Finding Similar Classes with a Simplified Metamodel

David Flater
National Institute of Standards and Technology
Gaithersburg, Maryland, U.S.A.

Abstract

To integrate conceptual models and other types of models, it is necessary to identify the portions of the models that overlap (i.e., find similar classes) and resolve any conflicts. Complete automation of this task is generally considered infeasible. However, it may be possible to provide some automated support to the integrator. This paper discusses a simplified metamodel and matching algorithm that have been implemented in a prototype tool called Similar Class Finder (SCF), which works with Unified Modeling Language (UML™) class diagrams. Simple tests of the prototype produce interesting successes and failures, suggesting that future work and refinement of the tool could be productive.

Keywords: UML, model, integration

1. Introduction

Model integration – more precisely, the integration of data models, information models, conceptual models, and/or structural models in a software context – is just one part of application integration. It is, however, a vitally important part. The very integrity of an integrated system depends on the expertise with which the concepts represented in these models are merged. This merging unfortunately tends to consume a great deal of expensive time.

To integrate models, it is necessary to identify the portions of the models that overlap (*i.e.*, find similar classes) and resolve any conflicts. There is a subtask, herein called the "hard semantic matching problem," that probably cannot be reliably automated because it is not done reliably even by people: determining when the abstractions identified by different models from different sources can safely be treated as identical. Long before automation of this task (using ontologies, knowledge bases, automated reasoning, *etc.*) was considered within information technology circles, 20th century philosopher Alfred Korzybski argued that confidence in the sameness of the different abstractions of different people is generally unwarranted [20]. Nevertheless, without solving the hard semantic matching problem, it may be possible to provide some automated support to reduce the time spent on model integration.

After the discussion of related work, this paper presents a simplified metamodel and matching algorithm that have been implemented in a prototype tool called Similar Class Finder (SCF), which works with Unified Modeling Language (UML) class diagrams [30]. It then documents some interesting successes and failures resulting from simple tests of the prototype and explores directions for future work.

2. Related work

2.1. Model integration

Work on Similar Class Finder was inspired by a presentation [31] about another tool, the Visual and Natural Language Specification Tool (VINST) [4][5]. The original intent was simply to do the same kind of thing for UML. However, the work ended up taking a different path. VINST judges the similarity of different entities using the corpus of information that is available in the context of ISO 10303, informally known as the Standard for the Exchange of Product Model Data [13], which includes full text definitions of terms in addition to the labels and structural information in EXPRESS models [14]. While VINST is more sophisticated in its use of full text definitions, SCF is more sophisticated in its identification of structural variants.

Although much of the work labeled "schema integration" or "ontology integration" is equivalent to or interrelated with the task of model integration, it is difficult to find relevant references to model integration by that name, at this time. Quite a lot of references exist for *enterprise* model integration, but most enterprise models are not the kind of models this paper addresses. Nonetheless, the approach described in Reference [9] is applicable to many kinds of models. Modeling languages are characterized using ontologies, effectively creating a new, unifying metamodel. This addresses the "language barrier" that exists when integrating models that are written in different modeling languages and supports formal reasoning about the structural content of models. Other information needed to support reasoning about the integration of different models can then be captured in "context" ontologies.

Rational Software Corporation's Rational Rose [27] includes a tool called Model Integrator that allows several UML models to be merged into one. However, automated integration of "unrelated" models (in the sense not derived from a common ancestor) was not in the original design intent of the tool [26], so its functionality in that regard has not yet been developed to the level of sophistication seen in existing schema integration methods.

2.2. Schema integration

Research in database schema integration that was *called* database schema integration peaked in the 1980s. However, as attention shifted to federated and heterogeneous information systems and to ontology-based research during the 1990s, much of that research continued to relate to schema integration (or vice-versa, depending on one's viewpoint). The fundamental problems of detecting semantic overlap and heterogeneity among different information sources were essential to all of this research, and they remain essential to our present topic.

