The NORMA Software Tool for ORM 2

Matthew Curland[1] and Terry Halpin[2]

[1] LogicBlox, USA
[2] LogicBlox, Australia and INTI International University, Malaysia
{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, ORM typically provides greater expressive power and semantic stability than provided by UML or industrial versions of ER. 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) [18], fact-orientation makes no use of attributes as a way to express facts, instead representing all ground assertions of interest as atomic (non-decomposable) facts that are either existential facts or elementary facts. An existential fact simply asserts the existence of an entity (e.g. There is a country named 'Australia'). An elementary fact predicates over individuals (objects that are either entities or values). For example, "The Country named 'Australia' is large" expresses a unary fact about an entity, and "The Country named 'Australia' has the Nickname 'Down Under'" expresses a binary fact that relates an entity to a value.

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).

Fact-oriented modeling approaches include *Object-Role Modeling* (*ORM*) [8], Cognition-enhanced Natural Information Analysis Method (CogNIAM) [16], the Predicator Set Model (PSM) [14], 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 [19]. For an overview of fact-oriented modeling approaches, including history and research directions, see [7].

Since the 1970s, various *software tools* have supported fact-orientation. Early tools based on NIAM include IAST and RIDL* (based on the RIDL language [15]). 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 supported second generation ORM (*ORM 2*) [6]. These include *Natural ORM Architect (NORMA)*, ActiveFacts [13], 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 [**9**].

Business rules are modeled in ORM 2 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**). *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. For example, grandparenthood facts may be derived from parenthood facts using the rule Person_1 is a grandparent of Person_2 **if** Person_1 is a parent of **some** Person_3 **who** is a parent of Person_2. Similarly, instances of the subtype Father may be derived from parenthood and gender facts using the rule **Each** Father **is a** Person **who** is male **and** is a parent of **some** Person_2.

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 [12].

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 [17]. 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, and these are then displayed graphically using auto-layout. Alternatively, object types and fact types may be entered directly on the current diagram by dragging elements from the toolbox and supplying appropriate names. Large ORM schemas are typically displayed using many pages of ORM diagrams.

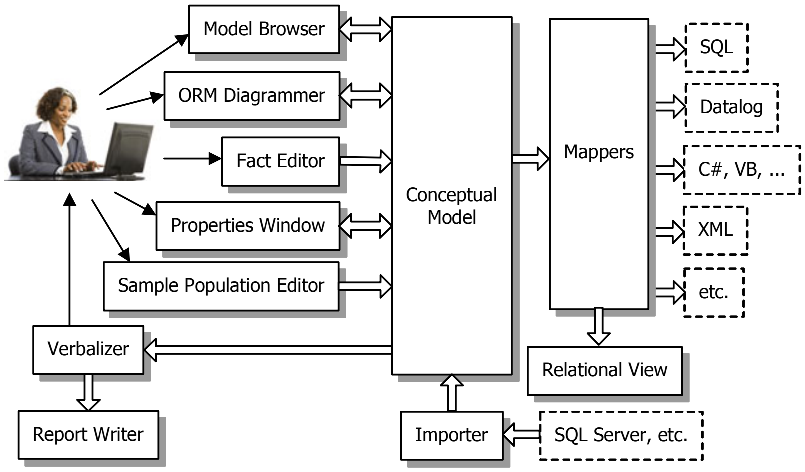


Fig. 1. Main components of NORMA

Currently, most 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 nontechnical people. We plan to allow complete ORM models, including constraints and derivation rules, to be entered textually in FORML. An initial prototype to provide this support has been developed as an extension to the Fact Editor [11].

The Model Browser tool window provides a hierarchical view of all model components, and allows model elements to be dragged onto diagram pages. Sample object and fact instances may be entered in tabular format in the Sample Population Editor, and automatically verbalized. Explanatory comments and informal descriptions may be added to model elements. These comments are included in the output displayed in the Verbalization Browser. Reports in HTML format may be automatically generated including complete verbalizations of ORM models with embedded hyperlinks for easy navigation. These reports are especially valuable for non-technical domain experts to validate and sign off on the model specification.

