4

Advanced Class Modeling

4.1 Figure A4.1 improves the class diagram in the exercise by changing some associations to aggregations. For a bill-of-material parts hierarchy, all parts except the root must belong to something.

Note that none of the aggregations are composition—the parts do not have coincident lifetime with the assemblies.

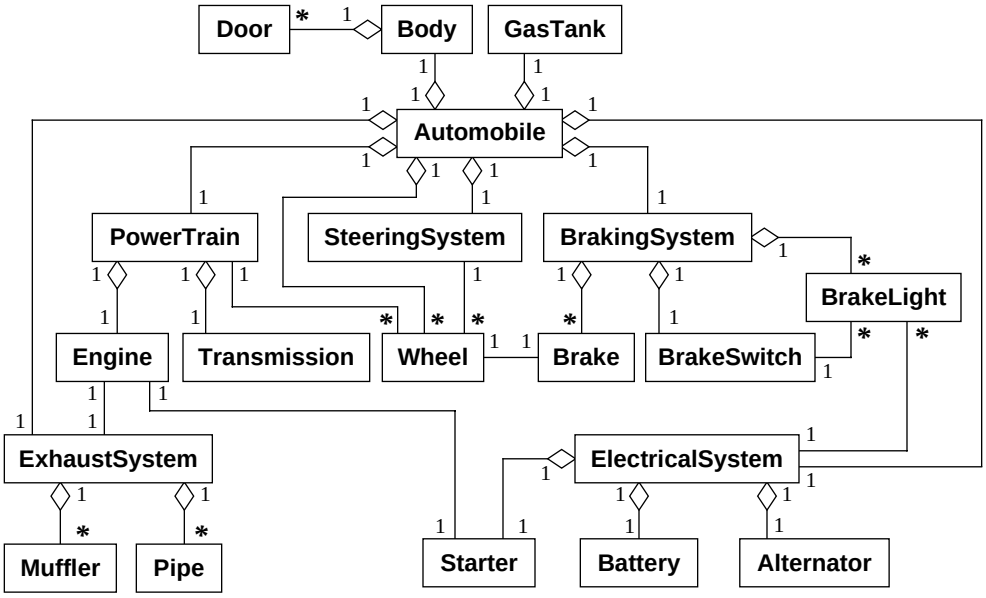

Figure A4.1 Portion of a class diagram of the assembly hierarchy of an automobile

4.2 The class diagram in Figure A4.2 generalizes the classes *Selection, Buffer,* and *Sheet* into the superclass *Collection*. This is a desirable revision. The generalization promotes code reuse because many operations apply equally well to the subclasses. Six aggregation relationships in the original diagram, which shared similar characteristics, have been reduced to two. Finally, the structure of the diagram now captures the constraint that each *Box* and *Line* should belong to exactly one *Buffer*, *Selection,* or *Sheet*.

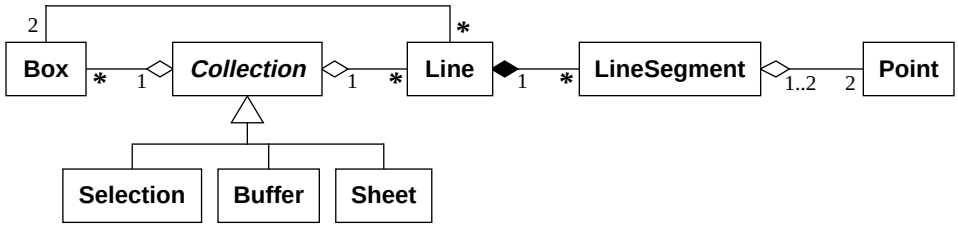

Figure A4.2 Generalization of the classes *Selection*, *Buffer*, and *Sheet* into the class *Collection*

4.3 a. *A country has a capital city.* Association. A capital city and a country are distinct things so generalization certainly does not apply. You could argue that a capital city is a part of a country and thus they are related by aggregation.

b. *A dining philosopher uses a fork.* Association. Dining philosophers and forks are completely distinct things and are therefore not in a generalization relationship. Similarly, neither object is a part of the other and the relationship is not aggregation.

c. *A file is an ordinary file or a directory file.* Generalization. The word "or" often is an indicator of generalization. *File* is the superclass and *OrdinaryFile* and *DirectoryFile* are subclasses.