Thousands of papers about schema integration were published. The most one can hope to do is cite a reasonable set of representative works. The selections cited here were found by breadth-first exploration of the references made by several convenient sources. Thus, the citation of any particular work is indicative only of its reference proximity to the original sources and should not be construed as an endorsement of one research program over another.

Reference [2] contains a survey of integration methodologies that were available circa 1986. By that time the various kinds of conflicts that can arise during schema integration had been analyzed in various ways, and it was acknowledged that "automatic conflict resolution is generally not feasible; close interaction with designers and users is required before compromises can be achieved in any real-life integration activity." [2]

Of particular interest to the task at hand are any automated or potentially automatable approaches to finding similar entities in different schemas. The survey mentions two references where "the integration system automatically assigns a 'degree of similarity' to pairs of objects, based on several matching criteria." In the first, Reference [1], these criteria appear to include similarities in the attributes and relationships of entities and in the attributes and involved entities of relationships in an Entity-Relationship model. However, the heuristic is not described in sufficient detail to reproduce it exactly. In the second, Reference [6], a measure of the degree of similarity is calculated based on detailed, formal assertions supplied by the database designer. A work published later by the same group of researchers, Reference [21], provides a more detailed analysis of attribute equivalence. Their prior analyses of object class equivalence [7] and relationship equivalence [23] are then revisited from the perspective of attribute equivalence. Similar work based on a further refined definition of attribute equivalence appears in Reference [28]. The automatable portions of these analyses depend on assertions that involve real-world state, "the underlying real world instances of the object being represented," or a similar notion for real-world attribute semantics. The rigor of the analyses therefore depends on the rigor with which it can be determined that two different abstractions refer to the same real-world object, which is another form of the hard semantic matching problem. As Reference [28] cautions, "Schema integration involves a subjective activity which relies heavily on the knowledge and intuition of the user about the application domain (domain of discourse), intended use of the integrated schema, and the systems that manage data modeled by the schemas. Thus, the results of a schema integration activity are not unique and *cannot* be generated totally automatically."

References [16], [17], and [29] document a line of work that deals with a related notion of "semantic proximity" between different database objects. Reference [17] begins with the assertion, "The fundamental question in interoperability is that of identifying objects in different databases that are semantically related, and then resolving the schematic differences among semantically related objects." Semantic proximity is defined as a function of context, abstraction, domain (values), and state (extents). The line of work includes a breakdown of schematic heterogeneities into domain incompatibilities, entity definition incompatibilities, data value incompatibilities, abstraction level incompatibilities, and schematic discrepancies, each of which is further broken down into different kinds of conflicts. Different taxonomies of schematic heterogeneities and resolution methods appear in many other works about schema integration and federated, distributed, heterogeneous, and/or multi-database systems, *e.g.*, References [3] and [19].

The database schema integration work that survived through the 1990s became increasingly mingled with research having to do with ontologies, *e.g.* in Reference [18]. Progress in this direction can be seen in Reference [12], where a "semantic dictionary" is used in the detection and resolution of semantic heterogeneity. Similarly, Reference [10] describes a semi-automatic tool for finding similar classes in "semantically enriched" relational schemas. However, this semantic enrichment corresponds more closely to the information contained in a UML class diagram. Their approach to finding similar classes focuses on generalization/specialization and aggregation, ignoring the attributes. Reference [8] describes an approach to finding similar classes based on reasoning about fuzzy terminological relationships between names as defined in a terminological knowledge base. Other progress in the ontology direction can be seen in Reference [33], where it is proposed to replace the assertion-based characterization of attribute semantics used in previously described work with a characterization in terms of a "concept hierarchy." With respect to automation, Reference [12] repeats the assertion that "automatic conflict resolution is in general infeasible" and Reference [10] makes a related assertion to the effect that detecting semantic relationships among different databases cannot be completely automated, whereas Reference [33] proposes to semi-automate the identification of concepts using natural language processing of a supplied data dictionary.

