

## A Catalog of Object Model Transformations

Michael Blaha\* and William Premerlani\*\*

\*OMT Associates Inc., Chesterfield, MO 63017 (blaha@acm.org)

\*\*GE Corporate R&D, Schenectady, NY 12301 (premerlani@crd.ge.com)

### Abstract

*The process of software development is gradually achieving more rigor. Proficient developers now construct software indirectly through the abstraction of models. Models allow a developer to focus on the essential aspects of an application and defer details. Transformations extend the power of models, as the developer can substitute refinement and optimization of models for tedious manipulation of code. This paper catalogs object modeling transformations that we have encountered in our application work.*

### 1. Introduction

Transformations have been a prominent theme in the literature [1] [2] [3] [4] [5]. At the first two working conferences on reverse engineering there was also much mention and interest in transformations. In principle, there are two kinds of transformations—on function and on data. This paper concerns transformations on data.

The literature explains some of the theory of transformations on data and motivates their use, but surprisingly only lists a small number of specific transformations. This paper presents our list of primitive transformations for object models. We do not claim our list is complete, but it is the most comprehensive list that we have found.

This paper uses the OMT notation [6] for modeling data structure. The OMT object model is essentially an extended Entity-Relationship model. The object model characterizes the static structure of things (objects). It looks at structure in terms of groups of analogous objects (classes), their similarities and differences (generalization), and their important relationships with one another (associations). The appendix summarizes the OMT notation that we use in this paper.

### 2. Transformation Concepts

A *transformation* is a mapping from the domain of object models to the range of object models. You can think of

a transformation as accepting a source object model pattern and yielding a target object model pattern. (A *pattern* is an excerpt of an object model with one or more parameters as placeholders for classes and associations.) You apply a transformation to an object model by aligning the source pattern with the model and then substituting the instantiated target pattern.

Figure 1 shows some transformations and their use. The top portion specifies two transformations—depending on the direction in which you read the diagram.

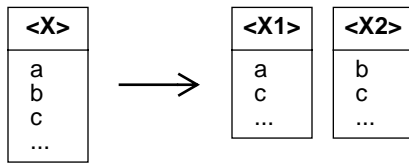
- Read from left to right, Figure 1 shows a transformation for partitioning a class. A designer may choose to split a class ( $X$ ) into two smaller classes ( $X1$ ,  $X2$ ) by apportioning and replicating information.
- Read from right to left, Figure 1 shows a merge transformation. Under some circumstances  $X1$  and  $X2$  can be combined to form  $X$ .

The bottom portion of Figure 1 shows an example for the transformations. A database may store both personal and business information for a person; we can represent the information with a single class or split the information into two classes. Both models are correct; the choice between models would depend on the purpose and scope of an application. If we have much personal and business information, it would be a good idea to separate them. If we have a modest amount of personal and business information, it may be easier just to combine them.

The angle brackets denote a placeholder for a class that must be instantiated when the transformation is used. Thus we substitute *Person* for  $X$  in our Figure 1 example.

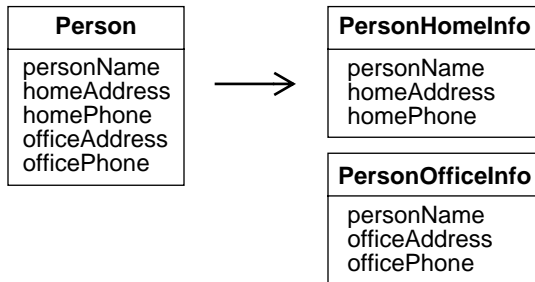
We use an arrow notation to indicate whether a transformation gains or loses information. A transformation in the direction of an arrow yields a model with equal or less semantic content. A designer need not add information to allow a transformation to proceed in the direction of an arrow. Conversely, transformation in a direction without an arrow yields a model with increased semantic content. The designer must assert some additional information to allow

**Transformations:**



{Note: Each attribute and association for X is replicated or apportioned. If X is a subclass in a generalization, then both X1 and X2 become subclasses in the generalization. If X is a superclass then the subclasses multiply inherit from X1 and X2.}

**Example:**



**Figure 1 Sample transformations and example**

such a transformation to proceed. A bidirectional arrow denotes an equivalence transformation.

We classify transformations into several categories depending on whether they gain or lose information.

- **Equivalence transformation.** The source and target object models describe sets of instances where each source instance corresponds to exactly one target instance. Similarly, each target instance corresponds to exactly one source instance. An equivalence transformation may lose or gain some incidental information from the object model, such as association names or role names.
- **Information-losing transformation.** The source model is more constrained than the target model. All instances of the source model can be described by the target model; some instances of the target model cannot be described by the source model. The transformation in Figure 1 loses information when applied from left to right (partition transformation). The X class has an implicit relationship between the attributes that may not be enforced with the separate classes.
- **Information-gaining transformation.** The source model is less constrained than the target model. All instances of the target model can be described by the source model; some instances of the source model cannot be described by the target model. The transformation in Figure 1 gains information when applied from right to left (merge transformation). The software developer must assert some

correspondence between X1 and X2 for the merge transformation to occur.

All transformations have a *local effect* on a model. A transformation only affects the constructs that align with the source pattern and the associations and generalizations that connect to these classes.

