A Semantic Approach for Schema Evolution and Versioning in Object-Oriented Databases

Enrico Franconi[1], Fabio Grandi[2] and Federica Mandreoli[2]

[1] Dept. of Computer Science, University of Manchester, UK
http://www.cs.man.ac.uk/~franconi/
franconi@cs.man.ac.uk
[2] Dip. di Elettronica, Informatica e Sistemistica, Università di Bologna, Italy
{fgrandi|fmandreoli}@deis.unibo.it

Abstract. In this paper a semantic approach for the specification and the management of databases with evolving schemata is introduced. It is shown how a general object-oriented model for schema versioning and evolution can be formalized; how the semantics of schema change operations can be defined; how interesting reasoning tasks can be supported, based on an encoding in description logics.

1 Introduction

The problems of schema evolution and versioning arose in the context of long-lived database applications, where stored data were considered worth surviving changes in the database schema [25]. According to a widely accepted terminology [19], a database supports *schema evolution* if it permits modifications of the schema without the loss of extant data; in addition, it supports *schema versioning* if it allows the querying of all data through user-definable version interfaces. For the sake of brevity, we will consider the schema evolution as a special case of the schema versioning where only the current schema version is retained.

With schema versioning, different schemata can be defined and selected by means of a suitable "coordinate system": symbolic labels are often used in design systems to this purpose, whereas time values are the elective choice for temporal applications. In a generalized versioning model [14, 15], as required by advanced applications (e.g. CAD/CAM, software engineering and GIS), the support of both temporal and design schema versions should be taken into account. In this context, object-oriented data models are usually considered, though schema versioning in relational databases [11] has also been studied deeply. In particular, the adoption of an object-oriented data model is the most common choice in the literature concerning schema evolution.

The contribution of this paper concerns the introduction of a semantic approach for the specification and the management of evolving database schemata. The proposal is based on a formal framework which consists of an object-oriented data model extended with schema versions and its encoding in a description logics which supports complex reasoning tasks. The proposed model extends the "snapshot" object-oriented model introduced in [1] and developed in [8], which is consistent with all the static features of UML/OMT and ODMG models. Moreover, it provides full support for the taxonomy of primitive schema changes usually considered in the literature [4]. Formal semantics for the schema (version) and for the supported schema change operations is introduced. It is then proved how both the schemata and the schema changes can be encoded as inclusion dependencies in a suitable description logics, based on $\mathcal{ALCQI}$ [8]. Furthermore, it is shown how several interesting tasks concerning evolving schemata can be solved using existing description logics reasoning procedures.

In particular, the adoption of our semantic approach for the solution of schema evolution and versioning problems is aimed at enhancing the global functionalities of a system supporting evolving schemata as follows:

- The complexity of schema changes becomes potentially unlimited: in addition to the classical schema change primitives, our approach enables the definition of complex and articulated schema changes by means of powerful interschema assertions [10].
- Techniques for consistency checking and classification can be automatically applied to any resulting schema. We define different notions of consistency, related to existence of a legal database for the global schema or for a single schema version, or related to the consistency of single classes within a consistent schema (version). Classification tasks we define involve the discovery of implicit inclusion/inheritance relationships between classes (e.g. subsumption [5]).
- The process of schema transformation can be formally checked. The provided semantics of the various schema change operations makes it possible to reduce the correctness proof to solvable reasoning tasks. As far as we know, this problem has not been considered before.

The paper is organized as follows. After a survey of the current status of the field, Section 3 first introduces the syntax and the semantics of the object-oriented model for evolving schemata and then formally defines the relevant reasoning problems supporting the design and the management of an evolving schema. Section 4 provides a provably correct encoding of the object-oriented model for evolving schemata into a description logic, so that theoretical results from the description logic field can be reused for the object-oriented model. Conclusions are finally drawn in Sec. 5.

2 Related Work

The problems of schema evolution and schema versioning support have been diffusely studied in relational and object-oriented database papers: [25] provides an excellent survey on the main issues concerned. The introduction of schema change facilities in a system involves the solution of two fundamental problems: the **semantics of change**, which refers to the effects of the change on the schema itself, and the **change propagation**, which refers to the effects on the underlying data instances. The former problem involves the checking and maintenance of schema consistency after changes, whereas the latter involves the consistency of extant data with the modified schema.

In the relational field [26, 11], the two problems are solved by specifying a precise semantics of the schema changes at intensional and extensional levels, for example via algebraic operations on catalogue and base tables. However, the related consistency problems have not been considered so far. The correct use of schema changes is completely under the database designer/administrator's responsibility, without automatic system aid and control.

In the object-oriented field, several solutions have been proposed for the "semantics of change" and for the "change propagation", and also the properties of complex schema changes and the consistency problems have been investigated to a certain extent.

In particular, two main approaches were followed to ensure consistency in pursuing the "semantics of change" problem. The first approach is based on the adoption of *invariants* and *rules*, and has been used, for instance, in the ORION [4] and O_2 [12] systems. The second approach, which was proposed in [24], is based on the introduction of *axioms*. In the former approach, the invariants define the consistency of a schema, and definite rules must be followed to maintain the invariants satisfied after each schema change. Invariants and rules are strictly dependent on the underlying object model, as they refer to specific model elements. In the latter approach, a sound and complete set of axioms (provided with an inference mechanism) formalizes the *dynamic schema evolution*, which is the actual management of schema changes in a system in operation. The approach is general enough to capture the behaviour of several different systems and, thus, is useful for their comparison in a unified framework. The compliance of the available primitive schema changes with the axioms automatically ensures schema consistency, without need for explicit checking, as incorrect schema versions cannot actually be generated.

For the “change propagation” problem, several solutions have been proposed and implemented in real systems, which can be ascribed to four main approaches:

1. *Immediate conversion (coercion)*: changes are propagated via immediate object conversion – used for instance in GemStone [23];
2. *Deferred conversion (lazy updates, screening)*: changes are propagated via deferred object conversion – used for instance in ORION [4];
3. *Filtering*: changes are never propagated: objects are assigned to different schema versions according to their semantics indeed – used for instance in CLOSQL [21];
4. *Hybrid*: uses or combines two or more of the previous approaches – used for instance in Sherpa [22] and O_2 [12].

In any case, simple *default* mechanisms can be used or user-supplied conversion functions must be defined for non-trivial extant object updates.

As far as complex schema changes are concerned, [20] considered sequences of schema change primitives to make up high-level useful changes, solving the propagation to objects problem with simple schema integration techniques. However, with this approach, the consistency of the resulting database is not guaranteed nor checked. In [6], high-level primitives are defined as *well-ordered* sets of primitive schema changes. Consistency of the resulting schema is ensured by the use of invariants’ preserving elementary steps and by *ad-hoc* constraints imposed on their application order. In other words, consistency preservation is dependent on an accurate design of high-level schema changes and, thus, still relies on the database designer/administrator’s skills.

This paper deals with the “semantics of change” problem, and tries to give a general answer to the problem of deciding the consistency-preserving property of any given sequence of elementary schema changes. The semantic and computational foundation of a general framework are laid down, which allows for a natural extension to consider also the “change propagation” problem (see Section 3.2).

3 An Object-Oriented Data Model for Evolving Schemata

In this Section we will define a general object-oriented model for evolving schemata which supports the taxonomy usually adopted for schema changes. To this end, we will first formally introduce the syntax and the semantics for the schema (version) and for the supported schema changes, and then formulate some interesting reasoning problems. To the best of our knowledge, this is the first attempt to formally define the semantics of schema changes based on the semantics of the underlying data model for the single schemata.

3.1 Syntax and Semantics

The object-oriented model we propose allows for the representation of multiple schema versions. It is an expressive version of the “snapshot” – i.e., single-schema – object-oriented model introduced by [1] and further extended and elaborated in its relationships with description logics by [8, 9]; in this paper we borrow the notation from [8]. The language embodies the features of the static parts of UML/OMT and ODMG and, therefore, it does not take into account those aspects related to the definition of methods. At the end of section 4 suggestions will be given on how to extend even more the expressiveness of the data model, both at the level of the schema language for classes and types and at the level of the schema change language.

