The NORMA Tool for ORM 2

Matthew Curland[1] and Terry Halpin[2]

[1]LogicBlox, USA
[2]LogicBlox, Australia and INTI Education Group, Malaysia
e-mail: {matthew.curland, terry.halpin}@logicblox.com

Abstract: Second generation Object-Role Modeling (ORM 2) is a prime exemplar of fact-orientation, an approach that models the underlying facts of interest in an attribute-free way, using natural sentences to identify objects and the roles they play in relationships. ORM 2 provides languages and procedures for modeling and querying information systems at a conceptual level as well as mapping procedures for transforming between ORM structures and other structures, such as Entity Relationship (ER) models, class models in the Unified Modeling Language (UML), relational database models, extensible markup language schemas (XSD), and datalog. This paper provides an overview of Natural ORM Architect (NORMA), an ORM 2 tool under development that is implemented as a plug-in to Microsoft Visual Studio. For data modeling purposes, NORMA typically provides greater expressive power and semantic stability than provided by tools based on ER or UML. NORMA's support for automated verbalization and sample populations facilitates validation with subject matter experts, and its live error-checking provides efficient feedback to modelers.

1 Introduction

Fact-oriented modeling is a conceptual approach (including languages and procedures) for modeling, transforming, and querying information, that specifies the fact structures of interest as well as the applicable business rules in terms of concepts that are intelligible to the business users. Unlike Entity-relationship modeling (ER) [4] and class diagramming in the Unified Modeling Language (UML) [17], fact-orientation makes no use of attributes as a way to encode facts, instead representing all ground assertions of interest as atomic (non-decomposable) facts that are either existential facts (e.g. There is a country named 'Australia') or elementary facts that predicate over first-order individuals (objects that are either entities or values).

Elementary facts are expressed using mixfix predicates, and are instances of *fact types*. For example, the UML attributes Person.isSmoker and Person.birthdate are modeled instead as Person smokes (unary fact type) and Person was born on Date (binary fact type). Higher arity fact types are allowed, for example Person played Sport for Country (a ternary) and Product in Year in Region sold in Quantity (a quaternary). This attribute-free approach facilitates natural verbalization and population of models (important for validating models with nontechnical domain experts), and promotes semantic stability (e.g. one never needs to remodel an attribute and associated access paths if one later wants to talk about an attribute).

Business rules are modeled as *constraints* or *derivation rules* that apply to the relevant business domain. Alethic constraints restrict the possible states or state transitions of fact populations (e.g. No Person is a parent of itself), while deontic constraints are obligations that restrict the permitted states or state transitions of fact populations (e.g. It is obligatory that each Doctor is licensed to practice). Derivation rules enable some facts or objects to be derived from others.

Fact-oriented modeling approaches include *Object-Role Modeling* (*ORM*) [8], Cognition-enhanced Natural Information Analysis Method (CogNIAM) [15], the Predicator Set Model (PSM) [13], and Fully-Communication Oriented Information Modeling (FCO-IM) [1]. The Semantics of Business Vocabulary and Business Rules (SBVR) initiative is fact-based in its use of attribute-free constructs [18]. For an overview of fact-oriented modeling approaches, including history and research directions, see [7].

Since the 1970s, various tools have been developed to support fact-orientation. Early tools based on NIAM include IAST and RIDL* (based on the RIDL language [14]). CogNIAM is currently supported by Doctool. FCO-IM is supported by the Case Talk tool. Related ontology tools include DOGMA Studio and Collibra. ORM tools began with InfoDesigner, which later evolved into InfoModeler, VisioModeler, and the ORM Source Model solution in Microsoft Visio for Enterprise Architects. ActiveQuery [3] is an ORM conceptual query tool released as a companion to InfoModeler.