**3. Transformations and Software Engineering**

Transformations have several purposes for software engineering.

- **Forward engineering.** By applying a series of transformations, a designer can simplify and optimize a model. Transformations allow you to consider implementation needs in a language and database independent manner.
- **Reverse engineering.** The reverse engineer takes implementation artifacts and deduces the underlying logical intent. Transformations allow you to explicitly note assumptions while processing implementation artifacts.
- **Schema integration.** Transformations are useful for mapping an enterprise model to application models. Such mappings are a prerequisite for allowing common data to flow between applications.
- **Schema evolution.** Transformations also facilitate data conversion. Each transformation can be associated with commands that migrate data from the source to the target.

Most software development tools currently provide little support for transformations. (Reference [4] describes an exception.) Nevertheless, it is still useful to think in terms of transformations. Eventually the tools will become more capable.

**4. Catalog of Primitive Transformations**

A *primitive transformation* is a transformation that cannot be decomposed into lesser transformations. This section presents the following primitive transformations. We do not claim our list is complete and our presentation is informal.

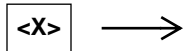
- Transformations on a single construct type.
  - T1. Remove or add a construct.
  - T2. Assert a construct is not derived or derived.
  - T3. Degrade or restrict multiplicity.
  - T4. Transform a multi-valued attribute.
  - T5. Reorder attributes.
  - T6. Transform an enumeration attribute.
  - T7. Partition a construct or merge constructs.
  - T8. Compose associations.
- Transformations on multiple construct types.
  - T9. Combine associated classes or partition class.
  - T10. Move an attribute across an association.
  - T11. Move an attribute across generalization.

- T12. Move an association across generalization.
- Modifying inheritance.
  - T13. Remove or add a subclass.
  - T14. Push subclass information up or specialize.
  - T15. Fragment multiple inheritance.
  - T16. Factor multiple inheritance.
- Conversions.
  - T17. Convert generalization to/from exclusive-or associations.
  - T18. Convert qualifier to/from link attribute.
  - T19. Convert link attribute to/from object attribute.
  - T20. Convert association to/from class.

#### 4.1 Remove or add a construct

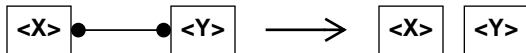
The transformations in T1 modify a model by removing or adding a construct. You may remove or add a class, association, object attribute, or link attribute. These transformations are especially helpful during reverse engineering, because you should rigorously track such decisions.

##### (a) Class:



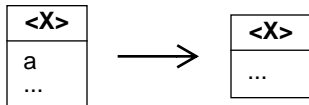
{Note: X has no attributes, associations, superclasses, or subclasses.}

##### (b) Association:



{Note: The association can be n-ary or an aggregation. The association cannot have any qualifiers or link attributes. Any multiplicity is possible.}

##### (c) Attribute:



{Note: T1c applies for both object attributes and link attributes.}

#### T1 Remove or add a construct

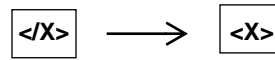
#### 4.2 Assert a construct is derived or not derived

The transformations in T2 allow you to mark a construct as not derived or derived. These transformations are helpful for fine adjustments to a model.

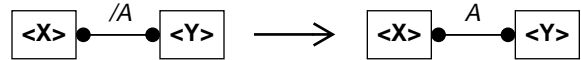
#### 4.3 Degrade or restrict multiplicity

As T3 shows, you may degrade or restrict the multiplicity of an association, object attribute, or link attribute. For example, during reverse engineering we may have

##### (a) Class:

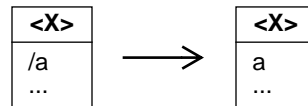


##### (b) Association:



{Note: The association can be n-ary or an aggregation.}

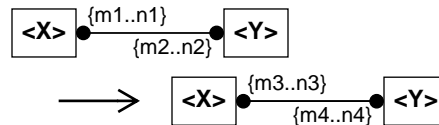
##### (c) Attribute:



{Note: T2c applies for both object attributes and link attributes.}

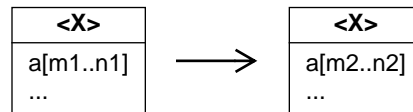
#### T2 Assert a construct is not derived or derived

##### (a) Association multiplicity:



{Note:  $m1 \geq m3$ ,  $n1 \leq n3$ ,  $m2 \geq m4$ ,  $n2 \leq n4$ . The associations can be n-ary or an aggregation and have qualifiers and link attributes.}

##### (b) Attribute multiplicity:



{Note:  $m1 \geq m2$ ,  $n1 \leq n2$ . T3b applies for both object attributes and link attributes.}

#### T3 Degrade or restrict multiplicity

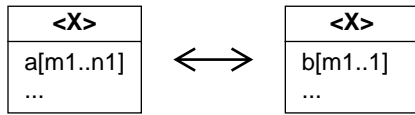
some prior understanding of an application and be able to reduce the multiplicity of an association from many-to-many to one-to-many. The notation  $m1..n1$  denotes a multiplicity range. (See Appendix.)

#### 4.4 Transform a multi-valued attribute

T4 shows transformations for a multi-valued attribute. You may concatenate a multi-valued attribute into a single attribute with a longer length or you may replace a multi-valued attribute with several parallel attributes. The transformations are equivalence transformations.

