FORML 2

Terry Halpin[1] and Jan Pieter Wijbenga[2]

[1] LogicBlox, Australia and INTI Education Group, Malaysia
[2] University of Groningen, The Netherlands
terry.halpin@logicblox.com, J.P.Wijbenga@student.rug.nl

Abstract. A conceptual schema of an information system specifies the fact structures of interest as well as the business rules that apply to the business domain being modeled. These rules, which may be complex, are best validated with subject matter experts, since they best understand the business domain. In practice, business domain experts often lack expertise in the technical languages used by modelers to capture or query the information model. Controlled natural languages offer a potential solution to this problem, by allowing business experts to validate models and queries expressed in language they understand, while still being executable, with automated generation of implementation code. This paper describes FORML 2, a controlled natural language based on ORM 2 (second generation Object-Role Modeling), featuring rich expressive power, intelligibility, and semantic stability. Design guidelines are discussed, as well as a prototype implemented as an extension to the open source NORMA (Natural ORM Architect) tool.

1 Introduction

A conceptual information model includes a conceptual schema as well as a population (set of instances that conform to the schema). Ideally, a conceptual schema specifies the fact structures of interest as well as the applicable business rules in terms of concepts that are intelligible to the business users. Business rules are constraints or derivation rules that apply to the relevant business domain. Alethic constraints restrict the possible states or state transitions of fact populations. Deontic constraints are obligations that restrict the permitted states or state transitions of fact populations. Derivation rules enable some facts to be derived from others.

Business rules may be complex (since the domain being modeled may itself be complex), and are best validated with subject matter experts, who best understand the business domain. In practice, business domain experts may lack the technical expertise required to understand the technical languages used by modelers to capture or query the information model. These languages may be graphical (e.g. class diagrams in the Unified Modeling language (UML) [29]), or textual (e.g. the Object Constraint Language (OCL) [30, 37] supplement to UML). *Controlled natural languages* (unambiguous subsets of natural languages with restricted grammar and vocabulary) offer a potential solution to his problem, by allowing business experts to validate models and queries expressed in language they understand, while the models/queries can still be executable, with automated generation of implementation code.

I. Bider et al. (Eds.): BPMDS 2010 and EMMSAD 2010, LNBIP 50, pp. 247–260, 2010.
© Springer-Verlag Berlin Heidelberg 2010

In *fact-oriented modeling* approaches, all facts are treated as instances of fact types, which may be existential (e.g. Patient exists) or elementary (e.g. Patient smokes, Patient is allergic to Drug). In attribute-based approaches such as Entity Relationship modeling (ER) [7] and UML's class diagramming technique, facts may be instances of attributes (e.g. Patient.isSmoker) or relationship types (e.g. Patient is allergic to Drug).

Fact-oriented modeling approaches include *Object-Role Modeling* (*ORM*) [18], Cognition-enhanced Natural Information Analysis Method (CogNIAM) [28], the Predicator Set Model (PSM) [25], and Fully-Communication Oriented Information Modeling (FCO-IM) [1]. The Semantics of Business Vocabulary and Business Rules (SBVR) initiative is also fact-based in its use of attribute-free constructs [31]. An overview of fact-oriented modeling approaches, including history and research directions, may be found in [16].

This paper discusses *Fact-Oriented Modeling Language version 2* (*FORML 2*), a controlled natural language based on second generation ORM (ORM 2) [13]. An introduction to ORM may be found in [14, 18], a thorough treatment in [22], and a comparison with UML in [17]. FORML 2 is a formal yet intelligible textual language with rich expressive power and high semantic stability. The body of this paper discusses its main features and design guidelines, as well as a prototype implementation of FORML 2 as an extension to the Natural ORM Architect (NORMA) tool.

The rest of this paper is structured as follows. Section 2 briefly overviews related work on high level textual languages, both within and outside the fact-oriented community, as well as providing some background on ORM. Section 3 provides a brief overview of the NORMA tool, and its modes of support for FORML 2. Section 4 discusses some of the underlying algorithms and design guidelines. Section 5 provides details of the formal grammar and implementation architecture used to support FORML 2 as an input language. Section 6 summarizes the main contributions and outlines future research directions.

