Uml2Rdbms Example

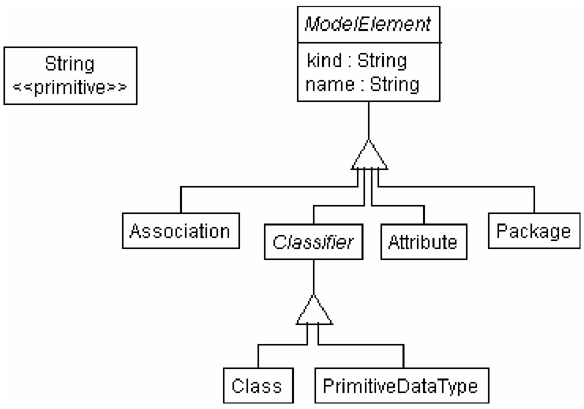
1 Introduction

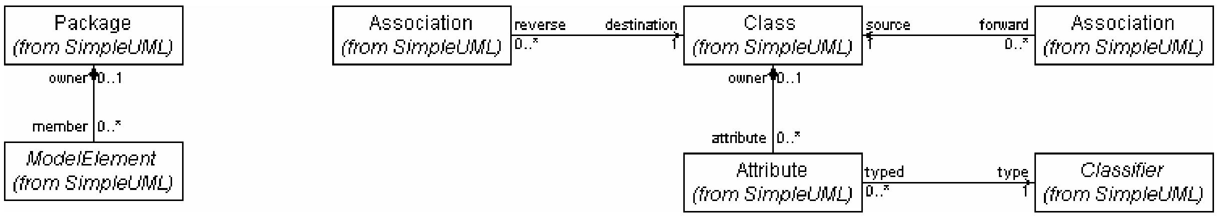
This example is an quick implementation of the UML to RDBMS discussion example on the omgqvt mailing list to demonstrate the UMLX concrete graphical syntax. It requires no off-sheet text (and could use slightly less on-sheet text by preserving attribute names graphically, rather than constraining them with OCL).

The set problem is: “A class maps on to a single table. A class attribute of primitive type maps on to a column of the table. Attributes of a complex type are drilled down to the leaf-level primitive type attributes; each such primitive type attribute maps onto a column of the table. An association maps on to a foreign key of the table corresponding to the source of the association. The foreign key refers to the primary key of the table corresponding to the destination of the association."

