Automatic Generation of Content Management Systems from EER-Based Specifications

Sebastiano Vigna
Dipartimento di Scienze dell'Informazione, Università degli Studi di Milano
vigna@acm.org

Abstract

ERW is an innovative open-source system for handling complex databases using a web browser. Once the details of an enhanced entity-relationship schema have been specified in XML, ERW generates a complete application that lets the user interact with the database. Then, *specification percolation* makes it possible to customise heavily the application while maintaining the flexibility of a model-driven approach.

1 Introduction

Entity-Relationship (ER) schemata are a popular conceptual model introduced by Chen [Chen:1976], and later extended in several ways; the more common extensions are usually termed *Extended ER* (EER).

ERW is a framework that, essentially, lets users modify instances of an EER schema. More in detail, one defines an EER schema using ERL, an XML-based language. Then, a Java™ tool reifies the schema into a set of SQL tables using a standard algorithm, and produces related documentation and diagrams. Moreover, it produces a set of definition files used by a run-time environment written in PHP. The run-time environment creates forms that let the user interact with the schema instance, with natural operations such as "associate this entity to this entity", and so on. In particular, the ontological data contained in the schema is used to offer different interfaces for types with a different ontological status (e.g., identification functions).

There are of course several tools that let you edit SQL databases using a browser. However, ERW does not let you edit SQL databases: rather, it defines a very precise mathematical semantics (based on bicategory theory) of EER schemata with a type system [Vigna:2002b], and it lets you edit an instance of a schema, as defined by the semantics.[1]

The design of ERW started from a number of major requirements, which were not available in any existing system we were aware of.

- **Support for a wide range of EER features**. A tool based on conceptual modelling is of no use if a sufficient number of sophisticated features is not available. Relationship types with attributes, weak entities with multiple owners, multiple inheritance, abstract (noninstantiable) entities should be part of the basic modelling capabilities.
- **Maintenance of *referential* and *logical* integrity of the database**. All cardinality constraints imposed at the EER level should be automatically enforced.
- **Built-in authorisation mechanisms**. Content management systems are routinely access concurrently by many users with different (possibly fine-grained) permission levels.

[1] As a side effect, the mathematical semantics allows one to prove validation results about a schema. Currently, ERW implements new algorithms that detect statically double ownership [Vigna:2002b] and unreachable instances.

- **Rich, intuitive user interface**. The powerful features offered by the W3C DOM should make it possible to build a rich and intuitive user interface.
- **Internationalisation**. ERW should provide language-dependent forms based on content negotiation, and full UTF-8 support.
- **DBMS independent**. A generic tool for content management should not be tied to any specific database.
- **Multimedia content**. A content management system is likely to contain multimedia data; it should provide a simple way to associate files to entities.
- **Scalability**. Both the size of the schema *and* the size of an schema *instance* should not be limited. Whereas scalability w.r.t. the schema size is usually not a problem, several efforts in the domain of web-based editing assume that the database will be small.
- **Clean semantics**. The notion of schema instance should be clearly defined by a mathematical model, and the SQL database data should closely reflect the model.
- **Open-source software based on open standards**. ERW should be entirely based on international standards and open-source tools.
- **Customisable, but not hardwired**. The editing forms should be largely customisable, but in a way that does not keep from continuing to extend and update the conceptual model. Vendor-provided tools often fail to support customised user-interface requirements.

ERW implements all these requirement leveraging upon well-known standards: EER schemata for conceptual modelling, XML for definition and configuration files, `printf`-like strings for label formatting, UTF-8 for character representation, regular expressions for content validation, HTML, ECMAScript and the W3C DOM for building a rich user interface, a subset of SQL-99 for interacting with the DBMS and the UN*X user/group authorisation mechanism.

2 A Look at the Schema Semantics

Albeit the semantics of an ERW database has been fully documented elsewhere [Vigna:2002b], we give here a brief review. There are two main peculiarities: relationship types are instantiated to *multirelations*, and there is a definite type system, similar to the one of object-oriented programming languages.