Related problems in information retrieval continued to be addressed (*e.g.*, Reference [24]) until this line of work was transformed by the emergence of the World Wide Web.

2.3. Ontology integration

To the extent that some definitions of ontology would admit a conceptual model in UML, there is some related work in ontology integration. Reference [22] describes a semi-automatic tool that finds similar concepts in different ontologies based on labels, context identifier tags, term-based matching rules, structure-based matching rules, and other rules.

3. Boxes & Stuff metamodel

The Boxes & Stuff metamodel is an abstraction that suppresses structural details that are distracting when comparing different UML class diagrams for integration purposes. See Figures 1 and 2. (Note: all figures appear at the end of the document.)

"Boxes" correspond to classes, which hopefully are abstractions of important domain concepts. Everything else is "stuff." The many different ways that UML provides for associating different things with classes are reduced to only one: a box has stuff. Whether that stuff is thought of as being in the box, on the box, associated with the box, connected to the box by a line, or whatever, is not germane. The choice between declaring an attribute whose type is class Snark and making an association to class Snark, and similar choices, are deliberately suppressed.

The Boxes & Stuff metamodel also loses constraints and various other adornments that can happen in UML. Clearly there will be cases when this information is critical to integration decisions, but for the purpose of a first prototype it suffices to ignore it.

The goal of recasting a UML class diagram into boxes and stuff is not to reproduce the UML model, but to break it into tiny, flattened pieces representative of how the model might look from the perspective of an observer sitting in the various boxes. That way, a similar class finder can quickly reach a decision about whether the view from inside of one box is similar to the view from inside another one in a different model.

Returning to Figures 1 and 2, two non-obvious features of stuff are apparent. The first is a flag called "many;" the second is an operation called "normalize." The purpose of these features becomes apparent in the following section.

4. Translating classes to boxes and stuff

The prototype mapping from a UML class diagram to boxes and stuff is as follows.

1. Classes are boxes. The name of the box is the name of the class.
2. Attributes are stuff.
   a. Stuff.name is the name of the attribute.
   b. Stuff.type is the name of the type of the attribute.
   c. The "many" flag is true if the upper bound on the multiplicity of the attribute is greater than 1, false otherwise.
   d. The type field and "many" flag are then normalized as described below.
3. Operations are stuff.
   a. Stuff.name is the name of the operation.
   b. Stuff.type is the name of the return type of the operation.
   c. The "many" flag is set to false.
   d. The type field and "many" flag are then normalized as described below.
   e. Parameters other than the return are ignored.
4. Associations are stuff.
   a. Only named AssociationEnds are considered. (This is a poor but practical substitute for testing the isNavigable flag, which is often disused.)
   b. Stuff.name is the role of the associate (i.e., the name of the AssociationEnd).
   c. Stuff.type is the name of the type of the associate.
   d. The "many" flag is true if the upper bound on the multiplicity of the association is greater than 1, false otherwise.
5. Specializations "inherit" the stuff of their "parents" – it is simply copied over.
6. Normalization of data types.
   a. All number types (integers, floats, doubles, fixed, etc.) become simply "number."

   b. All text types (chars, strings, wide char strings, etc.) become simply "text."
   c. If the type field refers to an aggregate type, the "many" flag is set to true and the contents of the type field are replaced by the name of the type being aggregated or "unknown" if it could be a mixture.

Normalization of data types to "number" and "text" is done on the premise that the choice among many possible numeric and text types is seldom significant when comparing different UML class diagrams for integration purposes. They would only serve to distract from any gross similarities that might exist.

5. Matching algorithm

The matching algorithm makes many passes through the models. The most similar boxes are identified first, while subsequent passes identify similarities with less and less confidence.

A mapping from names to canonical synonyms is built up in the early stages and utilized thereafter when making comparisons. It may be initialized to include user-supplied input if desired.

