Evolving Temporal Conceptual Schemas: The Reification Case
(extended version)

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

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

Evolving Temporal Conceptual Schemas: The Reification Case
(extended version)

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

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

Abstract

In this report we study temporal conceptual schema evolutions related to reification, a typical and complex modeling construct. Various types of reification are considered. Using a previously defined framework, we specify, only at conceptual level (and without descending to logical nor application levels), the effects of any possible evolution related to reification, thus reducing the complexity of the management of those changes.

TABLE OF CONTENTS

•

1 Introduction

This report deals with the evolution of temporal conceptual schemas of information systems. Specifically, we are interested in the reification case. Reification (also known as *associative entity*, *association type*, *association class* or *relationship-to-entity*) is a construct that is commonly used in the conceptualization of information systems. This construct can be applied in the first conceptualization of an information system or as a consequence of an evolution activity.

Ideally, evolution of information systems should be managed at conceptual schema level, and automatically propagated down to the logical schema(s) and application programs [HEH+94]. Unfortunately, existing technology is far from allowing this ideal situation. Few research efforts have been done in the (temporal) conceptual schemas evolution topic [TrSc92, LeMa95, JGJ+95, GSÖP98] (although database schema evolution is a widely investigated field [BCG+87, Zica92, Rodd94]), and no adequate tools have been provided to information systems designers. Information systems evolution must be managed not only at conceptual level, but also as changes at design and application levels, thus possibly incrementing their intrinsic complexity and complicating the evolution task. These evolution activities consume a large amount of resources, and this justifies the interest of improving the way information systems evolve.

Our work pretends to contribute partially to this improvement, focusing our efforts in the reification construct. Reification is a powerful modeling element widely used and present in most of the modeling languages. We think that its complexity gives good reason for any possible improvement in its management. In this report, we look for a way of completely specifying, only at conceptual level, any possible evolution related to reification. To achieve this, we apply here a framework that has been defined in [LoOl00]. This framework supports schema evolutions related to some common conceptual modeling elements (entity and relationship types, cardinality constraints, derivation rules, partitions...). We extend it to support also reification in a temporal context [Oliv99], and not only from the usual static point of view. We will see later that the temporal dimension originates different possible alternatives of reification.

The rest of this report is organized as follows: We review the framework and its main features in Section 2, and reification definition and its types in Section 3. Our contribution is presented in Section 4, which deals with reification schema evolutions. Finally, the conclusions of the report are presented in Section 5.

2 Framework Review

The framework is based on the well-known three-level architecture of an information system [ISO82], which is seen as the union of a conceptual schema (for short, schema), an information processor and an information base. In this view, the information base is a representation of the information system domain, whose structure is shaped by the schema. The information processor receives messages from the information system environment, modifies the information base, if necessary, and sends messages to the environment as a response, if needed.

To deal with evolution, a reflective approach is followed [Mant93]. The basic idea is to consider temporal schemas as the domain of a special meta information system. The framework (Figure 1) has two levels. In the lowest level, there are a conceptual schema, an information processor (IP) and an information base (IB), which correspond to the conventional information system. In the highest level, which corresponds to the meta information system, there is a conceptual meta schema, a meta information processor (MIP) and a meta information base (MIB).

There is a one to one correspondence between the MIB and the schema. That means that the evolution of the schema is achieved by modifying the MIB. The two processors (MIP, IP) are loosely coupled, and each one can use a different language. Different schema evolution operations are represented by corresponding meta external event types. When occurrences of those event types are produced, the MIP changes appropriately the MIB and, therefore, the schema changes too. The MIP can also generate external events and notify the IP of them, if changes in the IB are also necessary as a consequence of the schema change.

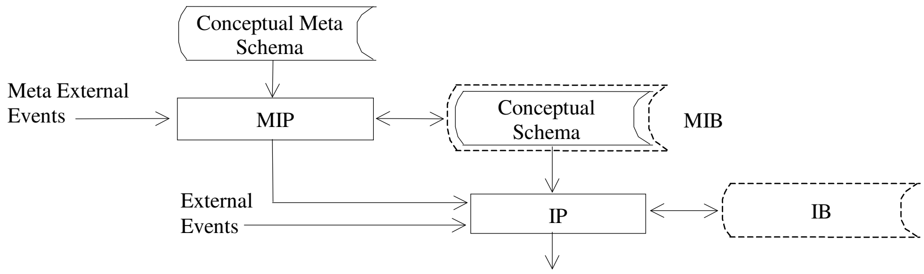

Figure 1: Framework for the evolution of temporal conceptual schemas

For the following descriptions, we choose the first order logic (FOL), with some common extensions, as modeling language, i.e. the language of the MIP and the IP. However, we also use the UML notation in some figures, for illustration purposes only. In our temporal view, time is assumed as linear, discrete and totally ordered [ThLo91], and time points are expressed in a common base unit, such as second, minute, or day.

2.1 Conceptual Meta Schema

The framework pretends to be general, and not linked to a specific conceptual modeling language. For that reason, only a few modeling elements, common to most languages (and enough for the definition of any possible schema) are taken into account to define a *minimal meta schema*. In the following, we review the main features of the core modeling elements considered, and the corresponding minimal meta schema. A more detailed description can be found in [LoOl00].

2.1.1 Schema Core Elements

Any conceptual schema can be defined in terms of five core elements, that must be included in the meta schema: entity types, relationship types, integrity constraints, derivation rules and external event types. All the occurrences of these core elements in a schema may vary over time. We denote by E_t, R_t, Ev_t, IC_t, DR_t, the set of (respectively) entity types, relationship types, external event types, integrity constraints and derivation rules of the schema at time t.

The entities of a domain are instances of *entity types* at particular time points. Entity types can be *lexical* or *non-lexical.* Almost all entity types are non-lexical, except those whose (constant) instances are all the words that can be formed in a certain language (e.g. *Integer*, *String*), which are lexical. With predicate $E(e,t)$, we represent in the IB the fact that entity e is instance of entity type E at time t.

The external events that can be produced in an information system domain are instances of *external event types* at particular time points. With predicate $Ev(ev,t)$ we represent in the IB the fact that event ev is instance of event type Ev at time t, i.e., event ev of type Ev has been produced at time t.