Schemata. For simplicity, in this review we do not introduce attributes in our schema definition. An *EER schema* (of binary relations) $\mathscr{S}$ is given by a set $\mathscr{E}$ of entity types, a set $\mathscr{R}$ of relationship types, a source function $s : \mathscr{R} \to \mathscr{E}$ and a target function $t : \mathscr{R} \to \mathscr{E}$. An entity type may also be *abstract*[2] or *weak*. Moreover, each relationship type has a source and a target *cardinality constraint*, which is a symbol out of `(0:1)`, `(1:1)`, `(0:N)`, `(1:N)`, `(0:M)`, `(1:M)`. The ordered pair of cardinality constraint of a relationship type is usually written as `(-:-)`→`(-:-)`. Finally, a relationship type may be marked optionally either `ISA`, in which case its constraint must be `(1:1)`→`(0:1)`, or `WEAK`, in which case its constraint must be `(1:1)`→`(-:N)`, and its source entity type must be weak. Moreover, every weak entity type must be the source of at least one `WEAK`-labelled relationship type.

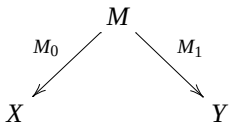
Whenever there is an `ISA` relationship type from E to F, E is said to be a *direct subtype* of F, and F a *direct supertype* of E. A *subtype* of E is either E or a direct subtype of a subtype of E (analogously for supertypes).

[2] The notion of abstract entity type is borrowed from object-oriented languages (notably Java). The idea is that it should be used for types that are necessary for a correct structuring of the type hierarchy, but that are "universals" (in the ontological sense) and thus have no instance themselves—they can be just instantiated through their subtypes.

Note that there are two new cardinality types: `(0:M)` and `(1:M)`. The `M` indicates that we are actually requiring a multirelation (see the next section for the exact semantics).

Instances. To define a schema instance, one introduces multirelations in the spirit of bicategory theory [Borceux:1994]:

Definition 1 A (binary) multirelation from set X to set Y is a set M endowed with two functions, the *left leg* M_0 and the *right leg* M_1:



Two elements $x \in X$ and $y \in Y$ are related if there is an $r \in M$ such that $M_0(r) = x$ and $M_1(r) = y$. It can happen that this is true for more than one element of M: in this case, the elements x and y are related "more than once".

Definition 2 An instance σ for a schema $\mathscr{S}$ is given by a map σ assigning to each entity type E in $\mathscr{E}$ a set $\sigma(E)$ and to each relationship type R in $\mathscr{R}$ a multirelation $\sigma(R)$ satisfying the following properties:

1. The left leg of $\sigma(R)$ must end in $\sigma(\mathrm{s}(E))$, and the right leg in $\sigma(t(E))$; in other words, if R is a relationship type from entity type E to entity type F, then $\sigma(R)$ must be a multirelation with a left leg ending in $\sigma(E)$ and a right leg ending in $\sigma(F)$.
2. A cardinality constraint[3] of the form `(1:-)` requires that the corresponding leg be a surjective function.
3. A cardinality constraint of the form `(-:1)` requires that the corresponding leg be an injective function.
4. A cardinality constraint of the form `(-:N)` requires that the multirelation is actually a relation (i.e., nothing is related twice).
5. Whenever a relationship type is marked `ISA`, the first leg is the identity and the second leg is an inclusion map, so that the source entity set is a subset of the target entity set.
6. Whenever entity types E and F have a common supertype and $x \in \sigma(E) \cap \sigma(F)$, there is a common subtype G of E and F such that $x \in \sigma(G)$.
7. If E is abstract, $\sigma(E)$ is exactly the union of $\sigma(F)$ when F ranges through the proper subtypes of E.[4]

Condition (6) is important as it forces *definite typing*. Essentially, every entity must have a type[5].

An *entity* is now a pair (E, x) such that E is the type of x. Note that we did not restrain entity sets to be disjoint, so (E, x) and (F, x) are actually *distinct entities*[6].

Representing schema instances in SQL. As we mentioned in the introduction, the semantics we defined has the purpose of specifying what exactly ERW stores in a schema instance. Thus, we must guarantee that when the EER schema is reified the reification algorithm supports the multirelation-based semantics we have just discussed.

[3] The wording of cardinality constraints may seem a bit unorthodox: however, it is easy to see that it is exactly equivalent to the standard *participation interpretation* of constraints.

[4] In other words, all instances are subtype instances. In particular, an abstract entity type without subtypes cannot have instances.

[5] If, for instance, `man` and `woman` are subtypes of `person`, it is not possible that x belongs to $\sigma($`man`$)$ *and* $\sigma($`woman`$)$ (unless you add a common subtype `hermaphrodite` of both `man` and `woman`).

[6] This fact parallels the common usage of numerical identifiers to represent set elements in SQL databases—the identifiers are not necessarily distinct in different tables.