More recently, a number of tools have been developed based on second generation ORM (*ORM 2*) [6]. These include *Natural ORM Architect (NORMA)*, ActiveFacts [12], and ORM-Lite. For data modeling purposes, the ORM 2 graphical notation is far more expressive than UML's graphical notation for class diagrams, and is also much richer than industrial ER notations. A detailed summary of the ORM 2 graphical notation is accessible at http://www.orm.net/pdf/ORM2GraphicalNotation.pdf. A thorough treatment of the theory and practice of ORM 2 may be found in [11].

The rest of this paper provides an overview of the NORMA tool, and is structured as follows. Section 2 summarizes the main components of NORMA. Section 3 illustrates some important capabilities of NORMA. Section 4 provides details of the implementation architecture. Section 5 summarizes the main contributions and outlines future research directions.

2 Overview of NORMA

NORMA is implemented as a plug-in to Microsoft Visual Studio. Most of NORMA is open-source, and a public domain version is freely downloadable [16]. A professional version of NORMA is also under development. Fig. 1 summarizes the main components of the tool. Users may declare ORM object types and fact types textually using the Fact Editor, or drag new elements off the toolbox. New model components are added to the conceptual model and displayed with graphical shapes on one or more ORM diagrams The Model Browser tool window also provides a hierarchical view of all model components. Sample object and fact instances may be entered in tabular format in the Sample Population Editor.

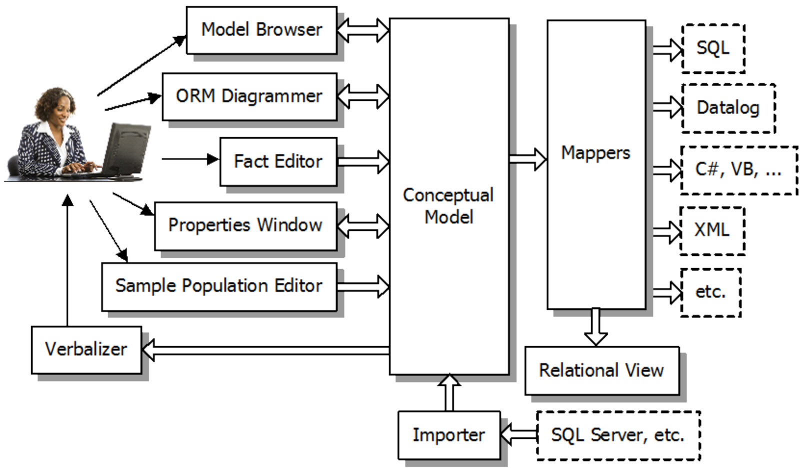


Fig. 1. Main components of NORMA

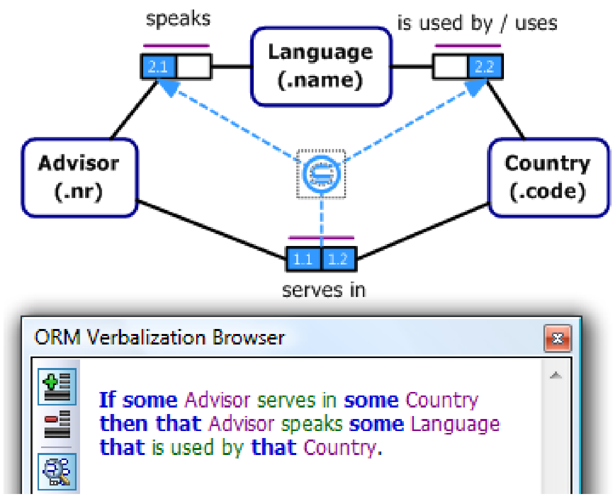
Currently, ORM constraints must be entered in the ORM diagrammer or the Properties Window. These constraints are automatically verbalized in FORML (Formal ORM Language), a controlled natural language that is understandable even by non-technical people. Our modeling team at Logicblox recently extended the Model Browser to enable derivation rules for both fact types and subtypes to be formally captured and stored in a rules component of the conceptual model based on the role calculus [5]. These derivation rules are also automatically verbalized.