1. Matching Boxes.

```
do {
  added_new_synonym = false;
  // Find Boxes with matching names and equivalent
     Stuff
  // Find Boxes that differ in name only and add to
     database of synonyms *
} while (added_new_synonym);
// Find Boxes with matching names that subset Stuff
// Find Boxes with matching names
// Find Boxes that subset Stuff *
// List leftovers
* Empty boxes are excluded in these passes.
```

2. Equivalence of sets of Stuff.
   Sets of Stuff are equivalent if all members of both sets have a match in the other.
3. Matching Stuff.
   Stuff matches if the names and data types match. (Data type involves both "type" and "many;" matching is defined below.)
4. Matching Box and Stuff names.
   a. Case-insensitive.
   b. 'z' = 's' (for U.K. versus U.S. spelling).
   c. Modulo database of synonyms.
   d. [Future work could add VINST-like rules for matching among different naming conventions.]
5. Matching data types.
   a. Data types match if cardinality and type names match as defined below.
   b. Data types also match if cardinality and type names match after unrolling one level of "holder classes," defined as Boxes that contain only one "Stuff."
6. Matching cardinality.
   Equivalence of the "many" booleans.
7. Matching type names.
   a. Case-insensitive.
   b. 'z' = 's' (for U.K. versus U.S. spelling).
   c. Common type names are "normalized" to a canonical synonym.
   d. Modulo database of synonyms.
   e. Type name "unknown" matches every type name.

6. Implementation and test

The algorithm descriptions above and the following results are accurate as of version 0.4.1 of the Similar Class Finder (SCF).

Similar Class Finder is a simple command-line application with the following usage:

```
SCF left-URL right-URL [thesaurus-URL] [-verbose]
```

The left and right URLs are of Extensible Markup Language (XML) Metadata Interchange (XMI) [32] files such as any UML modeling tool might be able to produce. The prototype was only tested with Poseidon for UML [11] Community Edition, Version 1.0.

Four tests were conducted in which SCF was fed XMI files but not supplied with any initial "thesaurus" (the database of synonyms). The first test was to try SCF on a UML translation of the example that was used in the VINST presentation mentioned under Related Work [31]. Figures 3 and 4 show the class diagrams that were compared: one from the context of race cars, the other from the context of mass market automobiles, "street cars."

Before running SCF, one manual change was made to the XMI resulting from the race cars model: the multiplicity of the entries attribute of class logbook was corrected to be 0..*. A limitation of the version of Poseidon that we used prevented this information from being stored in the original model.

This example plays to the strengths of SCF because the matching of attribute names is trivial. Due to limited space, the output will only be summarized here. SCF first matched the identical Weight and Distance classes in the two models, then correctly matched the class Car from the race cars model with the class Vehicle from the street cars model (different in name only). The class named Car in the street cars model does not match as well; it has attributes that are specific to street cars, such as air conditioning. Farther down in the listing, the classes named Date in both models were matched with considerably lower confidence since they had only a name in common.

Because it was derived from relatively simple EXPRESS models, the first test did not fully demonstrate the potential benefits of using a simplified metamodel. The second test, therefore, was to compare the race cars model used previously with an obfuscated version of itself. The obfuscations were:

- Distance was renamed to Length, its attributes were reversed and their data types were changed.
- Two of the attributes of Car were converted to associations, and one was replaced with an accessor operation.
- The troublesome entries attribute of Logbook was changed from type 0..* String to a generic List type.

The results were as follows. First, the Person and Logbook classes of the two models were matched (same names, equivalent stuff). Then Distance and Length were found to differ in name only and added to the database of synonyms, and subsequently most other classes were matched immediately with high confidence (matching names, equivalent stuff). The exceptions, Weight and Date, were matched with lower confidence than was strictly necessary (matching names, subset stuff) because the obfuscator incorrectly made the links from Weight and Date to Car navigable, resulting in extraneous "car" stuff on Weight and Date.

