

GENERALIZATION AND BLENDING IN THE GENERATION OF ENTITY-RELATIONSHIP SCHEMAS BY ANALOGY

Marco A. Casanova, Simone D.J. Barbosa, Karin K. Breitman and Antonio L. Furtado
Departamento de Informática – Pontifícia Universidade Católica do Rio de Janeiro
Rua Marquês de S. Vicente, 225 – Rio de Janeiro – CEP 22451-900, Brasil

Keywords: Schema Generation, Analogy, Blending, Lattices, Entity-Relationship Model, Logic Programming.

Abstract: To support the generation of database schemas of information systems, a five-step design process is proposed that explores the notions of generic and blended spaces and favours the reuse of predefined schemas. The use of generic and blended spaces is essential to achieve the passage from the source space into the target space in such a way that differences and conflicts can be detected and, whenever possible, conciliated.

1 INTRODUCTION

Designers of information systems soon learn that reusing their previous experience, and also that of other designers, is a rewarding strategy. Motivated by this remark, we have been working (Breitman et al., 2007; Barbosa et al., 2007) on methods and tools to abstract a *pattern* that captures the structure of a database schema regarded as a *source schema*, which is then repeatedly used to generate one or more *target schemas*. What makes this strategy viable is the perception of an *analogy* between source and target, expressed by “*target is like source*”. Additionally, the source schema should be a typical example among those that are analogously structured, and the terminology of its underlying domain should be familiar even to the less experienced designers. If these requirements are satisfied, it will be possible to instantiate the positions occupied by variables in the pattern by prompting the designer to indicate which name in the target schema being generated correspond to which name in the example source schema.

In the present paper, we expand our earlier method and introduce a five-step process that takes four spaces into consideration – the source, target, generic and blended spaces, as proposed in (Fauconnier & Turner, 1994) for widely different areas. We adopt the familiar Entity-Relationship (ER) model (Batini, Ceri & Navathe, 1992) and use the weak entity concept to illustrate the process.

The diagram in figure 1 represents the four spaces and shows how they are articulated in view of

the process, whereby, starting from the source, the target is gradually constructed.

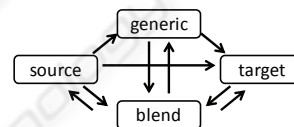


Figure 1: The four-space approach.

Informally, the generic space originates from the source by importing, in a generalized format, the elements for which corresponding elements in the target will eventually be characterized. In practice, both the source and the target will contain other non-corresponding elements, since analogy is rarely bijective. Viewing the diagram as a lattice (MacLane & Birkhoff, 1967), the generic constitutes the *meet* of the source and the target spaces and denotes the elements that correspond to each other in these two spaces. By contrast, the blended space reflects the *join* of source and target and inherits all their elements, corresponding or not. The blend is the space wherein one can detect whatever is incomparable or conflicting when putting together source and target, often calling for some form of adaptation (Turner, 1996; Fauconnier & Turner, 2002). Goguen (1999) formalized blending in category theory.

The text is organized as follows. Section 2 details the process we propose, section 3 extends it to operations, and section 4 contains the conclusions.

2 THE FIVE-STEP SCHEMA-GENERATION PROCESS

2.1 Example

We adopt a simple example to illustrate the proposed schema generation process. We start with a schema *fragment*, specifying *employees* and their *dependents*, which is probably the most frequently mentioned illustration of the weak entity concept in ER modeling. As a fragment, it only needs the elements relevant to characterize weak entities.