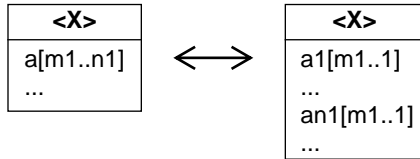
As an example of transformation T4, consider a *streetNumber* attribute of class *Address* that stores street location for a person or company. You could represent *streetNum-*

**(a) Concatenation:**



{Note:  $m1 = 0$  or  $1$ . (length of  $b$ ) =  $n1 * (\text{length of } a)$ . T4a applies for both object attributes and link attributes.}

**(b) Parallel attributes:**



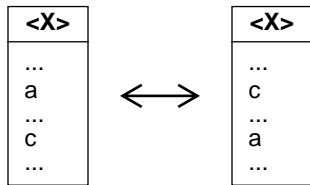
{Note:  $m1 = 0$  or  $1$ . T4b applies for both object attributes and link attributes.}

**T4 Transform a multi-valued attribute**

*ber* as an array of 3 lines of 80 characters each; a single string of 240 characters; or 3 parallel attributes, *streetNumber1*, *streetNumber2*, and *streetNumber3*, each of which are 80 characters long.

**4.5 Reorder attributes**

T5 interchanges two attributes. You may change the order of object attributes and link attributes. The transformation is an equivalence transformation and can be useful for making a model more understandable.



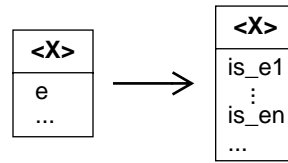
{Note: T5 applies for both object attributes and link attributes.}

**T5 Reorder attributes**

**4.6 Transform an enumeration attribute**

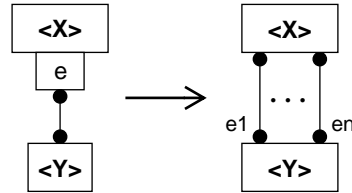
T6 shows transformations for enumeration attributes. T6a substitutes multiple boolean attributes for an object attribute or link attribute that is an enumeration. For example, you could replace the enumeration attribute *color* with values *red*, *green*, and *blue* with the attributes *isRed*, *isGreen*, and *isBlue*. The transformation from left to right loses information, because the boolean attributes are not constrained to be mutually exclusive. T6b transforms an association with an enumerated qualifier to/from multiple associations.

**(a) Enumeration attribute**



{Note:  $e$  is an enumeration with values  $e1..en$ . T6a applies for both object attributes and link attributes.}

**(b) Enumeration qualifier**



{Note:  $e$  is an enumeration with values  $e1..en$ . If all associations have the same multiplicity then the transformation is an equivalence transformation. The associations may be aggregations, *but X* must be the assembly or component for all the aggregations.}

**T6 Transform an enumeration attribute**

**4.7 Partition a construct or merge constructs**

You may also use transformations to partition a construct or merge constructs as T7 shows. In general a merge gains information; the designer must assert that the merge is logically meaningful before it can be allowed to occur. We presented T7a and an example earlier as Figure 1. Figure 2 and Figure 3 illustrate T7b and T7c respectively.

**4.8 Compose associations**

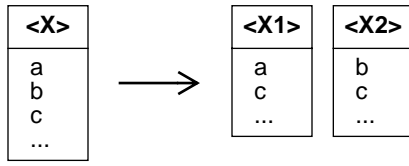
You can form a new association by composing other associations. You can then add the composed association as a derived association to an object model. Or you may remove a constituent association as we do for T8 so the composed association is no longer derived. T8 is similar to the partition/merge transformations in T7b. The difference is that the forward transformation in T7b partitions the set of links; every link on the left side is described by one of the associations on the right. In contrast T8 combines two or more links to form a new link.

Figure 4 shows an example of T8. A company has many divisions, each of which employs many persons. You could instead represent the persons as directly working for the company.

**4.9 Combine associated classes or partition class**

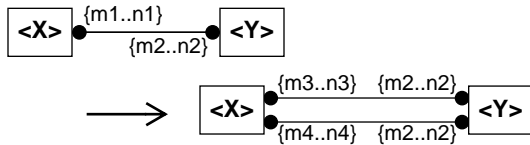
As T9a shows, you can sometimes combine two classes and an intervening association. T9b is an equivalence

(a) Class:



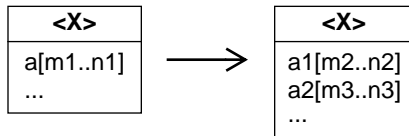
{Note: Each attribute and association for X is replicated or apportioned. If X is a subclass in a generalization, then both X1 and X2 become subclasses in the generalization. If X is a superclass then the subclasses multiply inherit from X1 and X2.}

(b) Association:



{Note:  $m1 \geq m3 + m4$ ,  $n1 \leq n3 + n4$ . The associations can be n-ary or aggregations and have qualifiers and link attributes.}

(c) Attribute:



{Note:  $m1 \geq m2$ ,  $m1 \geq m3$ ,  $n1 \leq n2$ ,  $n1 \leq n3$ . T7c applies for both object attributes and link attributes.}