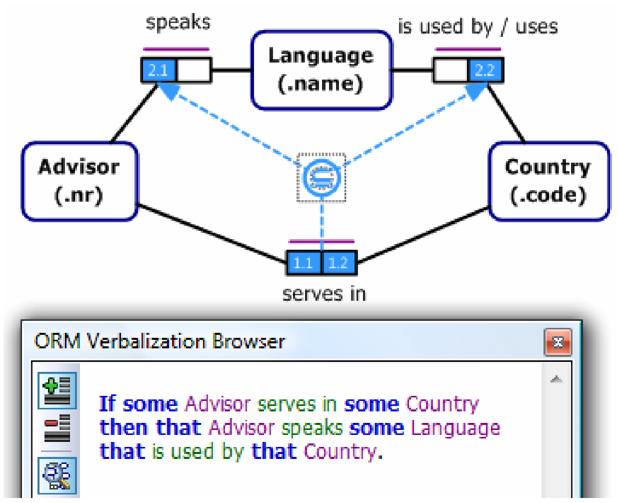
The VisualBlox team at Logicblox recently extended the Model Browser to enable derivation rules for both fact types and subtypes to be formally captured and stored in a rules component of the conceptual model based on the role calculus [5]. These derivation rules are also automatically verbalized.

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 [10], .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 reverse engineer relational schemas in SQL Server into ORM schemas. Import from further sources is planned.

Other components facilitate navigation and abstraction. For example, the Diagram Spy window facilitates views onto various pages as well as the copying of diagram selections from one page to another. Hyperlinks in the Verbalization Browser allow rapid navigation through a model. The Context Window allows one to focus on a given model element and progressively expand outwards to view its neighborhood in the global schema.

3 Some NORMA Highlights

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.



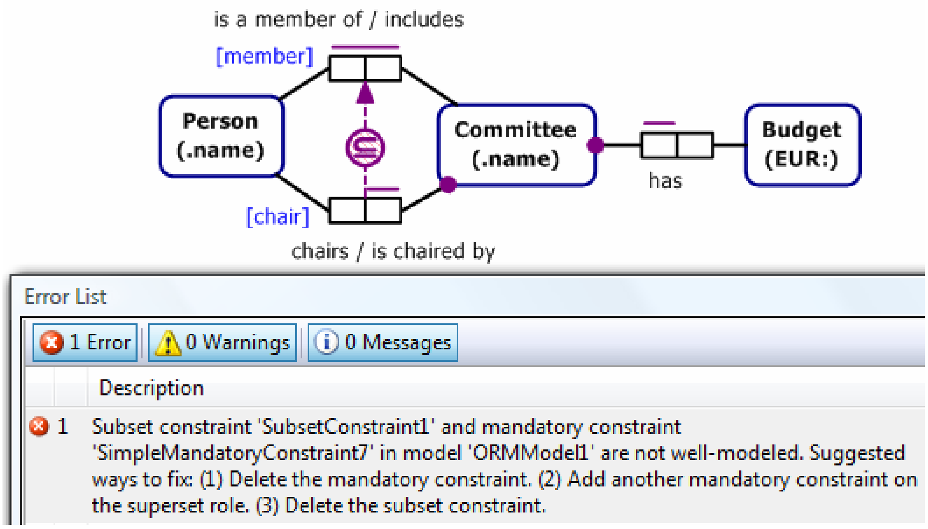
If some Advisor serves in some Country
then that Advisor speaks some Language
that is used by that Country.

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. In ORM, the term “role” means a part played in a relationship (of arity one or higher). 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.