The definition of an evolving schema $\mathcal{S}$ is based on a set of class and attribute names ($\mathcal{C}_\mathcal{S}$ and $\mathcal{A}_\mathcal{S}$ respectively) and includes a partially ordered set of schema versions. The initial schema version of $\mathcal{S}$ contains a set of class definitions having one of the following forms:

$$
\begin{array}{lr}
(S1) & \underline{\text{Class}}\ C\ \underline{\text{is-a}}\ C_1, \ldots ,C_h\ \underline{\text{disjoint}}\ C_{h+1}, \ldots ,C_k\ \underline{\text{type-is}}\ T. \\
 & \underline{\text{View-class}}\ C\ \underline{\text{is-a}}\ C_1, \ldots ,C_h\ \underline{\text{disjoint}}\ C_{h+1}, \ldots ,C_k\ \underline{\text{type-is}}\ T.
\end{array}
$$

A class definition introduces just necessary conditions regarding the type of the class – this is the standard case in object-oriented data models – while views are defined by means of both necessary and sufficient conditions. The symbol T denotes a type expression built according to the following syntax:

$$
\begin{array}{lll}
(S2) & T \rightarrow C\ \mid & \\
 & \quad \underline{\text{Union}}\ T_1, \ldots ,T_k\ \underline{\text{End}}\ \mid & \text{(union type)} \\
 & \quad \underline{\text{Set-of}}\ [\text{m,n}]\ T\ \mid & \text{(set type)} \\
 & \quad \underline{\text{Record}}\ A_1{:}T_1, \ldots ,A_k{:}T_k\ \underline{\text{End}}\ . & \text{(record type)}
\end{array}
$$

where $C \in \mathcal{C}_\mathcal{S}$, $A_i \in \mathcal{A}_\mathcal{S}$, and [m,n] denotes an optional cardinality constraint.

A schema version in $\mathcal{S}$ is defined by the application of a sequence of schema changes to a preceding schema version. The schema change taxonomy is built by combining the model elements which are subject to change with the elementary modifications, add, drop and change, they undergo. In this paper only a basic set of elementary schema change operators will be introduced; it includes the standard ones found in the literature (e.g., [4]); however, it is not difficult to consider the complete set of operators with respect to the constructs of the data model.

$$
\begin{array}{ll}
(S3) & M \rightarrow \underline{\text{Add-attribute}}\ C, A, T\ \underline{\text{End}}\ \mid \\
 & \quad\ \underline{\text{Drop-attribute}}\ C, A\ \underline{\text{End}}\ \mid \\
 & \quad\ \underline{\text{Change-attr-name}}\ C, A, A'\ \underline{\text{End}}\ \mid \\
 & \quad\ \underline{\text{Change-attr-type}}\ C, A, T'\ \underline{\text{End}}\ \mid \\
 & \quad\ \underline{\text{Add-class}}\ C, T\ \underline{\text{End}}\ \mid \\
 & \quad\ \underline{\text{Drop-class}}\ C\ \underline{\text{End}}\ \mid \\
 & \quad\ \underline{\text{Change-class-name}}\ C, C'\ \underline{\text{End}}\ \mid \\
 & \quad\ \underline{\text{Change-class-type}}\ C, T'\ \underline{\text{End}}\ \mid \\
 & \quad\ \underline{\text{Add-is-a}}\ C, C'\ \underline{\text{End}}\ \mid \\
 & \quad\ \underline{\text{Drop-is-a}}\ C, C'\ \underline{\text{End}}\ .
\end{array}
$$

In a framework supporting schema versioning, a mechanism for defining version coordinates is required. Such coordinates will be used to reference distinct schema versions which can then be employed as interfaces for querying extant data or modified by means of schema changes. We require that different schema versions have different version coordinates. At present, we omit the definition of a schema version coordinate mechanism and simply reference distinct schema versions by means of different subscripts.

Definition 1. *An evolving object-oriented schema is a tuple $\mathcal{S} = (\mathcal{C_S}, \mathcal{A_S}, \mathcal{SV}_0, \mathcal{M_S})$, where:*

- *$\mathcal{C_S}$ is a finite set of class names;*
- *$\mathcal{A_S}$ is a finite set of attribute names;*
- *$\mathcal{SV}_0$ is the initial schema version, which includes class definitions of the form specified in (S1) for some $C \in \mathcal{C_S}$;*
- *$\mathcal{M_S}$ is a set of modifications $\mathcal{M}_{ij}$, where i, j denote a pair of version coordinates. Each modification is a finite sequence of schema changes of the form specified in (S3).*

The set $\mathcal{M_S}$ induces a partial order $\mathcal{SV}$ over a finite and discrete set of schema versions with minimal element $\mathcal{SV}_0$. Hence $\mathcal{SV}_0$ precedes every other schema version and the schema version $\mathcal{SV}_j$ represents the outcome of the application of $\mathcal{M}_{ij}$ to $\mathcal{SV}_i$. $\mathcal{S}$ is called *elementary* if every $\mathcal{M}_{ij}$ in $\mathcal{M_S}$ contains only one elementary modification, and every schema version $\mathcal{SV}_i$ has at most one immediate predecessor. Without loss of generality, in the following we will consider only elementary evolving schemata.

Let us now introduce the meaning of an evolving object-oriented schema $\mathcal{S}$. Informally, the semantics is given by assigning to each schema version a possible legal database state – i.e., a legal instance of the schema version – conforming to the constraints imposed by the sequence of schema changes starting from the initial schema version.

Formally, an instance $\mathcal{I}$ of $\mathcal{S}$ is a tuple $\mathcal{I} =(\mathcal{O}^\mathcal{I}, \rho^\mathcal{I}, (\mathcal{I}_0, \ldots ,\mathcal{I}_n))$, consisting of a finite set $\mathcal{O}^\mathcal{I}$ of object identifiers, a function $\rho^\mathcal{I} : \mathcal{O}^\mathcal{I} \mapsto \mathcal{V}_{\mathcal{O}^\mathcal{I}}$ giving a value to object identifiers, and a sequence of version instances $\mathcal{I}_i$, one for each schema version $\mathcal{SV}_i$ in $\mathcal{S}$. The set $\mathcal{V}_{\mathcal{O}^\mathcal{I}}$ of values is defined by induction as the smallest set including the union of $\mathcal{O}^\mathcal{I}$ with all possible "sets" of values and with all possible "records" of values. Although the set $\mathcal{V}_{\mathcal{O}^\mathcal{I}}$ is infinite, we consider for an instance $\mathcal{I}$ the finite set $\mathcal{V}_\mathcal{I}$ of *active values*, which is the subset of $\mathcal{V}_{\mathcal{O}^\mathcal{I}}$ formed by the union of $\mathcal{O}^\mathcal{I}$ and the set of values assigned by $\rho^\mathcal{I}$ ([8]).

A version instance $\mathcal{I}_i =(\pi^{\mathcal{I}_i}, \cdot^{\mathcal{I}_i})$ consists of a total function $\pi^{\mathcal{I}_i} : \mathcal{C_S} \mapsto 2^{\mathcal{O}^\mathcal{I}}$, giving the set of object identifiers in the extension of each class $C \in C_\mathcal{S}$ for that version, and of a function $\cdot^{\mathcal{I}_i}$ (the *interpretation* function) mapping type expressions to sets of values, such that the following is satisfied:

$$\begin{aligned}
C^{\mathcal{I}_i} &= \pi^{\mathcal{I}_i}(C)\\
(\underline{\text{Union}}\ T_1, \ldots ,T_k\ \underline{\text{End}})^{\mathcal{I}_i} &= T_1^{\mathcal{I}_i} \cup \ldots \cup T_k^{\mathcal{I}_i}\\
(\underline{\text{Set-of}}\ [\text{m,n}]\ T)^{\mathcal{I}_i} &= \{\{\!|\ v_1, \ldots , v_k\ |\!\}\mid m \leq k \leq n, v_j \in T^{\mathcal{I}_i},\\
&\quad\ \text{for } j \in \{1, \ldots , k\}\}\\
(\underline{\text{Record}}\ A_1{:}T_1, \ldots ,A_k{:}T_k\ \underline{\text{End}})^{\mathcal{I}_i} &= \{[\![A_1 : v_1, \ldots , A_k : v_k, \ldots , A_h : v_h]\!] \mid h \geq k,\\
&\quad\ v_j \in T_j^{\mathcal{I}_i}, \text{for } j \in \{1, \ldots , k\},\\
&\quad\ v_j \in \mathcal{V}_{\mathcal{O}^\mathcal{I}}, \text{for } j \in \{k + 1, \ldots , h\}\}
\end{aligned}$$

where an open semantics for records is adopted (called *-interpretation in [1]) in order to give the right semantics to inheritance. In a set constructor if the minimum or the maximum cardinalities are not explicitly specified, they are assumed to be zero and infinite, respectively.

