A Framework for the Evolution of Temporal Conceptual Schemas of Information Systems
(Extended Version)

Juan-Ramón López[1,2], Antoni Olivé[1]

[1]Universitat Politècnica de Catalunya,
Departament de Llenguatges i Sistemes Informàtics.
e-mail: {jrlopez|olive}@lsi.upc.es
[2]Universidade da Coruña,
Departamento de Computación.

A Framework for the Evolution of Temporal Conceptual Schemas of Information Systems
(Extended version)

Abstract

This paper focuses on the problem of information systems evolution. Ideally, changes in the requirements of information systems should be defined and managed at the conceptual schema level, with an automatic propagation down to the logical database schema(s) and application programs. As one step more towards the achievement of this ideal approach, we propose here a framework for the evolution of temporal conceptual schemas of information systems.

Our framework uses a reflective architecture with two levels: meta schema and schema, and two loosely coupled information processors, one for each level. We define a temporal minimal meta schema, and we show, using some examples, how to extend this minimal meta schema to support any usual conceptual modeling construct. We also show how the framework can be used to specify different kinds of schema changes. These changes are defined as meta external events, with explicit and implicit effects over the meta information base/conceptual schema and the information base. Two detailed examples are given to illustrate this approach.

Keywords:
Temporal conceptual schemas, evolution, meta schemas, reflective architecture.

Table of contents

.

1 INTRODUCTION

This paper deals with the evolution of temporal conceptual schemas of information systems. We propose a general framework that allows defining schema evolutions and their effects. We illustrate the framework with a limited albeit representative number of widely used conceptual modeling constructs, define some evolution operations at the meta schema level, and analyze their impact on the conceptual schema and the information base.

Our work aims at contributing to the general field of information systems evolution. The field attracts many research efforts, due to the great practical importance of information systems evolution for many organizations. Evolution activities consume a large amount of resources, and this justifies the interest of research leading to improvements in the way information systems evolve [Lien83, Aria91, Sjøb93].

Evolution activities can be classified in several ways [SoGo79]. In order to characterize our contribution, we find it useful to start with the abstract database evolution framework presented in [HEH+94], that we rephrase as follows: Assume that an information system satisfies requirements R0. The information system comprises a conceptual schema CS0, one (or more) database(s) with logical schema(s) LS0 and extension(s) E0, and several application programs P0. When requirements change to R1, the information system must evolve. In the general case, the evolution implies changes leading to a new conceptual schema CS1, new logical schema(s) LS1 and application programs P1. Moreover, the extension(s) E0 may not be valid, and need to be converted to E1.

Ideally, the evolution of information systems should follow the strategy of "forward information system maintenance" [HEH+94]: changes should be applied to the conceptual schema, and from here they should propagate automatically down to the logical schema(s) and application programs. If needed, the extension(s) should be also converted automatically. In practice, however, organizations do not follow always this strategy and, due to the current state of the art, they do not have the tools required to perform the evolution in a completely automatic way.

In the context of the above framework, the aspect that has received more research effort is (logical) database schema evolution, including the propagation of changes to the extensional database. Data models that have been studied extensively include the relational and object-oriented models, both non-temporal and temporal. In several cases, research results have been incorporated into commercial or prototype database management systems (e.g., Orion [BCG+87], O_2 [Zica92], F2 [AlLe98]). We refer to [NgRi89, Casa90, Rodd94, Lemk94, Rodd95, Laut99] for extensive bibliographies and surveys.

We observe that the ideal strategy described above implies that the conceptual schema is the only description to be defined, and the basis for the specification of the evolution. All the others are internal to the system, and need not to be visible outside. This observation has led us to our framework, which we see as one step more towards that ideal strategy. Our framework is based on the commonly accepted approach that, from a conceptual point of view, an information system consists of a conceptual schema, an information base and an information processor. The conceptual schema defines all general domain knowledge relevant to the system, which includes entity and relationship types, all kind of integrity constraints and derivation rules, and the effects of external events. We take into account the temporal dimension of conceptual schemas and information bases, which is particularly relevant in the information systems field [ThLo91]. The information processor receives messages from the environment, reporting the occurrence of external events. In response to such events, the information processor changes the information base and/or issues messages making known the contents of the information base, in the way specified in the conceptual schema [ISO82].

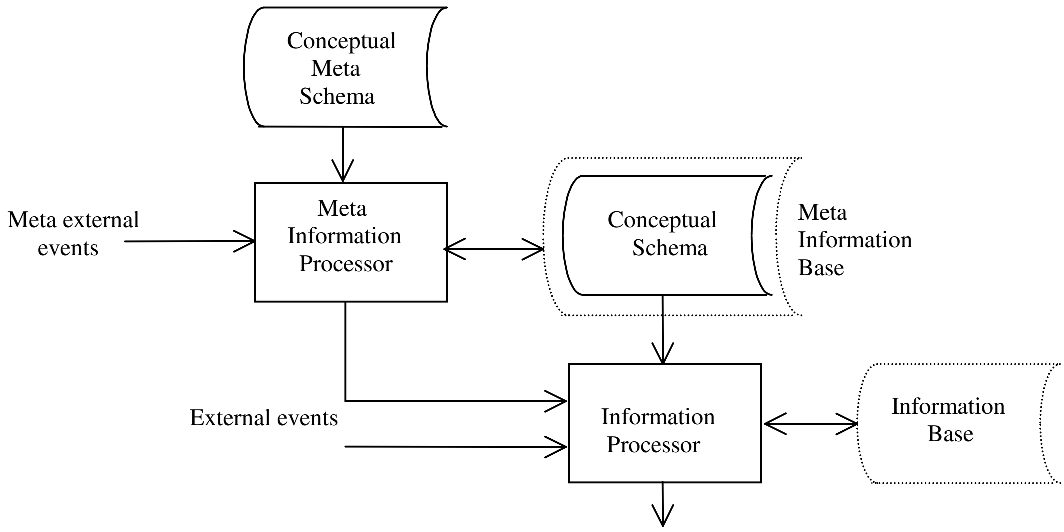

Figure 1. Framework for the evolution of information systems.

In this view, evolution can be accommodated elegantly by using the classical reflective architecture [Mant93, PeÖz93], as shown in Figure 1. A distinction is made between an information system and a meta information system. The first consists of a conceptual schema, an information base (IB) and an information processor (IP). The second consists of a conceptual meta schema, a meta information base (MIB) and a meta information processor (MIP). In essence, the MIB of the meta information system describes the conceptual schema of the information system [ISO82].

In our framework, possible changes in the conceptual schema are specified as meta external events in the conceptual meta schema. Occurrences of these events are reported to the MIP, which changes the MIB and indirectly the conceptual schema. In some cases, changes in the IB may be necessary too. These latter changes are not performed directly by the MIP. Instead of this, the MIP generates occurrences of external events, which are notified to the IP to perform the desired changes. In this way, the two processors are loosely coupled.

There is a lot of previous research and development work relevant to this framework. We can only give in the following a few comments relating a representative part of this work with our paper. More comments will be given in the rest of the paper.

[TrSc92] distinguishes between three levels of objects: data, schema and metaschema. Using the COCOON data model, schema evolution is achieved by invoking operations on metaobjects. Our framework incorporates that distinction, and adds the temporal dimension, general integrity constraints, general derivation rules and external events.

[LeMa95] describes SEA, Schema Evolution Assistant, a tool that allows the evolution of schemas in Chimera. Chimera is a rich data model that includes integrity constraints, derivation rules and triggers. SEA uses a reflective architecture [Lemk95], as we do. Initially, Chimera was non-temporal, but a temporal extension was published later [BeFG97]. SEA was developed for the non-temporal version. Our framework could be useful for a temporal version. On the other hand, we will see that our framework could be useful to extend the Chimera conceptual meta schema with additional constructs.

[JGJ+95, JJN+98] describe ConceptBase, a system based on the O-Telos language, a variant of Telos [MBJK90]. O-Telos deals uniformly with objects at different levels of abstraction. O-Telos has a kernel, from which all other constructs can be bootstrapped. We have followed the same idea with our minimal meta schema. However, our framework allows that the information and the meta information processors be different. This may be interesting from a practical point of view. The former might be, for example, a temporal SQL-based system, while the latter might be Prolog-based. On the other hand, we deal with the temporal dimension and focus more on schema evolution.

Finally, we mention the work done in the context of the TIGUKAT temporal ODBMS [PeÖz93, PeÖz97, GSÖP98]. The TIGUKAT schema objects include types, classes, behaviours, functions and collections. The model allows the representation of the temporal histories of real-world objects. We also adopt the temporal view, but conceptual schemas of information systems need to include other elements like integrity constraints, derivation rules, external events, etc. Evolution is performed by operations (add, drop and modify) on schema objects. These are similar to our meta external events, but we need many more to allow the evolution of other schema objects.

The remainder of this paper is organized as follows. In Section 2, we characterize temporal conceptual schemas and evolution at the temporal IB level. Our characterization is very general and could be adapted to any temporal conceptual model. In Section 3 we do the same for the conceptual meta schema and MIB, and define the correspondence between the MIB and the conceptual schema. We also show how usual conceptual modeling constructs can be defined at the conceptual meta schema level. Due to space limitations, it is impossible to describe all usual constructs. For this reason we only explain four of them, which we think are representative enough. In Section 4 we then define the meta external events that perform schema evolution. Again, we can only define a few of them, which are explained in some detail. Finally, Section 5 gives the conclusions and points out future work.