We express schemas with the help of clauses such as those below that introduce two entity classes, *employee* and *dependent*:

```
Schema: Emp_Dep
Clauses --
  entity(employee, empno)
  attribute(employee, empno)
  entity(dependent,
    [empno/depno-isdepof-empno, depno])
  attribute(dependent, depno)
  relationship(isdepof,
    dependent/0/n, employee/1/1)
  attribute(isdepof, family_tie)
```

The identifying attribute of *employee* is *empno*, whereas *dependent*, being a weak entity, relies on the identifying relationship *isdepof*, combined with the discriminating attribute *depno*. The identifying relationship is 1 to *n*, being total with respect to *dependent* and partial with respect to *employee*; these properties are indicated by associating pairs of minimum and maximum values for the participation of instances of each entity in relationship instances: at least 0 and at most *n* *dependents* can be related with exactly one *employee*. The relationship *isdepof* has attribute *family_tie*, whose values are *spouse* and *child*. Note that the fragment does not include, as unessential to the characterization of weak entities, certain basic properties of *employee*, such as those referring to the employment itself.

This schema will be used as the source schema, wherefrom target schemas based on the weak entity concept can be derived, through five consecutive steps, to be described in the sequel. As will be noticed, the process takes into due consideration some domain-independent consistency rules inherent in the ER model, such as the following:

- 1) all entity classes must have identifying properties;
- 2) relationships can only be defined between defined entity classes;
- 3) the deletion of an entity instance implies the deletion of all its properties;

- 4) if a relationship *R* is total with respect to one of its participating entity classes *E*, an instance of *R* cannot be deleted if it is the only one involving a given instance of *E*.

2.2 Step 1 - Generating the Pattern

From the source schema *Emp_Dep*, the Weak Entity pattern is obtained (Fig. 2) by substituting variables for the names of entities, relationships and attributes.

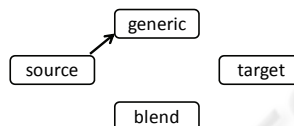


Figure 2: Generating the pattern.

The pattern contains *mappings* that associate the introduced variables with the corresponding source schema names. For example, variable *A* refers to *employee* wherever it occurs in the pattern.

```
Pattern: Weak Entity
Example schema: Emp_Dep
Clauses --
  entity(A, B)
  attribute(A, B)
  entity(C, [B/D-E-B, D])
  attribute(C, D)
  relationship(E, C/0/n, A/1/1)
  attribute(E, F)
Mappings --
A:employee
B:empno
C:dependent
D:depno
E:isdepof
F:family_tie
```

2.3 Step 2 - Generating the Target Schema

Suppose the designer wants to specify a *Bk_Ed* schema, about book editions, and realizes that this too involves the weak entity concept: the *editions* of a book are comparable to the *dependents* of an *employee*. The generation (Fig. 3) is basically done by specializing the clauses of the pattern (in the generic space), but also referring to the originating source space, to stress that the names figuring in the pattern mappings were extracted from it.

Each pattern variable is replaced by an appropriate name belonging to the underlying domain of *Bk_Ed*.

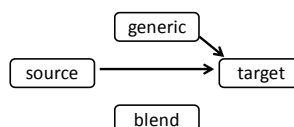


Figure 3: Generating the target schema.

Relying on the assumption of an intuitive understanding of the analogy between the two domains, the designer is prompted to supply the target schema names through queries of the form:

- What corresponds to <name in source>?

In our example, this would instantiate the pattern mappings as follows:

```
employee → book
empno → isbn
dependent → edition
depno → edno
isdepof → isedof
family_tie → nil
```

We note that the designer may, with limitations, deny one or more correspondences by replying `nil` as with the attribute `family_tie`. This is indeed the only element in this case that can be absent. Having informed `book` as corresponding to entity `employee`, the designer should be aware that the indication of what corresponds to `empno` is mandatory, since no entity can lack an identifier (cf. ER rule 1, section 2.1). Likewise, if nothing corresponds to `dependent`, the indication of `isedof` as corresponding to `isdepof` would be an error, because a binary relationship requires the presence of two participating entities (cf. ER rule 2). The absence of `isedof`, on the other hand, would defeat the purpose of the entire process – the weak entity concept makes no sense without an identifying relationship.