The relationships of a domain are instances of *relationship types* at particular time points. *Attributes* can be seen as particular binary relationship types, and they are not given any special treatment. With predicate $R(e_1,\dots,e_n,t)$ we represent in the IB the fact that $(e_1,\dots,e_n)$ is instance of an n-ary ($n \geq 2$) relationship type R in t, being $e_1,\dots,e_n$ the participant instances in the relationship. The *participants* in a relationship type can be entity types or external event types. In this report we assume that all the relationship types are *synchronous*.[1]

We call *concepts* to the set of entity, relationship and external event types. All concepts have a name. We allow *dynamic* and *multiple classification* of their instances. Concepts are *base* or *derived*. A concept is derived if its instances are obtained from a *derivation rule*, which is a formula written in a language understood by the IP (FOL, here). All other concepts are base. The IP calculates the extension of a derived concept when needed. As lexical entity types are constant, we assume that if they are derived its derivation rule is also constant and can not be changed.

Integrity constraints are also formulas in the language of the IP, which in this case express conditions that the IB must satisfy at each instant. Changes in the IB that violate any integrity constraint of the schema are not allowed. We do not make any distinction, in principle, between different kinds of integrity constraints (cardinality constraints, referential integrity constraints...).

There exists, for each concept, what we can call a *typing rule*. The typing rule for entity types states that, at each time instant t, entities can only be instances of entity types defined in the schema at t: $\forall\, e,t\ E(e,t) \rightarrow E \in E_t$. Similar typing rules are defined for relationship and external event types. For convenience, we assume that when a concept is removed from the schema, all its instances cease to exist in the IB.

In a temporal context, additional intrinsic properties are related to all concepts [Oliv99]. Here, we are interested in the *classification intervals* and *durability* features. The durability property can take two values: *Durable* or *Instantaneous*. We say that an entity type E is instantaneous if for any instance e and time t, if we have $E(e,t)$ then $E(e,t+1)$ does not hold or e is instance of E only at some subintervals of t. In any other case, E is durable. The definition is similar for relationship and external event types. For instance, all external event types are instantaneous. Also, given an instance c, we can find out its *set of classification intervals* with respect to a concept C, which we denote by $Ci(c,C)$. If C is durable, each classification interval is a

[1] An n-ary relationship type R is *synchronous* if $\forall\, e_1,\dots,e_n,t\ (R(e_1,\dots,e_i,\dots,e_n,t) \rightarrow E_i(e_i,t))$, being E_i (i=1,...,n) the participant types in R.

maximal set of consecutive time points in which *c* is instance of *C*. If *C* is instantaneous, we have one classification interval for each time point in which *c* is instance of *C*.

External event types have also related features: an event type definition includes the *parameters* of the event, its *preconditions* and its *effect rules*. The event type parameters can be expressed in terms of particular binary relationship types, in which the event type is one of the participants. Preconditions are constraints that must hold each time an instance of the event is produced. The effect rules are formulas that define the consequences of the event occurrence over the IB.

For modifying the IB, the effect rules generate instances of *structural event types*, which are implicitly defined in the schema. There is an insertion and a deletion event type for each non-lexical event type and each relationship type. Given an n-ary relationship type *R*, the induced effect in the IB of an event $Ins\_R((e_1,\dots,e_n),t)$ is that the relationship $(e_1,\dots,e_n)$ becomes an instance of *R* at *t*. The induced effect in the IB of an event $Del\_R((e_1,\dots,e_n),t)$ is that the relationship $(e_1,\dots,e_n)$ ceases to be an instance of *R* at *t*. The same applies to structural event types corresponding to entity types. If a concept is base, then its structural event types are base. If the concept is derived, then its corresponding event types are derived, and induced by the concept derivation rule [Oliv89].

Effect rules produce changes in the IB that can violate any of the constraints defined in the schema. It is allowed to define, for each integrity constraint, a set of *compensating rules*, which are appended to the effect rules of any event (if they have violated the constraint) to set the IB in a valid state. Compensating rules are, like all other rules described in this section, formulas in the language of the IP.

2.1.2 Minimal Meta Schema

In Figures 2 and 3 we present, using the UML notation, a meta schema which allows the definition of any possible schema in terms of the schema core elements that have been just described in the previous section. In fact, this meta schema can also be described, using the FOL language, in terms of those elements.

Entity, relationship and external event types, integrity constraints and derivation rules are all represented by meta entity types, and their features are represented using some extra meta entity types and also some meta relationship types. For instance, for the durability feature of concepts there exists in the meta schema a lexical meta entity type, *Durability*, which participates, with meta type *Concept*, in a meta relationship type called *HasDurab*.

Most of the meta entity types and meta relationship types are base. Only *Concept*, *EntityType* and *Rule* are derived. *Formula*, *String* and *Durability* are the only lexical meta entity types of the meta schema (e.g. *Durability* has two instances, *Instantaneous* and *Durable*; the instances of *Formula* are all the possible valid FOL sentences).

The minimal meta schema includes also some meta integrity constraints and derivation rules. Here we show only a few of them to serve as an example; for instance, the meta derivation rule corresponding to meta entity type *EntityType,* which is the following: [2]

MDR1: EntityType (e,t) ← LexicalEntityType(e,t) ∨ Non LexicalEntityType(e,t)

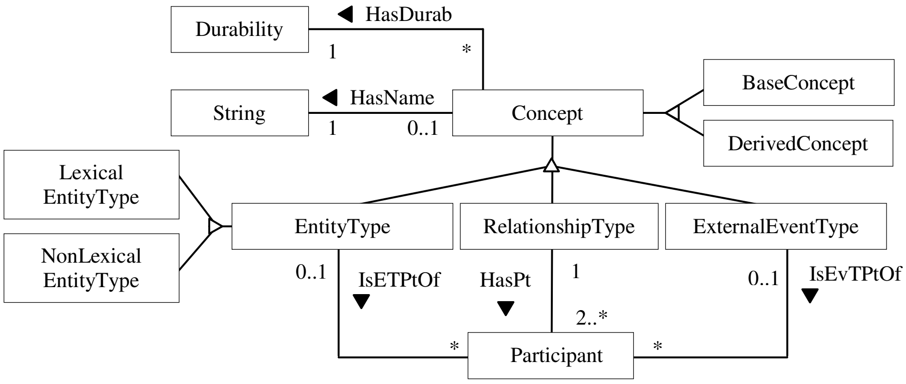

Figure 2 : Minimal conceptual meta schema (Part 1 of 2)

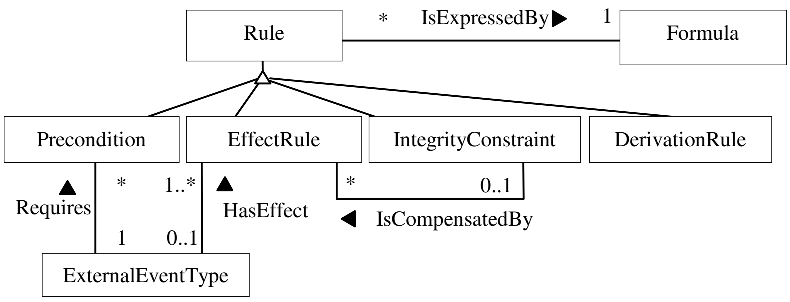

Figure 3: Minimal conceptual meta schema (Part 2 of 2)