A *legal* instance $\mathcal{I}$ of a schema $\mathcal{S}$ should satisfy the constraints imposed by the class definitions in the initial schema version and by the schema changes between schema versions.

Add-attribute $\mathbf{C}$, $\mathbf{A}$, $\mathbf{T}$ End	$\pi^{\mathcal{I}_j}(\mathbf{C}) = \pi^{\mathcal{I}_i}(\mathbf{C}) \cap \{o \in \mathcal{O}^{\mathcal{I}} \mid \rho^{\mathcal{I}}(o) = [\![\ldots, \mathbf{A}: v, \ldots]\!] \wedge v \in \mathbf{T}^{\mathcal{I}_j}\}$, $\pi^{\mathcal{I}_i}(D) = \pi^{\mathcal{I}_j}(D)$ for all $D \neq \mathbf{C}$
Drop-attribute $\mathbf{C}$, $\mathbf{A}$ End	$\pi^{\mathcal{I}_i}(\mathbf{C}) = \pi^{\mathcal{I}_j}(\mathbf{C}) \cap \{o \in \mathcal{O}^{\mathcal{I}} \mid \rho^{\mathcal{I}}(o) = [\![\ldots, \mathbf{A}: v, \ldots]\!]\}$, $\pi^{\mathcal{I}_i}(D) = \pi^{\mathcal{I}_j}(D)$ for all $D \neq \mathbf{C}$
Change-attr-name $\mathbf{C}$, $\mathbf{A}$, $\mathbf{A}'$ End	$\pi^{\mathcal{I}_i}(\mathbf{C}) \cap \{o \in \mathcal{O}^{\mathcal{I}} \mid \rho^{\mathcal{I}}(o) = [\![\ldots, \mathbf{A}: v, \ldots]\!]\} = \pi^{\mathcal{I}_j}(\mathbf{C}) \cap \{o \in \mathcal{O}^{\mathcal{I}} \mid \rho^{\mathcal{I}}(o) = [\![\ldots, \mathbf{A}': v, \ldots]\!]\}$, $\pi^{\mathcal{I}_i}(D) = \pi^{\mathcal{I}_j}(D)$ for all $D \neq \mathbf{C}$
Change-attr-type $\mathbf{C}$, $\mathbf{A}$, $\mathbf{T}'$ End	$\pi^{\mathcal{I}_i}(\mathbf{C}) \cap \{o \in \mathcal{O}^{\mathcal{I}} \mid \rho^{\mathcal{I}}(o) = [\![\ldots, \mathbf{A}: v, \ldots]\!] \wedge v \in \mathbf{T}'^{\mathcal{I}_j}\} = \pi^{\mathcal{I}_j}(\mathbf{C}) \cap \{o \in \mathcal{O}^{\mathcal{I}} \mid \rho^{\mathcal{I}}(o) = [\![\ldots, \mathbf{A}: v, \ldots]\!]\}$, $\pi^{\mathcal{I}_i}(D) = \pi^{\mathcal{I}_j}(D)$ for all $D \neq \mathbf{C}$
Add-class $\mathbf{C}$, $\mathbf{T}$ End	$\pi^{\mathcal{I}_i}(\mathbf{C}) = \emptyset$, $\rho^{\mathcal{I}}(\pi^{\mathcal{I}_i}(\mathbf{C})) \subseteq \mathbf{T}^{\mathcal{I}_j}$, $\pi^{\mathcal{I}_i}(D) = \pi^{\mathcal{I}_j}(D)$ for all $D \neq \mathbf{C}$
Drop-class $\mathbf{C}$ End	$\pi^{\mathcal{I}_j}(\mathbf{C}) = \emptyset$, $\pi^{\mathcal{I}_i}(D) = \pi^{\mathcal{I}_j}(D)$ for all $D \neq \mathbf{C}$
Change-class-name $\mathbf{C}$, $\mathbf{C}'$ End	$\pi^{\mathcal{I}_i}(\mathbf{C}) = \pi^{\mathcal{I}_j}(\mathbf{C}')$, $\pi^{\mathcal{I}_i}(D) = \pi^{\mathcal{I}_j}(D)$ for all $D \neq \mathbf{C}, \mathbf{C}'$
Change-class-type $\mathbf{C}$, $\mathbf{T}'$ End	$\pi^{\mathcal{I}_j}(\mathbf{C}) = \pi^{\mathcal{I}_i}(\mathbf{C}) \cap \{o \in \mathcal{O}^{\mathcal{I}} \mid \rho^{\mathcal{I}}(o) \in \mathbf{T}'^{\mathcal{I}_j}\}$, $\pi^{\mathcal{I}_i}(D) = \pi^{\mathcal{I}_j}(D)$ for all $D \neq \mathbf{C}$
Add-is-a $\mathbf{C}$, $\mathbf{C}'$ End	$\pi^{\mathcal{I}_j}(\mathbf{C}) = \pi^{\mathcal{I}_i}(\mathbf{C}) \cap \pi^{\mathcal{I}_i}(\mathbf{C}')$, $\pi^{\mathcal{I}_i}(D) = \pi^{\mathcal{I}_j}(D)$ for all $D \neq \mathbf{C}$
Drop-is-a $\mathbf{C}$, $\mathbf{C}'$ End	$\pi^{\mathcal{I}_i}(\mathbf{C}) = \pi^{\mathcal{I}_j}(\mathbf{C}) \cap \pi^{\mathcal{I}_j}(\mathbf{C}')$, $\pi^{\mathcal{I}_i}(D) = \pi^{\mathcal{I}_j}(D)$ for all $D \neq \mathbf{C}$

Table 1. Semantics of the schema changes.

Definition 2. *An instance $\mathcal{I}$ of a schema $\mathcal{S}$ is said to be legal if*

– *for each class definition Class C is-a $C_1, \ldots, C_h$ disjoint $C_{h+1}, \ldots, C_k$ type-is T in $\mathcal{SV}_0$ it holds that:*
  $C^{\mathcal{I}_0} \subseteq C_j^{\mathcal{I}_0}$ *for each* $j \in \{1, \ldots, h\}$,
  $C^{\mathcal{I}_0} \cap C_j^{\mathcal{I}_0} = \emptyset$ *for each* $j \in \{h+1, \ldots, k\}$,
  $\{\rho^{\mathcal{I}}(o) \mid o \in \pi^{\mathcal{I}_0}(C)\} \subseteq T^{\mathcal{I}_0}$;
– *for each view definition View-class C is-a $C_1, \ldots, C_h$ disjoint $C_{h+1}, \ldots, C_k$ type-is T in $\mathcal{SV}_0$ it holds that:*
  $C^{\mathcal{I}_0} \subseteq C_j^{\mathcal{I}_0}$ *for each* $j \in \{1, \ldots, h\}$,
  $C^{\mathcal{I}_0} \cap C_j^{\mathcal{I}_0} = \emptyset$ *for each* $j \in \{h+1, \ldots, k\}$,
  $\{\rho^{\mathcal{I}}(o) \mid o \in \pi^{\mathcal{I}_0}(C)\} = T^{\mathcal{I}_0}$;
– *for each schema change $\mathcal{M}_{ij}$ in $\mathcal{M}$, the version instances $\mathcal{I}_i$ and $\mathcal{I}_j$ satisfy the equations of the corresponding schema change type at the right hand side of Tab. 1.*

3.2 Reasoning Problems

According to the semantic definitions given in the previous section, several reasoning problems can be introduced, in order to support the design and the management of an evolving schema.