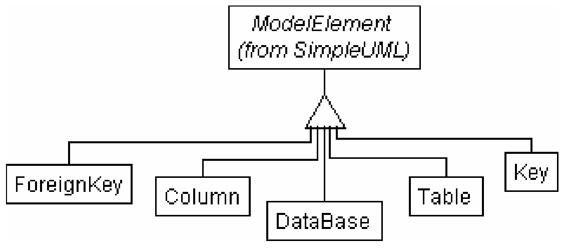
2 Information Models

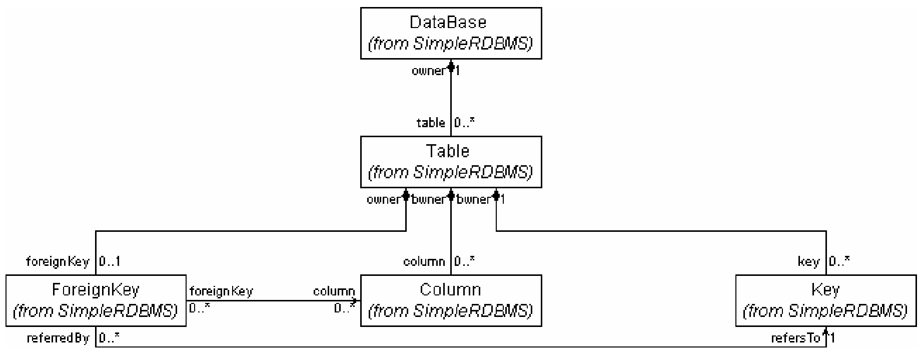
2.1 SimpleUML





2.2 SimpleRDBMS

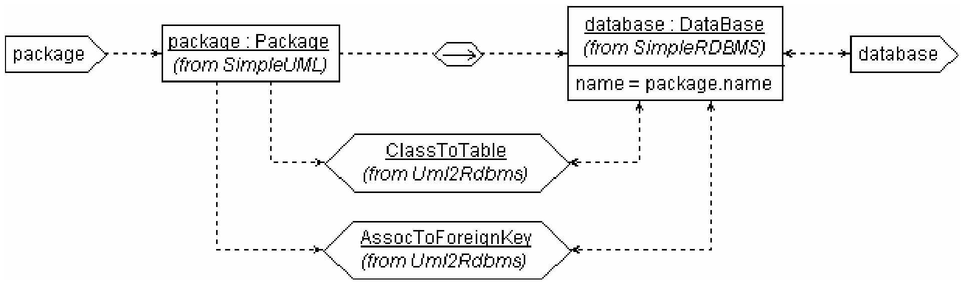




3 Transform Models

UMLX transformations must be invoked in a context, so further top level diagram may be needed to traverse ModelElement hierarchy from the model root to the Package. The extra elaboration required to handle multiple packages and package hierarchies is beyond the scope of the set problem, as is the extra level of transformation to drill down to inherited attributes[1].

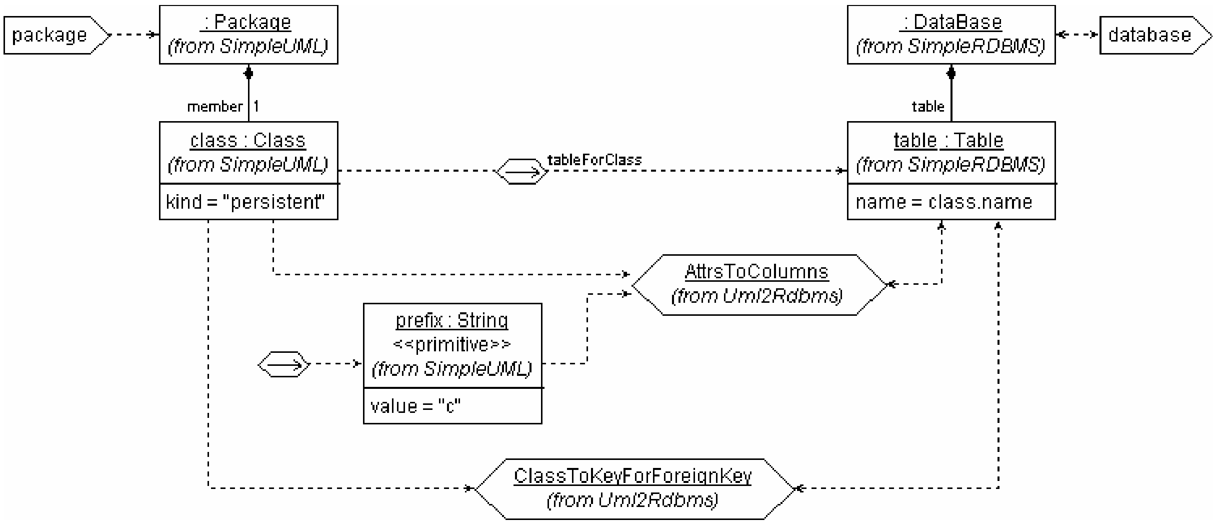
3.1 Uml2Rdbms.PackageToDataBase



Since the two sub-transformations create multiple tables we need a parent context to contain the generated multiplicity.