A meta derivation rule like this is needed for each partition[3] shown in the figures. Some meta integrity constraints, which for space reasons are not shown here, are also defined [LoOl00]. Some that we will need later are:[4]

MIC1: Concept(c,t) → BaseConcept(c,t) ∨ DerivedConcept(c,t)
with compensating rules:
M_Del_BaseConcept(c,t) ← M_Del_Concept(c,t) ∧ BaseConcept(c,t-1)
M_Del_DerivedConcept(c,t) ← M_Del_Concept(c,t) ∧ DerivedConcept(c,t-1)

MIC2: HasName(c,name,t) → Concept(c,t) ∧ String(name,t)
with compensating rule:
M_Del_HasName(c,name,t) ← M_Del_Concept(c,t) ∧ HasName(c,name,t-1)

MIC3: HasPt(r,p,t) → RelationshipType(r,t) ∧ Participant(p,t)
with compensating rule:
M_Del_HasPt(r,p,t) ← M_Del_RelationshipType(r,t) ∧ HasPt(r,p,t-1)

---

[2] Consecutive identifiers are assigned in this report to meta derivation rules and constraints to ease their reference.
[3] Remember that partitions are not considered, in this approach, a core modeling element, so, even in the meta schema, they will be expressed in terms of derivation rules and integrity constraints.
[4] To distinguish between meta schema and schema structural event types, the former are prefixed with '*M_*'.

MIC4: $Participant(p,t) \rightarrow |\{r \mid HasPt(r,p,t)\}| = 1$
with compensating rule:
$M\_Del\_Participant(p,t) \leftarrow M\_Del\_HasPt(r,p,t)$

MIC5: $IsETPtOf(e,p,t) \rightarrow EntityType(e,t) \wedge Participant(p,t)$
with compensating rule:
$M\_Del\_IsETPtOf(e,p,t) \leftarrow M\_Del\_Participant(p,t) \wedge IsETPtOf(e,p,t\text{-}1)$

MIC6: $IsEvTPtOf(e,p,t) \rightarrow ExternalEventType(e,t) \wedge Participant(p,t)$
with compensating rule:
$M\_Del\_IsEvTPtOf(e,p,t) \leftarrow M\_Del\_Participant(p,t) \wedge IsEvTPtOf(e,p,t\text{-}1)$

2.2 Correspondence Rules

The consistency between the MIB and the conceptual schema is achieved by defining a set of *correspondence rules*. These rules state and ensure that changes in the MIB have its counterpart in 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* (*NonLexicalEntityType*) then it is lexical (resp., non-lexical). Instances of *EntityType* have, at *t*, the durability and the name assigned through *HasDurab* and *HasName* at *t*.

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 $n+1$. Instances of *RelationshipType* have, at *t*, the durability and the name assigned through *HasDurab* and *HasName* at *t*.

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*. Instances of *ExternalEventType* have, at *t*, the durability and the name assigned through *HasDurab* and *HasName* at *t*.

2.3 Meta Schema Extensions

We will see that two additional constructs are necessary to deal with reification: cardinality constraints and frequency, which are not supported by the minimal meta schema.

It is possible to define some meta schema extensions to support other constructs used in conceptual modeling (e.g. partitions). Each meta schema extension shapes the structure of the representation, in the MIB, of instances of a construct. The extension must also include, expressly, some additional meta derivation rules. These rules translate MIB representations of the construct into minimal meta schema instances. That way, the correspondence rules of Section 2.2 are not affected and remain simple and completely effective.

Two extensions of the framework have been previously defined [LoOl00] to support cardinality and frequency, and we review them here. Only significative aspects are described.

2.3.1 Cardinality Constraints

Cardinality constraints are a special kind of static integrity constraints. The minimum (maximum) cardinality constraint for a relationship type and one of its participants indicates the minimum (maximum) number of relationships in which an instance of the participant can take part. For instance, the constraint formula for a participant entity type E_1 in a binary relationship type R with a *min* cardinality value would be the following:

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

The minimal meta schema is extended in the way showed in Figure 4. The cardinality constraints are represented by means of two base meta relationship types, *HasMinCard* and *HasMaxCard*, between *Participant* and *Integer*. Note that is possible for a participant not to have assigned minimum or maximum cardinalities (or both). In those cases a *0* and * cardinality is (respectively) assumed.

Consider only the *HasMinCard* case. To translate this representation into a generic integrity constraint, the *IntegrityConstraint* base meta type is changed into derived, with two subtypes: a base one (*ExplicitIntegrityConstraint*), which has as instances the integrity constraints defined explicitly by the designer, and a derived one (*ImplicitIntegrityConstraint*), which is populated with integrity constraints induced automatically by the system. A derived subtype of the latter is, among others, *ICMinCard*. The *IsExpressedBy* base relationship type between *Rule* and *Formula* is refined into a derived one between *ICMinCard* and *Formula*. The meta derivation rules corresponding to *ICMinCard* and *IsExpressedBy* are:

MDR2: ICMinCard (ic,t) ← HasMinCard(p,min,t) ∧ ic=Sym(ICMinCard, p)
MDR3: IsExpressedBy (ic,f,t) ← ICMinCard(ic,t) ∧ ICMinCardFormula(ic,f,t)

where *Sym(ICMinCard,p)* is a time independent system function that returns a symbol different from any other symbol in the MIB, and *ICMinCardFormula* returns in *f* the cardinality constraint formula corresponding to participant *p*. We omit here the details of this predicate because they are a lengthy sequence of trivial string manipulation operations.