2 TEMPORAL CONCEPTUAL SCHEMA AND INFORMATION BASE

In this section, we briefly characterize the elements of a temporal conceptual schema (or, for short, schema). We wish our work to be of general applicability and language-independent. For this reason, we focus here on a set of minimal core elements of schemas. Any schema can be expressed in terms of these elements. They are entity types, relationship types, derivation rules, integrity constraints and external events [BBJW97]. We represent them (and their instances in the IB) using the language of the first order logic (FOL), with some common extensions. In Section 3 we show how some usual constructs in conceptual modeling (cardinality constraints, specialization, generalization, partition, dynamic constraints, etc.) can be defined in the conceptual meta schema and translated into the elements described here.

2.1 Temporal Conceptual Schema

The entities of a domain are instance of *entity types* at particular time points [Bube77]. An entity may be instance of several entity types at the same time (multiple classification). On the other hand, an entity may change the entity types of which it is instance (dynamic classification). For example, an entity may be instance of *Employee* and *Single* at time *t*, and become *Married* at *t+1*.

In a temporal conceptual schema, entity types may be represented by binary predicates, whose first term is a symbol that denotes an entity in the domain, and the second a time point. In the IB, we represent by *E(e,t)* the fact that entity *e* is instance of entity type *E* at time *t*.

We assume that time is discrete, linear and totally ordered. Time points are expressed in a common base time unit (or *chronon* [TCG+93]), such as second or day. We also assume that a schema includes a special unary predicate *Time*, whose instances are the time points (in the chosen unit) of the information system lifespan. The extension of *Time* grows implicitly as time advances.

In general, the set of entity types defined in a schema is time-varying. We denote by E_t the set of entity types defined in a schema at time *t*. An entity type may cease to exist in a schema and appear again later. We adopt here the usual *typing rule*, which in our case requires that, for all entity types *E*:

$$\forall e,t\ (E(e,t) \rightarrow E \in E_t)$$

That is, we do not allow the IB to represent that *e* is instance of *E* at *t*, if *E* is not defined in the schema at *t*. For convenience, we assume that if an entity type *E* ceases to exist at time *t*, then all its instances at *t-1* (if any) also cease to exist at *t*. This assumption eases the definition of changes: if an entity type ceases to exist, we will not need to declare explicitly that all its instances cease too.

Entity types may be *lexical* or *non-lexical*. The former are those whose entities are words (or sentences) of the language used by the actors in the domain, while the latter are those whose entities are non-lexical objects in the domain [BBJW97]. Usual examples of lexical entity types are *String* and *Number*. For the sake of uniformity we represent lexical entity types also by binary predicates, even if lexical entities are usually instance of their types during the whole lifespan of an information system.

The relationships of a domain are instance of *relationship types* at particular time points. In a temporal schema, a relationship type with *n participants* may be represented by a predicate of degree *n+1*, where the first *n* terms are symbols that denote the participant entities, and the *n+1* term is a time point. Note that we do not restrict our schemas to binary relationship types. In the IB, we represent by $R(e_1,...,e_n,t)$ the fact that entities $e_1,...,e_n$ participate in a relationship instance of relationship type *R* at time *t*. We also say that $(e_1,...,e_n)$ is an instance of *R* at time *t*.

In general, the set of relationship types defined in a schema is time-varying, and we denote by R_t the set of relationship types defined in a schema at time *t*. The above typing rule also applies to relationship types.

Entity and relationship types may be *base* or *derived*. Instances of base types must be stated explicitly, as will be seen later, while those of derived types are defined by *derivation rules*. In a temporal schema, derivation rules must be considered also as time-varying. We denote by DR_t the set of derivation rules defined in a schema at time *t*. When needed, the IP computes (part of) the extension of a derived type at time *t* using the corresponding derivation rule in DR_t. Our framework allows derivation rules (and other expressions) to be written in the language understood by the IP. In the examples we use the FOL language. For example the derivation rule of entity type *Young* could be[1]:

$$\text{Young}(p,t) \leftarrow \text{Person}(p,t) \wedge \text{HasAge}(p,a,t) \wedge a \leq 18$$

A type may change from base to derived (or the other way around) at any time. For example, at some time point entity type *ImportantCustomer* could be derived in function of amount sold, balance, etc., and later become base if users wanted to state explicitly who is an important customer.

Integrity constraints are conditions that the IB must satisfy. We consider them also as time-varying. We denote by IC_t the set of integrity constraints defined in a schema at time *t*. It is assumed that at any time *t* the IB satisfies all integrity constraints defined in IC_t.

Our framework could be extended by distinguishing, in the schema, several kinds of integrity constraints (and of derivation rules). However, in this paper we prefer not to make this distinction here, but we do it in the conceptual meta schema. In that schema, we can define easily as many kinds of constraints as needed. The MIP will "generate" constraints in the language required by the IP.

The IB changes due to the occurrence of *external events*. As we did for entity types, we represent external event types in a schema by binary predicates, but now the first term denotes an event occurrence. In the IB we represent by *Ev(ev,t)* the occurrence of event *ev* of type *Ev* at time *t*. For example, *ChangeSalary(cs,t)* represents that the event *cs* of type *ChangeSalary* has occurred at time *t*. The parameters of the event are given by binary relationships with ordinary entities. For example, *ChangesSalaryOf(cs,emp,t)* represents the fact that event *cs* changes the salary of employee *emp*.

In general, the set of external event types defined in a schema is time-varying, and we denote by EE_t the set of external event types defined in a schema at time *t*. The above typing rule also applies to these types. External event types may be base or derived.

For each external event type, the schema includes the definition of its preconditions and effect rules (transactions). The *preconditions* are conditions that must be satisfied when an external event occurs. The *effect rules* define the changes produced in the IB by the event. Both preconditions and effect rules are formulas expressed in the language understood by the IP. As before, in this paper we will use the FOL language. For

[1] As usual, we assume that all free variables are universally quantified in front of a formula.

brevity, we do not take into consideration here postconditions, or triggering conditions for generated events, which would have a similar treatment.

We find convenient to allow the possibility of defining, for each integrity constraint, a set of *compensating* effect rules. The IP adds these rules to the effect rules of an event type when an occurrence of this event type leads to a violation of the integrity constraint. This, however, does not mean that violations can be prevented in all cases by compensating rules.

2.2 Information Base Evolution

The effect rules of an external event type define the changes to be produced in the IB when an instance of that type occurs. We model possible changes in the IB as structural events, and we then say that an external event induces structural events, which, in turn, change the IB. Other names used for structural events are elementary actions [ISO82] or basic events [MaOd95].

The *structural event types* are determined by the entity and relationship types defined in a conceptual schema and, thus, they are not defined by the designer. We represent structural event types by predicates, as follows:

- There is an insertion and a deletion event type for each non-lexical entity type *E*. We represent them by the binary predicates *Ins_E* and *Del_E*, respectively, where the first term is a symbol denoting an entity and the second a time point. The intended change produced in the IB by an event *Ins_E(e,t)* is that *e* becomes instance of *E* at time *t*. The intended change produced by an event *Del_E(e,t)* is that *e* ceases to be instance of *E* at *t*.

- There is an insertion and a deletion event type for each relationship type *R*. We represent them by the *n+1-ary* predicates *Ins_R* and *Del_R*, respectively, where the first *n* terms are symbols denoting entities and the *n+1* term is a time point. The intended change produced in the IB by an event $Ins\_R(e_1,...,e_n,t)$ is that $(e_1,...,e_n)$ becomes instance of *R* at time *t*. The intended change produced by an event $Del\_R(e_1,...,e_n,t)$ is that $(e_1,...,e_n)$ ceases to be instance of *R* at *t*.

For example, if a schema includes the binary relationship type *HasName*, between *Person* and *String*, there would be two structural event types: *Ins_HasName* and *Del_HasName*. To define that an occurrence of external event type *ChangeName*, with parameters given by relationship types *ChangesNameOf* and *HasNewName*, induces a change of a person's name, we could define the following simple effect rules for *ChangeName*:

$$\text{Del\_HasName}(p,n,t) \leftarrow \text{ChangeName}(cn,t) \wedge \text{ChangesNameOf}(cn,p,t) \wedge \text{HasName}(p,n,t-1)$$
$$\text{Ins\_HasName}(p,n',t) \leftarrow \text{ChangeName}(cn,t) \wedge \text{ChangesNameOf}(cn,p,t) \wedge \text{HasNewName}(cn,n',t)$$

A structural event type is base (derived) if the corresponding entity or relationship type is base (derived). External events induce directly only base structural events, while derived ones are induced indirectly by the derivation rules (there are several procedures to infer derived structural events; see, for instance, [Oliv89]). For example, if we have in a schema the derivation rule: $Young(p,t) \leftarrow Person(p,t) \wedge Age(p,a,t) \wedge a \leq 18$, then an increase of a person's age may induce a *Del_Young* event.

The extension of a base and non-lexical entity type *E* at time *t* is then defined declaratively by the *persistence axiom*:

$$E(e,t) \leftrightarrow Ins\_E(e,t_1) \wedge t_1 \leq t \wedge \neg\exists\, t_2\ (Del\_E(e, t_2) \wedge t_1 < t_2 \wedge t_2 \leq t)$$

Similar axioms define the extension of base relationship types. Persistence axioms are an implicit part of a schema. Note that persistence axioms are equivalent to the frame axioms in the situation and event calculus [VDD95].

The extension of lexical entity types is assumed to be constant during the information system lifespan, and known from the beginning. Thus, there are not structural event types defined for them. If a lexical entity type is derived, we assume that its derivation rule is constant.

3 TEMPORAL CONCEPTUAL META SCHEMA AND INFORMATION BASE

All conceptual models have a conceptual meta schema (or, for short, meta schema). A meta schema is like an ordinary schema, except that its domain is a schema. The elements of a schema are seen as entities, which are instance of appropriate entity types defined in the meta schema. Similarly, relationships between the elements of a schema are seen as instances of relationship types of the meta schema. The MIB has a representation of the schema.