T7 Partition a construct or merge constructs

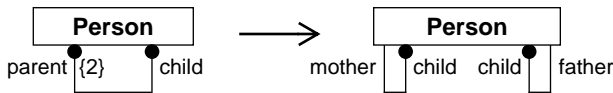


Figure 2 Example of T7b

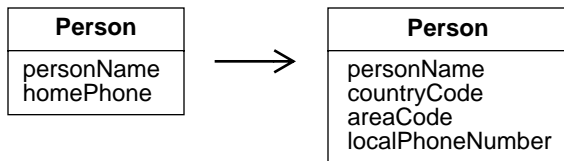
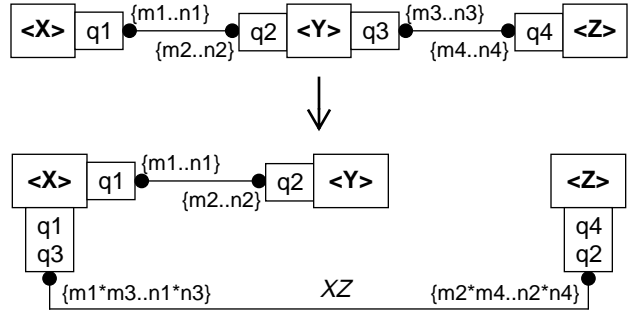


Figure 3 Example of T7c

transformation for the special case where X has one attribute, one association, and no generalizations.

Figure 5 illustrates T9a. An address is located within a city and a state. We could choose to simplify the model and combine cityName with addressNumber.



{Note: An association may be formed from an arbitrary number of associations. The associations can be n-ary, may have link attributes, and can be aggregations.}

T8 Compose associations

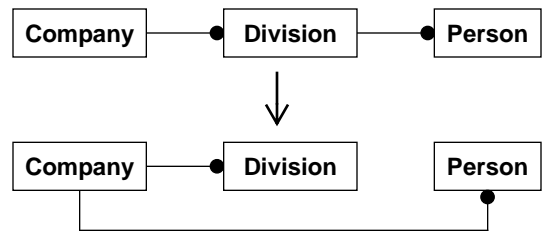
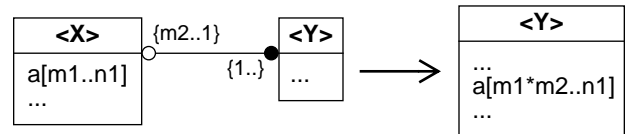


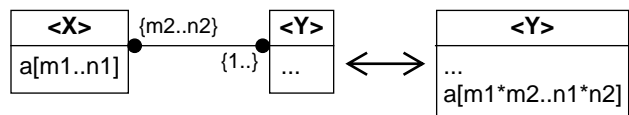
Figure 4 Example of T8

(a) Basic transformation



{Note: The transformation to the right adds all attributes, associations, and generalizations of X to Y. The association cannot have any qualifiers or link attributes.}

(b) Special case: X has one attribute, one association, and no generalizations.



T9 Combine associated classes or partition class

4.10 Moving an attribute across an association

You can move an attribute from one class to another as T10 shows. Alternatively, you can just add the migrated attribute and mark it as derived. The transformation is an equivalence transformation if  $m1$  and  $n2$  equal one and the association is not qualified.

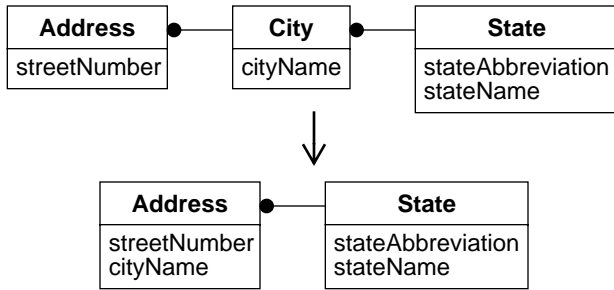
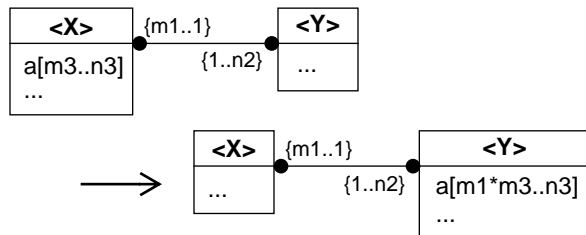


Figure 5 Example of T9



{Note:  $m1 = 0$  or  $1$ . The association may be an aggregation or have qualifiers and link attributes. The transformation is an equivalence transformation if  $m1$  and  $n2$  equal  $1$ .}

T10 Move an attribute across an association

#### 4.11 Moving attributes across generalization levels

Transformations T11a move an attribute down or up a generalization level. T11b is a special case where only one attribute is promoted to the superclass. Figure 6 illustrates T11a. A *FinancialInstrument* can be an *Asset* or a *Portfolio*. We can shift attributes across generalization levels during reverse engineering or as part of optimizing a model.