Definition 3. *Reasoning problems:*

a. *Global/local Schema Consistency: an evolving schema $\mathcal{S}$ is globally consistent if it admits a legal instance; a schema version $\mathcal{SV}_i$ of $\mathcal{S}$ is locally consistent if the evolving schema $\mathcal{S}_{\downarrow i}$– obtained from $\mathcal{S}$ by reducing the set of modifications $\mathcal{M}_{\mathcal{S}_{\downarrow i}}$ to the linear sequence of schema changes in $\mathcal{M}_{\mathcal{S}}$ which led to the version $\mathcal{SV}_i$ from $\mathcal{SV}_0$– admits a legal instance. In the following, a global reasoning problem refers to $\mathcal{S}$, while a local one refers to $\mathcal{S}_{\downarrow i}$.*

b. *Global/local Class Consistency: a class C is globally inconsistent if for every legal instance $\mathcal{I}$ of $\mathcal{S}$ and for every version $\mathcal{SV}_i$ its extension is empty, i.e., $\forall i.\ \pi^{\mathcal{I}_i}(C) = \emptyset$; a class C is locally inconsistent in the version $\mathcal{SV}_i$ if for every legal instance $\mathcal{I}$ of $\mathcal{S}_{\downarrow i}$ its extension is empty, i.e., $\pi^{\mathcal{I}_i}(C) = \emptyset$.*

c. *Global/local Disjoint Classes: two classes C, D are globally disjoint if for every legal instance $\mathcal{I}$ of $\mathcal{S}$ and for every version $\mathcal{SV}_i$ their extensions are disjoint, i.e., $\forall i.\ \pi^{\mathcal{I}_i}(C) \cap \pi^{\mathcal{I}_i}(D) = \emptyset$; two classes C, D are locally disjoint in the version $\mathcal{SV}_i$ if for every legal instance $\mathcal{I}$ of $\mathcal{S}_{\downarrow i}$ their extensions are disjoint, i.e., $\pi^{\mathcal{I}_i}(C) \cap \pi^{\mathcal{I}_i}(D) = \emptyset$.*

d. *Global/local Class Subsumption: a class D globally subsumes a class C if for every legal instance $\mathcal{I}$ of $\mathcal{S}$ and for every version $\mathcal{SV}_i$ the extension of C is included in the extension of D, i.e., $\forall i.\ \pi^{\mathcal{I}_i}(C) \subseteq \pi^{\mathcal{I}_i}(D)$; a class D locally subsumes a class C in the version $\mathcal{SV}_i$ if for every legal instance $\mathcal{I}$ of $\mathcal{S}_{\downarrow i}$ the extension of C is included in the extension of D, i.e., $\pi^{\mathcal{I}_i}(C) \subseteq \pi^{\mathcal{I}_i}(D)$.*

e. *Global/local Class Equivalence: two classes C, D are globally/locally equivalent if C globally/locally subsumes D and viceversa.*

Please note that the classical *subtyping* problem – i.e., finding the explicit representation of the partial order induced on a set of type expressions by the containment between their extensions – is a special case of class subsumption, if we restrict our attention to view definitions.

As to the *change propagation* task, which is one of the fundamental task addressed in the literature (see Sec. 2), it is usually dealt with by populating the classes in the new version with the result of queries over the previous version. The same applies for our framework: a language for the specification of views can be defined for specifying how to populate classes in a version from the previous data. Of course, at this point the problem of global consistency of an evolving schema $\mathcal{S}$ becomes more complex, since it involves the additional constraints defined by the data conversions: an instance would therefore be legal if it satisfies not only the constraints of Definition 2 but also the constraints specified by the views. Obviously, a schema $\mathcal{S}$ involving a schema change for which the corresponding semantics expressed by the equation in Tab. 1 and the associated data conversions are incompatible would never admit a legal instance. In general, the introduction of data conversion views makes all the reasoning problems defined above more complex. In this paper we do not deal with the change propagation task.

We will try to explain the application of the reasoning problems through an example. Let us consider an evolving schema $\mathcal{S}$ describing the employees of a company. The schema includes an initial schema version $\mathcal{SV}_0$ defined as follows:

```
Class Employee type-is Union Manager, Secretary, Worker End;
Class Manager is-a Employee disjoint Secretary, Worker ;
Class Secretary is-a Employee disjoint Worker ;
Class Worker is-a Employee;
View-class Senior type-is Record has_staff: Set-of [2,n] Worker End;
View-class Junior type-is Record has_staff: Set-of [0,1] Worker End;
```

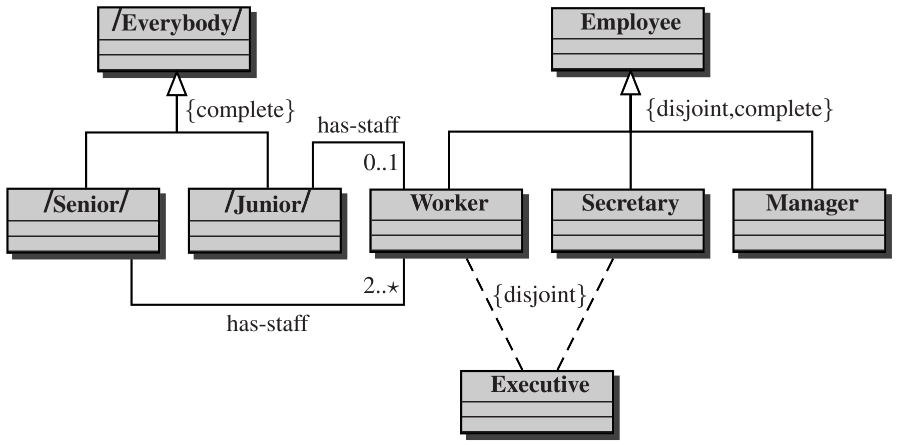

Fig. 1. The Employee initial schema version in UML notation.

Class `Executive` disjoint `Secretary`, `Worker`;
View-class `Everybody` type-is Union `Senior`, `Junior` End End;

Figure 1 shows the UML-like representation induced by the initial schema $\mathcal{SV}_0$; note that classes with names between slashes represent the views. The evolving schema $\mathcal{S}$ includes a set of schema modifications $\mathcal{M}_{\mathcal{S}}$ defined as follows:

($\mathcal{M}_{01}$)	Add-is-a `Secretary`, `Manager` End;
($\mathcal{M}_{02}$)	Add-is-a `Everybody`, `Manager` End;
($\mathcal{M}_{23}$)	Add-is-a `Everybody`, `Secretary` End;
($\mathcal{M}_{04}$)	Add-is-a `Executive`, `Employee` End;
($\mathcal{M}_{45}$)	Add-attribute `Manager`, `IdNum`, `Number` End;
($\mathcal{M}_{56}$)	Change-attr-type `Manager`, `IdNum`, `Integer` End;
($\mathcal{M}_{67}$)	Change-attr-type `Manager`, `IdNum`, `String` End;
($\mathcal{M}_{68}$)	Drop-class `Employee` End;

Let us analyse the effect of each schema change $\mathcal{M}_{ij}$ by considering the schema version $\mathcal{SV}_j$ it produces.

First of all, it can be noticed that in $\mathcal{SV}_0$ the `Junior` and `Senior` classes are disjoint classes and that `Everybody` contains all the possible instances of the record type. In fact, `Everybody` is defined as the union of view classes which are complementary with respect to the record type: any possible record instance is the value of an object belonging either to `Senior` or `Junior`.

`Secretary` is inconsistent in $\mathcal{SV}_1$ since `Secretary` and `Manager` are disjoint: its extension is included in the `Manager` extension only if it is empty (for each version instance $\mathcal{I}_1$, $\texttt{Secretary}^{\mathcal{I}_1} = \emptyset$). Therefore, `Secretary` is *locally inconsistent*, as it is inconsistent in $\mathcal{SV}_1$ but not in $\mathcal{SV}_0$.

The schema version $\mathcal{SV}_3$ is inconsistent because `Secretary` and `Manager`, which are both superclasses of `Everybody`, are disjoint and the intersection of their extensions is empty: no version instance $\mathcal{I}_3$ exists such that $\texttt{Everybody}^{\mathcal{I}_3} \subseteq \emptyset$. It follows that $\mathcal{S}$ is locally inconsistent with respect to $\mathcal{SV}_3$ and, thus, globally inconsistent (although is locally consistent wrt the other schema versions).

In $\mathcal{SV}_4$, it can be derived that `Executive` is locally subsumed by `Manager`, since it is a subclass of `Employee` disjoint from `Secretary` and `Worker` (`Manager`, `Secretary` and `Worker` are a partition of `Employee`).

The schema version $\mathcal{SV}_5$ exemplifies a case of attribute inheritance. The attribute `IdNum` which has been added to the `Manager` class is inherited by the `Executive` class. This means that *every* legal instance of $\mathcal{S}$ should be such that every instance of `Executive` in $\mathcal{SV}_5$ has an attribute `IdNum` of type `Number`, i.e., $\texttt{Executive}^{\mathcal{I}_5} \subseteq \{o \mid \rho^{\mathcal{I}}(o) = [\![\ldots, \texttt{IdNum} : v, \ldots]\!] \wedge v \in \texttt{Number}^{\mathcal{I}_5}\}$. Of course, there is no restriction on the way classes are related via subsumption, and multiple inheritance is allowed as soon as it does not generate an inconsistency.