d. *Files contain records.* Aggregation. The word "contain" is a clue that the relationship may be aggregation. A record is a part of a file. Some attributes and operations on files propagate to their constituent records.

e. *A polygon is composed of an ordered set of points.* Aggregation. The phrase "is composed of" should immediately make you suspicious that there is an aggregation. An ordered set of points is a part of a polygon. Some attributes and operations on a polygon propagate to the corresponding set of points.

f. *A drawing object is text, a geometrical object, or a group.* Generalization. Once again, the word "or" should prompt you to think of generalization. *DrawingObject* is the superclass. *Text*, *GeometricalObject*, and *Group* are subclasses.

g. *A person uses a computer language on a project.* Ternary association. *Person*, *ComputerLanguage*, and *Project* are all classes of equal stature. The association cannot be reduced to binary associations. None of these classes are a-kind-of or a-part-of another class. Thus generalization and aggregation do not apply.

h. *Modems and keyboards are input / output devices.* Generalization. The keyword "are" is the clue. *Modem* and *Keyboard* are the subclasses; *InputOutputDevice* is the superclass.

i. *Classes may have several attributes.* Association or aggregation. It depends on your perspective and the purpose of the model whether aggregation applies.

j. *A person plays for a team in a certain year.* Ternary association. *Person*, *Team*, and *Year* are distinct classes of equal stature.

k. *A route connects two cities.* Association. Either *Route* is a class associated with the *City* class, or *Route* is the name of the association from *City* to *City*.

l. *A student takes a course from a professor.* Ternary association. *Student*, *Course*, and *Professor* are distinct classes of equal stature.

4.4 Figure A4.3 shows a class diagram for a graphical document editor. The requirement that a *Group* contain 2 or more *DrawingObject*s is expressed as a multiplicity of 2..* on *DrawingObject* in its aggregation with *Group*. The fact that a *DrawingObject* need not be in a *Group* is expressed by the zero-one multiplicity.

It is possible to revise this diagram to make a *Circle* a special case of an *Ellipse* and to make a *Square* a special case of a *Rectangle*.

We presume that a *DrawingObject* belongs to a *Sheet* and has a coincident lifetime with it. Similarly, we presume that a *Sheet* belongs to one *Document* for its lifetime. Hence both are composition relationships.

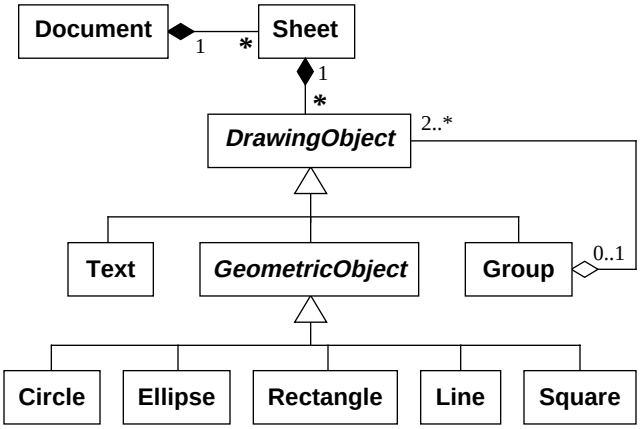

Figure A4.3 Class diagram for a graphical document editor that supports grouping

4.5 Figure A4.4 shows a class diagram with several classes of electrical machines. We have included attributes that were not requested.