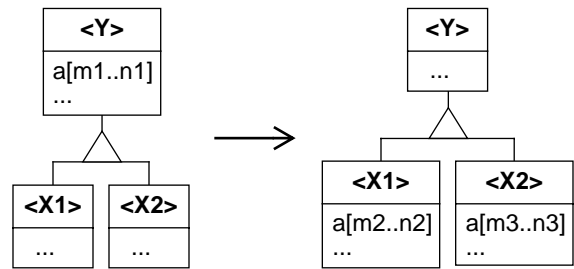
#### 4.12 Moving associations across generalization levels

Transformations T12a move an association down or up a generalization level. T12b is a special case where only one association is promoted to the superclass. Figure 7 illustrates T12a. A distribution list for electronic mail may contain many addresses; each address in turn may be the electronic mail address of a person or another distribution list.

#### 4.13 Remove or add a subclass

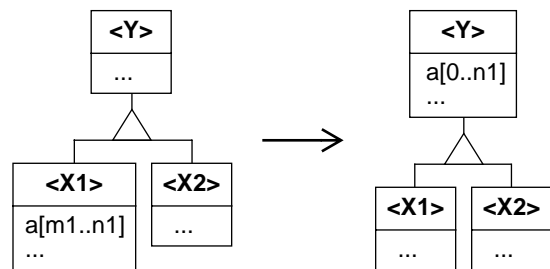
T13 removes or adds a subclass. By successively applying the left-to-right transformation, you can push all the information of a superclass down to its subclasses. Thus if *A* is the superclass of *C* and *C* is the superclass of *D* and *E*, two successive applications of T13 can collapse the inheritance hierarchy and make *A* a direct superclass of *D* and *E*.

(a) Basic transformation



{Note: The generalization must be exhaustive. There may be an arbitrary number of subclasses.  
 $m1 \geq m2, m1 \geq m3, n1 \leq n2, n1 \leq n3$ }

(b) Special case: Move an attribute for one subclass



{Note: There may be an arbitrary number of subclasses.}

T11 Move an attribute across generalization

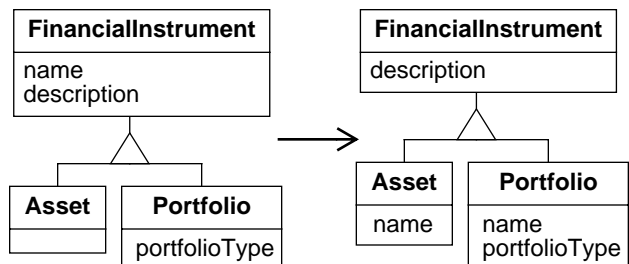
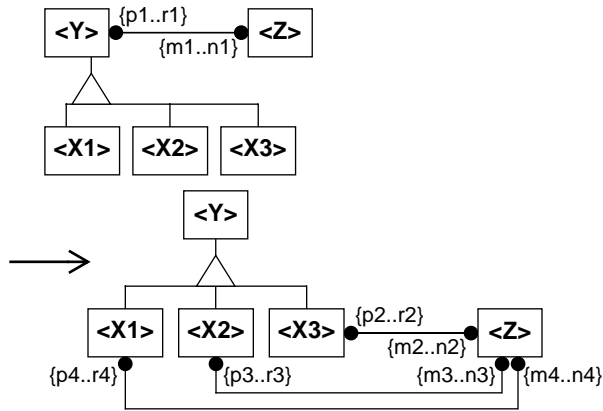


Figure 6 Example of T11a

#### 4.14 Push subclass information up or specialize

T14 can eliminate a generalization level by pushing the attributes, associations, and generalizations of subclasses up to the superclass. Sometimes it is helpful to eliminate generalization by manipulating the object model (as opposed to eliminating generalization during implementation). The transformation in T14 is an equivalence transformation for the special case where the subclasses have no attributes or associations.

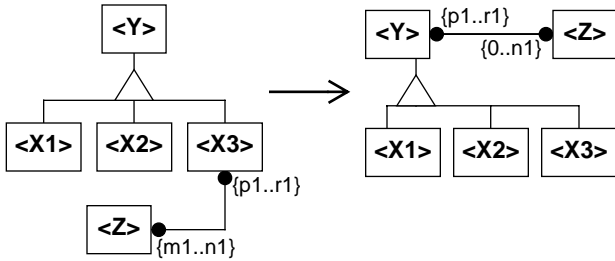
(a) Basic transformation



{Note: The generalization must be exhaustive. There may be an arbitrary number of subclasses. Class Z may be one of the subclasses.  $p1 \geq p2 + p3 + p4$ .  $q1 \leq q2 + q3 + q4$ .  $m1 \geq m2$ ,  $m1 \geq m3$ ,  $m1 \geq m4$ .  $n1 \leq n2$ ,  $n1 \leq n3$ ,  $n1 \leq n4$ . All associations can be n-ary or aggregations and may have qualifiers and link attributes. All associations must have the same qualifiers and link attributes for a transformation to the left.}

(b) Special case:

Move an association for one subclass



{Note: There may be an arbitrary number of subclasses. Class Z may be one of the subclasses. The association can be n-ary or an aggregation and may have qualifiers and link attributes.}