In this section, we first describe the minimal meta schema necessary to represent the core elements defined in the previous section. We then establish the correspondence rules between the MIB and the schema. However, this minimal meta schema is hardly satisfactory, because it does not take into account the constructs that have been developed in the conceptual modeling field. In the last part of this section we describe how the meta schema can be enriched to deal with those constructs.

3.1 Minimal meta schema

Figures 2 and 3 illustrate, using the UML language [RuJB99], the minimal meta schema[2]. We now briefly define it in terms of the core elements explained in the previous section. For clarity, we call sometimes these elements meta elements, in order to distinguish them from schema elements.

Entity types. The entity types are shown as rectangles in Figures 2 and 3. All entity types are non-lexical, except *Formula*, which is lexical. An instance of *Formula* is a formula in the language understood by the IP. For generality, we use here the FOL language. All entity types are base, except *Concept*, *EntityType* and *Rule*, which are derived. As we did in Section 2.1, in the MIB we represent by *E(e,t)* the fact that *e* is instance of meta entity type *E* at time *t*. For example, *EntityType(Employee,T)*.

[2] Note that, in the figures, we use partitions and cardinality constraints. However, these constructs will be translated into core elements in what follows.

Relationship types. The meta relationship types are shown as lines linking rectangles in Figures 2 and 3. All of them are binary and base. Note that a schema relationship type *R* may be of degree 2 or higher. *R* would be an instance of *RelationshipType* at *T* (Figure 2) and, if it has degree *n*, there would be *n* instances $HasPt(R,P_i,T)$ in the MIB.

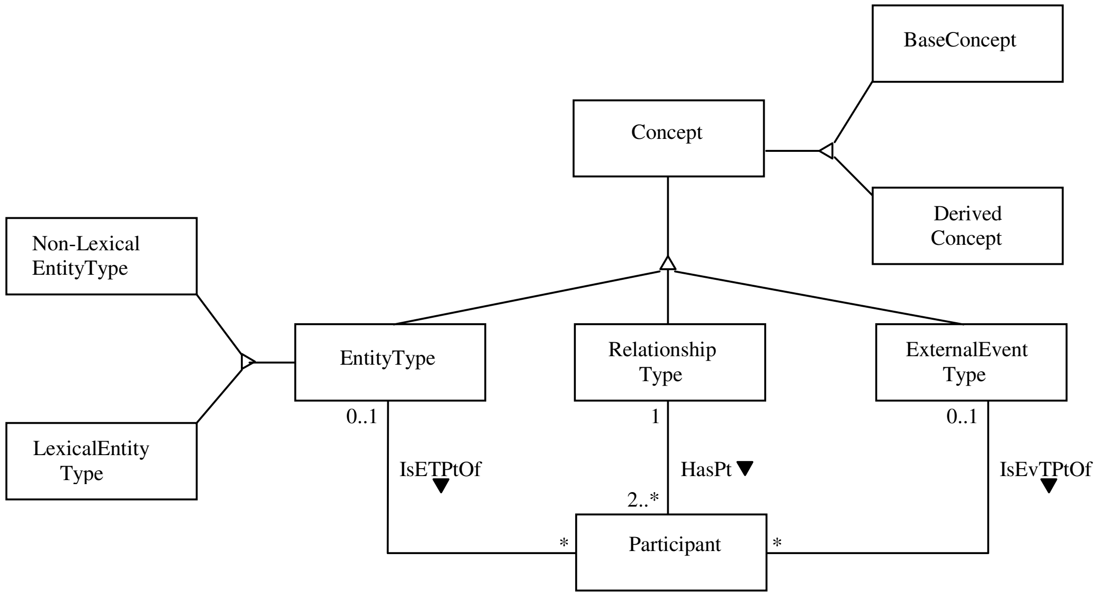

Figure 2. Minimal conceptual meta schema (Part 1 of 2)

Structural events. Meta structural events are defined as we did in Section 2.2. For clarity, we will prefix them with *M_*. For example, *M_Del_EntityType*, whose intended effect is that an entity type ceases to be instance of *EntityType* in the MIB.

Derivation rules. There are three meta derivation rules, which define *Concept*, *EntityType* and *Rule*. They are:

MDR1. Concept(c,t) ↔ EntityType(c,t) ∨ RelationshipType(c,t) ∨ ExternalEventType(c,t)
MDR2. EntityType(e,t) ↔ Non-LexicalEntityType(e,t) ∨ LexicalEntityType(e,t)
MDR3. Rule(r,t) ↔ Precondition(r,t) ∨ EffectRule(r,t) ∨ IntegrityConstraint(r,t) ∨ DerivationRule(r,t)

Integrity constraints. A schema is a complex structure subject to a large number of integrity constraints. This implies that the meta schema must include many meta integrity constraints, usually called *invariants* in the schema evolution field [BCG+87]. Here we can define only a few, including those that will be mentioned later.

MIC1. A lexical entity type cannot be at the same time non-lexical (recall that we allow multiple classification):

LexicalEntityType(e,t) → ¬Non-LexicalEntityType(e,t)

MIC2. A non-lexical entity type cannot be at the same time lexical. Formula as above.

MIC3. The participants in a relationship *IsETPtOf* must be an entity type and a participant:

IsETPtOf(e,p,t) → EntityType(e,t) ∧ Participant(p,t)

We define two compensating rules for MIC3. They state that when an entity type (or participant) is deleted so are the relationships *IsETPtOf* in which it participates:

M_Del_IsETPtOf(e,p,t) ← M_Del_EntityType(e,t) ∧ IsETPtOf(e,p,t-1)
M_Del_IsETPtOf(e,p,t) ← M_Del_Participant(p,t) ∧ IsETPtOf(e,p,t-1)

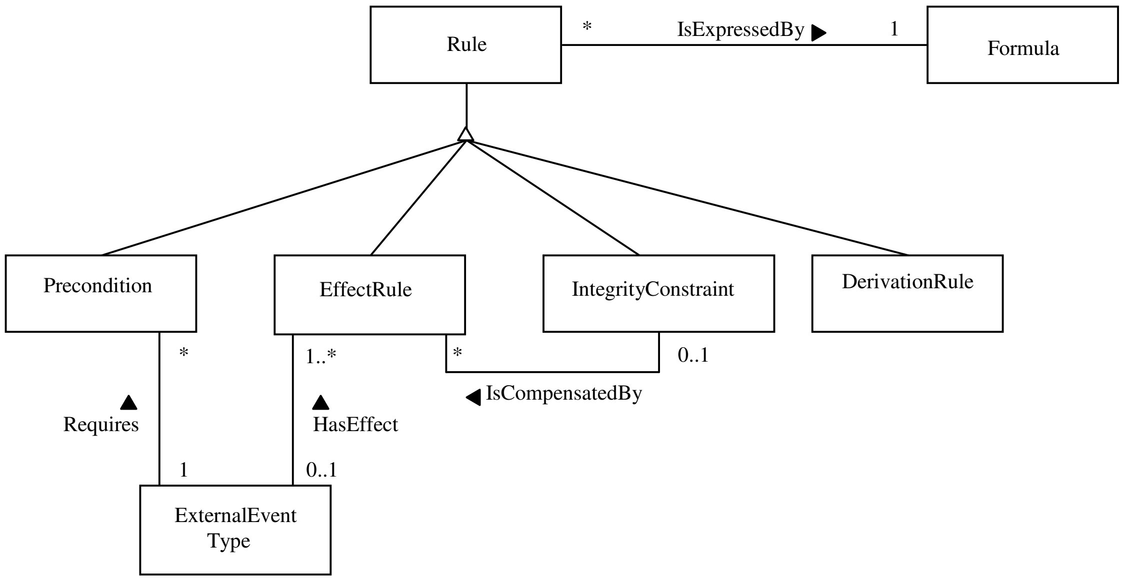

Figure 3. Minimal conceptual meta schema (Part 2 of 2)

MIC4. A participant is an entity type or an external event type:

Participant(p,t) → ∃e IsETPtOf(e,p,t) ∨ ∃ev IsEvTPtOf(ev,p,t)

MIC5. If a participant is an entity type, it cannot be an external event type:

IsETPtOf(e,p,t) → ¬∃ev IsEvTPtOf(ev,p,t)

MIC6. If a participant is an external event type, it cannot be an entity type. Formula as above.

MIC7. Only existing rules and formulas can be related in *IsExpressedBy*:

IsExpressedBy(r,f,t) → Rule(r,t) ∧ Formula(f,t)

with compensating rule:

M_Del_IsExpressedBy(r,f,t) ← M_Del_Rule(r,t) ∧ IsExpressedBy(r,f,t-1)

External event types. Meta external event types allow the evolution of schemas. The minimal meta schema includes meta event types that allow the insertion and deletion of the entity and relationship types shown in Figures 2 and 3. For example, there could be an external event type *NewIC*. Occurrences of *NewIC* would create an instance of *IntegrityConstraint* (Figure 3) and the associated instance of *IsExpressedBy*. We will discuss them in the next section.

3.2 Correspondence rules

We can now define the correspondence rules in our framework between the MIB and the schema:

CR1. There is a one-to-one correspondence between the instances at time *t* of *EntityType* in the MIB and the elements of E_t in the schema. If an instance of *EntityType* is also instance of *BaseConcept* (*DerivedConcept*) then it is base (resp., derived). If an instance of *EntityType* is also instance of *LexicalEntityType* (*Non-LexicalEntityType*) then it is lexical (resp., non-lexical).

CR2. There is a one-to-one correspondence between the instances at time *t* of *RelationshipType* in the MIB and the elements of R_t in the schema. If an instance of *RelationshipType* is also instance of *BaseConcept* (*DerivedConcept*) then it is base (resp., derived). If an instance *R* of *RelationshipType* is related (through *HasPt*) to *n* instances of *Participant*, then the degree of *R* in the schema is $n+1$.