ERW's approach is very simple, and inspired by common practise in databases. To superimpose our abstract semantics on the tuple semantics of relational databases, we first need to be able to speak about elements and multirelations without referring to attributes. Thus, all tables generated by the reification algorithm contain an `id` column, acting as primary key for all entities and relationships. Then, ERW follows common customs, using a single SQL column to represent relationship types that instantiate to partial functions, and a support table for the remaining ones. Subtyping is represented by set inclusion, exactly as it happens in a schema instance.[7]

This concrete structure can be exactly mimicked by our bicategorical semantics: we just have to restrict our sets to sets of natural numbers, and we will have a faithful mathematical reproduction of our database. Composition of multirelations by means of pullbacks [Vigna:2002b] now models exactly the natural join on foreign keys.

3 Customised Content Management

A serious problem faced by automatic software generation for content management is the existence of two contradictory goals: the application should be generated in a completely automatic way, to minimise efforts, but at the same time it should be highly customisable.

Often the solution is adding custom code to an automatically generated skeleton. However, even if this approach is extremely flexible, it is also very dangerous, as procedurally specified customisations end up being strongly tied to the skeleton itself.

ERW adopts a design pattern that we term *specification percolation*: the generation of an application is akin a fluid, percolating from the ERL specification to the user interface through a series of additional specifications, which act similarly to active membranes: they may filter some information, and even replace it with something else, but it should never happen that information is lost in the process or an inconsistent configuration is generated.

In Figure 1 we represent schematically the way information percolates through ERW's additional specifications. From the ERL file, which just describes an EER schema and restrictions on its instances (e.g., regular expressions for text fields), ERW can generate documentation about the reification process, graphical layouts hyperlinked to the documentation, and the SQL code that will create the database.

However, the most interesting part is how this information percolates to the user interface. ERW translates the ERL file into a set of PHP definition files, which essentially set up an associative array representing conveniently the information contained in the schema.

The first membrane on the route of this information is internationalisation: labels for attributes and enumerative type values are translated in a suitable language (usually negotiated with the browser).

Then, the administrator may set up *custom files* which alter (in a controlled way) the information represented in the definition files. This modifications may range from filtering options (specifying which fields are important to find quickly entities and relationships of a given type), labelling (`printf`-like formatting strings that use the attributes of the schema), and alternative user interfaces.

This information allows to tune several parts of the application, but it is not necessary to specify any part of it: for instance, filtering is performed by default on the first mandatory attribute, and labelling is handled analogously.

Then, the user may specify custom forms. These are completely redefined forms, which are described in an XHTML-like language that contains the basic elements, plus suitable elements to position the input controls for attributes and relations.

Again, specifying a form is not necessary: ERW is able to generate autonomously a default form. In doing so, it preserves whatever information has percolated, such as the attribute order (implicit in the ERL file),

[7] In other words, ERW keeps in sync the `id` columns of the tables belonging of a type hierarchy so that all rows with the same `id` represent the same element.

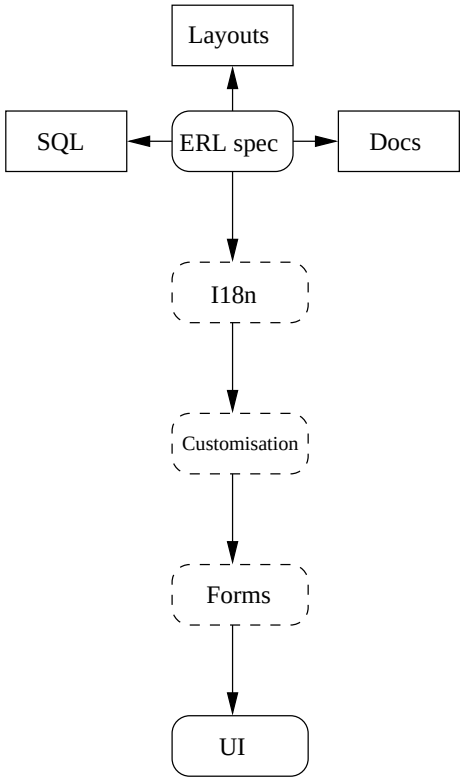


Figure 1: Specification percolation

filtering options, alternative user interface and so on. Of course, these may be all re-specified at the custom form level, but this is not necessary: a custom form may decide, for instance, to modify just the *order* in which attribute are presented, but not the specific interface that was chosen for each of them.