Using mappers, ORM schemas may be automatically transformed into various implementation targets, including relational database schemas for popular database management systems (SQL Server, Oracle, DB2, MySQL, PostgreSQL), datalog, .NET languages (C#, VB, etc.), and XML schemas. A Relational View extension displays the relational schemas in diagram form. The semantics underlying relational columns can be exposed by selecting them and automatically verbalizing the ORM fact types from which they were generated. An import facility can import ORM models created in some other ORM tools, and can reverse engineer relational schemas in SQL Server into ORM schemas. Import from further sources is planned.

Other components facilitate navigation and abstraction. For example, multiple concurrent windows viewing the same model allow shapes to be copied between diagrams, the ORM Diagram Spy and hyperlinks in the Verbalization Browser allow rapid navigation through a model, and the ORM Context Window automatically displays the global schema neighborhood of a selected ORM element.

3 Examples of Some NORMA Features

Feedback from industrial practitioners indicates that *automated verbalization* support is one of the most useful features of NORMA. Fig. 2 shows a screen shot from NORMA illustrating verbalization of a join subset constraint.



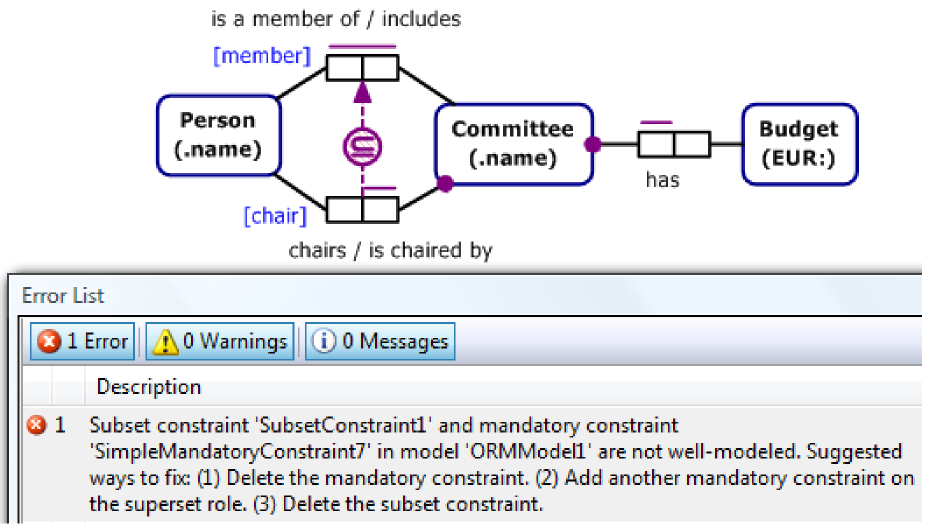
If some Advisor serves in some Country
then that Advisor speaks some Language
that is used by that Country.

Fig. 2. NORMA screenshot showing verbalization of a join-subset constraint

Here we have three binary fact types: Advisor serves in Country; Advisor speaks Language; Language is used by Country. Entity types are shown as named, soft rectangles with their reference mode in parenthesis. Logical predicates are depicted as a named sequence of role boxes connected to the object types whose instances play those roles. The bar over each predicate depicts a spanning uniqueness constraint, indicating that the fact types are *m:n*, and can be populated with sets of fact instances, but not bags.

The circled "⊆"connected by dashed lines to role pairs depicts a subset constraint. When the constraint shape is selected, NORMA displays role numbers to highlight the role sequence arguments to the constraint. In this example, the set of advisor-country instances of the role-pair (1.1, 1.2) are constrained to be a subset of the set of advisor-country instances populating the role-pair (2.1, 2.2) projected from the role path from Advisor through Language to Country. In passing through Language, a conceptual inner join is performed on its entry and exit roles, so this is an example of a join-subset constraint. The meaning of the constraint is clarified by the verbalization shown at the bottom of Fig. 2. Because every aspect of an ORM model can be automatically verbalized in such a high level language, non-technical domain experts can easily validate the rules without even having to see or understand the diagram notation.