CR3. There is a one-to-one correspondence between the set of *Formula* in which *IsExpressedBy* the instances of *DerivationRule* at time *t* in the MIB and the elements of DR_t in the schema.

CR4. There is a one-to-one correspondence between the set of *Formula* in which *IsExpressedBy* the instances of *IntegrityConstraint* at time *t* in the MIB and the elements of IC_t in the schema. If *Ic* is an instance of *IntegrityConstraint*, and *IsCompensatedBy(Ic,Eff,t)* and *IsExpressedBy(Eff,F,t)* then formula *F* is a compensating effect rule of *Ic* in the schema at *t*.

CR5. There is a one-to-one correspondence between the instances at time *t* of *ExternalEventType* in the MIB and the elements of EE_t in the schema. If an instance of *ExternalEventType* is also instance of *BaseConcept* (*DerivedConcept*) then it is base (resp., derived). If *Ev* is an instance of *ExternalEventType*, and *Requires(Ev,Pre,t)* and *IsExpressedBy(Pre,F,t)*, then formula *F* is a precondition of *Ev* in the schema at *t*. Similarly, if *Ev* is an instance of *ExternalEventType*, and *HasEffect(Ev,Eff,t)* and *IsExpressedBy(Eff,F,t)*, then formula *F* is an effect rule of *Ev* in the schema at *t*.

Correspondence rules similar to the above are present in a way or another in all frameworks with a reflective architecture. We emphasize the distinction between the MIP and the IP, make explicit the temporal correspondence, and leave undefined how the correspondence rules are implemented.

3.3 Conceptual modeling constructs

The above minimal meta schema is hardly satisfactory because it does not take into account the important number of special constructs that have been developed in the conceptual modeling field, and that are used in several conceptual models. These constructs are useful because they ease the definition of the schema and, at the same time, allow the development of efficient implementations. As an example, consider the cardinality constraints: it is much easier to simply define that the cardinality constraint of a participant in a relationship is 1..10 than to define the constraint in a general-purpose language.

The modeling constructs are particular types in the meta schema. A given schema may be defined using some of such constructs, and then the MIB will contain the corresponding entities as instances of those types. For example, the MIB could have the relationship *HasMinCard(Px,1,10)*, indicating that at time 10 the minimum cardinality of participant *Px* in its relationship is *one*.

However, there is not a direct correspondence between the "instantiation" of these constructs and the schema, but an indirect one. In our framework, the meta schema includes a set of derivation rules that "translate" the constructs into elements of the minimal meta schema. The above correspondence rules need not to be changed. In what follows we explain how to extend the minimal meta schema to incorporate the constructs in our framework. The general idea is to partition *IntegrityConstraint* and *DerivationRule* (Figure 3) into two subtypes: a base and a derived one (Figure 4). The base subtypes will correspond to integrity constraints or derivation rules defined explicitly by the designer, while the derived ones are defined by derivation rules of the meta schema. In what follows, we illustrate this idea with its application to a few albeit representative modeling constructs.

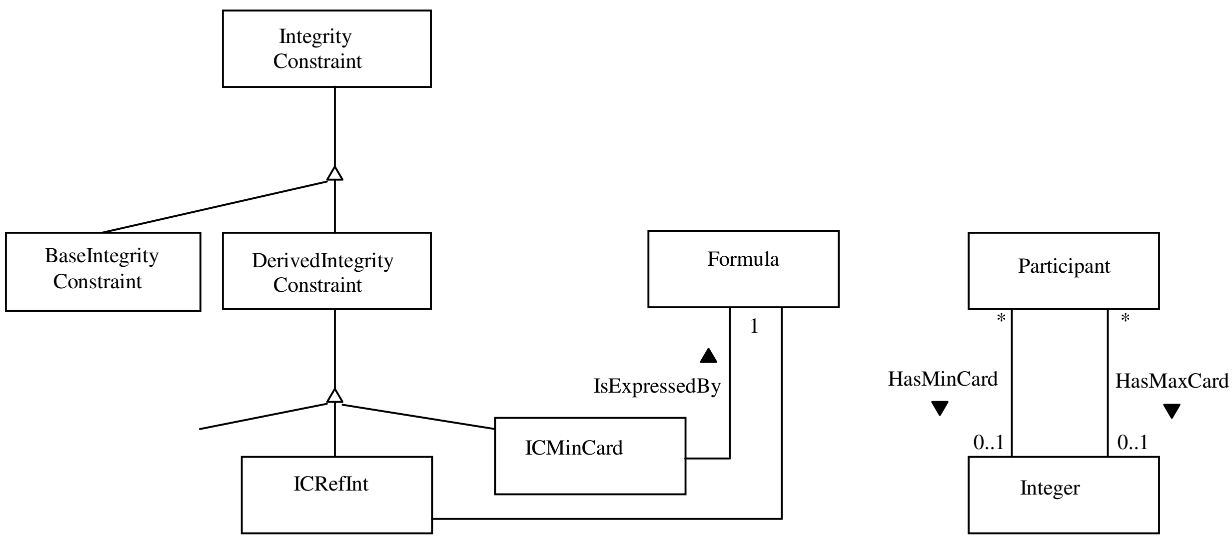

Figure 4. Meta schema conceptualization of cardinality and referential integrity constraints

3.3.1 Cardinality constraints

Figure 4 shows (again in the UML language) the conceptualization in the meta schema of the common cardinality constraints. The minimum (maximum) cardinality expresses the minimum (resp., maximum) number of times that an entity or event must (resp., can) participate in a relationship at a given time.

In the MIB, these cardinalities are expressed as instances of relationship types *HasMinCard* and *HasMaxCard*. We will focus only on the minimum cardinality; the maximum one is similar. Each minimum cardinality constraint is an instance of *ICMinCard*, which is a specialization of *DerivedIntegrityContraint*. A participant may not have minimum cardinality (i.e., zero) and then there is not a corresponding instance of *ICMinCard*. Entity type *ICMinCard* is derived. The corresponding derivation rule is:

MDR4. ICMinCard(s,t) ← HasMinCard(p,int,t) ∧ s = sym(ICMinCard,p)

where *sym(ICMinCard,p)* is a function that denotes a time-independent symbol (*s*), distinct from any other used in the MIB. The MIP could, for example, create this symbol when symbol *p* is created.

Now, we need to define the *Formula* in which the constraint *IsExpressedBy*. We refine the base relationship type between *Rule* and *Formula* to a derived one between *ICMinCard* and *Formula*. The derivation rule is:

MDR5. IsExpressedBy(s,f,t) ← ICMinCard(s,t) ∧ ICMinFormula(s,f,t)

In MDR5, *ICMinFormula(s,f,t)* is a predicate that gives in *f* the string representation of the corresponding formula. For a binary relationship type *R*, the general form of *f* for one of its participants (E_1) is:

$$\text{"}E_1\,(e_1,t) \rightarrow min \leq |\{e_2 \mid R(e_1,e_2,t)\}|\text{"}$$

where E_1 is given by *IsETPtOf* (or *IsEvTPtOf*), *min* by *HasMinCard* and *R* by *HasPt*. We omit here the details of predicate *ICMinFormula* because they are a lengthy sequence of trivial string manipulation operations.

Note that, by MDR4 and MDR5, the MIB will have a relationship *IsExpressedBy(s,f,t)* during the period in which *p* exists as participant and has a non-zero minimum cardinality. By CR4, during this period $f \in IC_t$, that is, *f* will be a schema integrity constraint, to be enforced by the IP.

3.3.2 Referential integrity constraints

A referential integrity constraint must be defined for each participant of a base relationship type. We assume that for derived relationship types the corresponding derivation rule is correct and, therefore, also are the types of the participants in its instances. To represent these constraints, we define in the meta schema (see Figure 4) a derived subtype of *DerivedIntegrityConstraint*, called *ICRefInt*, with the following derivation rule:

MDR6. ICRefInt(s,t) ← HasPt(r,p,t) ∧ BaseConcept(r,t) ∧ s=sym(ICRefInt,p)

where *sym* is again a function that gives us a time-independent symbol, that in this case denotes the referential integrity constraint related to a participant. We also refine the base relationship type *IsExpressedBy* between *Rule* and *Formula* to a derived one, in this case between *ICRefInt* and *Formula*, whose derivation rule is the following:

MDR7. IsExpressedBy(s,f,t) ← ICRefInt(s,t) ∧ ICRefIntFormula(s,f,t) where *ICRefIntFormula* is a predicate that gives in *f* the string representation of the corresponding formula[3]. For a binary relationship type *R*, the general form of *f* for one of its participants (E_1) is:

$$\text{“}R(e_1,e_2,t) \rightarrow E_1(e_1,t)\text{”}$$

where *R* is given by *HasPt*, and E_1 by *IsETPtOf* (or *IsEvTPtOf*).

3.3.3 Frequency

Cardinality and referential constraints are static. We now deal with a modeling construct corresponding to a dynamic constraint: frequency. A *Concept* has *single* frequency if all its instances can only be instance of the concept during one time interval. Otherwise, the frequency is *intermittent* [Oliv99]. For example, if persons can be instance of *Employee* during several distinct intervals, then *Employee* is intermittent. An example of single relationship type could be *Takes(Student, Course)* if after the period a student has been taking a course she cannot take it again. Note that, in fact, the intermittent frequency does not impose any constraint. We assume that external event types are always single.