After inspecting the resulting target schema, the designer's knowledge of the target domain must be used to check its clauses, with a special attention to:

- a) additions to the target schema, that have no correspondence in the source schema;
- b) modifications to be done in the generated clauses in the target schema.

Suppose that the designer judged that the addition and the modification below are necessary:

```
addition:      attribute(book, subject)
modification:  isedof - min-1:1
```

Then, the `Bk_Ed` target schema becomes:

```
Schema: Bk_Ed
Clauses --
entity(book, isbn)
attribute(book, isbn)
attribute(book, subject)
entity(edition,
  [isbn/edno-isedof-isbn, edno])
attribute(edition, edno)
relationship(isedof,
  edition/1/n, book/1/1)
```

2.4 Step 3 - Blending the Source and Target Schemas

The blended space is pictured as a confluence of the source and target spaces, taking into consideration the correspondences in the generic space (Fig.4).

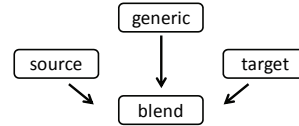


Figure 4: Blending the source and target schemas.

In the database schema-generation process, elements are obtained by joining each entity and relationship of the source schema with its counterpart in the target schema. To begin with, all information about each entity and relationship, contained in the various clauses of the two schemas, is collected in separate *frames*, structured as lists of property:value pairs.

Each property of an entity E is represented either by an attribute name, or by a binary relationship name tagged with 1 or 2 to indicate, respectively, whether E is the first or the second participant in the relationship. Since in the present example no restrictions are being imposed on the values, all value positions are filled with an underscore, a usual convention for an anonymous variable. The properties of a relationship R are similarly represented. They include the identifying attributes of the two participating entities, the minimum and maximum occurrences for the first and for the second participant, and other relationship attributes if any.

The frames extracted from `Emp_Dep` are:

```
frame of employee =
  [empno:_, isdepof/2:_]
frame of dependent =
  [depno:_, isdepof/1:_]
frame of isdepof =
  [depno:_, empno:_,
  min-1:0, max-1:n, min-2:1, max-2:1, fam-
  ily_tie:_]
```

and those taken from the `Bk_Ed` schema are:

```
frame of book =
  [isbn:_, subject:_, isedof/2:_]
frame of edition =
  [edno:_, isedof/1:_]
frame of isedof =
  [edno:_, isbn:_,
  min-1:1, max-1:n, min-2:1, max-2:1]
```

We now introduce a *join* operation on frames, specifying that, when applied to entity or relationship frames F_1 and F_2 , it results in a frame J, whose property-value pairs comprise:

- a) pairs $p_1:v_1$ from F_1 , for each property p_1 not corresponding to any property in F_2 ;
- b) pairs $p_2:v_2$ from F_2 , for each property p_2 not corresponding to any property in F_1 ;

c) pairs p_1 - p_2 : v_{1-2} , for each two corresponding properties p_1 and p_2 in F_1 and F_2 , respectively. Value v_{1-2} in item c is obtained by joining the two values v_1 and v_2 , according to the following criterion: if the values are identical constants, or at least one of them is a variable, v_{1-2} is the result of their *unification* (Knight, 1989); otherwise the result is a term formed by the two values prefixed by an asterisk to indicate that they are in *conflict*.

The frames characterizing the blended space are shown below. Non-corresponding properties and conflicting values are stressed (in italic, boldface; the symbol “ \vee ” denotes the join of two frames):

```

Femployee  $\vee$  Fbook =
  [empno-isbn:_,
   isdepof/2-isdepof/2:_,
   subject:_]
Fdependent  $\vee$  Fedition =
  [depno-edno:_,
   isdepof/1-isdepof/1:_]
Fisdepof  $\vee$  Fisdepof =
  [depno-edno:_,
   empno-isbn:_,
   min-1:*(0,1),
   max-1:n,
   min-2:1,
   max-2:1,
   family_tie:_]

```

A disclaimer is in order here. We considered only one simple type of conflict. If the designer is allowed to perform arbitrary modifications to the target schema initially obtained by instantiating the pattern variables (cf. step 2), other types of conflict may occur, calling for the specification of appropriate criteria to handle them. As noted in (Fauconnier & Turner, 2002), blending is, in general, a particularly complex task, requiring a great deal of creativity from the part of the designer, who may have to devise *ad hoc* ways to achieve consistency.