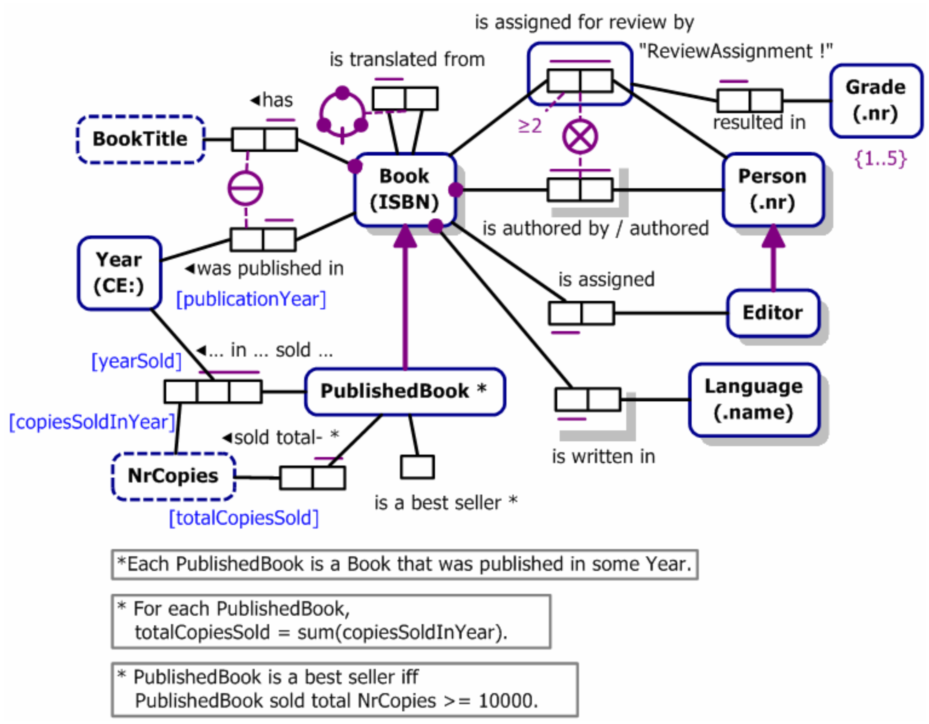
The circled "⊆"connected by dashed lines to role pairs depicts a subset constraint. When the constraint shape is selected, NORMA displays role numbers to highlight the role sequence arguments to the constraint. In this example, the set of advisor-country instances of the role-pair (1.1, 1.2) are constrained 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.



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. Example of live error detection involving a problematic constraint pattern



*Each PublishedBook is a Book that was published in some Year.

* For each PublishedBook,
totalCopiesSold = sum(copiesSoldInYear).

* PublishedBook is a best seller iff
PublishedBook sold total NrCopies >= 10000.

Fig. 4. NORMA screenshot illustrating rich support for graphic constraints

Fig. 4 shows a NORMA screenshot of one page from a larger ORM model concerning book publishing. This gives some idea of ORM's richly *expressive graphic constraint notation* for data modeling compared with UML and industrial ER. All fact types must have at least one reading, but may be given as many as desired. Here the authorship fact type has forward and inverse predicate readings, separated by a slash.

PublishedBook and Editor are subtypes, connected to their supertype by an arrow. PublishedBook is a derived subtype, as shown by its trailing asterisk, and the derivation rule supplied for it: **Each** PublishedBook **is a** Book **that** was published in **some** Year. This rule is expression in relational-style, using predicate readings. Optionally, roles may be assigned names, displayed in square brackets next to the role box. In this example, four roles are named. One reason to add role names in ORM is to enable derivation rules to be formulated in attribute-style (using role names for "attributes"). For example, the derived fact type PublishedBook sold total NrCopies is defined using the following attribute-style derivation rule: **For each** Publishedbook, totalCopiesSold = **sum**(copiesSoldInYear). Editor is an asserted subtype, so has no derivation rule. NORMA also supports semiderived types (some instances may be asserted and some derived).