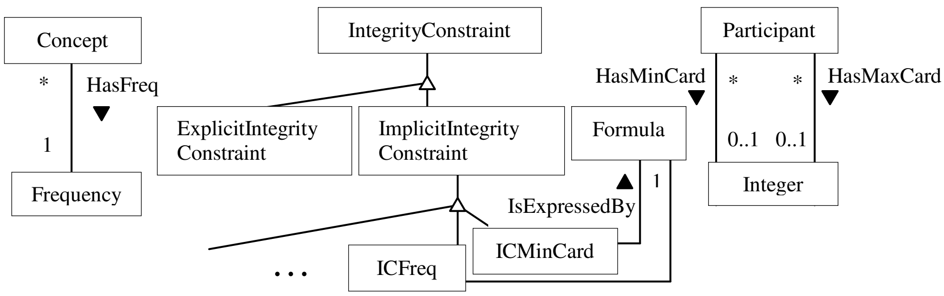

Figure 4 : Meta schema conceptualization of cardinality and frequency constraints

A defined meta integrity constraint that we will use later is:

MIC7: HasMinCard(p,min,t) → Participant(p,t) ∧ Integer(min,t)
with compensating rule:
M_Del_ HasMinCard(p,min,t) ← M_Del_Participant(p,t) ∧ HasMinCard(p,min,t-1)

2.3.2 Frequency Constraints

Frequency is a temporal feature of concepts that can take two values: *Single* or *Intermittent* [Oliv99]. A concept has a single frequency if all its instances have only one classification interval. Otherwise, it is intermittent. A constraint is not needed for an intermittent frequency value, but a single value can be translated into a (dynamic) *frequency integrity constraint*. The general formula of this constraint for an entity type *E* is:

$$\text{“}E(e,t) \rightarrow |Ci(e,E)|=1\text{”}$$

The meta schema extension for frequency is shown in Figure 4. As done in the cardinality constraints case, a new subtype of *ImplicitIntegrityConstraint* is defined: *ICFreq*. The main meta derivation rules required here are the following:

MDR4: ICFreq (ic,t) ← HasFreq(c,Single,t) ∧ ic=Sym(ICFreq, c)
MDR5: IsExpressedBy (ic,f,t) ← ICFreq(ic,t) ∧ ICFreqFormula(ic,f,t)

3 Reification Review

In this report we assume the reification types definition given in [Oliv99], and we reproduce it below for ease of reference.

Reification is a schema transformation in conceptual modeling of information systems. It consists in considering a relationship type as an entity type, in order to allow the use of the reified entity type as a participant in other relationship types. Formally, reification of an n-ary relationship type *R* is a schema transformation that produces a new *intrinsic* entity type *E* and n *intrinsic* relationship types $R_1,...,R_n$ which connect *E* with the n participant entity types in *R*.

Figure 5a shows an example of the result of a reification. A relationship type *WorksIn* between *Programmer* and *Project* is reified into a *Participation* intrinsic entity type and two intrinsic binary relationship types, *IsOf* and *IsIn*. The intrinsic entity type participates in a binary relationship type called *WHours* which represents a number of working hours.

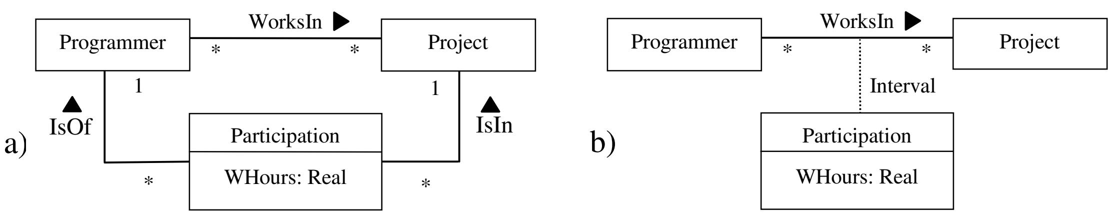

Figure 5: a) Reification result b) UML graphical representation of reification per interval.

From a temporal perspective, various different types of reification can be defined, and they can coexist for the same relationship type. The example includes only one reification, and its semantics differ depending on the reification type considered:

- Reification per Instant
- Reification per Classification Interval (or, in short, per Interval)
- Reification per Life Span

Assuming *reification per instant*, reified entity type *Participation* will have a different instance at time *t* per every *WorksIn* instance at *t*. Figure 6a shows how an instance *(Pg,Pr)* of *WorksIn* produces three instances, α, β and χ, of *Participation* at different time instants. With day as base time unit, this allows us to register the total number of *WHours* of a programmer in a project for each day she works in that project.

Assuming *reification per interval*, reified entity type *Participation* will have a different instance for each *WorksIn* instance and classification interval. Figure 6b shows how an instance *(Pg,Pr)* of *WorksIn* produces two instances, α and β, of *Participation* at two different time intervals. With day as base time unit, this allows us to register the total number of *WHours* of a programmer in a project for each set of consecutive days she works in that project. Figure 5b shows, as an example, the graphical representation in UML of a reification of this type. A special label indicates the reification type.

Assuming *reification per life span*, reified entity type *Participation* will have an instance per every *WorksIn* instance. Figure 6c shows how an instance *(Pg,Pr)* of *WorksIn* has always an associated instance α of *Participation.* With day as base time unit, this allows us to register the total number of *WHours* of a programmer in a project.

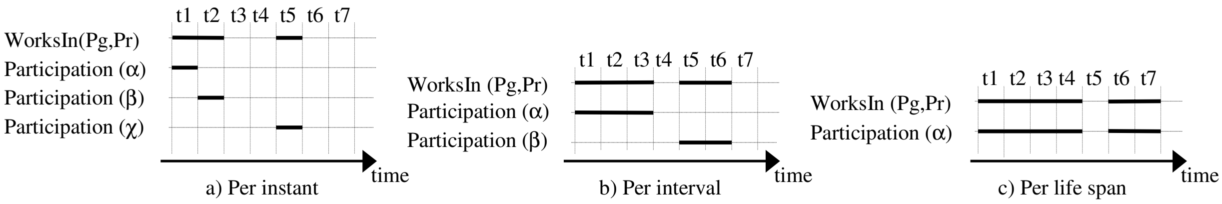

Figure 6 : Examples of different reification types.

A reification transformation is correct if the new intrinsic types satisfies four conditions. Two of them are the same for the three reification types:

a) Intrinsic relationship types R_i are constant with respect to *E* (each instance of *E* participates always in the same instance of each R_i).
b) Each instance of *E* participates in one and only one instance of each R_i at time *t*.

The other two are different for each reification type. Note the use of predicate *IncludedIn(t,ti)*, which indicates if time point *t* is included in the temporal interval *ti*.