2 Background and Related Research

Provision of a high level, executable rule language based on a formal subset of natural language that is intelligible to ordinary business users has the potential to revolutionize the way software systems are developed. It is not surprising therefore, that many attempts have been made, and are being made, to achieve this goal. The first language of this nature, Reference and Idea Language (RIDL) [27], was developed in the 1980s based on an early version of NIAM; the model declaration part was implemented in the RIDL* tool, but relationships were restricted to binaries, and the query part was never implemented.

Following RIDL, other fact-oriented modeling languages were developed. One of us specified the first version of FORML to capture ORM constraints in textual form, and in the 1990s provided the patterns used to automatically generate verbalizations of constraints in some early ORM tools (InfoDesigner, InfoModeler, VisioModeler, Microsoft Visio for Enterprise Architects). At that stage, FORML was mainly an output language (while fact types could be entered in FORML, constraints and derivation rules could not be entered textually, but were instead verbalized from models that had been entered diagrammatically). The Language for Information Structure and Access

Descriptions (LISA-D), based on PSM, was specified [25] by researchers at Radboud University, and later extended to the Fact Calculus [26], but was not fully implemented. ConQuer, an ORM-based language for conceptual queries, was implemented in the ActiveQuery tool, which generated SQL code from ConQuer queries [5]. However, ActiveQuery did not allow models to be entered in textual form, and is no longer available. The Constellation Query Language (CQL) is currently under development in the Active Facts tool [23] to provide ConQuer-like functionality (a BNF grammar for CQL is accessible at [24]).

Outside the fact-oriented modeling community, several high level textual modeling and/or query languages were developed. Of those based on information modeling approaches, the most widely adopted is OCL, a formal language used to augment UML class models with rules that cannot be expressed graphically in UML [30, 37]. While useful, OCL has three main drawbacks: its attribute-based nature leads to semantic instability; its rule contexts are restricted to classes; and OCL expressions are often too technical for business users to understand and hence validate.

For example, consider the ORM schema in Fig. 1(a). The entity types Person and Gender are depicted as named, rounded, solid rectangles with their reference modes in parenthesis. The value type PersonTitle is depicted with a dashed line. Relationships are depicted as named sequences of role boxes (a role is a part in a relationship). Solid dots depict mandatory role constraints, bars depict uniqueness constraints, the value constraint on gender code is listed in braces, and the circled subset operator with connectors depicts a join-subset constraint. The explicit mandatory, uniqueness, and value constraints verbalize in FORML thus: **Each** Person is of **exactly one** Gender; **Each** Person has **exactly one** PersonTitle; **Each** PersonTitle is restricted to **at most one** Gender; **The possible values of** Gender **are** 'M', 'F'. Some person titles (e.g. 'Mr', 'Mrs', 'Ms', 'Lady') are restricted to a single gender, while others are not (e.g. 'Dr', 'Prof.'). The fact type PersonTitle is restricted to Gender is used to record any such restrictions.