The Change-attr-type elementary schema change allows for the modification of the type of an attribute with the proviso that the new type is not incompatible with the old one, like in $\mathcal{M}_{56}$. In fact, the semantics of elementary schema changes as defined in Tab. 1 always refer to an *evolution* of the objects through the various versions; the only elementary change that can refer to *new* objects is Add-class. Notice that, for this reason, $\mathcal{M}_{67}$ leads to an inconsistent version if `Number` and `String` are defined to be non-empty disjoint classes. Thus, in order to specify a schema change involving a restructuring of the data and the creation of new objects – like in the case of the change of the type of an attribute with an incompatible new type – a sequence of Drop-class and Add-class should be specified, together with a data conversion view specifying how the data is converted from one version to the other.

The deletion of the class `Employee` in $\mathcal{SV}_8$ does not cause any inconsistency in the resulting schema version. In $\mathcal{SV}_8$ the `Employee` extension is empty and the former `Employee` subclasses continue to exist (with the constraint that their extensions are subsets of the extension of `Employee` in $\mathcal{SV}_6$). Notice that, in a classical object model where the class hierarchy is explicitly based on a DAG, the deletion of a non-isolated class would require a restructuring of the DAG itself (e.g. to get rid of dangling edges).

4 Reasoning with an Evolving Schema

In this section we establish a relationship between the proposed model for evolving schemata and the $\mathcal{ALCQI}$ description logic. To this end, we provide an encoding from an evolving schema into an $\mathcal{ALCQI}$ knowledge base Σ, such that the reasoning problems mentioned in the previous section can be reduced to corresponding description logics reasoning problems, for which extensive theories and well founded and efficient implemented systems exist. The encoding is grounded on the fact that there is a correspondence between the models of the knowledge base and the legal instances of the evolving schema.

We give here only a very brief introduction to the $\mathcal{ALCQI}$ description logic; for a full account, see, e.g., [7]. The basic types of a description logic are *concepts* and *roles*. A concept is a description gathering the common properties among a collection of individuals; from a logical point of view it is a unary predicate ranging over the domain of individuals. Inter-relationships between these individuals are represented by means of roles (which are interpreted as binary relations over the domain of individuals). The syntax rules at the left hand side of Figure 2 define valid concept and role expressions. Concepts are interpreted as sets of individuals—as for unary predicates—and roles as sets of pairs of individuals—as for binary predicates. Formally, an *interpretation* is a pair $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consisting of a set $\Delta^{\mathcal{I}}$ of individuals (the *domain* of $\mathcal{I}$) and a function $\cdot^{\mathcal{I}}$ (the *interpretation function* of $\mathcal{I}$) mapping every concept to a subset of $\Delta^{\mathcal{I}}$ and every role to a subset of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$, such that the equations at the right hand side of Figure 2 are satisfied.

$$
\begin{array}{lll}
C, D \rightarrow & A \mid & \\
& \top \mid & \top^{\mathcal{I}} = \Delta^{\mathcal{I}} \\
& \bot \mid & \bot^{\mathcal{I}} = \emptyset \\
& \neg C \mid & (\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}} \\
& C \sqcap D \mid & (C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}} \\
& C \sqcup D \mid & (C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}} \\
& \forall R.C \mid & (\forall R.C)^{\mathcal{I}} = \{i \in \Delta^{\mathcal{I}} \mid \forall j.\ R^{\mathcal{I}}(i,j) \Rightarrow C^{\mathcal{I}}(j)\} \\
& \exists R.C \mid & (\exists R.C)^{\mathcal{I}} = \{i \in \Delta^{\mathcal{I}} \mid \exists j.\ R^{\mathcal{I}}(i,j) \wedge C^{\mathcal{I}}(j)\} \\
& {\geq} nR.C \mid & ({\geq} nR.C)^{\mathcal{I}} = \{i \in \Delta^{\mathcal{I}} \mid \sharp\{j \in \Delta^{\mathcal{I}} \mid R^{\mathcal{I}}(i,j) \wedge C^{\mathcal{I}}(j)\} \geq n\} \\
& {\leq} nR.C & ({\leq} nR.C)^{\mathcal{I}} = \{i \in \Delta^{\mathcal{I}} \mid \sharp\{j \in \Delta^{\mathcal{I}} \mid R^{\mathcal{I}}(i,j) \wedge C^{\mathcal{I}}(j)\} \leq n\} \\
\\
R, S \rightarrow & P \mid & \\
& R^{-} & (R^{-})^{\mathcal{I}} = \{(i,j) \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \mid R^{\mathcal{I}}(j,i)\}
\end{array}
$$

Fig. 2. $\mathcal{ALCQI}$ concept and role expressions and their semantics.

A *knowledge base* is a finite set Σ of axioms of the form $C \dot{\sqsubseteq} D$, involving concept expressions C, D; we write $C \equiv D$ as a shortcut for both $C \dot{\sqsubseteq} D$ and $D \dot{\sqsubseteq} C$. An interpretation $\mathcal{I}$ satisfies $C \dot{\sqsubseteq} D$ if and only if the interpretation of C is included in the interpretation of D, i.e., $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$; it is said that C is subsumed by D. An interpretation $\mathcal{I}$ is a *model* of a knowledge base Σ iff every axiom of Σ is satisfied by $\mathcal{I}$. If Σ has a model, then it is *satisfiable*; thus, checking for KB satisfiability is deciding whether there is at least one model for the knowledge base. Σ *logically implies* an axiom $C \dot{\sqsubseteq} D$ (written $\Sigma \models C \dot{\sqsubseteq} D$) if $C \dot{\sqsubseteq} D$ is satisfied by every model of Σ. Reasoning in $\mathcal{ALCQI}$ (i.e., deciding knowledge base satisfiability and logical implication) is decidable, and it has been proven to be an EXPTIME-complete problem [7].

As in [8], the encoding of an object-oriented schema in an $\mathcal{ALCQI}$ knowledge base is based on the *reification* of type expressions – i.e., explicit individuals exist to denote values of complex types. We introduce the concept AbstractClass to represent the classes, the concepts RecType, SetType to represent types, the role value to model the association between classes and types, and the role member to specify the type of the elements of a set. In particular, a record is represented as an individual connected by means of (functional) roles – corresponding to attributes – to the fillers of its attributes. The mapping function ψ_i translates type expressions into $\mathcal{ALCQI}$ concepts as follows:

$$
\begin{aligned}
\psi_i(\mathrm{C}) &= C_i \\
\psi_i(\underline{\mathrm{Union}}\ T_1, \ldots, T_k\ \underline{\mathrm{End}}) &= \psi_i(T_1) \sqcup \ldots \sqcup \psi_i(T_k) \\
\psi_i(\underline{\mathrm{Set\text{-}of}}\ [\mathrm{m,n}]\ T) &= \mathsf{SetType} \sqcap \forall \mathsf{member}.\psi_i(T) \sqcap \\
&\quad {\geq} m\mathsf{member}.\top \sqcap {\leq} n\mathsf{member}.\top \\
\psi_i(\underline{\mathrm{Record}}\ A_1{:}T_1, \ldots, A_k{:}T_k\ \underline{\mathrm{End}}) &= \mathsf{RecType} \sqcap \exists A_1.\psi_i(T_1) \sqcap \ldots \sqcap \exists A_k.\psi_i(T_k)
\end{aligned}
$$

The translation function ψ_i is contextualised to the ith schema version, since a class in different schema version may have different extensions, and it is mapped into distinct concepts.