T12 Move an association across generalization

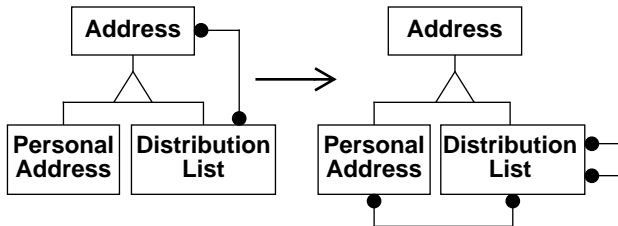
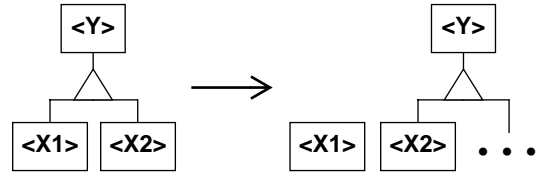


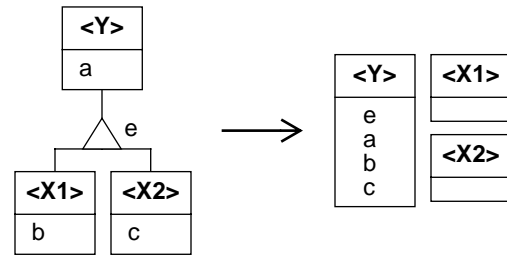
Figure 7 Example of T12a



{Note: The source generalization may or may not be exhaustive. There may be an arbitrary number of subclasses. For a transformation to the right, all the attributes, associations, and superclasses of Y must be added to X1. For a transformation to the left, X1 must have attributes, associations, and superclasses that match those of Y.}

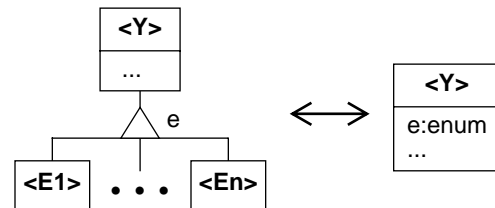
T13 Remove or add a subclass

(a) Basic transformation



{Note: The source generalization may or may not be exhaustive. There may be an arbitrary number of subclasses.}

(b) Special case: All subclasses have no attributes and no associations

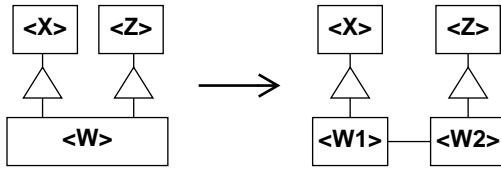


{Note: e is an enumeration domain with values E1..En. The subclasses may not have any attributes, associations, or other generalizations.}

T14 Push subclass information up or specialize

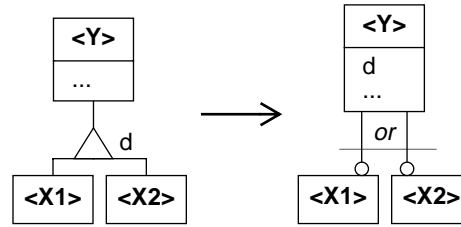
4.15 Multiple inheritance work-arounds

Similarly, we normally prefer to address work-arounds for multiple inheritance during implementation of the object model. But sometimes it is helpful to eliminate multiple inheritance by transforming an object model. T15 and T16 show two transformations: fragmenting and factoring of a subclass. You can also deal with multiple inheritance by converting one of the generalizations to exclusive-or associations using the transformation in T17.



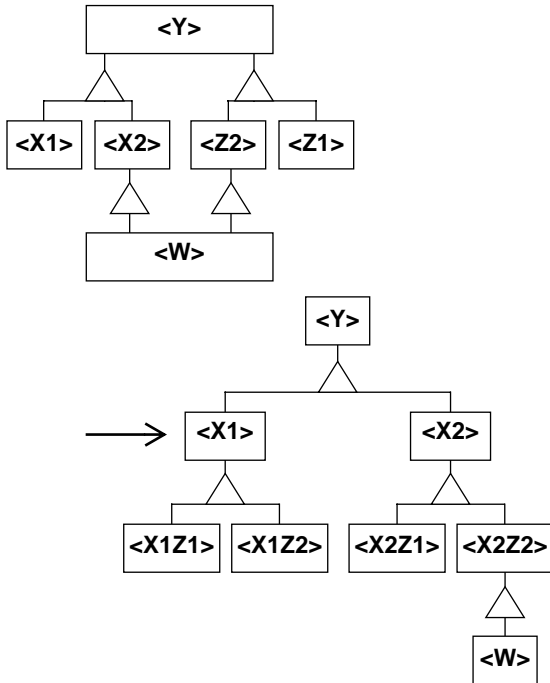
{Note: The generalizations may or may not be exhaustive. The generalizations may have an arbitrary number of subclasses. Attributes, associations, and other generalizations should be apportioned between W1 and W2.}

**T15 Fragment multiple inheritance**



{Note: The generalization must be exhaustive. There may be an arbitrary number of subclasses. The exclusive-or associations may not have any qualifiers or link attributes. T17 is an equivalence transformation if there is a single subclass.}

**T17 Convert generalization to/from exclusive-or associations**

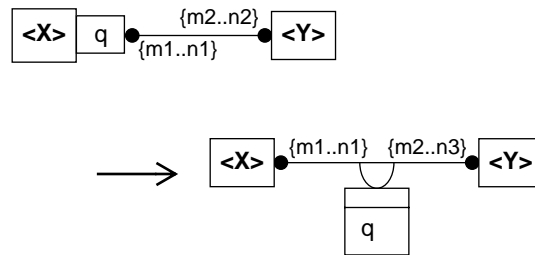


{Note: The generalizations may or may not be exhaustive. The generalizations may have an arbitrary number of subclasses. Classes X1Z1 and X2Z1 have the same attributes, associations as class Z1. Classes X1Z2 and X2Z2 have the same attributes and associations as class Z2.}