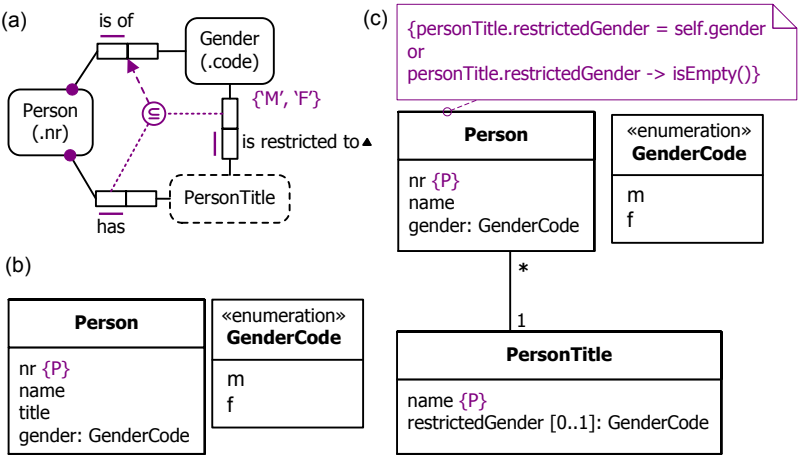

Fig. 1. (a) ORM schema captured (b) partly and (c) fully in UML

The join subset constraint ensures that the set of (Person, Gender) pairs projected from the person and gender roles in the path from Person through PersonTitle to Gender must be a subset of the (Person, Gender) pairs projected from the Person is of Gender fact type. The path at the subset end involves a conceptual join on PersonTitle. This constraint may be verbalized in FORML thus:

If a Person has **a** PersonTitle **that** is restricted to **some** Gender
then that Person is of **that** Gender.

Fig. 1(b) shows how a novice modeler might model this example in UML (although we've extended the UML notation here to include a {P} constraint for the primary value-based identifier). This loses the ability to express the subset constraint, because title is modeled as an attribute, so it cannot participate in an association. Attribute-based approaches are inherently unstable, because if ever one later needs to have an attribute play a role, the attribute needs to be remodeled, along with any rules or queries involving the attribute. Fig. 1 (c) shows how an experienced UML modeler might model this example. UML has no graphic notation for join subset constraints, so the subset constraint is instead captured by the OCL rule shown. This syntax is opaque to non-technical business people so is useless for validating the rule. While the intelligibility of OCL could in principle be addressed by a friendlier surface syntax, this has yet to occur, and would not solve the semantic instability problem.

Some textual languages for ER have been proposed (e.g. see section 16.3 of [22]), but these are limited in scope, and share with OCL the problem of semantic instability caused by an underlying attribute-based model.

Some business rules languages simply capture rules in a semi-natural language using patterns, while lacking the formal underpinnings to generate code (e.g. RuleSpeak [33]). Controlled natural languages are often linguistics-based, employing a formal, executable subset of natural language (typically English). Some of these languages use a linguistic, artificial intelligence approach, perhaps drawing upon existing public lexicons. Attempto Controlled English (ACE) [1, 35] supports a wide range of natural statements and queries, relying on interpretation rules (e.g. **and** has priority over **or**) to enable its text to be automatically and unambiguously translated into discourse representation structures, a syntactic variant of first-order logic. John Sowa's Common Logic Controlled English (CLCE) [36] has the full expressibility of first-order logic (FOL), while providing a semi-natural syntax that can be automatically translated into FOL. As discussed later, CLCE's use of untyped variables tends to make its expressions look more mathematical than natural.

In contrast to some controlled natural languages, Processable ENGlish (PENG) [35] uses a controlled lexicon of predefined function words as well as domain-specific content words that can be defined by the author on the fly. PENG texts can be deterministically translated into discourse representation structures or FOL for theorem proving. Like PENG, FORML restricts its lexicon to the model currently defined by the user. FORML derivation rules, constraints, and queries are constrained to the ORM model of interest (e.g. a derivation rule body for a new fact type must not refer to object types or fact types that are not already declared in the ORM schema).

Further details on controlled natural languages may be found on Rolf Schwitter's Website [34], as well as Jonathan Pool's review of controlled languages [32], which also includes an extensive list of references.

3 Overview of NORMA and Its FORML 2 Support

NORMA is an ORM 2 tool under development that is implemented as a plug-in to Microsoft Visual Studio. Most of NORMA is open-source, and a public domain version is freely downloadable [29]. Fig. 2 summarizes the main functions of the tool. Users may enter object types, fact types, and reference modes textually in FORML using the Fact Editor. These model components are stored in the conceptual model and automatically displayed in diagram form in the ORM diagrammer as well as in an explorer-layout in the Model Browser. Sample object and fact instances are entered in tabular format in the sample Population Editor.