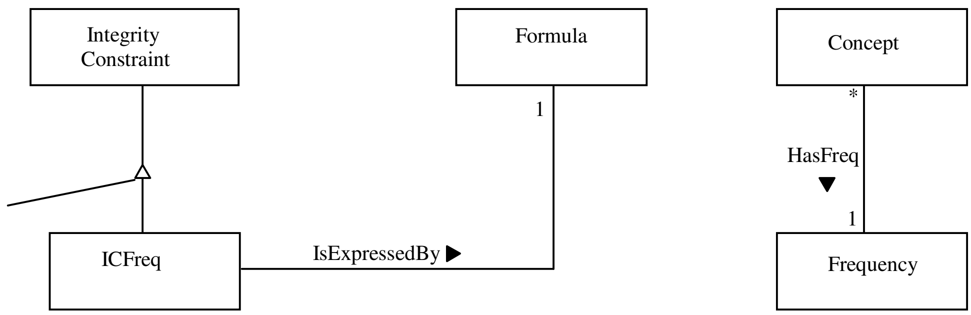

Figure 5. Meta schema conceptualization of frequency constraints

Figure 5 shows the conceptualization in the meta schema of this constraint. *Frequency* is a lexical entity type, with two instances: *Single* and *Intermittent*. All concepts are related to an instance of *Frequency* (through *HasFreq*). The integrity constraints are modeled as instances of *ICFreq*, an specialization of *DerivedIntegrityConstraint*. Entity type *ICFreq* is derived, with derivation rule:

MDR8.

$$ICFreq(s,t) \leftarrow (EntityType(c,t) \vee RelationshipType(c,t)) \wedge HasFreq(c,Single,t) \wedge s=sym(ICFreq,c)$$

By definition, an external event only holds for a time point and, therefore, external event types do not have an associated *ICFreq* constraint. The formula in which the *ICFreq* constraint *IsExpressedBy* is given by the derivation rule:

MDR9. $IsExpressedBy(s,f,t) \leftarrow ICFreq(s,t) \wedge ICFreqFormula(s,f,t)$

[3] This formula depends on the relationship type of the participant being *synchronous* or *asynchronous* (see [Oliv99] for more details), but for the sake of simplicity in this paper we are considering only synchronous relationship types.

where predicate *ICFreqFormula* gives in *f* the string representation of the formula. The general form of *f* for an entity type *E* is:

"E(e,t) ∧ E(e,t1) ∧ t1 < t → ¬∃t2(t1 < t2 < t ∧ ¬E(e,t2))"

Among the meta integrity constraints related to the frequency construct, we have:

MIC8. HasFreq(c,freq,t) → Concept(c,t) ∧ Frequency(freq,t)

with meta compensating rule:

M_Del_HasFreq(c,freq,t) ← M_Del_Concept(c,t) ∧ HasFreq(c,freq,t-1)

3.3.4 Partitions

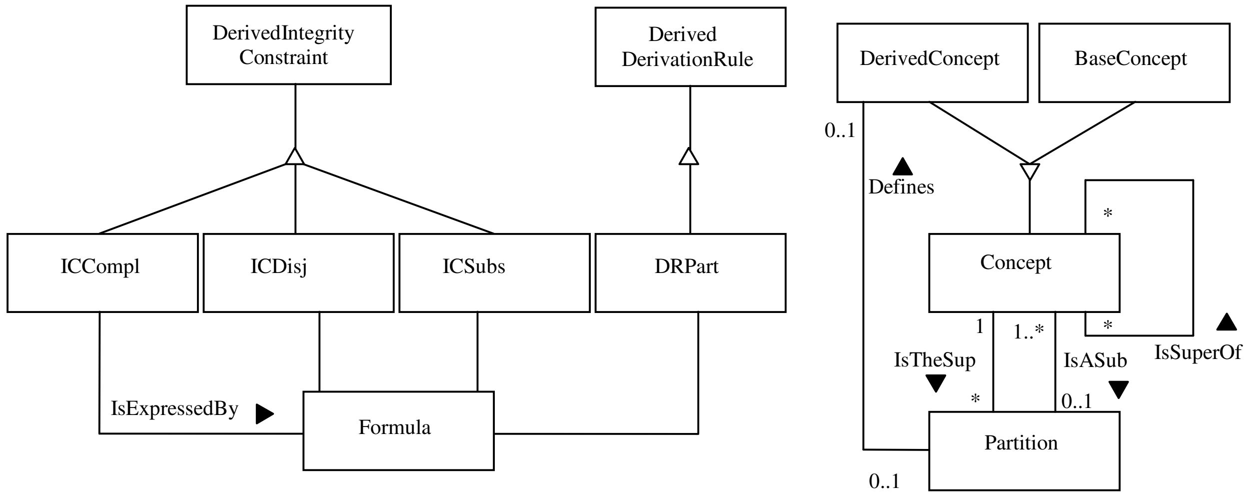

Figure 6. Meta schema conceptualization of partitions.

Partitions are an important modeling construct in many conceptual models [MaOd95, OlCS99]. All generalizations and specializations can be transformed into partitions. We are going to see that partitions are represented in the core elements of a schema by zero or one derivation rule, and zero or more integrity constraints. Figure 6 shows the conceptualization of partitions in the meta schema.
We allow the partition of any *Concept*. Thus, we deal also with partitions of relationship types [AnSC97]. A concept may be the superconcept (*IsTheSup*) of several partitions. A partition has only one superconcept, but it may have one or more subconcepts. We assume that a concept may be the subconcept (*IsASub*) of zero or only one partition. The concepts involved in a partition may be base or derived. Almost all combinations can be acceptable. One of the exceptions is when all concepts are base, because if all subconcepts are base, then the superconcept must be derived. For example, if P_1 is a partition with super *Person* and subs *Man* and *Woman*, and these are base, then *Person* must be derived, since it is the "union" of *Man* and *Woman*. In the meta schema, there is a relationship type (*Defines*) between *Partition* and *DerivedConcept*. An instance *Defines(P,C,T)*, where *C* must be a derived concept and the super of *P*, indicates that, at time *T*, concept *C* is derived as the union of the subconcepts of *P*. In the above example, we could have *Defines(P_1,Person,T)*. Entity type *Person* may be the super of another

partition P_2 (for example, into the base entity types *Married* and *Unmarried*) but it is defined only by one of them, because the other definition would be redundant.

Partitions require several meta integrity constraints. We indicate four of them here:

MIC9. If the super of a partition is an entity type, then all subs must be entity types:

$$\text{IsTheSup(sup,p,t)} \wedge \text{EntityType(sup,t)} \rightarrow \forall \text{sub (IsASub(sub,p,t)} \rightarrow \text{EntityType(sub,t))}$$

MIC10. All partitions have a superconcept.

$$\text{Partition(p,t)} \rightarrow \exists \text{sup IsTheSup(sup,p,t)}$$

MIC11. $\text{IsASub(c,p,t)} \rightarrow \text{Concept(c,t)} \wedge \text{Partition(p,t)}$
with compensating rules:

$$\text{M\_Del\_IsASub(c,p,t)} \leftarrow \text{M\_Del\_Concept(c,t)} \wedge \text{IsASub(c,p,t-1)}$$
$$\text{M\_Del\_IsASub(c,p,t)} \leftarrow \text{M\_Del\_Partition(c,t)} \wedge \text{IsASub(c,p,t-1)}$$

MIC12. In a partition of relationship types, the participants of a partition's subtype must be also subtypes of the participants of the partition's supertype (or the same). We omit the formula here, but mention that it involves relationship type *IsSuperOf* between concepts (Figure 6). This is a derived relationship type that will be used also in the next section. Its derivation rule is:

MDR10.

$$\text{IsSuperOf(e,e',t)} \leftarrow \text{IsTheSup(e,part,t)} \wedge \text{(IsASub(e',part,t)} \vee \text{(IsASub(e'',part,t)} \wedge \text{IsSuperOf(e'',e',t)))}$$

In the schema, we define a disjointness integrity constraint for each concept that is a sub of a partition[4]. In the above example, a *Man* cannot be a *Woman*, and the inverse. These integrity constraints are modeled as instances of *ICDisj*. Entity type *ICDisj* is derived, with derivation rule:

MDR11. $\text{ICDisj(s,t)} \leftarrow \text{IsASub(c,p,t)} \wedge \text{s = sym(ICDisj,c)} \wedge \text{IsASub(c',p,t)} \wedge \text{c} \neq \text{c'}$

The *Formula* in which *ICDisj IsExpressedBy* is given by the derivation rule:

MDR12. $\text{IsExpressedBy(s,f,t)} \leftarrow \text{ICDisj(s,t)} \wedge \text{ICDisjFormula(s,f,t)}$

where predicate *ICDisjFormula* gives in *f* the string representation of the formula. As an example, the general form of *f* for a base relationship type R_1 in a partition of a binary relationship type R into $R_1, \ldots, R_n$ is:

$$\text{"}R_1\text{(e1,e2,t)} \rightarrow \neg\, (R_2\text{(e1,e2,t)} \vee \ldots \vee R_n\text{(e1,e2,t))"}$$

In the schema, we define also a subset integrity constraint for each subconcept, except when the partition defines a derived superconcept. For example, if *Person* is a derived concept, defined by the partition into *Man* and *Woman* and also partitioned into base concepts *Married* and *Unmarried*, then we must ensure that all instances of *Married* or

[4] For flexibility, we allow a partition with only one subtype (Figure 6). In this particular case, it is obvious that disjointness constraints would not apply.