The advantage of a careful design based on specification percolation is that the application is highly customisable, but at the same time *every change to one of the specifications is immediately reflected in the user interface*, as it percolates freely through the following ones.

For instance, suppose that in a complex database, with a high degree of customisation and internationalisation, the administrator has to add quickly a new entity type. This is as simple as adding the new type to the ERL specification: there is no immediate need to add all customisation layers to the new type, as it is guaranteed that the information contained in the ERL file will percolate down to the user interface, producing a reasonable form in the default language. Layers of customisation can be added later, without disturbing the rest of the application.

4 The Web Architecture

The web interface offered by ERW is more similar to a dedicated client than to a typical web application. This is due to intensive client-side scripting and W3C DOM manipulation. Two highlights of the architecture are *stateless editing* and *remote database access* (see also [Vigna:2002a]).

Stateless editing. ERW is designed to let the user edit freely a database recording *no* information server-side. ECMAScript and the W3C DOM are used to show how relationships are added or deleted, and how their attributes are modified. *All data is stored in the ECMAScript state of the browser*. When the user submits the form, the ECMAScript state is suitably serialised and sent to the server.

Remote Database Access. ECMAScript provides no direct way to connect to a database. Usually, whenever the user has to choose among a set of elements the entire set is packed in a suitable HTML input element and sent to the browser. This approach, however, makes it impossible to edit large databases. Thus, ERW uses *remote scripting* to implement on top of HTTP a remote database access mechanism.

5 Related Work

There are of course several tools that provide content management using SQL databases. We briefly review the proposal we are aware of, and compare them with ERW. We note, however, that in several cases the tools are not publicly available, whereas ERW is free software downloadable from `http://erw.dsi.unimi.it/`.

WebML/AutoWeb/W3I3. This family of tools is oriented toward the generation of data-intensive web applications, and founds its modelling on HDM Lite, a simple ER model enriched with presentational and navigational information [Fraternali & Paolini:2000]. AutoWeb has a dual purpose w.r.t. ERW, as it is concerned more with *presenting* than with *managing* content; correspondingly, it has no support for advanced EER features or sophisticated web clients for database access.

WebCUS. WebCUS is a system that has a goal similar to AutoWeb. Apparently [Kerer, Kirda & Kurmanowytsch:2002] it requires a very concrete (SQL-like) description of the database. It uses MyXML as a graphical back-end to produce a web site.

WWWdb, Sashipa–Melba, mySQLAdmin, etc. These, and similar tools, allow one to edit a generic SQL database in more or less sophisticated forms, but do not provide EER modelling.

6 Conclusions

ERW has been used during the last three years at the DSI of the Università degli Studi di Milano. Implementation started in 2001, after several discussions with Massimo Santini and Paolo Boldi. Paolo Boldi helped in writing parts of ERtool, and made several useful comments. In February 2002 ERW was released to the public as free software.

A major advance in feature implementation and testing was the effect of a collaboration with the Università di Verona. Roberto Posenato and Alberto Belussi started a complete refactoring of the science faculty database using ERW. In doing so, they created a very complex schema, that required several new features. At the same time, the system was extensively tested by hundreds of concurrent users.

References

[Borceux:1994] Francis Borceux. *Handbook of Categorical Algebra 2*, volume 51 of *Encyclopedia of Mathematics and Its Applications*. Cambridge University Press, 1994.

[Chen:1976] Peter Pin-Shan Chen. The entity-relationship model: Toward a unified view of data. *ACM Transactions on Database Systems*, 1(1):9–36, 1976.

[Fraternali & Paolini:2000] P. Fraternali and P. Paolini. Model-driven development of web applications: The AutoWeb system. *ACM Trans. on Inform. Systems*, 28:323–382, 2000.

[Kerer, Kirda & Kurmanowytsch:2002] Clemens Kerer, Engin Kirda, and Roman Kurmanowytsch. A generic content-management tool for web databases. *Internet Computing*, 6(4):38–42, 2002.

[Vigna:2002a] Sebastiano Vigna. ERW: Entities and relationships on the web. In *Poster Proc. of Eleventh International World Wide Web Conference*, Honolulu, USA, 2002.

[Vigna:2002b] Sebastiano Vigna. Multirelational semantics for extended entity-relationship schemata with applications. In *Conceptual Modeling—ER 2002. 21st International Conference on Conceptual Modeling*, number 2503 in Lecture Notes in Computer Science, pages 35–49. Springer–Verlag, 2002.