2.5 Step 4 - Revising the Target (and Source) Schemas

The resulting blended space can be *reinject*ed into the derived target space, and even into the originating source space, if the designer admits the possibility of also reconsidering it (Figure 5).

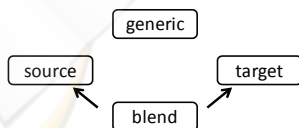


Figure 5: Revising the target (and source) schemas.

A convenient way to call the designer's attention to what was not used from the source schema is to display together, in frame format, the entire list of current properties of each entity and relationship in

the target schema, expanded as the result of blending. Such frames are directly obtained from the blend frames by reducing the paired names assigned to corresponding properties to their original names in the target space, while, naturally, keeping the names of the source space properties until now disregarded:

```

frame of bookemployee =
  [isbn:_, isdepof/2:_, subject:_]
frame of editiondependent =
  [edno:_, isdepof/1:_]
frame of isdepofisdepof =
  [edno:_, isbn:_, min-1:1, max-1:n, min-2:1,
   max-2:1, family_tie:_]

```

Surely, the designer may or may not judge appropriate to reconsider what was initially left out, in this case the relationship attribute *family_tie*. Would there be different "ties" between *edition* and *book*? With respect to its "parent" *book*, an *edition* may be classified as revised, corrected, expanded, abridged, and also simply as regular, which are some of the possible values for a new *ed_type* attribute for the *isdepof* relationship.

The reconsideration of a source schema for expansion is more rarely desirable, especially if one wishes to keep it as a fragment containing only the features necessary to characterize weak entities. But in the event that the designer wants to examine the possibility, the blend frames can be alternatively renamed as follows:

```

frame of employeebook =
  [empno:_, isdepof/2, subject:_]
frame of dependentedition =
  [depno:_, isdepof/1:_]
frame of isdepofisdepof =
  [depno:_, empno:_, min-1:0, max-1:n, min-
   2:1,max-2:1, family_tie:_]

```

What can be the "subject" of an *employee*? The subject of a *book* can be some fictional genre, but it can also be a professional field, such as engineering, or accounting, which may suggest a new attribute *profession* for the *employee* entity, with possible values including *engineer* and *accountant*. A further reduction of *Emp_Dep* to suppress the *family_tie* attribute is more likely to happen. This would become advisable if the attribute is systematically disregarded in a long series of target schema generations. Reconsidering a source schema, and consequently the pattern abstracted from it (as covered in step 5) is a case of *double-loop learning* (Argyris & Schön, 1995): the continuing use of a model providing clues for its correction and refinement.

2.6 Step 5 - Revising the Pattern

Since the generic space is often intended as a help to generate a plurality of target spaces, conflicts located at the blended space, as well as changes made at the source space from suggestions motivated by observing the blend, may entail the reconsideration of the generic space (Figure 6).

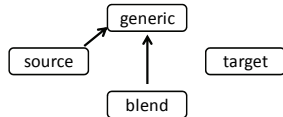


Figure 6: Revising the generic space.

In our example, the blend mirrors the fact that an identifying relationship must be total with respect to the weak entity, but no such requirement is imposed with respect to the entity on which it relies for identification. So the conflict registered in the property:value pair $\text{min-1} : * (0, 1)$ of the frame resulting from the join of F_{isdepof} with F_{isedof} should motivate the insertion of a *hotspot* (Pree, 1995) in the Weak Entity pattern, i.e., a place where the specification becomes flexible. The adopted notation, using a question mark as prefix, will signal that the designer should be queried about the min-1 property of the relationship denoted by variable E , and that the value supplied must be chosen as 0 or 1.

