INTRODUCTION

Modelling software design as the systematic transformation of formal specifications into efficient programs and building CASE tools that support it has long been considered one of the ultimate goals of software engineering. For instance, Balzer (1981) and Fikas (1985) consider that the process of developing a program [can be] formalized as a set of correctness-preserving transformations [...] aimed to compilable and efficient program production. In this context, according to Partsch and Steinbrüggen (1983), a transformation is a relation between two program schemes P and P’ (a program scheme is the [parameterized] representation of a class of related programs; a program of this class is obtained by instantiating the scheme parameters). It is said to be correct if a certain semantic relation holds between P and P’.

These definitions still hold for database schemas, which are a special kind of abstract program schemes. The concept of transformation is particularly attractive in this realm, though it has not often been made explicit (for instance, as a user tool) in current CASE tools.

A (schema) transformation is most generally considered to be an operator by which a data structure S1 (possibly empty) is replaced by another structure S2 (possibly empty) which may have some sort of equivalence with S1. Some transformations change the information contents of the source schema, particularly in schema building (adding an entity type or an attribute) and in schema evolution (removing a constraint or extending a relationship type). Others preserve it and will be called semantics-preserving or reversible.

Transformations that are proved to preserve the correctness of the original specifications have been proposed in practically all the activities related to schema engineering: schema normalization (Rauh & Stickel, 1995), DBMS schema translation (Rosenthal & Reiner, 1994), schema integration (McBrien & Poulouvasilis, 2003), schema equivalence (Jajodia, Ng, & Springsteel, 1983; Kobayashi, 1986), data conversion (Navathe, 1980), reverse engineering (Casanova & Amaral De Sa, 1984; Hainaut, Chandelon, Tonneau, & Joris, 1993), schema optimization (Halpin & Proper, 1995), database interoperability (McBrien & Poulouvasilis; Thiran & Hainaut, 2001), and others. The reader will find in Hainaut (1995) an illustration of numerous application domains of schema transformations.

Though it has been explored for more than 25 years, this concept has only been gaining wider acceptance for two years, as witnessed by the recent references Omelayenko and Klein (2003) and van Bommel (2004).

The goal of this paper is to develop and illustrate a general framework for database transformations in which most processes mentioned above can be formalized and analyzed in a uniform way. Section 2 describes the basics of schema transformations. Section 3 explains how practical transformations can be used for database engineering. The database design and reverse engineering processes are revisited in Section 4, where we give them a transformational interpretation. Section 5 concludes the article.

BACKGROUND

This section describes a general transformational theory that will be used as a basis for modelling database engineering processes. First, we define a wide-spectrum model from which operational models (i.e., those which are of interest for practitioners) can be derived. Then, we describe the concept of transformation and its semantics-preserving property.

A Data Structure Specification Model

Database engineering is concerned with building, converting, and transforming database schemas at different levels of abstraction and according to various paradigms. Some processes, such as normalization, integration, and optimization operate within a single model and require intra-model transformations. Other processes, such as logical design, use two distinct models, namely, source and target. Finally, some processes—among others, reverse engineering and federated database development—can operate on an arbitrary number of models (or on a hybrid model made up of the union of these models) as we will see later on. The generic entity-relationship model (GER) is a wide-spectrum formalism intended to encompass most popular operational models, whatever their abstraction level and their underlying paradigms (Hainaut, 1996).
The GER includes, among others, the concepts of schema, entity type, entity collection, domain, attribute, relationship type, key, as well as various constraints. In this model, a schema is a description of data structures (Figure 1). It is made up of specification constructs, which can be, for convenience, classified into the usual three abstraction levels, namely, conceptual, logical, and physical:

- A **conceptual schema** comprises entity types, super/subtype (is-a) hierarchies, relationship types, roles, attributes (multi/single-valued; atomic/compound), identifiers (or unique keys), and various constraints.
- A **logical schema** comprises such constructs as record types, fields, arrays, foreign keys, redundant fields, etc.
- A **physical schema** comprises files, record types, fields, access keys (a generic term for index, calc key, etc), physical data types, bag and list multivalued attributes, and other implementation details.

Since it includes the main concepts of most operational models, the GER can be used to precisely specify each of them thanks to a specialization mechanism. According to the latter, each construct of model M is a specialization of a construct of the GER model. For instance, tables, columns, primary keys, and foreign keys are specializations of, respectively, entity types, attributes, primary identifiers, and referential attributes. In addition, schemas in M must satisfy specific assembly rules, such as the following: *each entity type has at least one attribute*. As an important consequence, all intra- and inter-model transformations are specializations of GER-to-GER transformations. For instance, the standard ERA-to-SQL transformation is modelled by a chain of transformations from the ERA specialization of the GER to its SQL specialization.

The GER model has been given a formal semantics in terms of an extended NF2 model (Hainaut, 1996). This semantics allows us to analyze the properties of transformations and particularly to precisely describe how and under which conditions they propagate and preserve the information contents of schemas.

**Transformation: Definition**

The definitions that will be stated are model-independent. In particular, they are valid for the GER model, so that the examples will be given in the latter. We denote by M the model in which the source and target schemas are expressed, by S the schema on which the transformation is to be applied, and by S’ the schema resulting from this application.

A transformation $\Sigma$ consists of two mappings $T$ and $t$ (Figure 2):

---

**Figure 1.** A typical hybrid schema made up of conceptual constructs (e.g., entity types PERSON, CUSTOMER, EMPLOYEE, and ACCOUNT; relationship type of; identifiers Customer ID of CUSTOMER); logical constructs (e.g., record type ORDER, with various kinds of fields, including an array, foreign keys ORIGIN and DETAIL.REFERENCE); and physical objects (e.g., table PRODUCT with primary key PRO_CODE and indexes PRO_CODE and CATEGORY, table space PRODUCT.DAT). The identifier of ACCOUNT, stating that the accounts of a customer have distinct Account numbers, makes ACCOUNT a dependent, or weak, entity type. The cardinality constraint of a role follows the participation interpretation and not the UML look-across semantics.