Unmarried are also instances of *Person*. These integrity constraints are modeled as instances of *ICSubs*. Entity type *ICSubs* is derived, with derivation rule:

MDR13. ICSubs(s,t) ← IsASub(c,p,t) ∧ IsTheSup(sup,p,t) ∧ ¬ Defines(p,sup,t) ∧ s = sym(ICSubs,c)

The *Formula* in which *ICSubs IsExpressedBy* is given by the derivation rule:

MDR14. IsExpressedBy(s,f,t) ← ICSubs(s,t) ∧ ICSubsFormula(s,f,t)

where predicate *ICSubsFormula* gives in *f* the string representation of the formula. The general form of *f* for a base entity type E_1 in a partition of a derived entity type *E* into $E_1,..., E_n$ would be:

"$E_1(e,t) \rightarrow E(e,t)$"

Finally, we also include in the schema a completeness integrity constraint for each partition, except when the partition defines a derived superconcept. In the above example, we must ensure that all instances of *Person* are also instance of *Married* or *Unmarried*. These integrity constraints are modeled as instances of *ICCompl*. Entity type *ICCompl* is derived, with the derivation rule:

MDR15. ICCompl(s,t) ← IsTheSup(c,p,t) ∧ ¬ Defines(p,c,t) ∧ s = sym(ICCompl,p)

The *Formula* in which *ICCompl IsExpressedBy* is given by the derivation rule:

MDR16. IsExpressedBy(s,f,t) ← ICCompl(s,t) ∧ ICComplFormula(s,f,t)

where predicate *ICComplFormula* gives in *f* the string representation of the formula. The general form of *f* for a base entity type *E* partitioned into $E_1,..., E_n$ would be:

"$E(e,t) \rightarrow (E_1(e,t) \vee ... \vee E_n(e,t))$"

Now, we describe the schema derivation rules implied by partitions. If a derived concept is defined by a partition, then the schema must include its derivation rule. We model it as an instance of *DRPart*, which is a derived entity type subtype of *DerivationRule* (Figure 6). Its derivation rule is:

MDR17. DRPart(s,t) ← Defines(p,c,t) ∧ s = sym(DRPart,c)

Note again the use of *sym(DRPart,c)*, which denotes a symbol we use to refer to the derivation rule. We refine the base relationship type *IsExpressedBy* between *Rule* and *Formula* to a derived one between *DRPart* and *Formula*. The derivation rule is:

MDR18. IsExpressedBy(s,f,t) ← DRPart(s,t) ∧ DRPartFormula(s,f,t)

where predicate *DRPartFormula* gives the string representation of *f*. The general form of *f* in a partition of a derived entity type *E* into $E_1,..., E_n$ is:

"$E(e,t) \leftarrow (E_1(e,t) \vee ... \vee E_n(e,t))$"

4 SCHEMA EVOLUTION

In our framework, schema evolution is performed by meta external events. When the designer wants to change a schema, she generates an instance of the appropriate meta external event type. The corresponding effect rules will induce meta structural events, which change the MIB. By the correspondence rules (Section 3.2), such changes imply changes to the schema.

The set of external event types included in the meta schema define the possible schema evolutions, either applied individually or by means of their composition. There is no other way to evolve a schema. Given that usually a schema may change in many different ways, it is necessary to include many external event types in the meta schema. For example, [AlLe98] defines 38 possible F2 schema changes. More are needed for richer schemas. We cannot show here the complete set of needed external event types. Rather, we are going to show how external event types and their effects can be defined in our framework. Other external event types would be defined similarly.

Some meta external event types change only the schema, without affecting the IB. For example, *M_RemoveIntegrityConstraint* could have only the effect rule:

M_Del_BaseIntegrityConstraint(ic,t) ← M_RemoveIntegrityConstraint(rm,t) ∧ RemovesIC(rm,ic,t)

which, by CR4, would remove *ic* from the set of schema integrity constraints IC_t. Note that, in this case, the relationship *IsExpressedBy* between *ic* and its formula at *t* would be deleted by the compensating rule of MIC7 (Section 3.1).

Other meta external event types require changing the schema and the IB. We can distinguish two cases here: when the required changes to the IB are given by the typing rule (Section 2.1), and when they must be defined by explicit effect rules. An example of the former could be *M_RemoveRelType*, with appropriate preconditions and the effect rule:

M_Del_RelationshipType(r,t) ← M_RemoveRelType(rm,t) ∧ RemovesRT(rm,r,t)

which, by CR2, would remove *r* from the set of the schema relationship types R_t and, by the typing rule, would delete existing instances of *r* at *t* in the IB. We will show a more complete example of this case in Section 4.1 below.

In other cases, the changes implied by the typing rule are not enough. We need to define an appropriate external event type in the schema, generate an occurrence of it and notify the IP of this occurrence. In principle, we have two options here: either to define subtypes of *ExternalEventType*, similarly to what we have done in the previous section for integrity constraints, or to generate directly an instance of *ExternalEventType*. We prefer this latter option, because the former requires including in the meta schema a large number of subtypes. Thus, we will generate "on the fly" an instance *EvT* of *ExternalEventType*, without parameters, and then we notify the IP of an occurrence of *EvT*. We will use in the meta effect rules the notation:

EvT ←

to mean that the MIP notifies the IP of an occurrence of *EvT*. We will show a detailed example of this case in Section 4.2 below.

4.1 M_RemoveEntityType

We start with a rather simple external event type, *M_RemoveEntityType*. It has only one parameter, given by relationship type *Removes* between that type and *Non-LexicalEntityType*. A particular instance of *M_RemoveEntityType*, occurring at time *T* and removing *E*, represents the design decision that, from time *T*, *E* ceases to be instance of *EntityType* and, therefore, the IB must not record the instances of entity type *E*. However, *E* may be again instance of *EntityType* later. *E* may be base or derived. For illustration purposes, we only deal here with the simple case when *E* does not participate in any relationship type and is not involved in any partition.

Preconditions

We define first the preconditions. One of them is that *E* is a non-lexical entity type at time *T-1*. Formally,

M_RemoveEntityType(rm,t) ∧ Removes(rm,e,t) → Non-LexicalEntityType(e,t-1)

The if-part of the rule serves only to declare that an instance of the event type has occurred and to give name to its parameters. We omit this part in the next rules. Then, the other preconditions are:

¬∃participant IsETPtOf(e,participant,t-1)
¬∃partition IsTheSup(e,partition,t-1)
¬∃partition IsASub(e,partition,t-1)

Effect rules

The first effect rule induces the meta structural event that deletes *E* from *Non-LexicalEntityType*:

M_Del_Non-LexicalEntityType(e,t) ← M_RemoveEntityType(rm,t) ∧ Removes(rm,e,t)

As we did before, we omit the if-part of this rule in the next ones. Now, we must remove *E* from *BaseConcept* or from *DerivedConcept*:

M_Del_BaseConcept(e,t) ← BaseConcept(e,t-1)
M_Del_DerivedConcept(e,t) ← DerivedConcept(e,t-1)

Finally, if *E* is derived, we must delete its derivation rule:

M_Del_DerivationRule(s,t) ← DerivedConcept(e,t-1) ∧ s = sym(DerivationRule,e)

Induced effects on the MIB

The above effect rules are the only ones that need to be defined. We now reason (informally) about induced effects on the MIB:

- By MDR1 and MDR2 (Section 3.1), *e* ceases to be instance of *Concept* and *EntityType*: MDR1 and MDR2 induce the derived structural events *M_Del_Concept(e,t)* and *M_Del_EntityType(e,t)*.
- By the compensating effect rule of MIC8 (Section 3.3.3), the event *M_Del_HasFreq(e,freq,t)* is induced if *HasFreq(e,freq,t-1)* holds.
- By MDR8 and MDR9 there is not an *ICFreq* corresponding to *e* at *t*.
- *M_Del_DerivationRule* induces *M_Del_Rule*. By the compensating rule of MIC7, *M_Del_IsExpressedBy(s,f,t)* is induced if *IsExpressedBy(s,f,t-1)*.

Induced effects on the schema and IB

We now reason about the induced effects on the schema and the IB:

- By CR1, *e* is not an entity type in the schema at *t* (that is, $e \notin E_t$)
- By the typing rule, all instances of *e* at *t-1* cease to be instances of *e* at *t*.
- By CR3, all instances of *DerivationRule* that have been deleted at *t* cease to exist in the schema at *t*.
- By CR4, all instances of *IntegrityConstraint* that have been deleted at *t* cease to exist in the schema at *t*.

Remarks

The removal of an entity type, or any other schema change, can affect indirectly to integrity constraints, derivation rules, or external event types defined explicitly in the schema by the designer. Some changes in those elements can be needed. The analysis of those changes results in a very complex task [Lemk94, Zica92], that is outside the scope of this paper.

4.2 M_ChangeParticipantToSET.

We now deal with a complex external event type, *M_ChangeParticipantToSET*. It has two parameters, given by relationship types *Changes*, with *Participant*, and *HasNewType*, with *EntityType*. An occurrence *M_ChangeParticipantToSET(CP,T)* of this event type, with *Changes(CP,P,T)* and *HasNewType(CP,NT,T)* represents the fact that, from time *T*, the participant *P* of a relationship type *R* changes its type to a supertype or a subtype[5], the entity type *NT*. This means that, from *T*, instances of *R* are only valid if their corresponding participant is instance of *NT*.

We consider the general case in which the relationship type has any degree, and is base or derived, subtype in zero or one partition, and super in any number of partitions. Changing a participant of a binary relationship type (attribute) has been studied extensively in the literature. However, we have not found the study of the change in our

[5] *ToSET* stands for *"**To** a **S**uper or a **S**ub **E**ntity **T**ype"*

general settings.

If the relationship type is base, some of its instances may become invalid after the change. We have two possible strategies here: either to reject the change or to delete explicitly the invalid instances. The two possibilities must be offered to the designer. We could define a different external event type for each strategy, or add a new parameter to the event type, indicating the desired strategy [Zica92]. In the following, we assume the existence of two different external event types, and present here the one that performs the strategy of deleting explicitly the invalid instances. In that case, if the relationship type is involved in a partition, deletion of some of its instances must be propagated through the partition hierarchy. For base types, the propagation must be done explicitly. For derived types, the associated derivation rules might have to be corrected in the right way.