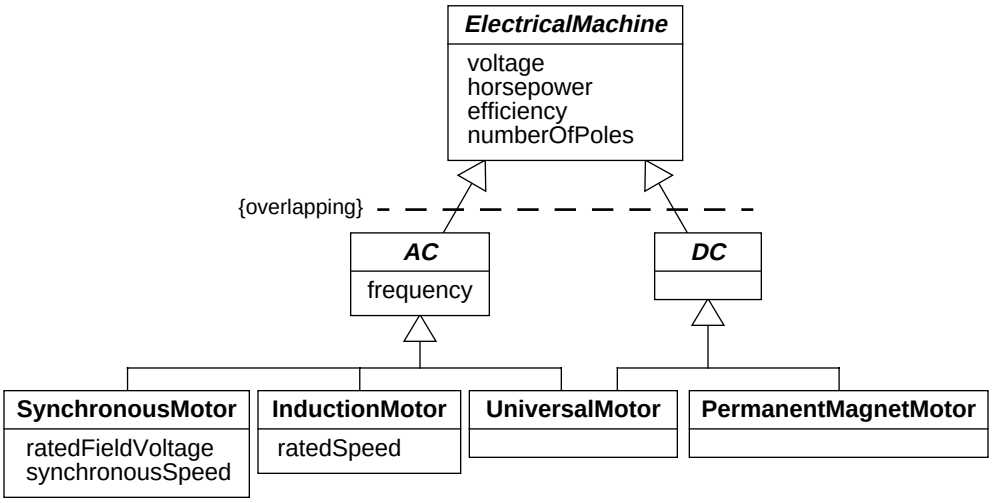

Figure A4.4 Partial taxonomy for electrical machines

4.6 Figure A4.5 converts the overlapping combination of classes into a class of its own to eliminate multiple inheritance. (Instructor's note: you may want to give the students a copy of our answer to the previous exercise.)

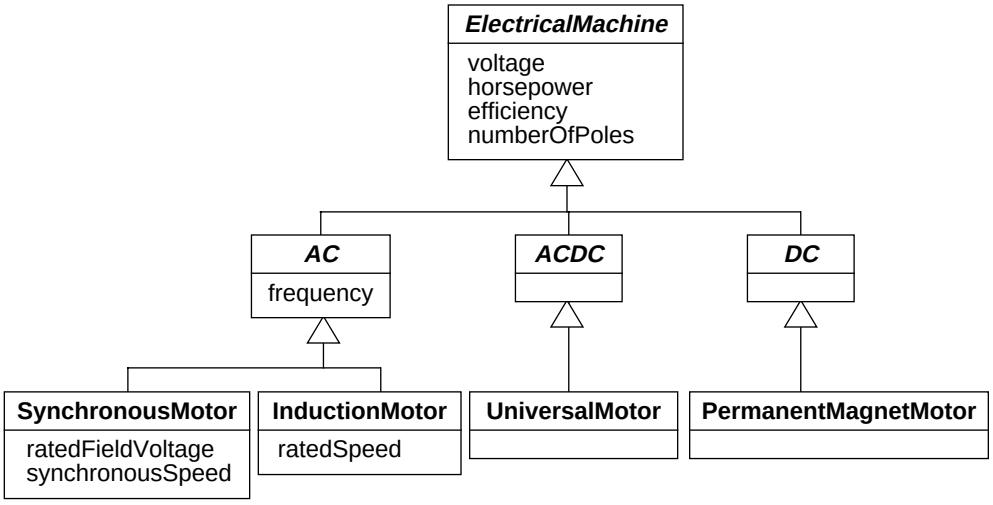

Figure A4.5 Elimination of multiple inheritance

4.7 Figure A4.6 is a metamodel of the following UML concepts: class, attribute, association, association end, multiplicity, class name, and attribute name.

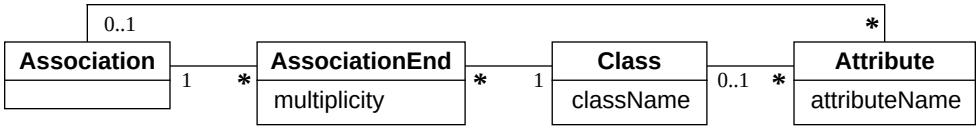

Figure A4.6 Metamodel for some UML concepts

4.8 Figure A4.7 treats the metamodel as a class diagram that can be described by itself. The metamodel in Figure A4.6 is self-descriptive. In general some metamodels are self-descriptive and some are not. (Instructor's note: you may want to give the students a copy of our answer to the previous exercise.)