- Reification per instant:
  c) E and R_i's are instantaneous and single.
  d) There is a one-to-one correspondence between the instances of *R* and *E* at any time *t*:
  $\forall e_1,...,e_n,t\ (R(e_1,...,e_n,t) \rightarrow \exists!\ e\ (E(e,t) \wedge R_1(e,e_1,t) \wedge .... \wedge R_n(e,e_n,t)))$
  $\forall e,e_1,...,e_n,t\ (E(e,t) \wedge R_1(e,e_1,t) \wedge .... \wedge R_n(e,e_n,t) \rightarrow R(e_1,...,e_n,t))$

- Reification per interval:
  c) E and R_i's are single. The value of durability for *E* and R_i's is the same as that of *R*.
  d) There is a one-to-one correspondence between the populations of *R* and *E* at any time *t*, and there is only one entity for each classification interval of a reified relationship:
  $\forall e_1,...,e_n,t,ti\ (R(e_1,...,e_n,t) \wedge ti \in Ci((e_1,...,e_n),R) \wedge IncludedIn(t,ti)$
  $\rightarrow \exists!\ e\ (E(e,t) \wedge R_1(e,e_1,t) \wedge ... \wedge R_n(e,e_n,t) \wedge ti \in Ci(e,E)))$
  $\forall e,e_1,...,e_n,t,t_i,\ (E(e,t) \wedge R_1(e,e_1,t) \wedge .... \wedge R_n(e,e_n,t) \wedge ti \in Ci(e,E) \rightarrow (R(e_1,...,e_n,t) \wedge ti \in Ci((e_1,...,e_n),R)))$

- Reification per life span.
  c) The values of durability and frequency for *E* and R_i's are the same as those of *R*.
  d) There is a one-to-one correspondence between the populations of *R* and *E* at any time *t*, and there is only one entity for all classification intervals of a reified relationship:
  $\forall e_1,...,e_n,t\ (R(e_1,...,e_n,t) \rightarrow \exists!\ e\ (E(e,t) \wedge R_1(e,e_1,t) \wedge .... \wedge R_n(e,e_n,t) \wedge Ci((e_1,...,e_n),R)=Ci(e,E)))$
  $\forall e,e_1,...,e_n,t\ (E(e,t) \wedge R_1(e,e_1,t) \wedge .... \wedge R_n(e,e_n,t) \rightarrow (R(e_1,...,e_n,t) \wedge Ci((e_1,...,e_n),R)=Ci(e,E)))$

4 Reification Evolution

After the necessary background definitions and reviews, we can center now our attention into the real motivation of this report: to deal with conceptual schema evolutions associated to reification. Some possible types of schema changes related to the reification construct are:

- Reification of a relationship type.
- Removal of a reification of a relationship type.

Moreover, changes related to other constructs can affect indirectly to reifications:

- Change of the frequency of a reified relationship type.
- Change of the durability of a reified relationship type.
- Removal of a reified relationship type.

We want to specify the effects of those possible changes at conceptual level, without descending to the design nor application level. Therefore, we apply the framework just reviewed to the reification case.

We define an extension of the meta schema to support the definition, in the MIB, of reifications of the three types described. Two meta external event types, *M_AddReification* and *M_RemoveReification,* are also defined. They specify the effects of the first two schema change types mentioned above. We will see that the extension can be defined in such a way that is not necessary to define specific meta external event types for the other three schema change types.

4.1 Reification Meta Schema Extension

To define an extension of the meta schema, we assume as derived the intrinsic types that result from a reification. The intrinsic entity type and the intrinsic relationship types are populated by a schema derivation rule which infers their instances from the instances of the reified relationship type.

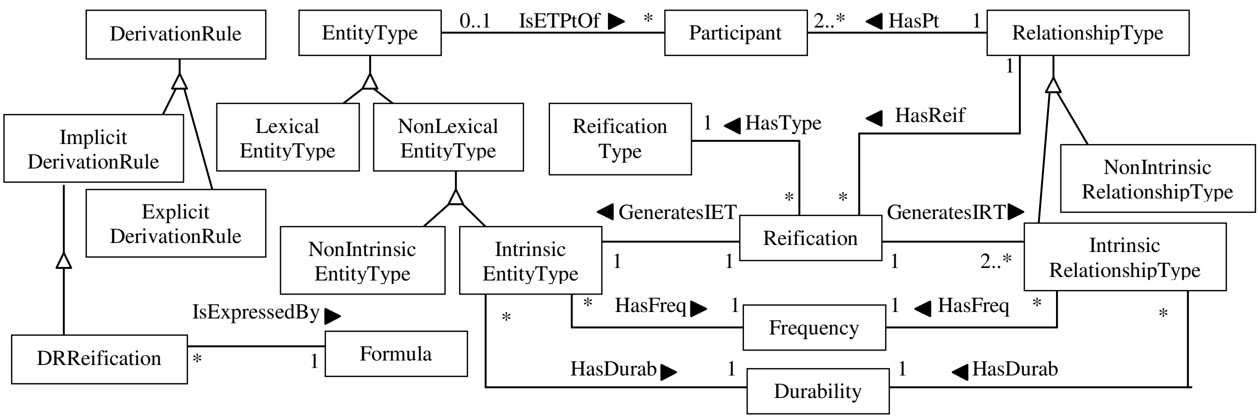

Figure 7: Meta schema conceptualization of reification

Figure 7 shows the conceptualization of reification. It includes some new base meta entity and relationship types. We allow the reification of any *RelationshipType*. A *Reification* must have a *ReificationType* (a lexical meta entity type). We consider useful to partition the (now derived) meta entity types *NonLexicalEntityType* and *RelationshipType* into an intrinsic and a non-intrinsic base subtypes. *IntrinsicEntityType* and *IntrinsicRelationshipType* have as instances the (derived) schema intrinsic types generated as a consequence of a reification.

Extending the meta schema requires also several meta integrity constraints and meta derivation rules. We indicate here those that we will need later:

MIC8: Reifications generate intrinsic entity types.
GeneratesIET(re,ie,t) → Reification(re,t) ∧ IntrinsicEntityType(ie,t)
with compensating rule:
M_Del_GeneratesIET(re,ie,t) ← M_Del_Reification(re,t) ∧ GeneratesIET(re,ie,t-1)

MIC9: Every intrinsic entity type is generated by one and only one reification.
IntrinsicEntityType(ie,t) → |{re | GeneratesIET(re,ie,t)} | =1
with compensating rule:
M_Del_IntrinsicEntityType(ie,t) ← M_Del_GeneratesIET(re,ie,t)

MDR6: Reifications generate intrinsic relationship types. *GeneratesIRT* is derived.
GeneratesIRT(re,ir,t) ← GeneratesIET(re,ie,t) ∧ IsETPtOf(ie,pi,t) ∧ HasPt(ir,pi,t)

MIC10: Every intrinsic relationship type is generated by one and only one reification.
IntrinsicRelationshipType(ir,t) → |{re | GeneratesIRT(re,ir,t)} | =1
with compensating rule:
M_Del_IntrinsicRelationshipType(ir,t) ← M_Del_GeneratesIRT(re,ir,t)

MIC11: Only relationship types can be reified.
HasReif(r,re,t) → Reification(re,t) ∧ RelationshipType(r,t)
with compensating rules:
M_Del_HasReif(r,re,t) ← M_Del_Reification (re,t) ∧ HasReif(r,re,t-1)
M_Del_HasReif(r,re,t) ← M_Del_RelationshipType(r,t) ∧ HasReif(r,re,t-1)

MIC12: Each Reification refers to one and only one relationship type.
Reification(re,t) → |{r | HasReif(r,re,t)} | =1
with compensating rule:
M_Del_Reification(re,t) ← M_Del_HasReif(r,re,t)

MIC13: Reifications have types.
HasType(re,ty,t) → Reification(re,t) ∧ ReificationType(ty,t)
with compensating rule:
M_Del_HasType(re,ty,t) ← M_Del_Reification(re,t) ∧ HasType(re,ty,t-1)

This meta schema extension defines how reification information must be organized in the MIB. But not all this information must be explicitly stated by the designer. We find convenient to define some meta derivation rules that allow the automatic inference of the frequency and durability properties of the intrinsic types of a reification.