At the time of writing, the public domain version requires ORM constraints and derivation rules to be entered in the ORM diagrammer or the Properties Window. 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 [9].

Using mappers, ORM schemas may be auto-transformed into various implementation targets, including relational schemas for popular DBMSs (SQL Server, Oracle, DB2, MySQL, etc.), datalog, .NET languages (C#, VB etc.), and XML schemas. A Relational View extension displays the relational schemas in diagram form.

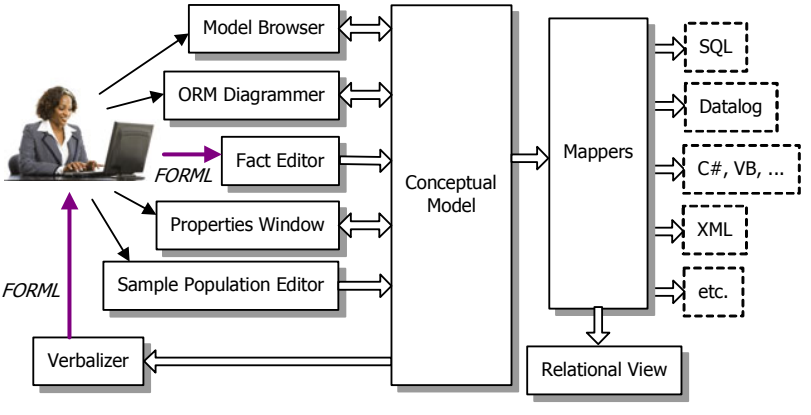

Fig. 2. Overview of main aspects of the NORMA tool

To facilitate validation of ORM models with domain experts, and to provide feedback to modelers on the meaning of ORM diagrams, the Verbalizer automatically verbalizes the models (or any selected part of them) in FORML. This makes use of FORML as an *output language*. For example, Fig. 3 is a screenshot from NORMA showing two fact types with spanning uniqueness constraints and a pair-exclusion constraint. For this screenshot, the top uniqueness constraint and the exclusion constraint have been selected. The rather verbose verbalization of the uniqueness constraint clarifies the *m:n* and set-based nature of the review fact type, and the verbalization of the exclusion constraint is also easily understood. Domain experts can validate the verbalization without needing to understand or even view the diagrams.

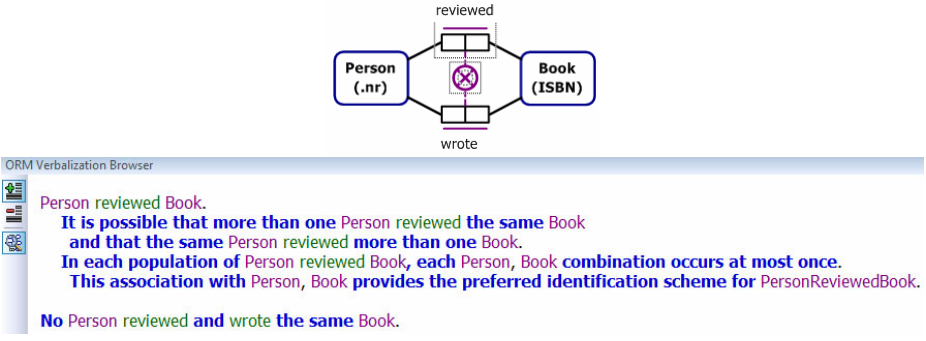

Fig. 3. NORMA screenshot showing verbalization of two selected constraints

Verbose *output* verbalizations are acceptable, since they are automatically generated, and are intended to clarify the semantics for nontechnical users. Indeed, in practice, industrial practitioners have typically rated the automated verbalization support in NORMA to be one of its most valuable features. However, for *inputting* FORML expressions, more concise formulations are often desirable. For example, the uniqueness constraint verbalized in Fig. 3 may be declared simply by appending "m:n" (for many-to-many) to the fact type entry.