**T16 Factor multiple inheritance**

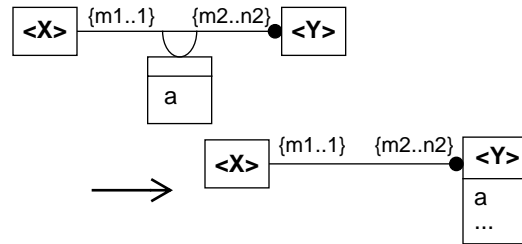
**4.16 Conversions**

Our last category of transformations deals with conversions from one kind of object modeling construct to another. T17 converts a generalization to/from exclusive-or associations. T18 converts a qualifier to/from a link attribute. T19 converts a link attribute to/from an object attribute. Consequently the composition of transformations in T18 and T19 can convert a qualifier to an object attribute. T20 converts an association to/from a class. In T20 the transformation from left to right loses the derivation of identity for the association class.



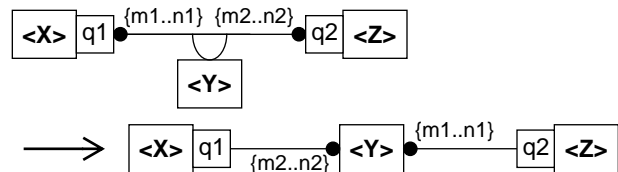
{Note: n2 < n3. n2 is often 1. The association may be an aggregation.}

**T18 Convert qualifier to/from link attribute**



{The association may be an aggregation.}

**T19 Convert link attribute to/from object attribute**



{Note: The association can be n-ary.}

**T20 Convert association to/from class**

Figure 8 illustrates successive application of transformations T19 and T18 during reverse engineering. We used these transformations when reverse engineering the data dictionary of a relational database. First we apply T19 to convert *table#* from an object attribute to a link attribute. Then we apply T18 to convert *table#* to a qualifier.

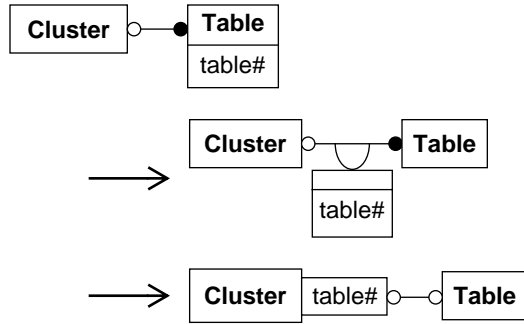


Figure 8 Example of T19 and T18

### 5. Conclusions

Transformations are a key concept for improving the rigor of software engineering. Transformations allow the developer to perform design while remaining within the realm of models. This paper has listed the transformations on data that we have discovered during our industrial work. Our list is more extensive than any we have seen in the literature. Our presentation has been informal, but nevertheless we believe this paper to be an important contribution to the literature.

One of the most promising areas for further work is to deepen the theoretical base—develop a more formal treatment of transformations and better document the catalog of transformations. We have encountered two difficulties with such a formalism. First of all, the topic of transformations is “heavy” reading; a formal mathematical treatment will be even more difficult to read. Secondly, a full theory of transformations is intrinsically complex. We find it difficult to fully specify the preconditions, postconditions, and invariants for transformations; it is easy to overlook details. Also we are not sure how to define completeness and prove that a set of transformations is complete.

### Related Work

Hainaut has written several excellent papers about transformations. Reference [3] describes the TRAMIS database software development tool. TRAMIS is notable for its semantic support for simple transformations. Reference [4] discusses the philosophy for the DB-MAIN tool that is the successor to TRAMIS. DB-MAIN supports forward engineering (the progression through analysis, design, and

implementation), reverse engineering, and evolution of populated databases. DB-MAIN incorporates many transformations and can replay transformations against a model; replay of transformations is helpful when a model changes or the developer wishes to reconsider the evolution of a model.

The papers [3] and [2] describe equivalence transformations, transformations with no loss or gain of information. Hainaut and Rosenthal only use equivalence transformations, so the designer cannot lose information from analysis. In contrast [1] and this paper adopt a more permissive viewpoint; designers may use transformations that add or lose information, but should be aware of the consequences. Transformations that add information are especially needed during reverse engineering to compensate for optimizations, imperfect decisions, and missing information.

[1] presents the most comprehensive list of transformations that we have seen in the literature. This paper presents additional transformations. Figure 9 cross references the transformations of [1] against those in this paper. [1] presents several transformations for mapping between simple attributes and compound attributes that we regard as too minor to list.