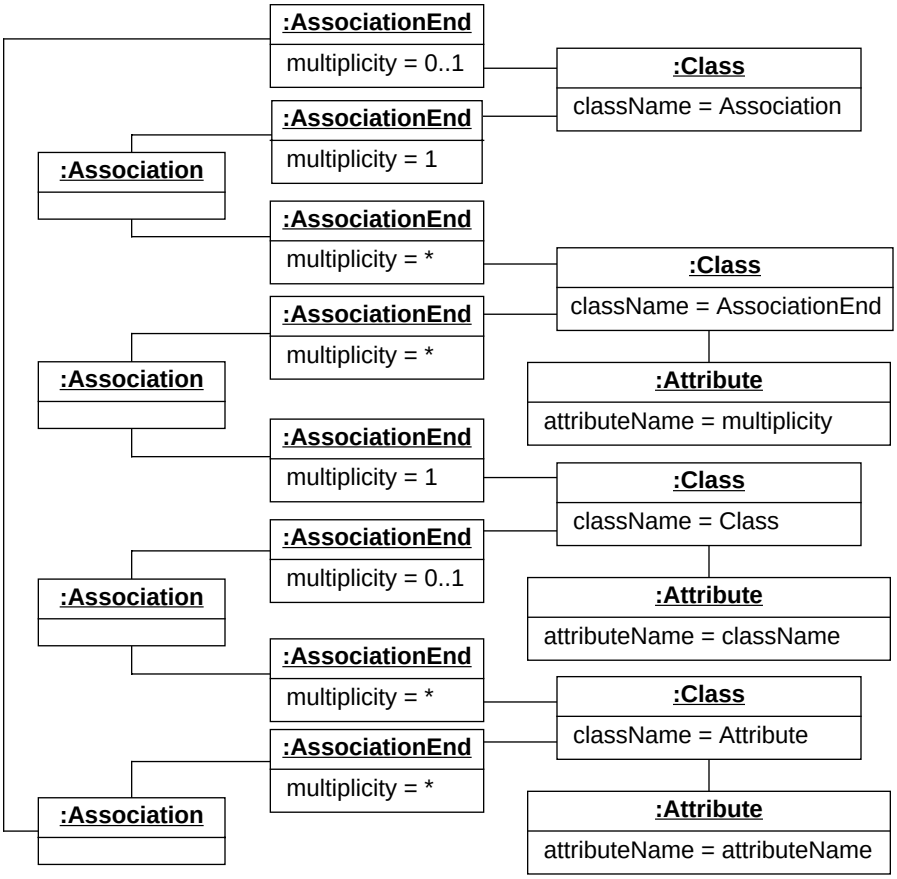

Figure A4.7 Instance diagram for the metamodel

4.9 Figure A4.8 revises the metamodel so that an attribute belongs to exactly one class or association. (Instructor's note: you may want to give the students a copy of our answer to Exercise 4.7.)

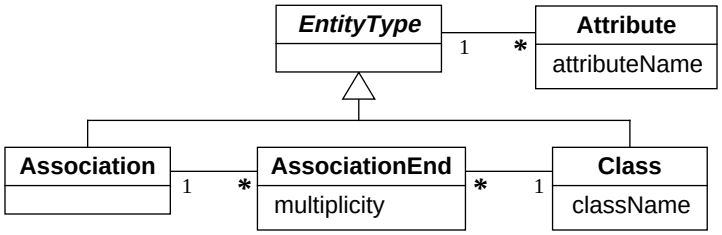

Figure A4.8 UML metamodel where an attribute belongs to exactly one class or association

4.10 The class diagram in Figure E4.3 does support multiple inheritance. A class may have multiple generalization roles of subclass participating in a variety of generalizations.

4.11 To find the superclass of a generalization using Figure E4.3, first query the association between *Generalization* and *GeneralizationRole* to get a set of all roles of the given instance of *Generalization*. Then sequentially search this set of instances of *GeneralizationRole* to find the one with *roleType* equal to *superclass*. (Hopefully only one instance will be found with *roleType* equal to *superclass*, which is a constraint that the model does not enforce.) Finally, scan the association between *GeneralizationRole* and *Class* to get the superclass.

Figure A4.9 shows one possible revision which simplifies superclass lookup. To find the superclass of a generalization, first query the association between *Generalization* and *SuperclassRole.* Then query the association between *SuperclassRole* and *Class* to find the corresponding instance of *Class*.

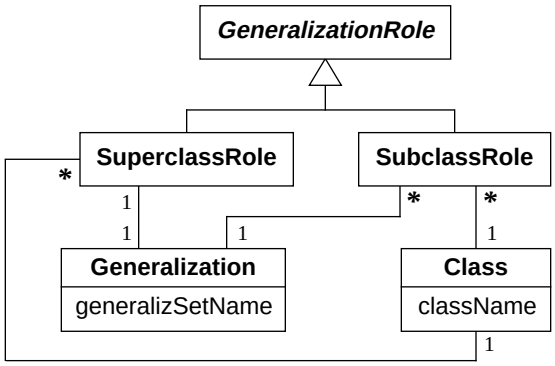

Figure A4.9 Metamodel of generalizations with separate subclass and superclass roles

Figure A4.10 shows another metamodel of generalization that supports multiple inheritance. To find the superclass of a generalization using this metamodel, simply query the *Superclass* association.