ORM's rich expressivity makes full verbalization of ORM models a non-trivial task. One challenging aspect is verbalization of constraints or derivation rules involving join paths, of which the subset constraint in Fig. 1(a) provides a simple example. Some basic patterns for join constraint verbalization were considered in [18], and the role calculus framework for capturing derivation rules was introduced in [8]. The professional version of NORMA has just been extended to support verbalization of join constraints and of derivation rules entered in the Model Browser.

Previously, use of FORML as an *input language* has been restricted to entry of object types, fact types, and reference modes. It has been a long term goal for NORMA to enable users to enter full ORM models (including constraints and derivation rules) in purely textual form using the FORML language. As two initial but significant steps to meet this goal, in the last several months we have refined the FORML grammar, and implemented a prototype to enable most FORML constraints and derivation rules to be entered in an extended form of the Fact Editor.

An even longer term goal for FORML is to support its use as a *conceptual query language*. We have designed FORML to include query capability, and the work we have done on support of derivation rules can be leveraged to provide this capability (a query may be viewed as a request to derive information using asserted or derived fact types). However, in this paper our discussion of FORML focuses on its use for expressing *constraints and derivation rules*. The next section discusses the main features of FORML as well as various design guidelines used in its specification. Implementation aspects are discussed in Section 5.

4 FORML 2 Features and Design Guidelines

Formally, FORML 2 is based on sorted, first order logic plus bag and set comprehension, as well as making use of basic modal operators. Apart from a list of predefined functions, FORML predicates and types are restricted to those that have been declared in the ORM schema at the time the constraint or derivation rule is specified. By default, the *modality* of constraints is assumed to be alethic, although alethic modality may be explicitly included by prepending "**It is necessary that**" to constraints in positive form (e.g. **It is necessary that each** Person was born on **at most one** Date) or inserting "**It is impossible that**" to constraints in negative form (e.g. **For each** Person, **it is impossible that that** Person was born on **more than one** Date). For deontic counterparts, "**necessary**" and "**impossible**" is replaced by "**obligatory**" and "**forbidden**" respectively.

In FORML, object type names start with a capital letter to distinguish them from predicate text, and formal words are highlighted (e.g. by bolding, coloring, or use of special delimiters). Fact types may be objectified, and then treated as object types. For example: Patient is allergic to Drug (**objectify as** Allergy); Allergy was detected on Date. *Hyphen binding* (forward or reverse) may be used to bind modifiers to terms, ensuring quantifiers are placed for natural verbalization. For example, "most- influential Person" and "Person of highest -influence" keep the modifiers with Person irrespective of quantifiers.

FORML expressions typically read more naturally than expressions in other fact-oriented languages. For example, the LISA-D based fact calculus rule "NO Official-paper of A Car being returned BUT NOT being returned" [24] is verbalized in FORML as "**No** Car **that** is returned has **some** OfficialPaper **that** is **not** returned".

FORML allows use of *pronouns* such as "**that**" (for impersonal types) and "**who**" (for personal types) as well as *typed variables, possibly subscripted* (e.g. Person_1) instead of untyped variables (e.g. x, y, z_1, z_2) for *correlation*. Compare the CLCE rule

> If some person x is a parent of some person y, and the person y is a parent of some person z, then the person x is a grandparent of the person z.

with the FORML rule, rendered in *relational style* (which uses predicate readings)

> Person_1 is a grandparent of Person_2 **iff**
> Person_1 is a parent of **some** Person_3 **who** is a parent of Person_2.

Here, "**iff**" may be expanded to "**if and only if**". Variables in the head of a rule are assumed to be universally quantified, but such quantifications may be made explicit by prepending a **for-each** clause, e.g.

> **For each** Person_1 **and** Person_2, Person_1 is a grandparent of Person_2 **if and only if**
> Person_1 is a parent of **some** Person_3 **who** is a parent of Person_2.

FORML rules may also be rendered in *attribute style* (which uses role names) using either dot notation or of-notation (which reverse the dot order). For example, assuming the role name "parent" is declared, the above rule may be stated in attribute style as either of the following: **For each** Person, grandparent = parent.parent.; **For each** Person, grandparent = parent **of** parent.

