikipedia. Wikipedia, the free encyclopedia. Temporal Databases. see `http://en.wikipedia.org/wiki/Temporal_database`.