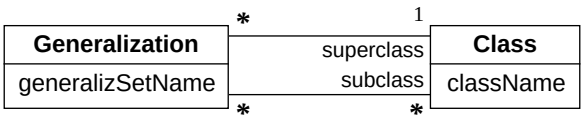

Figure A4.10 Simplified metamodel of generalization relationships

We do not imply that the metamodel in Figure A4.10 is the best model of generalization, only that it simplifies the query given in the exercise. The choice of which model is best depends on the purpose of the metamodel.

The following query finds the superclass given a generalization for Figure E4.3.

- aGeneralization.GeneralizationRole->SELECT(roleType='superclass').Class

The following query finds the superclass given a generalization for Figure A4.9.

- aGeneralization.SuperclassRole.Class

The following query finds the superclass given a generalization for Figure A4.10.

- aGeneralization.superclass

4.12 The metamodel in Figure E4.3 does not enforce the constraint that every generalization has exactly one superclass. In this figure, a *Generalization* has many *GeneralizationRole*s; each *GeneralizationRole* may have a *roleType* of *superclass* or *subclass*. Figure A4.9 and Figure A4.10 are improved metamodels and enforce the constraint that every generalization has exactly one superclass.

4.13 Figure A4.11 is the instance diagram that corresponds to Figure E4.3 and Figure E4.4.

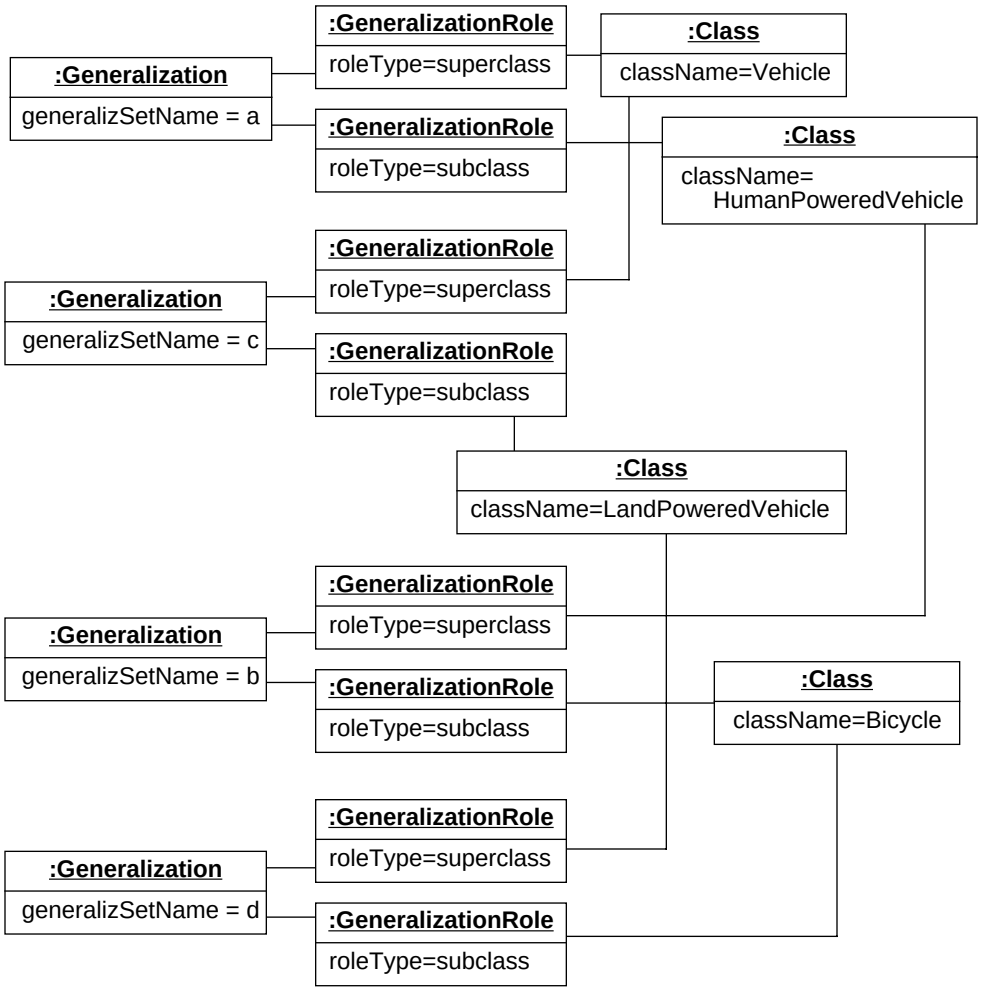

Figure A4.11 Instance diagram for multiple inheritance

4.14 Figure A4.14 shows our answer to the exercise.

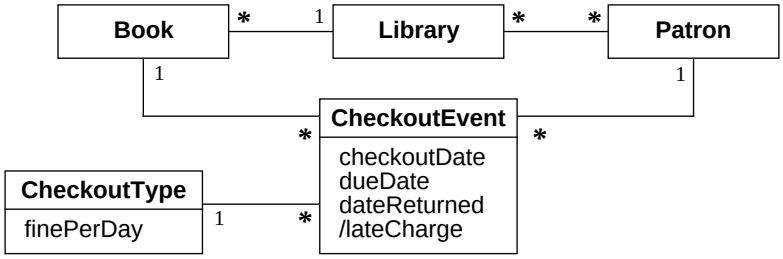

Figure A4.12 Class diagram for library book checkout system