Attribute style typically gives more compact expression for arithmetic derivation rules, e.g. **For each** Invoice, total = **sum**(lineItem.(quantity * unitprice). *Mixed style* (which allows a combination of relational and attribute styles) is also permitted.

Conjunction and *disjunction* is handled using "**and**" and "**or**" (treated as inclusive-or). Like English (but not ACE), FORML gives these operators equal precedence, so commas or brackets are used where necessary to disambiguate combinations. *Negations* may be expressed by prepending "**It is false that**" but may often be rendered more naturally using "**no**", e.g. **Each** NonDriver **is a** Person **who** drives **no** Car.

Currently, FORML rules assume that all types and predicates are predeclared, and no linguistic machinery is employed to recognize variations in number (e.g. plurals) or voice etc. Hence all object type names should be singular (e.g. Person, not Persons or People). *Pluralization* in rules is obtained by using "**instances of**". For example, instead of **Each** Person is a child of **at most** 2 Persons, we instead say **Each** Person is a child of **at most** 2 **instances of** Person. Moreover, NORMA is incapable of deducing that the fact type reading "Person drives Car" is equivalent to "Person does drive Car" (thus eliminating generation of alternate negations such as **Each** NonDriver **is a** Person **who** does **not** drive **any** Car.). However, users are free to manually provide multiple readings for the same fact type, any of which may then be used in rules.

FORML rules may navigate freely across ORM schemas, introducing joins, operators, and functions, so care is needed to ensure disambiguation. For example, Fig. 3 includes three equivalent derivation rules for the subtype RecentlyLicensedDrivingDoctor. Rule (1), in relational style, uses instance-level "**is**" predicates to navigate to supertypes. The other rules in attribute style need to disambiguate reference to the relevant licenseDate role. Rule (2) uses an "**as**" clause to *cast* DrivingDoctor as Doctor, then accesses the closest licenseDate role. Rule (3) is similar, but uses the implicit rolename "asDoctor" for the casting.

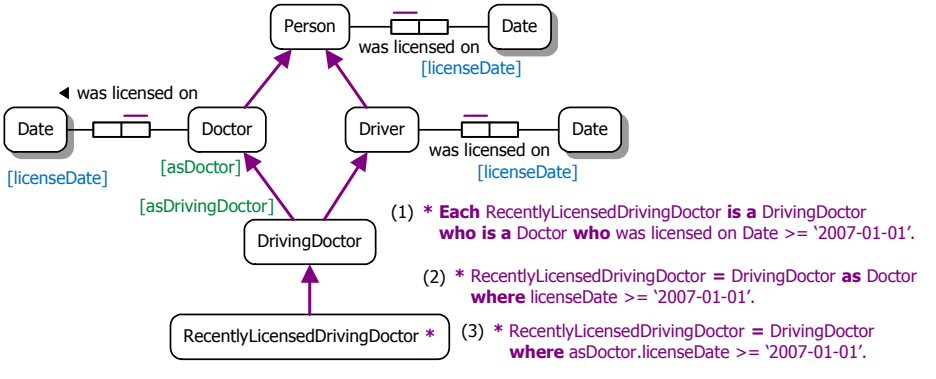

Fig. 4. Three equivalent subtype definitions with unambiguous reference to licenseDate

This illustrates the following procedure. For a given subtype, if multiple far roles have the same name, then disambiguate as follows:

If a direct far role of the subtype has that role name
then that role is chosen
else if only one of its supertype chains includes a supertype with a far role with that name
    then choose the first such supertype far role found (moving up the chain)
    else explicitly include the relevant subtyping connections before the rolename
        (use dot notation or prepend "**as** " to the name of each sub/supertype being navigated to).