A feature of NORMA that is especially useful to modelers is its *live error checking* capability. Modelers are notified immediately of errors that violate a metarule that has been implemented in the underlying ORM metamodel. Fig. 3 shows an example where the subset constraint is marked with red fill because it is inconsistent with other constraints present. In this case, the committee role of being chaired is declared to be mandatory (as shown by the solid dot on the role connection), while the committee role of including a member is declared to be optional. But the subset constraint implies that if a committee has a chair then it must have that person as a member. So it is impossible for the two fact types to be populated in this situation. NORMA not only detects the error but suggests three possible ways to fix the problem.



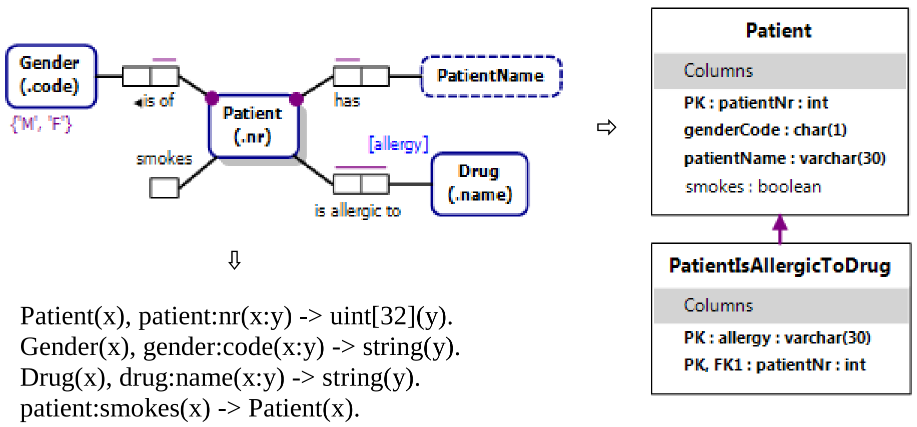
Error List

1 Error | 0 Warnings | 0 Messages

	Description
1	Subset constraint 'SubsetConstraint1' and mandatory constraint 'SimpleMandatoryConstraint7' in model 'ORMModel1' are not well-modeled. Suggested ways to fix: (1) Delete the mandatory constraint. (2) Add another mandatory constraint on the superset role. (3) Delete the subset constraint.

Fig. 3.

Fig. 4 illustrates a very simple example of *mapping* from ORM to a relational schema as well as a datalog schema. For a detailed discussion of a much more complex example, as well as comparisons with ER and UML, see [9].



```
Patient(x), patient:nr(x:y) -> uint[32](y).
Gender(x), gender:code(x:y) -> string(y).
Drug(x), drug:name(x:y) -> string(y).
patient:smokes(x) -> Patient(x).
patient:gender[x]=y -> Patient(x), Gender(y).
patient:name[x]=y -> Patient(x), string(y).
patient:drug:isAllergicTo(x,y) -> Patient(x), Drug(y).
Patient(x) -> patient:gender[x]=_.
Patient(x) -> patient:name[x]=_.
gender:code(_:y) -> y="M"; y="F".
```

Fig. 4. Simple example of mapping an ORM schema to a relational schema and datalog

4 Implementation

The NORMA designer is built primarily on the Domain Specific Language (DSL) Toolkit from the Visual Studio SDK, as well as general Visual Studio extension points. The implementation of NORMA adds multiple framework services to DSL to enable a highly modularized and extensible system. Extension points are available for file importers, additional DSL models and designers that interact with the core NORMA models, and extension points for artifact generation. All core NORMA components have exactly the same architecture as the extension models, except that the core models cannot be removed from the list of current extensions.

DSL was chosen because it was a model-driven system, providing for a large percentage of the required code to be generated. The generated code defines a transacted object model with standard notifications that enable responsive secondary model changes in response to atomic changes in the object model. A particularly important feature of DSL is the built-in support for *delete closures*, which provide notifications for elements that are pending deletion but have not yet been detached from the model. Delete closures enable NORMA to minimize the parts of the model that require revalidation in response to a model change, which in turn enables extremely responsive incremental validation irrespective of model size.