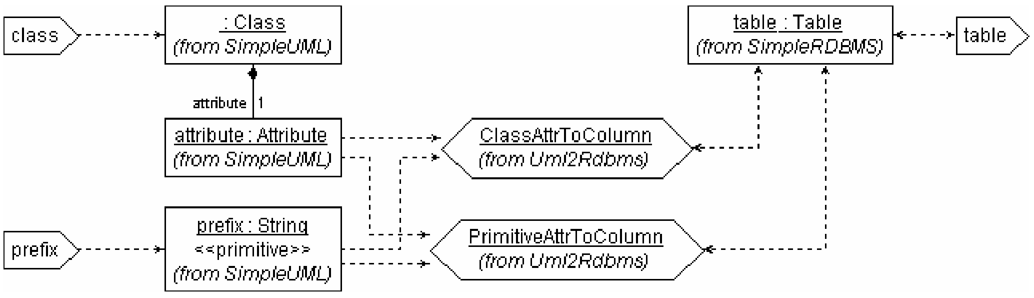
[1] An extra recursion from ClassToTable and ClassAttrToTable is required, aided by a list (a MultiInstance) of derived members if occluded attributes are legal and to be suppressed.

3.2 Uml2Rdbms.ClassToTable



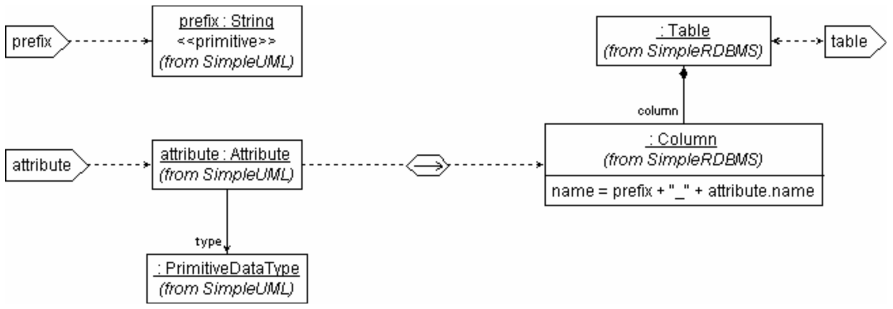
Each class contributes a table with identified as `tableForClass(class)`. The hierarchical column name prefix is seeded with a 'c'. Two sub-transformations are applied within the context of the class and evolved table.

3.3 Uml2Rdbms.AttrsToColumns



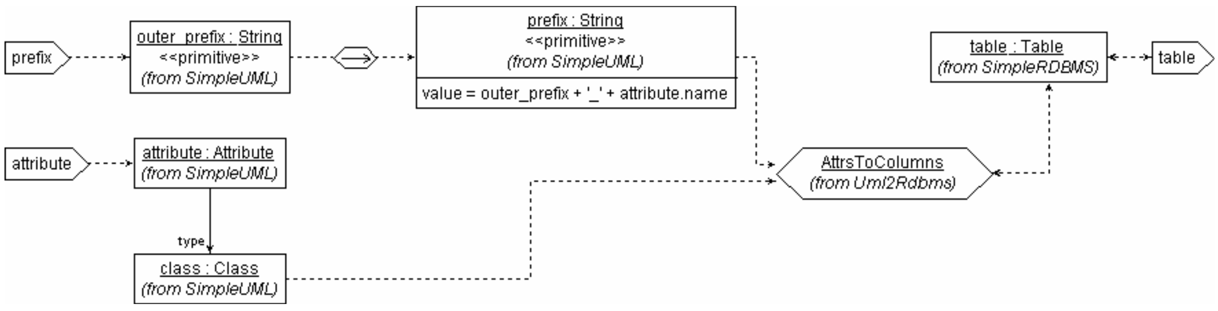
Each attribute contributes one or more columns via the appropriate transformation strategy.

3.4 Uml2Rdbms.PrimitiveAttrToColumn



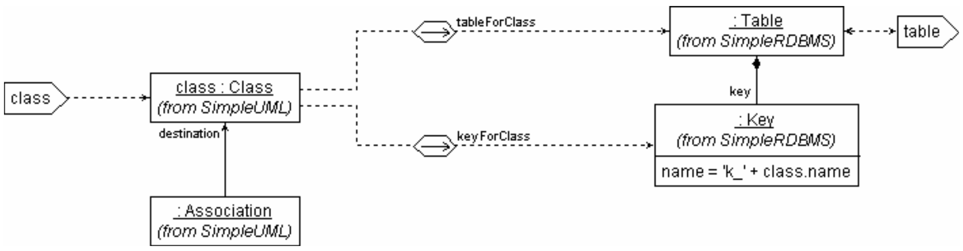
A primitive attribute contributes a column with a hierarchically prefixed column name.