While FORML permits all type variables to be subscripted (as in ConQuer [5]), it also includes various procedures to minimize the use of subscripts, thus allowing more natural formulation. For example, object variables introduced in the body of a fact type derivation rule are assumed to be existentially quantified. However, "**some**" may be prepended to make the quantification explicit, and "**that**" must be prepended when correlating unsubscripted variables to previous, unsubscripted occurrences of that variable. For example, the following two explicit formulations are allowed, but may be abbreviated to the later formulations:

*Person drives imported- Car **iff** Person drives Car **that** is imported from **some** Country.
*Person$_1$ is a father **iff** Person$_1$ is a parent of **some** Person$_2$ **and** Person$_1$ is male.

*Person drives imported- Car **iff** Person drives Car **that** is imported from Country.
*Person$_1$ is a father **iff** Person$_1$ is a parent of Person$_2$ **and** Person$_1$ is male.

There is no space here to cover all of FORML's disambiguation rules, but the above examples are representative of the kinds of rules adopted.

5 FORML 2 Grammar and Implementation

After specifying a basic input grammar for FORML 2 in Extended Backus-Naur Form (EBNF) [23], supplemented by some metarules (e.g. fact type readings referenced in rules must be pre-declared), we implemented a prototype of the grammar as an extension to NORMA's Fact Editor. To test the grammar, we used the ANTLR parser generator [30] and the ANTLRWorks development environment [4]. ANTLRWorks accepts EBNF-grammars and also supports a range of non-context-free parsing aids (e.g. gated semantic predicates). Our main reasons for choosing ANTLR include good tool support, code generation to C#, LL(*) "infinite" lookahead, and non-EBNF features like semantic predicates. These features facilitated quick iterations of the test/implement cycle [20]. A test suite of sample rules was constructed in collaboration with some ORM modelers. The test suite and a sample parse tree is accessible at http://home.kpn.nl/wijbe113/.

A FORML sentence is a sequence of textual items such as object type names, full or partial predicate readings, role names, *formal items* (operators, quantifiers, pronouns, etc.), constants (e.g. individual names or numbers), and punctuation marks (e.g., ",", "."). Object type terms start with a capital letter. Subscripts distinguish object variables of the same type. Formal items are comprised of *pseudo-reserved words*, and are displayed in a different *text style* (e.g. **bold**). Words within formal items are not fully reserved, since some of them may be used in a predicate reading. For example, in the following derivation rule, the third "a" is a formal item, but not the other instances of "a". While informally all four instances of "a" have the meaning of an existential quantifier, only the bolded "a" is formally interpreted as such.

Person$_1$ is a grandfather of Person$_2$ **iff** Person$_1$ is a parent of **a** Person$_3$ **who** is a parent of Person$_2$.

We refer to this syntax as *front-end FORML* since we plan to support it in the user interface (an extended Fact Editor) for inputting FORML text. However, while the verbalizer fully supports rich text controls for output FORML, to expedite the implementation of the prototype for input FORML, we delayed rich text editing support for that, instead using a *back-end FORML* grammar that distinguishes formal items and

subscripts by addition of special characters. For example, formal items are included in braces, and subscripts are indicated by prepending dollar signs. The above rule appears in back-end FORML thus: Person$1 is a grandfather of Person$2 {iff} Person$1 is a parent of {a} Person$3 {who} is a parent of Person$2{.}. For convenience, the other examples in this paper are displayed in front-end FORML.

Now consider the following derivation rule. The first occurrence of "**that**" appears just after an object term (Office), so is a relative pronoun used to perform a conceptual join on Office. The second occurrence of "**that**" precedes an object term (Building) so is an indicative pronoun referencing an earlier occurrence of that term. In back-end FORML these occurrences are distinguished as {that} and {jointhat}.

Employee works in Building **iff** Employee works in **some** Office **that** is in **that** Building.