The NORMA implementation relies on a number of enhancements to the DSL tooling and runtime components. Most extensions are necessary because many of the DSL supporting SDK components assume that the complete metamodel is known at all times, whereas NORMA makes the opposite assumption—the complete metamodel and associated rules are not known until a model file is opened and the set of extension models are read from the root XML element in the model file. Allowing multiple models also required significantly more flexibility for serialization of NORMA models than for a standard DSL model. The NORMA modeling framework includes multiple extensions to the DSL rules engine to enable compartmentalized model validation that minimizes incremental processing within a model and isolates dependent models from other models that they do not even know are loaded.

Extensions to NORMA can be classified in the following areas:

1. *Importers* allow XML data sources with schemas not supported first-hand by the NORMA designers (including older NORMA file formats) to be transformed automatically on load. Most importers are XSLT transformations, although additional wizards can be registered with Visual Studio to first translate a non-XML data source into XML suitable for import.
2. *Primary Domain Models* (domain model is a DSL term for a metamodel) are extensions that provide model elements and validation rules for runtime execution inside the designer. Extension models provide schematized XML components and a mapping from the in-memory model to the XML elements, load fixup mechanisms to reach an internally consistent state at load completion, and rules to maintain consistency after the model is loaded. Provided extensions in this category include the core ORM metamodel (*FactType*, *ObjectType*, etc) and the relational metamodel, which contains elements such as *Table*, *Column*, and *ReferenceConstraint*.

3. *Presentation Models* have the same characteristics as primary domain models and are modeled similarly. However, presentation elements are treated as views on the underlying model, not the model itself. The *ORM Diagram* and *Relational View*, along with their contained shapes, are defined in presentation models. A single element from a primary domain model may be associated with multiple presentation shapes.
4. *Bridge Models* are also domain models with the special function of relating two primary domain models. Bridge models consist of relationships between two other models, plus generation settings that control the current relationships between the models. Primary domain models are designed to be standalone so that transformations can be performed in either direction. This allows, for example, an importer to be written for the XML schema of the relational model that is generated directly from a database and has no ORM information. Bridge models enable changes in one primary model to be applied to another loaded standalone model while maintaining the relationships between the source and target.
5. *Shell Components* are views and editors targeting specific parts of an in-memory ORM model, such as the Model Browser and Fact Editor tool windows discussed earlier.
6. *Artifact Generators* produce non-ORM outputs such as DDL, class models, and other implementations mapped from an ORM model. In general, artifact generators are much easier to create than DSL models because generators deal with a static artifact—namely a snapshot of the ORM model in XML form—and do not have to worry about the change management that is the bulk of the implementation cost for domain models. NORMA's generation system supports a dependent hierarchy of generated files based on output format. A single generator can request inputs of both the standard ORM format (with required extensions specified by the generator) and any other formats produced by other registered generators.

   The hierarchical generation process allows analysis to be performed once during generation, and then reused for multiple other generators. NORMA provides XML schemas for all intermediate formats, allowing extension generators to understand and leverage existing work. Some examples of intermediate formats we use are DCIL (relational data constructs), DDIL (XML representation of data-definition constructs, created from the DCIL format), and PLiX (an XML representation of object-oriented and procedural code constructs). These intermediate XML formats are transformed into target-specific text artifacts. DDIL is directly transformed to either SQL Server or Oracle specific DDL formulations, and PLiX generates C#, VB, PHP, and other languages—all without changing the intermediate file formats.

   Another advantage of the use of well-defined intermediate structures is the ability to modify these structures during artifact generation. For example, attempting to decorate an ORM model with auditing constructs is extremely invasive at the conceptual model level. However, adding auditing columns to each table is a simple transform from DCIL to DCIL with additional columns for each table, with the modified file continuing in the generation pipeline.