The hyphen after "total" in the predicate "sold total" invokes hyphen-binding to bind the adjective "total" to its noun for automated verbalization. This causes the uniqueness constraint on this fact type to verbalize as "**Each** PublishedBook sold **at most one** total NrCopies". Without the hyphen, the "**at most one**" quantifier would appear after "total" instead of before it. NORMA supports both forward and reverse hyphen binding.

The circled bar connecting the BookTitle and publicationYear roles depicts an external uniqueness constraint (each book title, publication year combination applies to at most one Book). The annotated circle next to the book translation fact type depicts an acyclic ring constraint, constraining the translation relationship to exclude any cycles. ORM includes a large list of such ring constraints (irreflexive, asymmetric, antisymmetric, intransitive, etc.). The "≥ 2" on the review assignment role is a frequency constraint (each book that is assigned for review is assigned to at least two reviewers). ORM's mandatory role and frequency constraints work properly with fact types of any arity, unlike UML's multiplicity constraints which fail to cater for some constraint patterns on *n*-ary associations (e.g. see pp. 361-362 of [12]).

The soft rectangle around the "is assigned for review by" predicate objectifies the relationship as ReviewAssignment. The exclamation mark "!" after the name of this objectification type indicates that it is independent (instances of it may exist independently of playing in other facts). The {1..5} annotation on Grade depicts a value constraint, restricting its grade numbers to be in the range 1 to 5 inclusive. The circled "X" between the review assignment and authorship fact types is a pair-exclusion constraint indicating that no book can be reviewed by one of its authors.

The shadows around the object types Book, Person, Editor and Language indicate that these types also appear elsewhere in the global schema. In the NORMA tool, if you right-click on such a duplicated shape and choose "Select on Diagram" you will be presented with a list of all the other pages on which that shape appears; and selecting the desired page will then take you to that page with the shape highlighted.

Similarly, the shadows around the "is authored by" and "is written in" predicates indicate that these fact types appear elsewhere in the global schema. Fig. 5 shows a fragment from another page in the global schema, where these two facts types appear. The circled subset constraint runs from the (language, person) pairs projected from the join path Person authored Book that is written in Language to the (language, person) pairs in the fact type Person is fluent in Language. The constraint is *deontic*, as shown by the small "o" (for "obligatory") on the constraint symbol, as well as its blue color (in contrast to alethic constraints, which are colored violet). The automated verbalization shown below indicates the deontic nature by starting with "**It is obligatory that**".

It is obligatory that
if some Person authored **some** Book **that** is written in **some** Language
then that Person is fluent in **that** Language.

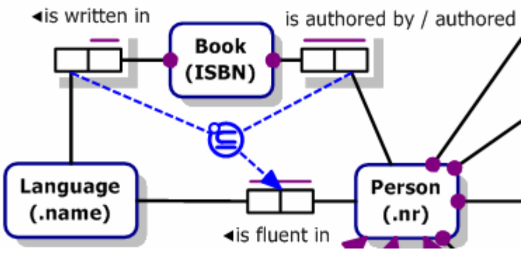
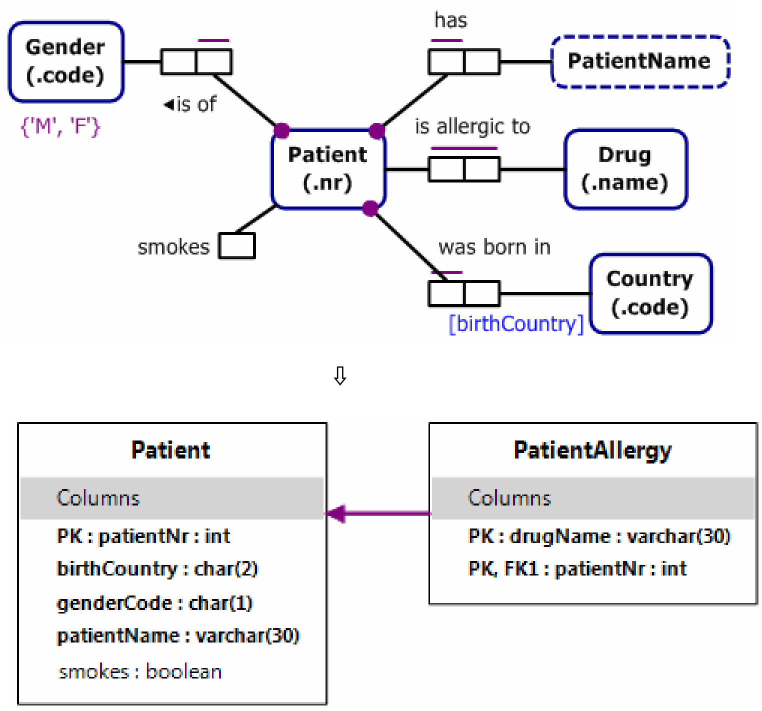