3.5 Uml2Rdbms.ClassAttrToColumn



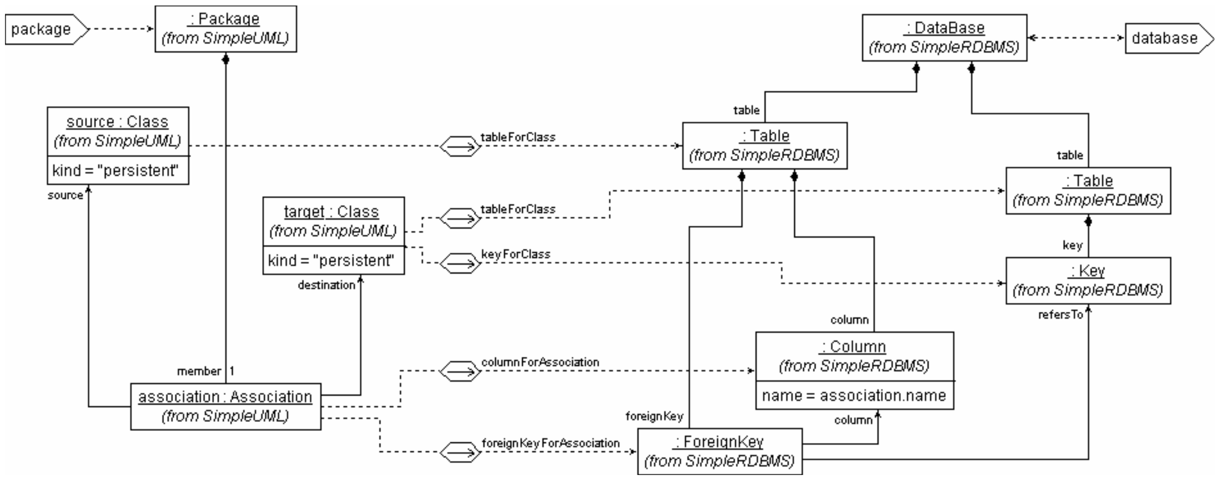
A class attribute drills down building up a hierarchical name prefix.

3.6 Uml2Rdbms.ClassToKeyForForeignKey



If the class is the destination of an association a target for foreign key references will be necessary.

3.7 Uml2Rdbms.AssocToForeignKey



Each association contributes a foreign key and a column to the source, with the foreign key referring to the key of the destination.

For-each `association` whose `source` and `destination` are persistent classes, create a foreign key in the table identified by the signature `tableForClass(source)` referring to the key identified by `keyForClass(target)` which is created within the table identified by `tableForClass(target)`.

[This differs from the DSTC example in creating the Key only if there is an association that requires it. The DSTC behaviour can be matched by creating the Key directly in ClassToTable.]

[The explicit `columnToAssociation` and `foreignKeyForAssociation` are provided to ease comparison with DSTC example, they are not needed until the example is extended with concurrent activity that needs to correlate on association products.]

[It might seem that the Key could be created directly in this transformation, rather than in ClassToKeyOfForeignKey, but this would cause problems for inter-package associations, which are beyond the scope of the problem, except in so far as UMLX exposes a difficulty. UMLX requires that all LHS instances that are used for value are reachable by composition relationships with respect to inputs. The `source` and `target` are not reachable as drawn and so can only be used for identity. Creation of the key would require `name='k_'+target.name` and so would require a package path to be established to reach the target, which would imply an unattractive 2D drill-down through the package hierarchy to reach the independent parents of `association` and `target`. In this case, the separate transformation avoids the problem. In the more complicated case, there are considerable benefits in establishing a flat root, nodes and arcs intermediate or pivot model before starting serious multi-pass rearrangements.]