On the other hand, if the relationship type changed is derived, the corresponding derivation rule needs to be changed too, so the invalid instances will be deleted implicitly.

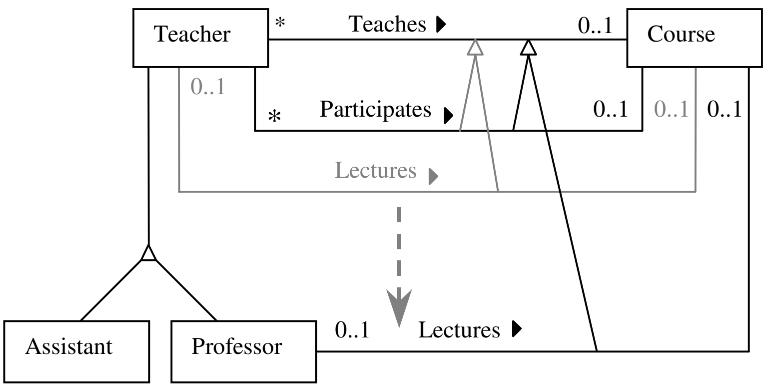

Figure 7. An example of changing a participant in a relationship type.

4.2.1 An introductory example

This is a more complex change than the one exposed in Section 4.1, so we will begin the explanation with an example. Assume a university information system (see Figure 7) with two base entity types, *Teacher* and *Course*, and also a base relationship type *Teaches* between *Teacher* and *Course*, partitioned into two subtypes, *Lectures* (base) and *Participates* (derived), with the same participants as the supertype. A teacher participates in a course if she teaches but does not lecture it. Any teacher can lecture courses or participate in them. A teacher can teach in at most one course. A course can have at most one lecturer, but any number of participants.

Assume now that the domain changes, and *Teacher* is transformed into a derived type, partitioned into *Professor* and *Assistant*. Now, among other changes, only a professor can be the lecturer of a course. That implies changing the participant of the relationship type *Lectures* from *Teacher* to *Professor*. The effects of this change, in our framework, would be the following:

The type of the participant must be explicitly changed in the MIB:

M_Del_IsETPtOf (Teacher, P, T)
M_Ins_IsETPtOf (Professor, P, T)

with P being the symbol representing the participant being changed, and T the time instant when the change is done.

As *Lectures* is base, P has a referential integrity constraint that must change. This change is performed automatically by MDR7, which induces:

M_Del_IsExpressedBy(S,'*Lectures(tch,c,t)* $\rightarrow$ *Teacher(tch,t)*', T)
M_Ins_IsExpressedBy(S,'*Lectures(tch,c,t)* $\rightarrow$ *Professor(tch,t)*', T)

with S=*sym(ICRefInt,P)*.

Some of the instances of *Lectures* may violate the above new integrity constraint. In that case, they must cease to exist. As *Lectures* is base, we must define a new external event type in the conceptual schema to explicitly eliminate its invalid instances; and also to eliminate them as instances of *Teaches*, since it is a base supertype of *Lectures*. We use the special function *newSymbol()* to obtain new symbols in the MIB, when needed. The explicitly defined effects in the MIB must be the following:

et=newSymbol()
M_Ins_ExternalEventType(et,T)

er_1=newSymbol()
M_Ins_EffectRule(er_1,T)
M_Ins_HasEffect(et,er_1,T)
M_Ins_IsExpressedBy(er_1,'*Del_Lectures(tch,c,t)*$\leftarrow$*Lectures(tch,c,t-1)*$\wedge$ $\neg$*Professor(tch,t-1)*',T)

er_2=newSymbol()
M_Ins_EffectRule(er_2,T)
M_Ins_HasEffect(et,er_2,T)
M_Ins_IsExpressedBy(er_2,'*Del_Teaches(tch,c,t)* $\leftarrow$ *Del_Lectures(tch,c,t)*',T)

Finally, we have to notify the IP of an occurrence of the external event type we have just defined:

et $\leftarrow$

4.2.2 General case

Once presented the example, we now formalize the preconditions and effect rules of the event type. We omit the literals *M_ChangeParticipantToSET(cp,t)*, *Changes(cp,p,t)* and *HasNewType(cp,nt,t)* in the if-part of the rules.

Preconditions

EntityType(nt,t-1)
Participant(p,t-1)
IsETPtOf(oldtype,p,t-1) $\rightarrow$ nt $\neq$ oldtype $\wedge$ (IsSuperOf(oldtype,nt,t-1) $\vee$ IsSuperOf(nt,oldtype,t-1))

Effect Rules

We must change explicitly the type of the corresponding participant:

M_Del_IsETPtOf(oldtype,p,t) ← IsETPtOf(oldtype,p,t-1)
M_Ins_IsETPtOf(nt,p,t)

If the relationship type is base and we are changing the participant to a subtype (as in the example of Figure 7), we must define an event type for eliminating possible invalid instances of the relationship type in the IB:

et=newSymbol()
M_Ins_ExternalEventType(et,t) ←
HasPt(r,p,t-1) ∧ BaseConcept(r,t-1) ∧ IsETPtOf(oldtype,p,t-1) ∧ IsSuperOf(oldtype,nt,t-1)

For the event type just created, we need an effect rule to explicitly delete the invalid instances of the base relationship type:

M_Ins_EffectRule(er,t) ∧ M_Ins_HasEffect(et,er,t) ∧ M_Ins_IsExpressedBy(er,f,t) ←
M_Ins_ExternalEventType(et,t) ∧ er=newSymbol() ∧ DelInvalidRelsFormula(p,nt,f,t)

where predicate *DelInvalidRelsFormula* gives in *f* the string representation of the formula describing the effect rule. An example of this kind of formula for a relationship type *R* that changes a participant to a new entity type E_i could be the following:

$$"Del\_R(e_1,\ldots,e_i,\ldots,e_n,t) \leftarrow R(e_1,\ldots,e_i,\ldots,e_n,t\text{-}1) \wedge \neg E_i(e_i,t\text{-}1)"$$

We also need an effect rule for each base type in the hierarchy of partitions of the relationship type changed, to propagate the possible deletion of any of its instances:

M_Ins_EffectRule(er,t) ∧ M_Ins_HasEffect(et,er,t) ∧ M_Ins_IsExpressedBy(er,f,t) ←
M_Ins_ExternalEventType(et,t) ∧ HasPt(r,p,t-1) ∧ er=newSymbol() ∧
(IsSuperOf(r,r',t-1) ∨ IsSuperOf(r',r,t-1) ∨ (IsSuperOf(r'',r',t-1) ∧ IsSuperOf(r'',r,t-1) ∧ r' ≠ r)) ∧
BaseConcept(r',t-1) ∧ PropagateInvalidRelsFormula(r,r',f,t)

where predicate *PropagateInvalidRelsFormula* gives in *f* the string representation of the formula describing the effect rule. An example of this kind of formula, for a relationship R that has a base sub/supertype R', could be the following:

$$"Del\_R'(e_1,\ldots,e_n,t) \leftarrow Del\_R(e_1,\ldots,e_n,t) \wedge R'(e_1,\ldots,e_n,t\text{-}1)"$$

Finally, if the external event type has been created, we must notify the IP of it:

et ← M_Ins_ExternalEventType(et,t)

Induced effects on the MIB

– The *Formula* in which *IsExpressedBy* the *ICRefInt* corresponding to the relationship type (if base), changes according to the new type, by MDR7.

Induced effects on the schema and IB

– By CR4, if the referential integrity constraint has changed its formula in the MIB, it is also changed in the schema.

– By CR5, if a new instance of *ExternalEventType* is created in the MIB, a new external event type is defined in the schema, with the corresponding effect rules defined in the MIB as instances of *EffectRules*. At the same time, the IP is notified of an occurrence of that new event type, and it must perform the changes defined by the effect rules.

– By the effect rule defined by *DelInvalidRelsFormula*, if the relationship type changed is base, its invalid instances are deleted. By the effect rules defined by *PropagateInvalidRelsFormula*, the deletion is propagated to other relations.

Remarks

1. As pointed out at the beginning of this section, if one or more derived relationship types are involved in the change (for instance, the relationship whose participant is being changed), it can also be necessary to change their corresponding derivation rules. But the change of a derivation rule is a meaningful change type by itself, and it must be modeled as an independent external event type; as a consequence, in such cases we will need to compose an occurrence of *ChangeParticipantToSET* and one or more occurrences of the external event type for changing derivation rules. Due to space limitations we are forced to omit in this paper the description of such event type. We think anyway that the example shown here is complete enough to understand the use of our framework.

2. Changing a participant in a relationship type may cause problems with cardinality constraints. We can see it in the previous example (Figure 7): if we assume now that a course must have one (and only one) lecturer (instead of 0..1), changing the participant from *Teacher* to *Professor* may cause a violation of that constraint. Some courses can result without an associated lecturer if some instances of *Lectures* must disappear because of the participant *Teacher* not being a *Professor.* In that case, the change of participant would be rejected.

   This is a case where a change in the requirements of the domain does not temporally match the reality: there is an exceptional situation in which some instances do not respect the imposed new constraints. We must wait until the domain adapts to the new requirements to impose them. In the example, we must relax the original cardinality constraint (so that a *Course* can have or not an assigned lecturer) to make the change of participant possible. Once changed the participant and assigned new *Professors* (as lecturers) to the unassigned *Courses*, it will be possible to restore the original cardinality.