Meta relationship types *HasFreq* and *HasDurab* define, respectively, the frequency and the durability of a concept. We refine them into derived ones for *IntrinsicEntityType* and *IntrinsicRelationshipType* participants, as shown in Figure 7. The meta derivation rules for intrinsic entity types are (according to condition *c)* defined in Section 3):

MDR7: HasFreq(ie,f,t) ← GeneratesIET(re,ie,t) ∧ HasType(re,ty,t) ∧ HasReif(r,re,t) ∧
(((ty=Instant ∨ ty=Interval) ∧ f=Single) ∨ (ty=Life ∧ HasFreq(r,f,t)))
MDR8: HasDurab(ie,d,t) ← GeneratesIET(re,ie,t) ∧ HasType(re,ty,t) ∧ HasReif(r,re,t) ∧
((ty=Instant ∧ d=Instantaneous) ∨ ((ty=Interval ∨ ty=Life) ∧ HasDurab(r,d,t)))

Intrinsic relationship types are assigned the same durability and frequency values as their corresponding intrinsic entity type is, using the following meta derivation rules:

MDR9: HasFreq(ir,f,t) $\leftarrow$ GeneratesIET(re,ie,t) $\wedge$ GeneratesIRT(re,ir,t) $\wedge$ HasFreq(ie,f,t)
MDR10: HasDurab(ir,d,t) $\leftarrow$ GeneratesIET(re,ie,t) $\wedge$ GeneratesIRT(re,ir,t) $\wedge$ HasDurab(ie,d,t)

In the beginning of this section we have assumed that intrinsic entity and relationship types are derived. Their corresponding derivation rules fulfil conditions *a)*, *b)* and *d)* defined in Section 3. Their formulas can also be automatically inferred and represented in the MIB.

To infer these formulas, we partition *DerivationRule* (Figure 7) into a base subtype (*ExplicitDerivationRule*) and a derived one (*ImplicitDerivationRule*). *ExplicitDerivationRule* has as instances derivation rules of the schema defined explicitly by the designer. *ImplicitDerivationRule* has as instances derivation rules of the schema automatically inferred.

A derived subtype of the latter is, among others, *DRReification.* We refine the base meta relationship type *IsExpressedBy* between *Rule* and *Formula* into a derived one between *DRReification* and *Formula.* The meta derivation rules required are:

MDR11: DRReification(dr,t) $\leftarrow$ Reification(re,t) $\wedge$ dr=sym(DRReification,re)
MDR12: IsExpressedBy(dr,f,t) $\leftarrow$ DRReification(dr,t) $\wedge$ DRReifFormula(dr,f,t)

where predicate *DRReifFormula* gives in *f* the corresponding formula of the intrinsic types derivation rule (one rule is enough). The general form of the formula for an n-ary relationship type *R* reified into a derived entity type *E* and n derived relationship types $R_1,...,R_n$ is, depending on the reification type:[5]

- Reification per instant:
  "$E(e,t) \wedge R_1(e,e_1,t) \wedge ... \wedge R_n(e,e_n,t) \leftarrow R(e_1, ... ,e_n,t) \wedge e=newSymbol()$"
- Reification per time interval:
  If *R* is instantaneous:
  "$E(e,t) \wedge R_1(e,e_1,t) \wedge ... \wedge R_n(e,e_n,t) \leftarrow R(e_1,...,e_n,t) \wedge e=newSymbol()$"
  If *R* is durable:
  "$E(e,t) \wedge R_1(e,e_1,t) \wedge ... \wedge R_n(e,e_n,t) \leftarrow R(e_1, ... ,e_n,t) \wedge ((E(e,t-1) \wedge R_1(e,e_1,t-1) \wedge ... \wedge R_n(e,e_n,t-1))$
  $\vee (\neg R(e_1, ... ,e_n,t-1) \wedge e=newSymbol()))$"
- Reification per life span:
  "$E(e,t) \wedge R_1(e,e_1,t) \wedge ... \wedge R_n(e,e_n,t) \leftarrow R(e_1, ... ,e_n,t) \wedge ((\exists t'\ E(e,t') \wedge R_1 (e,e_1,t') \wedge ... \wedge R_n(e,e_n,t') \wedge (t'<t))$
  $\vee ((\neg\exists t'\ R(e_1, ... ,e_n,t') \wedge (t'<t)) \wedge e=newSymbol()))$"

4.2 External Event Types

As we saw in Section 2, schema evolution is performed in the framework by meta external events. When a designer wants to do a change in the 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 2.2), such changes imply changes in the schema, and maybe in the IB. We present here the external event types which allow to perform schema changes that involve reifications.

4.2.1 Meta external event type M_AddReification

We define the meta external event type *M_AddReification(rer,t)*, which has three parameters. The first one is given by the meta relationship type *ReifInto* with *RelationshipType*, and represents the relationship type *R* that is going to be reified. The second one is given by the

[5] Note that, in what follows, we use the special predicate *newSymbol*() to obtain new MIB or IB symbols.

meta relationship type *ReifName* with *String*, and represents the name of the new entity *E*. And the last one is given by the meta relationship type *ReifType* with *ReificationType*, that represents the type of the reification.

Parameters: ReifInto(rer,r,t) ∧ ReifName(rer,name,t) ∧ ReifType(rer,type,t)

Preconditions. It can not exist another reification of the same type for the relationship type considered. The if-part of the rule gives us the event type parameters and will be omitted for clarity in the following.