Fig. 5. NORMA screenshot fragment illustrating a deontic constraint

While Fig. 4 and Fig. 5 show many of the constraint varieties in ORM, they by no means cover all of them. A complete list of ORM constraints, with examples, may be accessed online at http://www.orm.net/pdf/ORM2GraphicalNotation.pdf. Further explanation of the first page of the book publisher example, as well as comparative schemas in UML and ER, may be found in [9].

Fig. 6 illustrates a very simple example of *mapping* from ORM to a relational schema. NORMA provides several mechanisms for controlling how the names of tables and columns are generated (e.g. here the use of the role name "birthCountry" ensures that the column name will be the same). The relational view displays columns that are not nullable in bold, and marks primary key columns and foreign key columns by "PK" and "FK" respectively. The SQL code for creating the relational schema is automatically generated for the main industrial DBMSs.

For traceability and validation purposes, the ORM verbalization is accessible directly from the relational columns, so touching any column automatically provides the underlying semantics. For example, Fig. 7 is a NORMA screenshot showing the verbalization displayed when the birthCountry column is selected. This feature is extremely useful for validating relational models with clients, and can be used even if they are not familiar with the ORM schema from which the relational schema was generated. This example also includes some sample data entered in the ORM model.



Patient

Columns

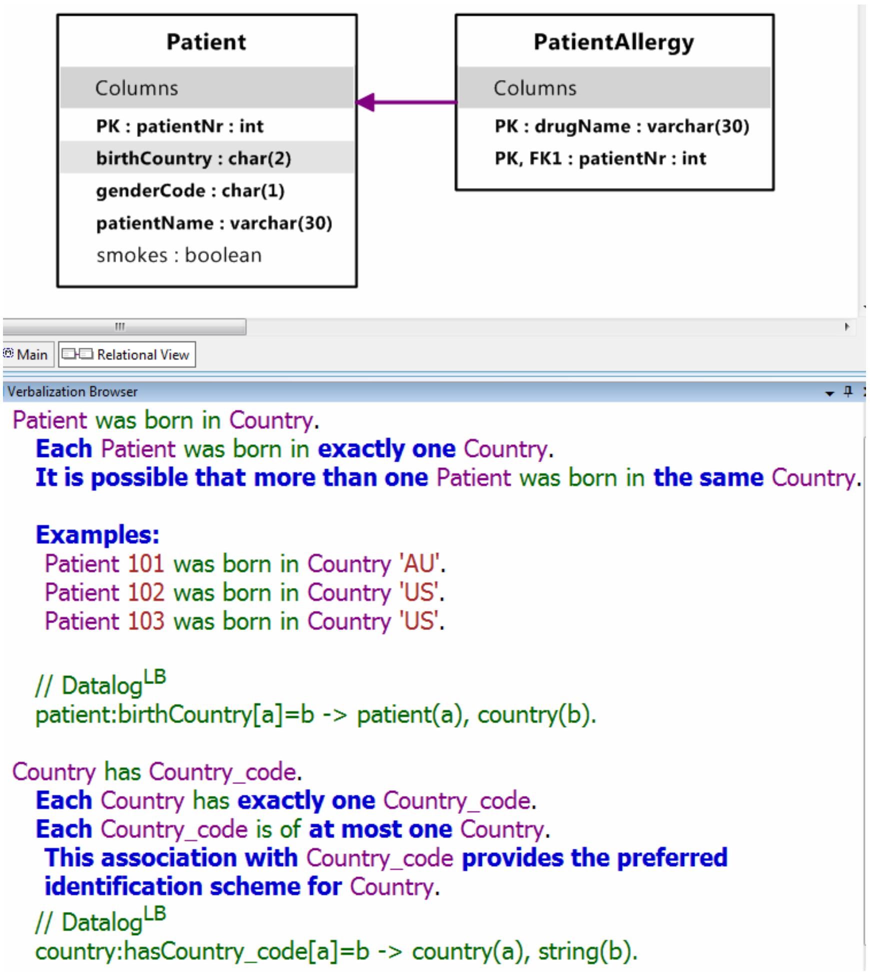
PK : patientNr : int
birthCountry : char(2)
genderCode : char(1)
patientName : varchar(30)
smokes : boolean