The third test was to compare UML translations of the Application Resource Model (ARM) (see Figure 5) and the Module-Interpreted Model (MIM) (see Figure 6) of the person and organization module from a draft ISO specification [15]. Again, some manual edits were made to the XMI to correct the multiplicity of some attributes. As with the first two tests, there is something to make the job of SCF easier: the ARM and the MIM are related by a design methodology that effectively makes them different views of the same information. However, in this case, the limitations of SCF began to show. The Address entities were matched with high confidence (matching names, equivalent stuff), but Organization and Organisation were only found to have equivalent names, and Person was only found to have a subset (75%) of the stuff of Person_in_organisation. The confidence with which the Organization classes were matched could be improved by recognizing the refactoring of the name attribute(s) and the substantial overlap that exists despite the fact that neither set of attributes is a subset of the other. However, the matching of the Person class with Person_in_organization is more troublesome. An examination of the models shows that Person_in_organization might be better matched to Person_and_organization. However, the semantics are still not quite equivalent. One could argue that the pairwise comparisons made by SCF are simply inadequate in this case. Future work should examine the possibilities for expanding the algorithm to find many-to-many correspondences.

The last test was to compare the previously used MIM model with an analogous person and organization model derived from the Product Data Management (PDM) Enablers standard from the Object Management Group™ [25] (see Figure 7). The PDM Enablers model used an association class that was not supported in the version of Poseidon that was used, so a normal class was substituted. The results were disappointing. Nothing was matched at any higher confidence than matching class names (Person, Organization). The similarity between Party and Address went completely unnoticed, as did the match between PersonOrganization and Person_and_organization. There is clearly still much work to be done if the matching algorithm is to perform well on models from significantly different sources.

7. Conclusion

The Boxes & Stuff metamodel eliminates several kinds of shallow "obfuscations" that cause similar models to appear different, thus increasing the effectiveness of a similar class-finding tool. The next step is to make progress in recognizing the deeper "obfuscations" that result from refactorings, different vocabularies, and different naming conventions. Although similar problems have been researched for many years with no conclusive victories, it may be productive to revisit this line of work from the perspective of a simplified metamodel. An extension of the matching algorithm to perform many-to-many comparisons might help with refactorings. The full text definition processing done by VINST [4][5] sought to address the vocabulary and naming convention problems, so a merging of ideas between VINST and SCF might yield additional progress. Finally, it is likely that some of the information that was ignored in the conversion to Boxes & Stuff, notably the distinction between aggregation and composition, should be used to enable a more accurate translation and better identification of structural variants.

8. References

[1] Carlo Batini and Maurizio Lenzerini, "A Methodology for Data Schema Integration in the Entity Relationship Model," IEEE Transactions on Software Engineering, v. 10, n. 6, November 1984, pp. 650-663.
[2] C. Batini, M. Lenzerini, and S. B. Navathe, "A Comparative Analysis of Methodologies for Database Schema Integration," ACM Computing Surveys, v. 18, n. 4, December 1986, pp. 323-364.
[3] Yuri Breitbart, Peter L. Olson, and Glenn R. Thompson, "Database Integration in a Distributed Heterogeneous Database System," in Proceedings of the Second IEEE Conference on Data Engineering, February 1986.
[4] H. Dalianis, "The VINST approach: Validating and Integrating STEP AP Schemata Using a Semi Automatic Tool,"

in Proceedings of the Conference on Integration in Manufacturing, October 1998.

[5] H. Dalianis and E. Hovy, "Integrating STEP Schemata using Automatic Methods," in Proceedings of the ECAI-98 Workshop on Applications of Ontologies and Problem-Solving Methods, August 1998, pp. 54-66.

[6] R. Elmasri, J. Larson, and S. Navathe, "Schema Integration Algorithms for Federated Databases and Logical Database Design," Honeywell Corporate Systems Development Division, Report CSC-86-9:8212, January 1986.