Definition 4. *The $\mathcal{ALCQI}$ knowledge base $\Sigma = \psi(\mathcal{S})$ corresponding to the object-oriented evolving schema $\mathcal{S} = (\mathcal{C}_{\mathcal{S}}, \mathcal{A}_{\mathcal{S}}, \mathcal{SV}_0, \mathcal{M}_{\mathcal{S}})$ is composed by the following axioms:*

– *Axioms on basic types:*

$$
\begin{aligned}
\mathsf{AbstractClass} &\sqsubseteq \exists \mathsf{value}.\top \sqcap {\leq} 1 \mathsf{value}.\top \\
\mathsf{RecType} &\sqsubseteq \forall \mathsf{value}.\bot \\
\mathsf{SetType} &\sqsubseteq \forall \mathsf{value}.\bot \sqcap \neg \mathsf{RecType}
\end{aligned}
$$

Add-attribute $\mathbf{C}$, $\mathbf{A}$, $\mathbf{T}$ End	$\psi_j(\mathbf{C}) \equiv \psi_i(\mathbf{C}) \sqcap \forall \mathsf{value}.(\mathsf{RecType} \sqcap \exists \mathbf{A}.\psi_j(\mathbf{T}))$, $\psi_i(D) \equiv \psi_j(D)$ for all $D \neq \mathbf{C}$
Drop-attribute $\mathbf{C}$, $\mathbf{A}$ End	$\psi_i(\mathbf{C}) \equiv \psi_j(\mathbf{C}) \sqcap \forall \mathsf{value}.(\mathsf{RecType} \sqcap \exists \mathbf{A}.\top)$, $\psi_i(D) \equiv \psi_j(D)$ for all $D \neq \mathbf{C}$
Change-attr-name $\mathbf{C}$, $\mathbf{A}$, $\mathbf{A}'$ End	$\psi_i(\mathbf{C}) \sqcap \forall \mathsf{value}.(\mathsf{RecType} \sqcap \exists \mathbf{A}.\top) \equiv \psi_j(\mathbf{C}) \sqcap \forall \mathsf{value}.(\mathsf{RecType} \sqcap \exists \mathbf{A}'.\top)$, $\psi_i(D) \equiv \psi_j(D)$ for all $D \neq \mathbf{C}$
Change-attr-type $\mathbf{C}$, $\mathbf{A}$, $\mathbf{T}'$ End	$\psi_i(\mathbf{C}) \sqcap \forall \mathsf{value}.(\mathsf{RecType} \sqcap \exists \mathbf{A}.\psi_j(\mathbf{T}')) \equiv \psi_j(\mathbf{C}) \sqcap \forall \mathsf{value}.(\mathsf{RecType} \sqcap \exists \mathbf{A}.\top)$, $\psi_i(D) \equiv \psi_j(D)$ for all $D \neq \mathbf{C}$
Add-class $\mathbf{C}$, $\mathbf{T}$ End	$\psi_i(\mathbf{C}) \equiv \bot$, $\psi_j(\mathbf{C}) \sqsubseteq \mathsf{AbstractClass} \sqcap \forall \mathsf{value}.\psi_j(\mathbf{T})$, $\psi_i(D) \equiv \psi_j(D)$ for all $D \neq \mathbf{C}$
Drop-class $\mathbf{C}$ End	$\psi_j(\mathbf{C}) \equiv \bot$, $\psi_i(D) \equiv \psi_j(D)$ for all $D \neq \mathbf{C}$
Change-class-name $\mathbf{C}$, $\mathbf{C}'$ End	$\psi_i(\mathbf{C}) \equiv \psi_j(\mathbf{C}')$, $\psi_i(D) \equiv \psi_j(D)$ for all $D \neq \mathbf{C}, \mathbf{C}'$
Change-class-type $\mathbf{C}$, $\mathbf{T}'$ End	$\psi_j(\mathbf{C}) \equiv \psi_i(\mathbf{C}) \sqcap \forall \mathsf{value}.\psi_j(\mathbf{T}')$, $\psi_i(D) \equiv \psi_j(D)$ for all $D \neq \mathbf{C}$
Add-is-a $\mathbf{C}$, $\mathbf{C}'$ End	$\psi_j(\mathbf{C}) \equiv \psi_i(\mathbf{C}) \sqcap \psi_i(\mathbf{C}')$, $\psi_i(D) \equiv \psi_j(D)$ for all $D \neq \mathbf{C}$
Drop-is-a $\mathbf{C}$, $\mathbf{C}'$ End	$\psi_i(\mathbf{C}) \equiv \psi_j(\mathbf{C}) \sqcap \psi_j(\mathbf{C}')$, $\psi_i(D) \equiv \psi_j(D)$ for all $D \neq \mathbf{C}$

Table 2. The axioms induced by the schema changes.

– *For each class definition Class C is-a $C_1, \ldots, C_h$ disjoint $C_{h+1}, \ldots, C_k$ type-is T in $\mathcal{SV}_0$:*
  $\psi_0(C) \sqsubseteq \mathsf{AbstractClass} \sqcap \psi_0(C_1) \sqcap \ldots \sqcap \psi_0(C_h) \sqcap \forall \mathsf{value}.\psi_0(T)$
  $\psi_0(C) \sqsubseteq \neg\psi_0(C_{h+1}) \sqcap \ldots \sqcap \neg\psi_0(C_k)$
– *For each view definition View-class C is-a $C_1, \ldots, C_h$ disjoint $C_{h+1}, \ldots, C_k$ type-is T in $\mathcal{SV}_0$:*
  $\psi_0(C) \sqsubseteq \mathsf{AbstractClass} \sqcap \psi_0(C_1) \sqcap \ldots \sqcap \psi_0(C_h)$
  $\psi_0(C) \sqsubseteq \neg\psi_0(C_{h+1}) \sqcap \ldots \sqcap \neg\psi_0(C_k)$
  $\psi_0(C) \equiv \forall \mathsf{value}.\psi_0(T)$
– *For each attribute in $\mathcal{A}_{\mathcal{S}}$:*
  $\exists A_i.\top \sqsubseteq \leq 1 A_i.\top$
– *For each schema modification $\mathcal{M}_{ij} \in \mathcal{M}_{\mathcal{S}}$ a corresponding axiom from Tab. 2.*

Based on the results of [9], we prove that the translation is correct, in the following sense. A precise correspondence between legal instances of $\mathcal{S}$ and models of the derived knowledge base Σ exists. The semantic correspondence is exploited to devise a correspondence between reasoning problems at the level of evolving schemata and reasoning problems at the level of the description logic.

Theorem 1. *A schema $\mathcal{S}$ is globally consistent – i.e., it admits a of legal instance – if and only if the corresponding $\mathcal{ALCQI}$ knowledge base Σ is satisfiable.*

In order to prove this theorem, we need to prove a lemma stating the correspondence between legal instances of an evolving schema and description logics models. As already pointed out in [9], the $\mathcal{ALCQI}$ knowledge base $\psi(\mathcal{S})$ resulting from the translation of an object-oriented schema $\mathcal{S}$ may admit models that do not have a counterpart among legal instances of $\mathcal{S}$. In particular, in a model of $\psi(\mathcal{S})$ there may be *bad* cycles involving only instances of SetType and RecType which can not be put in correspondence with a legal instance. Since the set of possible active values associated with each object identifier is bound by the depth of the schema, bad cycles can be unfolded obtaining individuals having a finite representation of the same depth of the schema. Therefore, given a schema $\mathcal{S}$ and its translation $\psi(\mathcal{S})$, the *m-unfolded version* $\mathcal{I}_{|m}$ of a finite interpretation $\mathcal{I}$ of $\psi(\mathcal{S})$ corresponds to the interpretation obtained by truncating at depth m each infinite representation generated in the process of unfolding.

Lemma 1. *For each object-oriented schema $\mathcal{S}$ of depth m, the following mappings exist:*

1. *$\alpha_{\mathcal{S}}$ from instances of $\mathcal{S}$ into finite interpretations of $\psi(\mathcal{S})$ and $\alpha_{\mathcal{V}}$ from active values of instances of $\mathcal{S}$ into domain elements of the finite interpretation of $\psi(\mathcal{S})$ such that:*
   For each legal instance $\mathcal{I}$ of $\mathcal{S}$, $\alpha_{\mathcal{S}}(\mathcal{I})$ is a finite model of $\psi(\mathcal{S})$, and for each type expression T of $\mathcal{S}$ and each $v \in \mathcal{V}_{\mathcal{I}}$, $v \in T^{\mathcal{I}_i}$ iff $\alpha_{\mathcal{V}}(v) \in (\psi_i(T))^{\alpha_{\mathcal{S}}(\mathcal{I})}$, where $\mathcal{I}_i$ denotes the version instance of the schema version $\mathcal{SV}_i \in \mathcal{SV}$.
2. *$\beta_{\mathcal{S}}$ from finite interpretations of $\psi(\mathcal{S})$ into instances of $\mathcal{S}$, $\beta_{\mathcal{SV}i}$ from finite interpretations of $\psi(\mathcal{S})$ into instances of $\mathcal{SV}_i \in \mathcal{SV}$ and $\beta_{\mathcal{V}}$ from domain elements of the m-unfolded versions of the finite interpretations of $\psi(\mathcal{S})$ into active values of instances of $\mathcal{S}$, such that:*
   For each finite model $\mathcal{I}$ of $\psi(\mathcal{S})$, $\beta_{\mathcal{S}}(\mathcal{I})$ is a legal instance of $\mathcal{S}$, and for each concept $\psi_i(T)$, which is the translation of a type expression T of $\mathcal{S}$ with respect to the schema version $\mathcal{SV}_i$, and each $d \in (\Delta)^{\mathcal{I}_{|m}}$, $d \in (\psi_i(T))^{\mathcal{I}_{|m}}$ if and only if $\beta_{\mathcal{V}}(d) \in T^{\beta_{\mathcal{SV}i}(\mathcal{I})}$.