PatientAllergy

Columns

PK : drugName : varchar(30)
PK, FK1 : patientNr : int

Fig. 6. Simple example of mapping an ORM schema to a relational schema



Patient was born in Country.
Each Patient was born in exactly one Country.
It is possible that more than one Patient was born in the same Country.

Examples:
Patient 101 was born in Country 'AU'.
Patient 102 was born in Country 'US'.
Patient 103 was born in Country 'US'.

```
// DatalogLB
patient:birthCountry[a]=b -> patient(a), country(b).
```

Country has Country_code.
Each Country has exactly one Country_code.
Each Country_code is of at most one Country.
This association with Country_code provides the preferred identification scheme for Country.

```
// DatalogLB
country:hasCountry_code[a]=b -> country(a), string(b).
```

Fig. 7. Verbalization for the selected birthCountry column in the relational schema

This screenshot also shows some of the Datalog^{LB} code that can be generated if the predicate mapping extension is enabled (this extension is not currently available in the open source version of NORMA). Datalog^{LB} is an advanced form of typed datalog that provides rich support for complex rules as well as high performance. Current research at LogicBlox is extending NORMA to generate complete Datalog^{LB} code for ORM models, including advanced constraints and derivation rules [10]. For this trivial model, the Datalog^{LB} code may be set out as shown. For a more complex example, 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).
Country(x), country:code(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:birthCountry[x]=y -> Patient(x), Country(y).
Patient(x) -> patient:gender[x]=_.
Patient(x) -> patient:name[x]=_.
gender:code(_:y) -> y="M"; y="F".
```

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 may be classified into 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. An example

of a non-XML based importer is the database import wizard, which uses several steps to translate the schema of an existing database into an ORM model.

a. The import wizard identifies the type of DBMS target and uses DBMS-specific code to generate a temporary XML file with a .orm extension. This XML file uses the same database structure representation as the relational artifact generator mentioned below.
b. Visual Studio opens the file with the NORMA designer, which is associated with the .orm file extension.
c. The NORMA designer—which recognizes exactly one root format plus one format per extension—examines the root element in the XML file and determines whether the root file format or one or more of the extension formats are not native or do not correspond to the current recognized extension versions.
d. The designer looks for a registered import transform that recognizes the XML format and then executes the transform, producing a new XML file. Steps c and d are repeated as often as needed to produce a file format that can be directly read by the designer. For the database import case, three transforms are needed. Two of the transform steps prompt the user for transform parameters, which is another import feature supported directly by NORMA.
e. This import mechanism is also used for file format changes that are occasionally required as NORMA evolves. To protect against data loss, opening an existing file with an outdated format will result in a user notification offering to block the file save operation (SaveAs remains enabled).

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. An extension model provides an XML schema for serialized elements, a mapping between the in-memory model and the XML elements, a load fixup mechanism to reach an internally consistent state at load completion, and rules to maintain consistency for user-initiated changes after the model is loaded. Provided primary domain models include the core ORM metamodel (*FactType*, *ObjectType*, etc) and the relational extension metamodel, with elements such as *Table*, *Column*, and *ReferenceConstraint*.