[7] R. Elmasri and S. B. Navathe, "Object Integration in Database Design," in Proceedings of the IEEE COMPDEC Conference, April 1984.

[8] P. Fankhauser, M. Kracker, and E. Neuhold, "Semantic vs. Structural Resemblance of Classes," SIGMOD Record, v. 20, n. 4, December 1991, pp. 59-63.

[9] Florence Fillion, Christopher Menzel, Thomas Blinn, and Richard J. Mayer, "An Ontology-Based Environment for Enterprise Model Integration," in Proceedings of the IJCAI Workshop on Basic Ontological Issues in Knowledge Sharing, August 1995. (Pages not numbered)

[10] Manuel García-Solaco, Malú Castellanos, and Fèlix Saltor, "Discovering Interdatabase Resemblance of Classes for Interoperable Databases," in Proceedings of the IEEE RIDE-International Workshop on Interoperability in Multidatabase Systems, 1993.

[11] Gentleware AG, Poseidon for UML product, http://purl.org/net/dflater/org/gentleware/poseidon.

[12] Joachim Hammer and Dennis McLeod, "An Approach to Resolving Semantic Heterogeneity in a Federation of Autonomous, Heterogeneous Database Systems," International Journal of Intelligent and Cooperative Information Systems, v. 2, n. 1, March 1993, pp. 51-83.

[13] ISO 10303:1994, *Industrial automation systems and integration — Product data representation and exchange.* Available from ISO, http://purl.org/net/dflater/org/iso.

[14] ISO 10303-11:1994, *Industrial automation systems and integration — Product data representation and exchange — Part 11: Description methods: The EXPRESS language reference manual.* Available from ISO, http://purl.org/net/dflater/org/iso.

[15] ISO/CD TS 10303-1011:2001(E), *Industrial automation systems and integration — Product data representation and exchange — Part 1011: Application module: Person organization.* Available from ISO, http://purl.org/net/dflater/org/iso.

[16] V. Kashyap and A. Sheth, "Schema Correspondences between Objects with Semantic Proximity," Technical Report DCS-TR-301, Dept. of Computer Science, Rutgers University, October 1993.

[17] V. Kashyap and A. Sheth, "Schematic and Semantic Similarities between Database Objects: A Context-based Approach," Very Large Data Bases (VLDB) Journal, v. 5, n. 4, 1996, pp. 276-304.

[18] Vipul Kashyap and Amit Sheth, "Semantic Heterogeneity in Global Information Systems: The Role of Metadata, Context and Ontologies," in Michael P. Papazoglou and Gunter Schlageter, eds., *Cooperative Information Systems: Current Trends and Directions*, Academic Press, 1998, pp. 139-178.

[19] Won Kim, Injun Choi, Sunit Gala, and Mark Scheevel, "On Resolving Schematic Heterogeneity in Multidatabase Systems," Distributed and Parallel Databases, v. 1, 1993, pp. 251-279.

[20] Alfred Korzybski, *Science and Sanity: An Introduction to Non-Aristotelian Systems and General Semantics*, 5th edition, Institute of General Semantics (http://purl.org/net/dflater/org/igs), 1994.

[21] James A. Larson, Shamkant B. Navathe, and Ramez Elmasri, "A Theory of Attribute Equivalence in Databases with Application to Schema Integration," IEEE Transactions on Software Engineering, v. 15, n. 4, April 1989, pp. 449-463.

[22] Prasenjit Mitra, Gio Wiederhold and Jan Jannink, "Semi-automatic Integration of Knowledge Sources," in Proceedings of Fusion '99, July 1999.

[23] S. B. Navathe, T. Sashidhar, and R. Elmasri, "Relationship Merging in Schema Integration," in Proceedings of the 10th International Conference on Very Large Databases, 1984.