4.15 Figure A4.13 presents one possible metamodel for BNF grammars. A BNF grammar consists of many production rules. Each production rule has many or-clauses which in turn reference many identifiers. An identifier may be a terminal such as a character string or a number or may be a non-terminal in which case it is defined by another production rule.

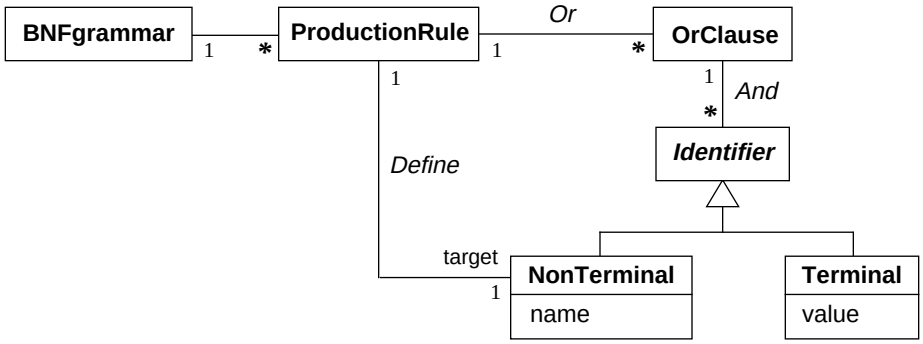

Figure A4.13 Metamodel for BNF grammars

In general, there are at least three ways to express BNF grammars: "railroad" diagrams as shown in the exercise, textual production rules, and state machine diagrams. The metamodel in Figure A4.13 describes the underlying meaning of a BNF grammar and is independent of the manner used to express it. Thus the metamodel applies equally well to each of these three ways of expressing a BNF grammar.

If you were developing software to automate the drawing of BNF "railroad" diagrams, you would need a model of the BNF logic as Figure A4.13 shows and another model to store the corresponding graphic representation. The graphical model would de-

scribe the size of rectangles and circles, the placement of geometric shapes, and the stopping and starting position for lines and arrows.

4.16 The simple class model in Figure A4.14 is sufficient for describing the given recipe data.

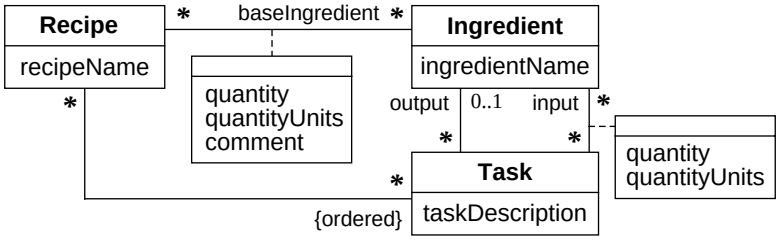

Figure A4.14 A simple class model for recipes

4.17 Figure A4.15 shows our initial solution to the exercise—merely adding an association that binds original ingredients to substitute ingredients. This model has two flaws.