Due to space limitations, the proof is only sketched in the following (it can be found in complete form in [13]).

Proof. (1) Given a legal instance $\mathcal{I}$ we define an interpretation $\alpha_{\mathcal{S}}(\mathcal{I}) = ((\Delta)^{\alpha_{\mathcal{S}}(\mathcal{I})}, (\cdot)^{\alpha_{\mathcal{S}}(\mathcal{I})})$ as follows:

– $\Delta^{\alpha_{\mathcal{S}}(\mathcal{I})} = \{\alpha_{\mathcal{V}}(v) \mid v \in \mathcal{V}_{\mathcal{I}}\}$ where $\alpha_{\mathcal{V}} : \mathcal{V}_{\mathcal{I}} \rightarrow \Delta^{\alpha_{\mathcal{S}}(\mathcal{I})}$ maps every $\mathcal{V}_{\mathcal{I}}$ element into a distinct element of $\Delta^{\alpha_{\mathcal{S}}(\mathcal{I})}$. We denote with Δ_{id}, Δ_{rec} and Δ_{set} the elements of $(\Delta)^{\alpha_{\mathcal{S}}(\mathcal{I})}$ corresponding to object identifiers, record and set values, respectively.
– The interpretation of atomic concepts is defined as follows:
$(\psi_i(C))^{\alpha_{\mathcal{S}}(\mathcal{I})} = \{\alpha_{\mathcal{V}}(o) \mid o \in \pi^{\mathcal{I}_i}(C)\}$, for every $\psi_i(C)$ corresponding to $C \in \mathcal{C}_{\mathcal{S}}$ where $\mathcal{SV}_i \in \mathcal{SV}$, $(\mathsf{AbstractClass})^{\alpha_{\mathcal{S}}(\mathcal{I})} = \Delta_{id}$, $(\mathsf{RecType})^{\alpha_{\mathcal{S}}(\mathcal{I})} = \Delta_{rec}$, $(\mathsf{SetType})^{\alpha_{\mathcal{S}}(\mathcal{I})} = \Delta_{set}$.
– The interpretation of atomic roles is defined as follows:
$(A)^{\alpha_{\mathcal{S}}(\mathcal{I})} = \{(d_1, d_2) \mid d_1 \in \Delta_{rec}$ and $\alpha_{\mathcal{V}}^{-}(d_1) = [\![\ldots, A : \alpha_{\mathcal{V}}^{-}(d_2), \ldots]\!]\}$, for every A corresponding to an attribute name $A \in \mathcal{A}_{\mathcal{S}}$, $(\mathsf{member})^{\alpha_{\mathcal{S}}(\mathcal{I})} = \{(d_1, d_2) \mid d_1 \in \Delta_{set}$ and $\alpha_{\mathcal{V}}^{-}(d_1) = \{\ldots, A : \alpha_{\mathcal{V}}^{-}(d_2), \ldots\}\}$, $(\mathsf{value})^{\alpha_{\mathcal{S}}(\mathcal{I})} = \{(d_1, d_2) \mid (\alpha_{\mathcal{V}}^{-}(d_1), \alpha_{\mathcal{V}}^{-}(d_2)) \in \rho^{\mathcal{I}}\}$

Then, we can prove by induction on their structure that for each type expression T of $\mathcal{S}$ and each $v \in \mathcal{V}_{\mathcal{I}}$, $v \in T^{\mathcal{I}_i}$ iff $\alpha_{\mathcal{V}}(v) \in (\psi_i(T))^{\alpha_{\mathcal{S}}(\mathcal{I})}$. From the second part of the thesis and since $\mathcal{I}$ is legal, we can also prove that $\alpha_{\mathcal{S}}(\mathcal{I})$ is a finite model of $\psi(\mathcal{S})$.

(2) Given a finite model $\mathcal{I}$ of $\psi(\mathcal{S})$ of depth m, we define a legal instance $\beta_{\mathcal{S}}(\mathcal{I})$ as follows:

- $\beta_{\mathcal{V}} : (\Delta)^{\mathcal{I}|m} \to \mathcal{V}_{\beta_{\mathcal{S}}(\mathcal{I})}$ maps every $(\Delta)^{\mathcal{I}|m}$ element into a distinct element of $\mathcal{V}_{\beta_{\mathcal{S}}(\mathcal{I})}$ such that the following conditions are satisfied:
  - $\mathcal{O}^{\beta_{\mathcal{S}}(\mathcal{I})} \subseteq \mathcal{V}_{\beta_{\mathcal{S}}(\mathcal{I})}$ is the set $\{\beta_{\mathcal{V}}(d) \mid d \in (\mathsf{AbstractClass})^{\mathcal{I}|m}\}$;
  - if $d \in (\mathsf{RecType})^{\mathcal{I}|m}$, $(d, d_i) \in (A_i)^{\mathcal{I}|m}$ for $i \in \{1, \dots, k\}$ where k is the greatest arity, then $\beta_{\mathcal{V}}(d) = [\![A_1 : \beta_{\mathcal{V}}(d_1), \dots, A_k : \beta_{\mathcal{V}}(d_k)]\!]$;
  - if $d \in (\mathsf{SetType})^{\mathcal{I}|m}$, $(d, d_i) \in (\mathsf{member})^{\mathcal{I}|m}$ for $i \in \{1, \dots, k\}$ where k is the greatest arity, then $\beta_{\mathcal{V}}(d) = \{\beta_{\mathcal{V}}(d_1), \dots, \beta_{\mathcal{V}}(d_k)\}$;
- for each $C \in \mathcal{C}_{\mathcal{S}}$, for each $\mathcal{SV}_i \in \mathcal{SV}$, $\pi^{\beta_{\mathcal{SV}_i}(\mathcal{I})}(C) = \{\beta_{\mathcal{V}}(d) \mid d \in (\psi_i(C))^{\mathcal{I}|m}\}$;
- $\rho^{\beta_{\mathcal{S}}(\mathcal{I})} = \{(o, v) \mid \beta_{\mathcal{V}}(d_1) = o, \beta_{\mathcal{V}}(d_2) = v, (d_1, d_2) \in (\mathsf{value})^{\mathcal{I}|m}\}$.

Then, we can prove by induction on the structure of the concept expression that for each concept $\psi_i(T)$, which is the translation of a type expression T of $\mathcal{S}$ with respect to the schema version $\mathcal{SV}_i$, and each $d \in (\Delta)^{\mathcal{I}|m}$, $d \in (\psi_i(T))^{\mathcal{I}|m}$ if and only if $\beta_{\mathcal{V}}(d) \in T^{\beta_{\mathcal{SV}_i}(\mathcal{I})}$. From the second part of the thesis and since $\mathcal{I}$ is a finite model of depth m of $\psi(\mathcal{S})$ we can also prove that $\beta_{\mathcal{S}}(\mathcal{I})$ is a legal instance of $\mathcal{S}$. □

Using standard techniques in description logics, it is possible to reduce the reasoning problems at the level of the evolving schema (as defined in Definition 3) to knowledge base satisfiability problems in the description logic.

Corollary 1. *Given $\mathcal{S}$, the reasoning problems defined in Definition 3 can be polynomially reduced to knowledge base satisfiability problems on $\psi(\mathcal{S})$. The complexity of the above reasoning problems in $\mathcal{S}$ is thus in EXPTIME.*

Please note that this decidability result does not contrast with the undecidability results with functional and inclusion dependencies, since only unary dependencies are involved in this formalism. Moreover, the worst case complexity in EXPTIME does not imply bad practical computational behaviour in the real cases: in fact, a preliminary experimentation with the description logic system FaCT [18, 17] shows that reasoning problems in realistic scenarios of evolving schemata are solved very efficiently.