[24] Yannis Papakonstantinou, Hector Garcia-Molina, and Jennifer Widom, "Object Exchange Across Heterogeneous Information Sources," in Proceedings of the IEEE International Conference on Data Engineering, Taipei, Taiwan, March 1995, pp. 251-260.

[25] Product Data Management Enablers v1.3 specification, Object Management Group, 2000. Available from http://purl.org/net/dflater/omgdoc/formal/2000-11-11.

[26] Rational Software Corporation, "Issues using Model Integrator to merge 'unrelated' models," Technote 7382, 1999-04-23. Available at http://purl.org/net/dflater/org/rational/technotes/7382.

[27] Rational Software Corporation, Rational Rose product, http://purl.org/net/dflater/org/rational/rose.

[28] A. P. Sheth, S. K. Gala, and S. B. Navathe, "On automatic reasoning for schema integration," International Journal of Intelligent and Cooperative Information Systems, v. 2, n. 1, 1993, pp. 23-50.

[29] Amit Sheth and Vipul Kashyap, "So Far (Schematically) yet So Near (Semantically)," in Proceedings of the IFIP TC2/WG2.6 Conference on Semantics of Interoperable Database Systems, DS-5, November 1992. Also in IFIP Transactions A-25, North Holland, 1993.

[30] Unified Modeling Language v1.4 specification, Object Management Group, 2001. Available from http://purl.org/net/dflater/omgdoc/formal/2001-09-67.

[31] Evan Wallace, "Adventures in Discovering Latent Semantic Links: Experiments with the VINST Tool," presentation given at NIST, 2001-10-03.

[32] XML Metadata Interchange v1.1 specification, Object Management Group, 2000. Available from http://purl.org/net/dflater/omgdoc/formal/2000-11-02.

[33] C. Yu, W. Sun, S. Dao, and D. Keirsey, "Determining relationships among attributes for Interoperability of Multidatabase Systems," in Proceedings of the First International Workshop on Interoperability in Multidatabase Systems, April 1991.

9. Figures

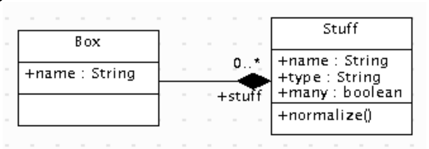

Figure 1. Boxes & Stuff metamodel in UML

Box:
  name: text
  stuff: many Stuff
Stuff:
  name: text
  type: text
  many: boolean
  normalize: void