M_AddReification(rer,t) ∧ ReifInto(rer,r,t) ∧ ReifName(rer,name,t) ∧ ReifType(rer,type,t)
→ ¬ ∃ re (HasReif(r,re,t-1) ∧ HasType(re,type,t-1))

Effect rules. A first set of effect rules induces the meta structural events that create a new intrinsic and derived entity type *ie*, with the name given as an event parameter.

ie=newSymbol()
M_Ins_IntrinsicEntityType(ie,t)
M_Ins_DerivedConcept(ie,t)
M_Ins_HasName(ie,name,t)

Now we must create a new reification *re*, linked to its corresponding relationship type *r* and the new entity type *ie* in which *r* is going to be reified. The reification type is also established.

re=newSymbol()
M_Ins_Reification(re,t)
M_Ins_GeneratesIET(re,ie,t)
M_Ins_HasReif(r,re,t)
M_Ins_HasType(re,type,t)

Finally, for each participant entity type *ei* of *r*, a new derived intrinsic relationship type *ri* must be created, linking the new intrinsic entity type *ie* and *ei*. The correct cardinality constraints must be established for both participants in each *ri*. A system-generated name is assigned to the new intrinsic relationship types. If the designer wants to change these names, she can do it later. The following effect rule works for participants that are entity types:

M_Ins_IntrinsicRelationshipType(ir,t) ∧ M_Ins_DerivedConcept(ir,t) ∧ M_Ins_HasName(ir,ir,t) ∧
M_Ins_Participant(pi',t) ∧ M_Ins_HasPt(ir,pi',t) ∧ M_Ins_IsETPtOf(ie,pi',t) ∧
M_Ins_Participant(pi'',t) ∧ M_Ins_HasPt(ir,pi'',t) ∧ M_Ins_Is**ET**PtOf(ei,pi'',t) ∧
M_Ins_HasMinCard(pi',1,t) ∧ M_Ins_HasMaxCard(pi',1,t)
← ir=newSymbol() ∧ pi'=newSymbol() ∧ pi''=newSymbol() ∧ HasPt (r,pi,t-1) ∧ Is**ET**PtOf(ei,pi,t-1)

And the following effect rule works for participants that are event types:

M_Ins_IntrinsicRelationshipType(ir,t) ∧ M_Ins_DerivedConcept(ir,t) ∧ M_Ins_HasName(ir,ir,t) ∧
M_Ins_Participant(pi',t) ∧ M_Ins_HasPt(ir,pi',t) ∧ M_Ins_IsETPtOf(ie,pi',t) ∧
M_Ins_Participant(pi'',t) ∧ M_Ins_HasPt(ir,pi'',t) ∧ M_Ins_Is**EvT**PtOf(ei,pi'',t) ∧
M_Ins_HasMinCard(pi',1,t) ∧ M_Ins_HasMaxCard(pi',1,t)
← ir=newSymbol() ∧ pi'=newSymbol() ∧ pi''=newSymbol() ∧ HasPt (r,pi,t-1) ∧ Is**EvT**PtOf(ei,pi,t-1)

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 partition MDR's (Section 2.1.2 and 4.1), *ie* becomes instance of *NonLexicalEntityType, EntityType* and *Concept*, and $ir_1,...,ir_n$ begin to be instances of *RelationshipType* and *Concept.*
- By MDR6 (Section 4.1), *Reification re* is associated to each generated *IntrinsicRelationshipType* (i.e. $ir_1,...,ir_n$) in an instance of *GeneratesIRT*.
- By MDR7 and MDR9 (Section 4.1), each $ie,ir_1,...,ir_n$ *HasFreq* at *t*. By MDR4 and MDR5 (Section 2.3.2), the *Formula* in which *IsExpressedBy* each corresponding *ICFreq* is defined.

- By MDR8 and MDR10 (Section 4.1), each $ie,ir_1,...,ir_n$ *HasDurab* at *t*.
- By MDR2 and MDR3 (Section 2.3.1), the *Formula* in which *IsExpressedBy* each *ICMinCard* for each *pi' Participant* is defined. The same happens with *ICMaxCard*.
- By MDR11 and MDR12 (Section 4.1), the *Formula* in which *IsExpressedBy* the required *DRReification* for the intrinsic types $ie,ir_1,...,ir_n$ is defined.
- By MDR's (trivial and not shown in this report) of the minimal meta schema and its extensions, the induced instance of *DRReification* becomes instance of *ImplicitDerivationRule*, *DerivationRule* and *Rule*. Also, the induced instances of *ICFreq, ICMinCard* and *ICMaxCard* become instances of *ImplicitIntegrityConstraint*, *IntegrityConstraint* and *Rule*.

Induced effects on the schema and IB.

- By CR1, *ie* is a derived non-lexical entity type of the schema at t (that is, $ie \in E_t$) with the defined durability, frequency and name.
- By CR2, $ir_1, ..., ir_n$ are derived binary relationship types of the schema at *t* (that is, $r_1, ..., r_n \in R_t$) with the defined durability, frequency, and cardinalities.
- By CR3, the *Formula* assigned to the *DRReification* induced is a derivation rule of the schema at *t*.
- By CR4, each *Formula* assigned to the *ICFreq*'s, *ICMinCard*'s and *ICMaxCard*'s induced is an integrity constraint of the schema at *t*.
- The new intrinsic entity and relationship types are populated in the IB, from the instances of the reified relationship type, by the derivation rule defined in Section 4.1, which induces the appropriate structural insertion events.

4.2.2 Meta external event type M_RemoveReification

Now, we define the meta external event type *M_RemoveReification(rer,t)*. It has one parameter, given by the meta relationship type *RemReif* with *Reification*, which represents the reification that is going to be removed.

Parameters: RemReif(rer,re,t)

Preconditions. It must exist the reification that we want to remove. The if-part of the rule gives us the event type parameters and will be omitted in the following.

M_RemoveReification (rer,t) $\wedge$ RemReif(rer,re,t) $\rightarrow$ Reification(re,t-1)

Effect rules. The only effect rule needed is:

M_Del_Reification(re,t)

Induced effects on the MIB.