As a final remark, it should be noted that the high expressiveness of the description logic constructs can capture an extended version of the presented object-oriented model, which includes not only taxonomic relationships, but also arbitrary boolean constructs, inverse attributes, n-ary relationships, and a large class of integrity constraints expressed by means of $\mathcal{ALCQI}$ inclusion dependencies [8].

The last point suggests that axioms modelling schema changes can be freely combined in order to transform a schema in a new one. Some combination can be defined at database level by introducing new non-elementary primitives. For the sake of brevity, we list some interesting examples we have considered:

Generalisation/Specialisation of an Attribute. The attribute A of a class C becomes an attribute of a C's superclass (generalisation) or of a subclass (specialisation).
Split of an Attribute. In class $\mathbf{C}$ the attribute $\mathbf{A}$ is split into two attributes $\mathbf{A}'$ and $\mathbf{A}''$.
Split of a Class. The class $\mathbf{C}$ is split into two classes $\mathbf{C}'$ and $\mathbf{C}''$.
Union of Attributes/Classes. Dual of the corresponding split operations.
Intersection/Difference of Classes. The new class C is the intersection (difference) of classes $\mathbf{C}'$ and $\mathbf{C}''$.
Decomposition of a class into a set of classes. One or more attributes in a class C are replaced by references to newly defined classes built from the substituted attributes [6].

5 Conclusions and Further Work

This paper deals with the support of database schema evolution and versioning by introducing a general framework based on a semantic approach. The reducibility of a general object-oriented conceptual model to the proposed framework made it possible to provide a sound foundation for the purposes stated in the Introduction. In particular, the adoption of a description logic for the framework specification implies the availability of powerful services which can be proved decidable.

We are currently working to extend the framework to include a view language for data conversion, for which the evaluation, consistency, and containment problems (under the constraints given by the evolving schema) could still be proved decidable. Once this view language is available, it would be possible to use it also for accessing the data through the schema versions, in both the case when the schema evolves but a single database is maintained, and the more general case when a different database is associated to every schema version [11]. Legacy applications could reuse the same query formulation related to a version of the schema different from the one modelling the actual data.

More complex is the case when a query – possibly over more than one conceptual schema – requires an answer from more than one pool of data. This is the case when a particular object maintains its identity over different version – i.e., it evolves with the schema by varying its *structural* properties – and it is requested to have an overview of its evolution over time [3, 2]. In this case an explicit treatment of time is required in the formal framework. By adopting a temporally extended conceptual data model with *implicit* time [16], and by assuming that objects in the same database have implicitly always the same timestamp – the one labelling the database version, the framework proposed in this paper easily lifts up to cover the case of having multiple pools of data. In fact, the set of pools can be considered as a unique temporal database where each pool represents a temporal snapshot[1], and queries involving several schemata are encoded as temporal queries involving terms in different time points.

References

1. S. Abiteboul and P. Kanellakis. Object identity as a query language primitive. *Journal of the ACM*, 45(5):798–842, 1998. A first version appeared in SIGMOD'89.
2. Alessandro Artale and Enrico Franconi. Schema integration of temporal databases. Technical report, University of Manchester, 1999.
3. Alessandro Artale and Enrico Franconi. Temporal ER modeling with description logics. In *Proc. of the International Conference on Conceptual Modeling (ER'99)*. Springer-Verlag, November 1999.
4. J. Banerjee, W. Kim, H.-J. Kim, and H. F. Korth. Semantics and Implementation of Schema Evolution in Object-Oriented Databases. In *Proc. of the ACM-SIGMOD Annual Conference*, pages 311–322, San Francisco, CA, May 1987.
5. S. Bergamaschi and B. Nebel. Automatic Building and Validation of Multiple Inheritance Complex Object Database Schemata. *International Journal of Applied Intelligence*, 4(2):185–204, 1994.
6. P. Brèche. Advanced Principles of Changing Schema of Object Databases. In *Proc. of the 8th Int'l Conf. on Advanced Information Systems Engineering (CAiSE)*, pages 476–495, Crete, Greece, May 1996.

[1] The case corresponds to the multi-pool solution for temporal schema versioning of snapshot data in the [11] taxonomy.

7. Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Daniele Nardi. Reasoning in expressive description logics. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*. Elsevier, 1999. To appear.
8. Diego Calvanese, Maurizio Lenzerini, and Daniele Nardi. Description logics for conceptual data modeling. In Jan Chomicki and Günter Saake, editors, *Logics for Databases and Information Systems*, pages 229–263. Kluwer Academic Publisher, 1998.
9. Diego Calvanese, Maurizio Lenzerini, and Daniele Nardi. Unifying class-based representation formalisms. *Journal of Artificial Intelligence Research*, 11:199–240, 1999.
10. T. Catarci and M. Lenzerini. Representing and Using Interschema Knowledge in Cooperative Information Systems. *Journal of Intelligent and Cooperative Systems*, 2(4):375–398, 1993.
11. C. De Castro, F. Grandi, and M. R. Scalas. Schema Versioning for Multitemporal Relational Databases. *Information Systems*, 22(5):249–290, 1997.
12. F. Ferrandina, T. Meyer, R. Zicari, G. Ferran, and J. Madec. Schema and Database Evolution in the O_2 Object Database System. In *Proc. of the 21st Int'l Conf. on Very Large Databases (VLDB)*, pages 170–181, Zurich, Switzerland, September 1995.
13. Enrico Franconi, Fabio Grandi, and Federica Mandreoli. A semantic approach for schema evolution and versioning – proof of lemma. Available at `http://www.cs.man.ac.uk/~franconi/proof-0200.ps`.
14. F. Grandi and F. Mandreoli. ODMG Language Extensions for Generalized Schema Versioning Support. In *Proc. of ECDM'99 Workshop (in conj. with ER)*, Versailles, France, November 1999.
15. F. Grandi, F. Mandreoli, and M. R. Scalas. A Generalized Modeling Framework for Schema Versioning Support. In *Proc. of 11th Australasian Database Conference (ADC 2000)*, Canberra, Australia, January 2000.
16. H. Gregersen and C. S. Jensen. Temporal Entity-Relationship Models - A Survey. *IEEE Transaction on Knowledge and Data Engineering*, 11(3):464–497, 1999.
17. I. Horrocks. FaCT and iFaCT. In *Proceedings of the International Workshop on Description Logics (DL'99)*, pages 133–135, 1999.
18. I. Horrocks, U. Sattler, and S. Tobies. Practical reasoning for expressive description logics. In H. Ganzinger, D. McAllester, and A. Voronkov, editors, *Proceedings of the 6th International Conference on Logic for Programming and Automated Reasoning (LPAR'99)*, number 1705 in Lecture Notes in Artificial Intelligence, pages 161–180. Springer-Verlag, 1999.
19. C. S. Jensen, J. Clifford, S. K. Gadia, P. Hayes, and S. Jajodia et al. The Consensus Glossary of Temporal Database Concepts - February 1998 Version. In O. Etzion, S. Jajodia, and S. Sripada, editors, *Temporal Databases - Research and Practice*, pages 367–405. Springer-Verlag, 1998. LNCS No. 1399.
20. S.-E. Lautemann. A Propagation Mechanism for Populated Schema Versions. In *Proc. of the 13th International Conference on Data Engineering (ICDE)*, pages 67–78, Birmingham, U.K., April 1997.
21. S. Monk and I. Sommerville. A Model for Versioning of Classes in Object-Oriented Databases. In *Proc. of the 10th British National Conf. of Databases (BNCOD)*, pages 42–58, Aberdeen, Scotland, July 1992.
22. G. T. Nguyen and D. Rieu. Schema Evolution in Object-oriented Database Systems. *Data and Knowledge Engineering*, 4:43–67, 1989.
23. D. J. Penney and J. Stein. Class Modification in the GemStone object-oriented DBMS. In *Proc. of the Int'l Conf. on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA)*, pages 111–117, Orlando, FL, December 1987.
24. R. J. Peters and M. T. Özsu. An Axiomatic Model of Dynamic Schema Evolution in Objectbase Systems. *ACM Transaction on Database Systems*, 22(1):75–114, 1997.
25. J. F. Roddick. A Survey of Schema Versioning Issues for Database Systems. *Information and Software Technology*, 37(7):383–393, 1996.
26. J. F. Roddick and R. T. Snodgrass. Schema Versioning. In *The TSQL2 Temporal Query Language*, pages 427–449. Kluwer, 1995.