The first problem is that the model awkwardly handles interchangeable ingredients. For example, in some recipes you can freely substitute butter, margarine, and shortening for each other. Figure A4.15 would require that we store each possible pair of ingredients. Thus we would have the following combinations of original and substitute ingredients—(butter, margarine), (butter, shortening), (margarine, butter), (margarine, shortening), (shortening, butter), and (shortening, margarine).

The second problem is that the substitutability of ingredients does not always hold, but can depend on the particular recipe.

Figure A4.16 shows a better class model that remedies both flaws.

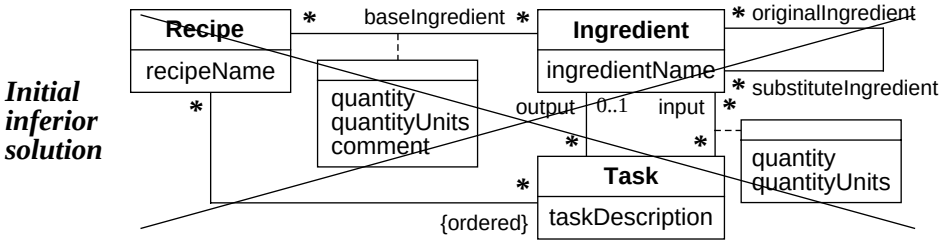

Figure A4.15 Initial class model for recipes with alternate ingredients

4.18 Figure A4.17 shows a class model for the NASAA form. We could have made *Recommendation* an association class, but we promoted it to a class to handle the unusual situation where a broker has multiple suggestions for a security in the same call. Your model could vary a bit from ours, depending on the precise interpretation of the form.