Description	Transform number [Batini-92]	Transform number (this paper)
Remove/add class	B1	T1a
Remove/add association	B2	T1b
Remove/add attribute	B4, B5, T6, T7	T1c
Partition/merge class	T3	T7a
Partition/merge association	T4	T7b
Partition/merge attribute	T8	T7c
Compose/decompose association	T5	T8
Combine/partition classes & association	T1	T9
Remove/add subclass	B3	T13
Push subclass up / specialize	Figure 6.9	T14a
Convert enum attribute to/from generalization	T2	T14b

Figure 9 Cross reference of transformations in this paper with [Batini-92]

Chapter 9 of [5] discusses schema transformations for the ORM approach to modeling. ORM is an extension of the NIAM approach to modeling [8].

Reference [7] has further explanation of the benefit of transformations for dealing with database applications.

## References

- [1] Carlo Batini, Stefano Ceri, and Shamkant B. Navathe. *Conceptual Database Design: An Entity-Relationship Approach*. Redwood City, California: Benjamin/Cummings, 1992.
- [2] Arnon Rosenthal and David Reiner. Theoretically Sound Transformations for Practical Database Design. *Proceedings of the Entity-Relationship Conference*, 1987.
- [3] JL Hainaut, M Cadelli, B Decuyper, and O Marchand. TRAMIS: a transformation-base database CASE tool. *Proceedings 5th International Conference on Software Engineering and Applications*, Toulouse, 7-11 December 1992, EC2 Publish., 1992.
- [4] JL Hainaut, V Englebert, J Henrard, JM Hick, and D Roland. Database evolution: the DB-MAIN approach. *Proceedings of the Entity-Relationship Conference*, Manchester, UK, 1994.
- [5] Terry Halpin. *Conceptual Schema and Relational Database Design*, Second edition. Sydney, New South Wales: Prentice Hall Australia, 1995.
- [6] J Rumbaugh, M Blaha, W Premerlani, F Eddy, and W Lorenzen. *Object-Oriented Modeling and Design*. Prentice Hall, Englewood Cliffs, New Jersey, 1991.
- [7] Michael Blaha and William Premerlani. *Object-Oriented Modeling and Design for Database Applications*. Prentice Hall, Englewood Cliffs, New Jersey, to be published in early 1997.
- [8] G Verheijen and J Van Bekkum. NIAM: An information analysis method. Unpublished paper. Information Systems Department, Control Data Corporation, The Netherlands, 1982.

## Appendix. Summary of OMT Notation

Figure 10 summarizes OMT [6] constructs that are included in this paper.

A class describes objects with common attributes, behavior, and semantic intent. A class is denoted by a rectangle. Attributes may be suppressed or displayed in the second portion of the class box. We may specify attribute multiplicity in brackets after the attribute name. The lower value for attribute multiplicity is usually 0 (nulls allowed) or 1 (nulls not allowed). The upper value for attribute multiplicity is usually 1 (single valued) or infinite (many, multi-valued).

Generalization organizes classes by their similarities and differences and is denoted by a triangle. Simple generalization apportions superclass instances among the subclasses. You may optionally list a discriminator attribute next to the generalization triangle; the discriminator indicates the subclass instance that applies for each superclass instance. OMT also supports several forms of multiple inheritance.

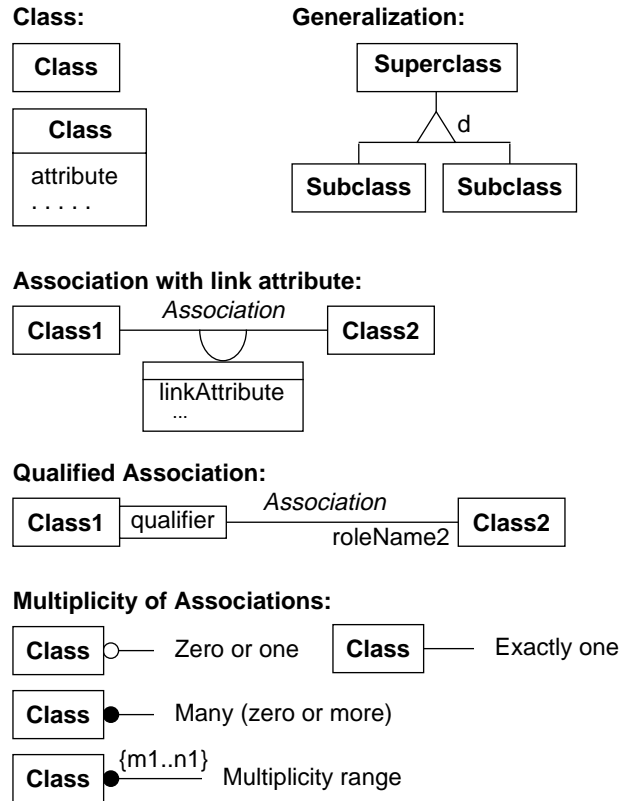


Figure 10 OMT syntax used in this paper

An association relates instances of two or more classes and is indicated by a line. Multiplicity specifies how many instances of one class may relate to an instance of an associated class. We use the multiplicity range in this paper to describe the multiplicity; the lower multiplicity (*m1*) is usually zero or one and the upper multiplicity (*n1*) is usually one or infinite (many).

An association may be qualified in which case the qualifier attribute further refines the multiplicity. For example a directory has many files but the combination of directory and file name corresponds to one file. A role is one end of an association and may be assigned an explicit name as we have shown for class2 of the qualified association. The instances of an association may be described with attributes that we call link attributes. A box attached to the association by a loop denotes a link attribute.