Figure 2. Metamodel as boxes and stuff

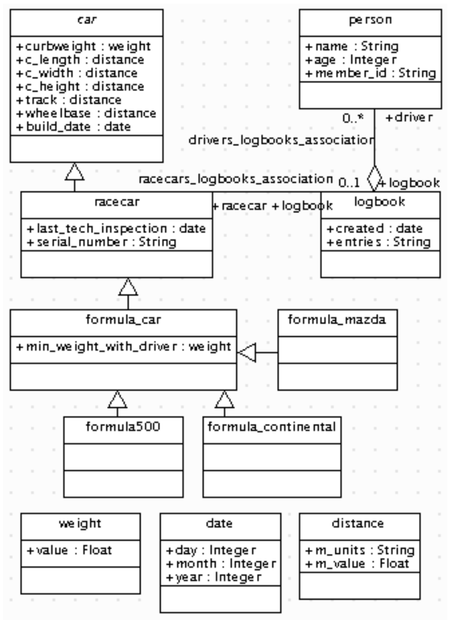

Figure 3. Race cars model

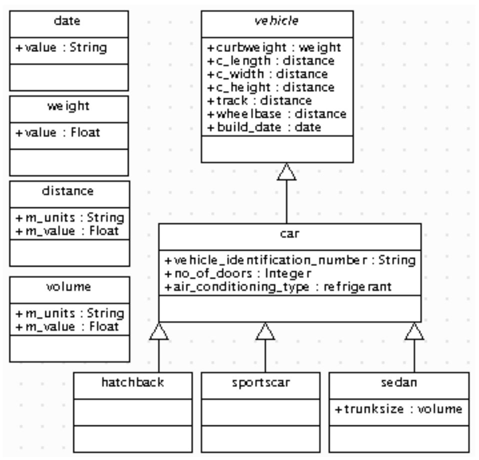

Figure 4. Street cars model

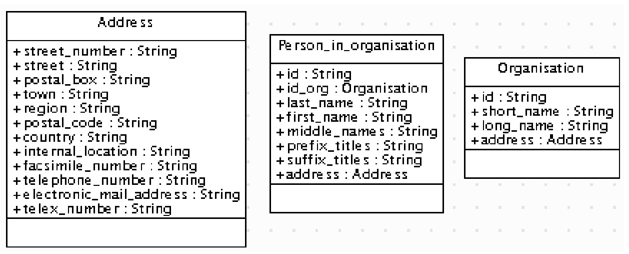

Figure 5. Translation of Application Resource Model

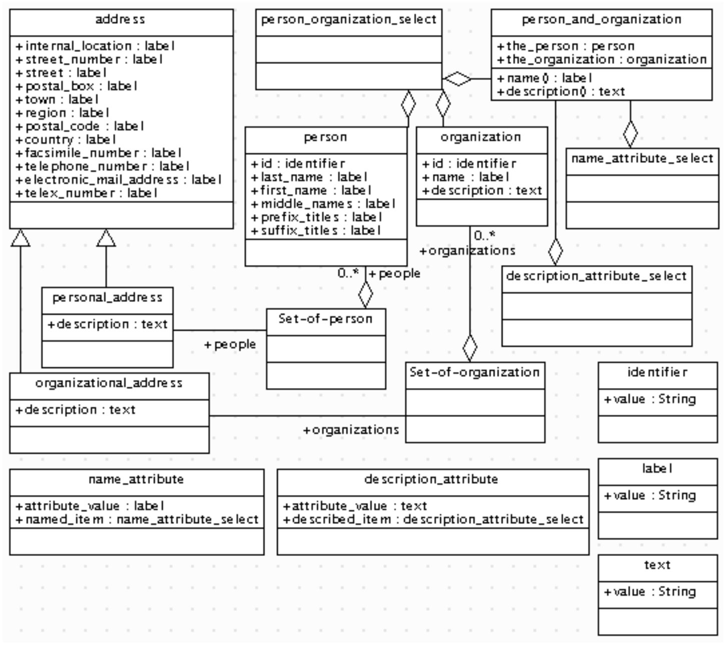

Figure 6. Translation of Module-Interpreted Model

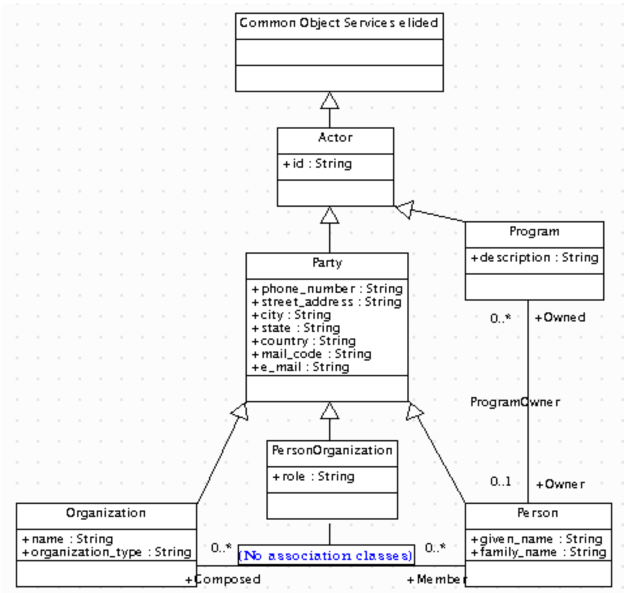

Figure 7. People and organizations in the Product Data Management Enablers