The ability to dynamically validate model state enables a scalable platform by not requiring a full model analysis for incremental changes. Although NORMA validation errors are conceptually deontic constraints, each error is modeled as a normal element in the domain model, just like *ObjectType* or *FactType*. The transacted object model framework provided by DSL supports fine-grained tracking of atomic model changes, allowing rules to record elements related to a change, and then verify the error state for affected elements later in the transaction.

One of the challenges in incremental error validation is knowing when all changes directly caused by user input have been entered into the model. Validating affected elements before all external changes are known results in duplication of work and expensive analysis with an incomplete state. The NORMA team has developed a system known as *delayed validation* that enables model validation to register an element/validator pair for delayed processing. The validator function is called once for each registered element after normal input has been completed. Validators run inside a transaction, so they can modify the contents of the model. Placing error validation objects in the models also means that validation is not required during undo/redo operations.

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. Presentation model changes are performed near the end of a transaction to ensure that all underlying model changes are complete.

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 targeting the XML schema of the in-memory relational model that can be fully defined without corresponding ORM elements in the file. However, this design means that elements in a primary relational model cannot directly reference elements in the underlying ORM model. Bridge models store these relationships between primary models and allow changes in one standalone model to be applied to another standalone model generated from it.

It is very important during bridge analysis used to generate one standalone model from another to have the complete set of changes from the source model. To facilitate this, the NORMA delayed validation mechanism prioritizes validator execution based on both model dependencies and explicit prioritization directives in the code. Model dependency analysis is sufficient in many cases. For example, a presentation model depends on the referenced primary model, so presentation validation is automatically performed after the primary model validation is complete. For bridge models, the default behavior is not correct because the bridge validators will naturally run after (not in-between) validation of the two models it is bridging.

For cases where the default validator ordering is insufficient, validation routines use .NET attributes to explicitly run their validators *before*, *with*, or *after* another model. By running bridge validation after native validators for another model, we ensure a consistent and complete state of the source model. Careful placement of the bridge validators allows the bridge models to trigger changes based on both primary elements and derived error state changes. For example, since the generated relational model requires a *FactType* to be completely specified (all role players specified with identification schemes with known data types plus an allowed uniqueness constraint pattern), the bridge model can watch for addition and deletion of validation errors like *RolePlayerRequiredError* and *FactTypeRequiresInternalUniquenessConstraintError* instead of reanalyzing underlying relationships.

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. Shell components interact with domain models by initiating transactions that modify the underlying model in response to user input, then responding to *model events* to update the shell display. Model events are a playback of a transaction log after the transaction is initially committed or repeated with an undo and redo operations. ORM model files can be loaded independently of any Visual Studio hosted shell components, allowing external tools to load an ORM model without Visual Studio running.