Moreover, if at step 4 a new attribute such as *profession* is added to the source schema, or if the *family_tie* relationship attribute is removed from it, the pattern must be modified accordingly, so that it will continue to reflect the Emp_Dep schema. If all these modifications occur, after deleting the lines

```

attribute(E, F)
F:family_tie
  
```

and adding or modifying three lines (in boldface), the Weak Entity pattern would become:

```

Emp_Dep
Clauses --
  entity(A, B)
  attribute(A, B)
  attribute(A, G)
  entity(C, [B/D-E-B, D]) attribute(C, D)
  relationship(E, C/?(0,1)/n, A/1/1)
Mappings --
A:employee
B:empno
G:profession
C:dependent
D:depno
E:isdepof
  
```

3 TOWARDS THE DESIGN OF OPERATIONS

In (Furtado et al., 2007) we added, both to schemas and patterns, clauses defining *operations* in terms of their *pre-* and *post-conditions* (Fikes & Nilsson, 1971). Without going into details, we now give one example of the repercussion of conflicts detected at the blending stage on the design of operations. Suppose that an operation named *end_coverage* has been defined over the source schema, allowing to remove a child C of an employee E from the list of dependents of E , if the *birth_year* of C (an additional attribute of dependent) precedes a currently determined limit. Note that the deletion of the literal $\text{dependent}([E, C])$ should cause the deletion of all properties of the entity instance C , in view of ER rule 3. On the other hand, note that the repeated execution of *end_coverage* is allowed, legitimately, to leave an employee with no dependents.

```

end_coverage(C, E)
pre-cond: dependent([E, C]),
         family_tie([E, C], child),
         birth_year([E, C], Y),
         Y < b_ylimit.
post-cond: -dependent([E, C]).
  
```

Also suppose that, when prompted to determine an operation corresponding to *end_coverage*, the designer responded with *weed*, to represent a practice known as *weeding library collections* (Slote, 1997). Analogously to *end_coverage*, *weed* discards editions whose year of publication, *ed_year* (from *birth_year*), came before a designated year. A conservative librarian would very likely demand that systematic discarding be restricted to *regular* editions, expressed by an attribute *ed_type* (from *family_tie*), as considered earlier.

However, straightforward renaming and the replacement of *child* by *regular* is not sufficient here to avoid a conflict of the generated *weed* operation when blending, namely, the totality property of *isedof* with respect to *book*, combined here with ER rule 4. One solution to the conflict is to ensure that the *book* itself remains, by keeping its newest edition, as illustrated below:

```

weed(E, B)
pre-cond: edition([B, E]),
         ed_type([B, E], regular),
         ed_year([B, E], Y),
         edition([B, En]),
         ed_year([B, En], Yn),
         Yn > Y,
         Y < ed_ylimit.
post-cond: -edition([B, E]).
  
```

Further refined versions may specify different values of *ed_ylimit* for different subjects, in view of con-

stantly updated studies to determine the period of obsolescence for publications belonging to each so-called Dewey class (Kramer, 2002).

4 CONCLUDING REMARKS

We have run experiments with the current version of the schema-generating process, using an interactive logic programming tool. Also, although simple, the weak entity example helped us gain a better understanding of design by analogy and blending.

Much work remains to be done, especially to extend the process as described in section 2, in order to cope with an ampler variety of conflicts, and to develop semi-automatic algorithms or heuristics to recommend adequate strategies for handling the different situations that may arise in practice.

A more comprehensive treatment of the schema generation problem calls for the study of additional topics. Patterns to model the same concept can be obtained from different source schemas, perhaps resulting in distinct versions with permissible variations, which in turn could be classified and selected by the designer according to the case on hand. Moreover, generating additional versions of the pattern provides a means to check the resulting patterns for conflicts and integrity constraints.