5 CONCLUSIONS

We have presented a framework that allows the definition of the evolution of information systems, at the conceptual level. The framework is based on a reflective architecture, with two loosely coupled conceptual processors. The first is the application-level processor, which behaves according to the rules specified by the conceptual schema. In our framework, this schema is temporal. The second processor, called the meta information processor, defines the schema used by the former. It behaves according to the rules specified by a meta schema, which is also temporal. The meta external events correspond to evolution operations of the schema.

Our framework allows the definition of all conceptual modeling constructs in the meta schema, either those developed in the conceptual modeling field or new ones considered useful in specific projects. This has been illustrated with four representative constructs. On the other hand, our framework allows the definition of any evolution operation, and this has been illustrated also by two particular operations, which have been analyzed in detail.

The evolution of information systems is a very important topic that still lacks a general and satisfactory solution. Fortunately, there is an impressive amount of previous and on-going research in this field (specially related to databases). This paper aims at contributing to the field by analyzing the evolution of information systems (and not only databases), defining the evolution of conceptual schemas (and not only logical database schemas) and considering the temporal dimension (thus extending the usual non-temporal view). On the other hand, our framework is general, can be adapted to many (if not all) temporal conceptual models and allows several practical implementations.

Our work can be extended in at least five directions. First, we could define, in the meta schema, other modeling constructs (aggregation, materialization, etc.). Second, we could develop the complete (both at schema and instance-level [Casa90]) set of meta external event types, for the minimal meta schema and for the considered constructs. Other complex external event types could be added to that set to facilitate the evolution to the designer [Brèc96]. Of course, this should be done on the basis of existing work. Third, a prototype implementation could be developed, preferably using two processors of different type. The work on temporal databases is relevant here [TCG+93]. In view of practical applications, the two processors should be loosely coupled, as suggested by our framework. Fourth, that prototype could include a tool to assist and guide the designer in the definition and analysis of evolution operations [Zica92, EwOr93]. For instance, warning her about external event types possibly affected by a particular schema change. Finally, building on top of existing knowledge and experience, one could attempt the development of an industrial system, with several possible degrees of ambition.

ACKNOWLEDGMENTS

The authors are indebted to Dolors Costal, Cristina Gómez and Camilo Ocampo for their helpful comments. This work has been partially supported by CICYT program projects TIC99-1048-C02-1 and TEL99-0335-C04-02.

REFERENCES

[AlLe98] Al-Jadir, L.; Léonard, M. "Multiobjects to Ease Schema Evolution in an OODBMS", Proc. ER'98, Singapore, LNCS 1507, Springer, pp. 316-333.

[AnSC97] Analyti, A.; Spyratos, N.; Constantopoulos, P. "Property Covering: A Powerful Construct for Schema Derivations", Proc. ER'97, LNCS 1331, Springer-Verlag, pp. 271-284.

[Aria91] Ariav, G. "Temporally oriented data definitions: Managing schema evolution in temporally oriented databases", Data & Knowledge Eng., 6, pp. 451-467.

[BBJW97] Boman, M.; Bubenko jr. J.A.; Johannesson, P.; Wangler, B. "Conceptual Modelling", Prentice Hall, 269 p.

[BCG+87] Banerjee, J.; Chou, H-T.; Garza, J.F.; Kim, W.; Woelk, D.; Ballou, N. "Data Model Issues for Object-Oriented Applications". ACM TOIS Vol. 5, No. 1, January, pp. 3-26.

[BeFG97] Bertino, E.; Ferrari, E.; Guerrini, G. "T_Chimera: A Temporal Object-Oriented Data Model", TAPOS 3(2), pp.103-125.

[Brèc96] Brèche, P. "Advanced Primitives for Changing Schemas of Object Databases". 8[Th]. Intl. Conf. CAiSE'96, LNCS 1080, Springer, pp. 476-495.

[Bube77] Bubenko, J.A. jr. "The Temporal Dimension in Information Modelling", In "Architecture and Models in Data Base Management Systems", North-Holland.

[Casa90] Casais, E. "Managing Class Evolution in Object-Oriented Systems", In Object Management, D.C. Tsichritzis (ed.), pp. 133-195.

[EwOr93] Erwald, C.A.; Orlowska, M.E. "A procedural approach to schema evolution", Proc. CAiSE'93, LNCS 685, Springer, pp. 22-38.

[GSÖP98] Goralwalla, I.; Szafron, D.; Özsu, T.; Peters, R. "A Temporal Approach to Managing Schema Evolution in Object Database Systems". Data&Knowledge Eng. 28(1), October, pp. 73-105.

[HEH+94] Hainaut, J-L.; Englebert, V.; Henrard, J.; Hick, J-M.; Roland, D. "Database Evolution: the DB-MAIN Approach". 13[th]. Intl. Conf. on the Entity-Relationship Approach - ER'94, LNCS 881, Springer-Verlag, pp. 112-131.

[ISO82] ISO/TC97/SC5/WG3. "Concepts and Terminology for the Conceptual Schema and Information Base", J.J. van Griethuysen (ed.), March.

[JGJ+95] Jarke, M.; Gallersdörfer, R.; Jeusfeld, M.A.; Staudt, M.; Eherer, S. "ConceptBase - a deductive object base for meta data management". Journal of Intelligent Information Systems, 4(2), pp. 167-192.

[JJN+98] Jeusfeld, M.A.; Jarke, M.; Nissen, H.W.; Staudt, M. "ConceptBase - Managing Conceptual Models about Information Systems". In P. Bernus, K. Mertins, G. Schmidt (eds.): Handbook on Architectures of Information Systems, Springer-Verlag, pp. 265-285.

[Laut99] Lautemann, S.-E. http://www.dbis.informatik.uni-frankfurt.de/~lauteman/research/prefe-rences.ps.gz

[LeMa95] Lemke, T.; Manthey, R. "The Schema Evolution Assistant: Tool Description", IDEA.DE.22.O.004, University of Bonn.

[Lemk94] Lemke, T. "Schema Evolution in OODBMS: A Selective Overview of Problems and Solutions", IDEA.WP.22.O.002, University of Bonn.

[Lemk95] Lemke, T. "DDL = DML?. An Exercise in Reflective Schema Management for Chimera", IDEA.WP.22.O.003, University of Bonn.

[Lien83] Lientz, B. "Issues in Software Maintenance", ACM Computing Surveys, 15(3), September, pp. 271-278.

[MBJK90] Mylopoulos, J.; Borgida, A.; Jarke, M.; Koubarakis, M. "Telos: Representing knowledge about information systems", ACM TOIS, 8(4), pp. 325-362.

[MaOd95] Martin, J.; Odell, J. "Object-Oriented Methods: A Foundation", Prentice Hall.

[Mant93] Manthey, R. "Beyond Data Dictionaries: Towards a Reflective Architecture of Intelligent Database Systems", DOOD'93, Springer-Verlag, pp. 328-339.

[NgRi89] Nguyen, G.T.; Rieu, D. "Schema evolution in object-oriented database systems", Data&Knowledge Eng. 4, pp. 43-67.

[OlCS99] Olivé, A.; Costal, D.; Sancho, M-R. "Entity Evolution in IsA Hierarchies", Proc. ER'99, LNCS 1728, Springer, pp. 62-80.

[Oliv89] Olivé, A. "On the design and implementation of information systems from deductive conceptual models", VLDB'89, Amsterdam, pp. 3-11.

[Oliv99] Olivé, A. "Relationship Reification: A Temporal View", Proc. CAiSE'99, LNCS 1626, Springer, pp. 396-410.

[PeÖz93] Peters, R.J.; Özsu, T. "Reflection in a Uniform Behavioral Object Model". Proc. ER'93, Arlington, LNCS 823, Springer-Verlag, pp. 34-45.

[PeÖz97] Peters, R.J.; Özsu, T. "An Axiomatic Model of Dynamic Schema Evolution in Objectbase Systems". ACM TODS Vol. 22, No. 1, March, pp. 75-114.

[Rodd94] Roddick, J.F. "Schema Evolution in DataBase Systems – An Updated Bibliography", ACM SIGMOD Rec., 21(4), May, pp. 35-40.

[Rodd95] Roddick, J.F. "A Survey of Schema Versioning Issues for Database Systems", Inf. Softw. Technol, 37(7), pp. 383-393.

[RuJB99] Rumbaugh, J.; Jacobson, I.; Booch, G. "The Unified Modeling Language Reference Manual", Addison-Wesley.

[Sjøb93] Sjøberg, D. "Quantifying Schema Evolution", Information and Software Technology, Vol. 35, No. 1, January, pp. 35-44.

[SoGo79] Sockut, G.H.; Goldberg, R.P. "Database Reorganization – Principles and Practice", ACM Computing Surveys, 11(4), December, pp. 371-395.

[TCG+93] Tansel,A.;Clifford,J.; Gadia,S. et al. Temporal Databases: Theory, Design and Implementation. Benjamin/Cummings.
[ThLo91] Theodoulidis, C.; Loucopoulos, P. "The Time Dimension in Conceptual Modelling", Information Systems, 16(3), pp. 273-300.
[TrSc92] Tresch, M.; Scholl, M.H. "Meta Object Management and its Application to Database Evolution", 11th. Intl. Conf. on the Entity-Relationship Approach - ER'92, LNCS 645, Springer-Verlag, pp. 299-321.
[VDD95] Van Belleghem, K.; Denecker, M.; De Schreye, D. "Combining situation calculus and event calculus", Procs. ICLP, MIT Press, pp. 83-97.
[Zica92] Zicari, R. "A Framework for Schema Updates in Object-Oriented Database System", in Bancilhon,F.; Delobel,C.; Kanellakis, P. (ed.) "Building an Object-Oriented Database System - The Story of O_2", Morgan Kaufmann Pub., pp. 146-182.