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 the modified file conforming to the same schema but containing additional columns for each table. After all intra-format transforms are complete, the modified file is passed to the next stage in the generation pipeline.

5 Conclusion

This paper provided a brief overview of the NORMA tool and its support for ORM 2, a conceptual data modeling approach with greater semantic stability and richer graphical constraint expressibility than UML class modeling and industrial ER modeling. Key features of NORMA's support for ORM 2 were highlighted, including live error checking, and validation by automated verbalization and sample populations.

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 [11]. Research is also under way to extend NORMA with support for dynamic rules [2].

References

1. Bakema, G., Zwart, J., van der Lek, H.: Fully Communication Oriented Information Modelling. Ten Hagen Stam (2000)
2. Cao, V.-T., Halpin, T.: Formal Semantics of Dynamic Rules in ORM. In: Meersman, R., Tari, Z., Herrero, P. (eds.) OTM-WS 2008. LNCS, vol. 5333, pp. 699–708. Springer, Heidelberg (2008)
3. Bloesch, A., Halpin, T.: Conceptual queries using ConQuer-II. In: Embley, D.W. (ed.) ER 1997. LNCS, vol. 1331, pp. 113–126. Springer, Heidelberg (1997)
4. Chen, P.P.: The entity-relationship model—towards a unified view of data. ACM Transactions on Database Systems 1(1), 9–36 (1976)
5. Curland, M., Halpin, T., Stirewalt, K.: A Role Calculus for ORM. In: Meersman, R., Herrero, P., Dillon, T. (eds.) OTM 2009 Workshops. LNCS, vol. 5872, pp. 692–703. Springer, Heidelberg (2009)
6. Halpin, T.: ORM 2. In: Meersman, R., Tari, Z., Herrero, P. (eds.) OTM-WS 2005. LNCS, vol. 3762, pp. 676–687. Springer, Heidelberg (2005)
7. Halpin, T.: Fact-Oriented Modeling: Past, Present and Future. In: Krogstie, J., Opdahl, A., Brinkkemper, S. (eds.) Conceptual Modelling in Information Systems Engineering, pp. 19–38. Springer, Berlin (2007)
8. Halpin, T.: Object-Role Modeling. In: Liu, L., Tamer Ozsu, M. (eds.) Encyclopedia of Database Systems. Springer, Berlin (2009)
9. Halpin, T.: Object-Role Modeling: Principles and Benefits. International Journal of Information Systems Modeling and Design 1(1), 32–54 (2010)
10. Halpin, T., Curland, M., Stirewalt, K., Viswanath, N., McGill, M., Beck, S.: Mapping ORM to Datalog: An Overview. In: Meersman, R., Dillon, T., Herrero, P. (eds.) On the Move to Meaningful Internet Systems 2010: OTM 2010 Workshops. LNCS. Springer, Heidelberg (2010) (to appear)
11. Halpin, T., Wijbenga, J.P.: FORML 2. In: Nurcan, S., Ukor, R. (eds.) BPMDS 2010 and EMMSAD 2010. LNBIP, vol. 50, pp. 247–260. Springer, Heidelberg (2010)
12. Halpin, T., Morgan, T.: Information Modeling and Relational Databases, 2nd edn. Morgan Kaufmann, San Francisco (2008)
13. Heath, C.: ActiveFacts (2009), `http://dataconstellation.com/ActiveFacts/`

14. ter Hofstede, A., Proper, H., van der Weide, T.: Formal definition of a conceptual language for the description and manipulation of information models. Information Systems 18(7), 489–523 (1993)
15. Meersman, R.: The RIDL Conceptual Language. In: Int. Centre for Information Analysis Services, Control Data Belgium, Brussels (1982)
16. Nijssen, M., Lemmens, I.M.C.: Verbalization for Business Rules and Two Flavors of Verbalization for Fact Examples. In: Meersman, R., Tari, Z., Herrero, P. (eds.) OTM-WS 2008. LNCS, vol. 5333, pp. 760–769. Springer, Heidelberg (2008)
17. NORMA (Natural ORM Architect) tool download site for public-domain version, `http://www.ormfoundation.org/files/folders/norma_the_software/default.aspx`
18. Object Management Group 2009, UML 2.2 Specifications, `http://www.omg.org/uml`
19. Object Management Group 2008, Semantics of Business Vocabulary and Business Rules (SBVR), `http://www.omg.org/spec/SBVR/1.0/`