- The compensating rule (c.r.) of MIC11 (Section 4.1) induces *M_Del_HasReif(r,re,t)*.
- The c.r. of MIC13 (Section 4.1) induces *M_Del_HasType(re,ty,t)*.
- The c.r. of MIC8 (Section 4.1) induces *M_Del_GeneratesIET(re,ie,t)*.
- The c.r. of MIC9 (Section 4.1) induces *M_Del_IntrinsicEntityType(ie,t)*.
- MDR6 (Section 4.1) induces *M_Del_GeneratesIRT(re,ir,t)* and the c.r. of MIC10 (Section 4.1) induces *M_Del_IntrinsicRelationshipType(ir,t),* for each intrinsic relationship type *ir* of reification *re.*
- By partition MDR's (Section 2.1.2 and 4.1), *ie* ceases to be instance of *NonLexicalEntityType, EntityType* and *Concept*, and $ir_1,...,ir_n$ cease to be instance of *RelationshipType* and *Concept.* By the c.r. of MIC1 and MIC2 (Section 2.1.2), they cease to be instance of *DerivedConcept* and their corresponding instances of *HasName* are removed.
- By MDR7 and MDR9 (Section 4.1), each $ie,ir_1,...,ir_n$ ceases to *HasFreq* at *t*. By MDR4 and MDR5, the *Formula* in which *IsExpressedBy* each corresponding *ICFreq* ceases to exist.
- By MDR8 and MDR10 (Section 4.1), each $ie,ir_1,...,ir_n$ ceases to *HasDurab* at *t*.
- The c.r.'s of MIC3, MIC4, MIC5 and MIC6 (Section 2.1.2) induce *M_Del_HasPt(ir,pi',t)*, *M_Del_HasPt(ir,pi'',t)*, *M_Del_Participant(pi',t)*, *M_Del_Participant(pi'',t)* and *M_Del_IsETPtOf(ei,pi'',t)* (or *M_Del_IsEvTPtOf(ei,pi'',t)*) for each removed $ir_1,...,ir_n$.

- The c.r. of MIC7 (Section 2.3.1) induces *M_Del_HasMinCard(pi',1,t)*. The same happens in the *HasMaxCard* case.
- By MDR2 and MDR3 (Section 2.3.1), the *Formula* in which *IsExpressedBy* each *ICMinCard* for each *pi'* participant ceases to be defined. The same happens with *ICMaxCard*.
- By MDR11 and MDR12 (Section 4.1), the *Formula* in which *IsExpressedBy* the required *DRReification* for the intrinsic types $ie, ir_1, ..., ir_n$ ceases to be defined.
- By MDR's (trivial and not shown in this report) of the meta schema and its extensions, the induced instance of *DRReification* ceases to be instance of *ImplicitDerivationRule*, *DerivationRule* and *Rule*. Also, the induced instances of *ICFreq*, *ICMinCard* and *ICMaxCard* cease to be instances of *ImplicitIntegrityConstraint*, *IntegrityConstraint* and *Rule*.

Induced effects on the schema and IB.

- By CR1, *ie* ceases to be a derived non-lexical entity type of the schema at t (that is, $ie \notin E_t$).
- By CR2, $ir_1, ..., ir_n$ cease to be derived binary relationship types of the schema at *t* (that is, $r_1, ..., r_n \notin R_t$).
- By CR3, the *Formula* assigned to the induced *DRReification* ceases to be a derivation rule of the schema at *t*.
- By CR4, each *Formula* assigned to the induced *ICFreq*'s, *ICMinCard*'s and *ICMaxCard*'s ceases to be an integrity constraint of the schema at *t*.
- By the typing rule, instances of $ie, ir_1, ..., ir_n$ cease to exist in the IB.

4.2.3 Other meta external event types.

We had defined some schema changes which could indirectly affect reification:

- Changing the frequency or durability of a reified relationship type: The framework includes generic meta external event types for this kind of schema changes. These changes affect to the frequency and durability properties of the intrinsic entity and relationship types, and, in some cases, to their derivation rule. As those information is automatically inferred (MDR7 to MDR12), is not necessary to define a new event type specific for reified relationship types.
- Removal a reified relationship type: The framework includes generic external meta event types for this kind of schema change. Removal of a reified relationship type requires the removal of its reifications. Its easy to see that the meta compensating rules defined in Section 4.1 (see MIC11 and MIC12) induce the reification information removal from the MIB. Thus, a new meta event type specific for reified relationship types is not required.

5 Conclusions

In this report we have studied temporal conceptual schema evolutions related to the reification construct. As the temporal dimension has been taken into account, some different kinds of reification have been considered.

To deal with evolution, we have applied a previously defined framework for temporal conceptual schemas evolution. We have seen that, with only some extensions, the framework easily supports reification in all of its types, and is able to manage any potential schema change which involves it. The effects of the possible changes have been specified at conceptual level only, and not at design nor application level, thus increasing the understandably of the specification and reducing the complexity of the management of changes.

References

[BCG+87] Banerjee, J., Chou, H-T., Garza, J.F., Kim, W., Woelk, D., Ballou, N.: Data Model Issues for Object-Oriented Applications. In: ACM TOIS Vol.5, No.1, January (1987) 3-26

[GSÖP98] Goralwalla, I., Szafron, D., Özsu, T., Peters, R.: A Temporal Approach to Managing Schema Evolution in Object Database Systems. In: Data&Knowledge Eng. 28(1), October (1998) 73-105

[HEH+94] Hainaut, J-L., Englebert, V., Henrard, J., Hick, J-M., Roland, D.: Database Evolution: the DB-MAIN Approach. In: 13th. Intl. Conf. on the Entity-Relationship Approach - ER'94, LNCS 881, Springer-Verlag (1994) 112-131

[ISO82] ISO/TC97/SC5/WG3: Concepts and Terminology for the Conceptual Schema and Information Base. J.J. van Griethuysen (ed.), March (1982)

[JGJ+95] Jarke, M., Gallersdörfer, R., Jeusfeld, M.A., Staudt, M., Eherer, S.: ConceptBase - a deductive object base for meta data management. In: Journal of Intelligent Information Systems, 4(2) (1995) 167-192

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

[LoOl00] López, J.R.; Olivé, A.: A framework for the evolution of temporal conceptual schemas of information systems. In: Proc. Conference in Advanced Information Systems Engineering - CAiSE'00 LNCS 1789, Springer-Verlag (2000) 369-386. An extended version can be found at: Research Report LSI-00-14-R, LSI, UPC, http://www.lsi.upc.es/dept/techreps/2000.html.

[Mant93] Manthey, R.: Beyond Data Dictionaries: Towards a Reflective Architecture of Intelligent Database Systems. In: DOOD'93, Springer-Verlag (1993) 328-339

[Oliv89] Olivé, A.: On the design and implementation of information systems from deductive conceptual models. In Proc. VLDB'89, Amsterdam, The Netherlands (1989) 3-11

[Oliv99] Olivé, A.: Relationship reification: A temporal View. In: Proc. CAiSE'99, Heidelberg, Germany, LNCS 1626, Springer (1999) 396-410

[Rodd94] Roddick, J.F.:Schema Evolution in DataBase Systems – An Updated Bibliography. In: ACM SIGMOD Rec., 21(4), May (1994) 35-40.

[ThLo91] Theodoulidis, C.I.; Loucopoulos, P.: The time dimension in conceptual modeling. In: Information Systems, Vol 16, No. 3, Pergamon Press (1991) 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.

[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.

LIST OF FIGURES