5 Conclusion

This paper provided a brief overview of the NORMA tool and its support for ORM 2. Major recent work not reported on here because of space restrictions includes deep support for entry of formal derivation rules and their automated verbalization, as well as a prototype implementation of FORML 2 as an input language. Details on the latter may be found in [10]. Research is also under way to extend NORMA with support for dynamic rules [2].

References

1. Bakema, G., Zwart, J., & van der Lek, H. 2000, *Fully Communication Oriented Information Modelling*. Ten Hagen Stam.
2. Balsters, H. & Halpin, T. 2008, 'Formal Semantics of Dynamic Rules in ORM', *On the Move to Meaningful Internet Systems 2008: OTM 2008 Workshops*, eds. R. Meersman, Z. Tari, P. Herrero et al., Monterrey, Mexico, Springer LNCS 5333, pp. 699-708.
3. Bloesch, A. & Halpin, T. 1997, 'Conceptual queries using ConQuer-II', *Proc. ER'97: 16th Int. Conf. on conceptual modeling,* Springer LNCS, no. 1331, pp. 113-126.
4. Chen, P. P. 1976, 'The entity-relationship model—towards a unified view of data'. *ACM Transactions on Database Systems*, 1(1), pp. 9−36.
5. Curland, M., Halpin, T. & Stirewalt, K. 2009, 'A Role Calculus for ORM', *On the Move to Meaningful Internet Systems 2008: OTM 2009 Workshops*, eds. R. Meersman, P. Herrero et al., Vilamoura, Portugal, Springer LNCS 5872, pp. 692-703.
6. Halpin, T. 2005, 'ORM 2', *On the Move to Meaningful Internet Systems 2005: OTM 2005 Workshops*, eds R. Meersman, Z. Tari, et al., Cyprus. Springer LNCS 3762, pp 676-87.
7. Halpin, T. 2007, 'Fact-Oriented Modeling: Past, Present and Future', *Conceptual Modelling in Information Systems Engineering,* eds. J. Krogstie, A. Opdahl & S. Brinkkemper, Springer, Berlin, pp. 19-38.
8. Halpin, T. 2009, 'Object-Role Modeling', *Encyclopedia of Database Systems*, ed. L, Liu & M. Tamer Ozsu, Springer-Verlag, Berlin.
9. Halpin, T. 2010, 'Object-Role Modeling: Principles and Benefits', *International Journal of Information Systems Modeling and Design*, Vol. 1, No. 1, IGI Global, pp. 32-54.
10. Halpin, T. & Wijbenga, J. P. 2010, 'FORML 2', *Enterprise, Business-Process and Information Systems Modeling*, eds. I. Bider et al., Springer LNBIP 50, pp. 247–260.
11. Halpin, T. & Morgan, T. 2008, *Information Modeling and Relational Databases, Second Edition*, Morgan Kaufmann, San Francisco.
12. Heath, C. 2009. ActiveFacts Website: http://dataconstellation.com/ActiveFacts/.
13. ter Hofstede, A., Proper, H. & van der Weide, T. 1993, 'Formal definition of a conceptual language for the description and manipulation of information models', *Information Systems*, vol. 18, no. 7, pp. 489-523.
14. Meersman, R. 1982, *The RIDL Conceptual Language*, Int. Centre for Information Analysis Services, Control Data Belgium, Brussels.
15. Nijssen, M., & Lemmens, I. 2008, 'Verbalization for Business rules and Two Flavors of Verbalization for Fact Examples', *On the Move to Meaningful Internet Systems 2008: OTM 2008 Workshops*, eds. R. Meersman, Z. Tari, P. Herrero et al., Springer LNCS 5333, pp. 760-769.
16. NORMA (Natural ORM Architect) tool download site for public-domain version: http://www.ormfoundation.org/files/folders/norma_the_software/default.aspx.
17. Object Management Group 2009, *UML 2.2 Specifications*. Online at: www.omg.org/uml.
18. Object Management Group 2008, *Semantics of Business Vocabulary and Business Rules (SBVR)*. Online at: http://www.omg.org/spec/SBVR/1.0/.