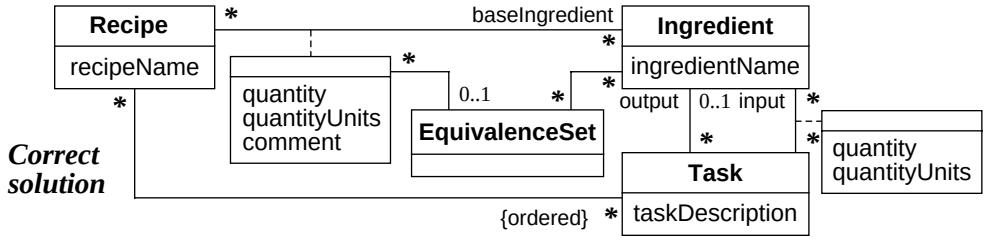

Figure A4.16 Correct class model for recipes with alternate ingredients

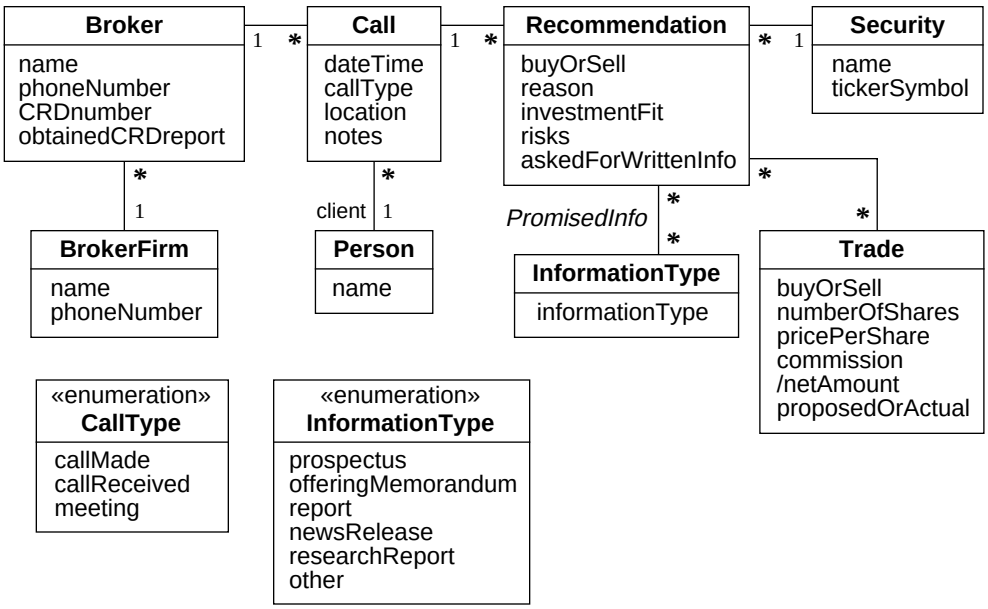

Figure A4.17 Class diagram for NASAA form

4.19 Figure A4.18 shows an inferior answer.

- *meaningType* has values *normal*, *colloquial*, and *slang*.
- *grammarType* has values *noun*, *verb*, *adjective*, *adverb*, and *preposition.*
- The words in a dictionary are alphabetically ordered. The meanings for a word are ordered by frequency of usage.
- The meaning of a word depends on the word and the grammar type.

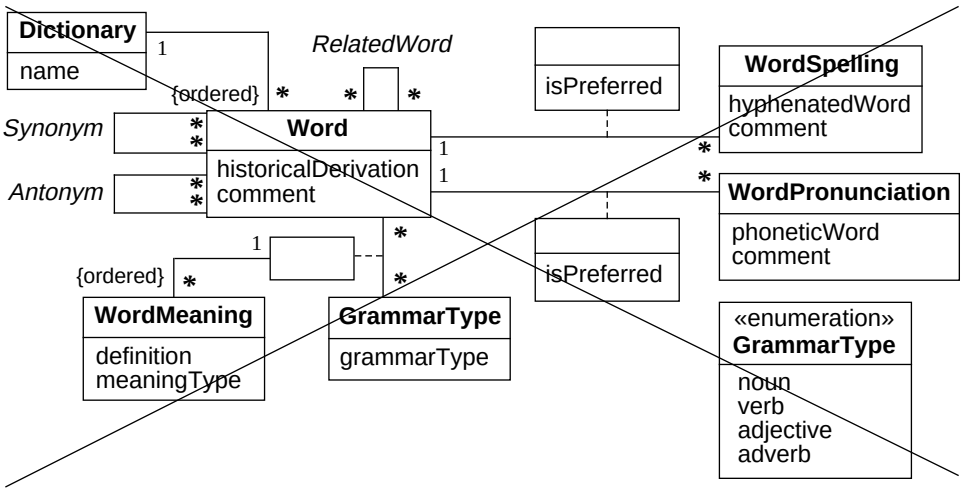

Figure A4.18 An inferior model for a dictionary

The problem with the *RelatedWord*, *Synonym*, and *Antonym* associations is their symmetry. We must store each possible pair of combinations. Figure A4.19 shows a better model.

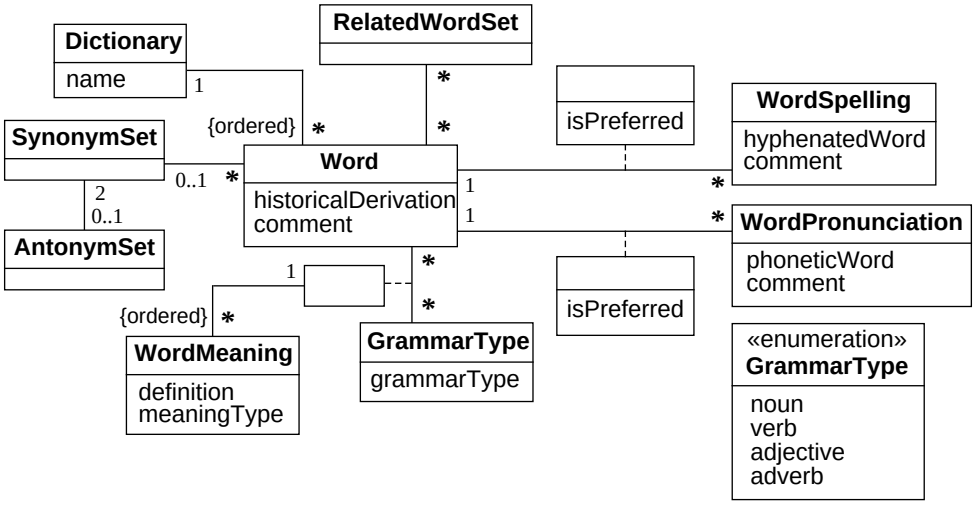

Figure A4.19 A better model for a dictionary