Early studies on analogy and metaphor (Lakoff & Johnson, 1980) argued for the use of multiple sources to characterize a target possessing many properties, which would naturally be grouped according to the originating source. The computational effort of some problem-solving algorithms could be then reduced, by considering only the properties that have been derived from a few designated sources (Holyoak & Thagard, 1996).

When generic and blend represent the confluence of spaces associated with the same underlying domain, they can give rise to new conceptual spaces, through a process sometimes called *categorization* (Fauconnier & Turner, 1994). When different underlying domains are involved, the resulting blend is populated with hybrid entities. Conflating persons, objects or events is a powerful literary practice, and, surprisingly, offers sometimes intuitive clues to solve problems, as in the Buddhist monk riddle expounded in (Turner, 1996). A blend conflating persons and books, for instance, might make sense in a Digital Storytelling application aiming to teach children how to use the facilities of a library. Other Computer Science areas have drawn significantly from the notions of analogy (Barbosa & de Souza, 2001) and blending (Imaz & Benyon, 2007).

ACKNOWLEDGEMENTS

This work is partially supported by CNPq under grants 301497/2006-0 and 311794/2006-8.

REFERENCES

- Argyris, C., & Schön, D. A. (1995). *Organizational Learning II: Theory, Method, and Practice*. New Jersey, NJ: FT Press.
- Breitman, K. K., Barbosa, S. D. J., Casanova, M. A., & Furtado, A. L. (2007). Conceptual modeling by analogy and metaphor. *Proceedings of CIKM 2007*.
- Barbosa, S. D. J., Breitman, K. K., Furtado, A. L., & Casanova, M. A. (2007). Similarity and analogy over application domains. *Proceedings of SBBD 2007*.
- Barbosa, S. D. J., & de Souza, C. S. (2001). Extending software through metaphors and metonymies. *Knowledge-Based Systems*, 14, 15-27.
- Batini, C., Ceri, S., & Navathe, S. (1992). *Conceptual Design – an Entity-Relationship Approach*. New Jersey, NJ: Benjamin Cummings.
- Fauconnier, G., & Turner, M. (1994). *Conceptual projection and middle spaces*. Technical Report 9401, University of California, San Diego.
- Fauconnier, G., & Turner, M. (2002). *The Way We Think*. New York, NY: Basic Books.
- Fikes, R. E., & Nilsson, N. J. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2 (3-4).
- Furtado, A.L., Casanova, M.A., Barbosa, S.D.J., & Breitman, K.K. (2007). *Plot mining as an aid to characterization and planning*. Technical Report MCC 07/07, PUC-Rio.
- Goguen, J. (1999). An Introduction to Algebraic Semiotics, with Application to User Interface Design. In C. Nehaniv (Ed.) *Computation and Metaphor, Analogy and Agents*. Springer-Verlag.
- Holyoak, K., & Thagard, P. (1996). *Mental Leaps*. Cambridge, MA: The MIT Press.
- Imaz, M., & Benyon, D. (2007). *Designing with Blends*. Cambridge, MA: The MIT Press.
- Knight, K. (1989). Unification: "A Multidisciplinary Survey". *ACM Computing Surveys*, Vol. 21, No. 1, March.
- Kramer, P. K. (2002). *Weeding as Part of Collection Development*. ISLMA Report. DuPage Library System.
- Lakoff, G., & Johnson, M. (1980). *Metaphors We Live By*. University of Chicago Press.
- MacLane, S., & Birkhoff, G. (1967) *Algebra*. MacMillan.
- Pree, W. (1995). *Design Patterns for Object-Oriented Software Development*. Boston, MA: Addison-Wesley.
- Slote, S. J. (1997). *Weeding Library Collections: Library Weeding Methods*. Libraries Unlimited.
- Turner, M. (1996). *The Literary Mind*. New York, NY: Oxford University Press.