The body of this rule (the part after "**iff**") contains two parts: Employee works in **some** Office; and **that** is in **that** Building. The second part starts with a join pronoun that stands for the referenced object variable. Each of these parts is called a Quantified Correlated Reading Instance (QCRI). Every QCRI contains a reading of exactly one fact type. This reading may have any arity (1 or more). It may have front text preceding the first object term, end text after the last object term, as well as hyphen-bound text. Each object type term may be subscripted, and may be preceded by quantifiers (e.g. "**some**", "**each**", "**no**") or by the relative pronoun "**that**".

As a simple example of an attribute-style rule, consider the following derivation rule, which navigates using far-role names.

For each Window, area = height * width.

In back-end FORML, this is written {For-each} Window, [area] = [height] * [width] {.}. Fig. 5 shows the parse tree for this input, as generated by the ANTLRWorks interpreter.

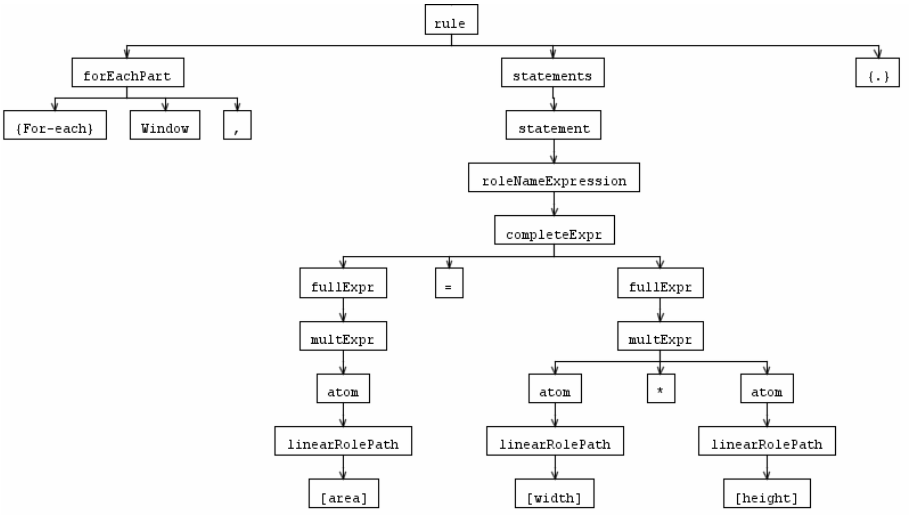

Fig. 5. Parse tree generated by ANTLRWorks for the area derivation rule

Another nice feature of the ANTLRWorks tool is its ability to automatically display the EBNF as railroad diagrams. Fig. 6 shows an example from our grammar.

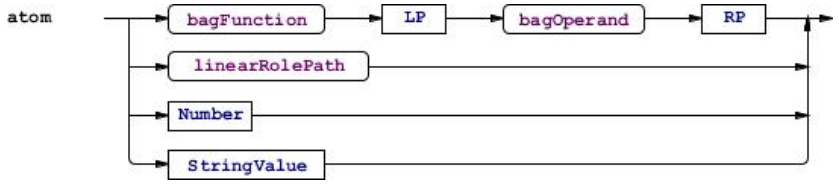

Fig. 6. Railroad diagram for EBNF definition of atom

Validating back-end FORML requires more than simple syntactic checks based on the EBNF grammar. For example, fact type readings referenced in rules must already be present in the ORM schema. When some text is parsed as a QCRI, its fact type reading is extracted and compared to the fact type base. The extraction normalizes whitespace, deletes subscripts and quantifiers, and if the QCRI is contracted then the last object type name from the previous QCRI is prepended to the reading. The following ANTLR rule matches a contractedQCRI.

```
contractedQCRI
@after{
$linearPath::lastOT = LastOTFrom($linearPath::lastOT,$pt);
}
        :       pt+=LCWord+
                (
                (quantifier ( pt+=LCWord+)?)? pt+=ObjectType Sub?
                (pt+=LCWord+ (quantifier ( pt+=LCWord+ )? )? pt+=ObjectType)*
                pt+=LCWord*)?
                // validating semantic predicate
                {IsReading($linearPath::lastOT,$pt)}?;
```

The two sections in braces contain the semantic actions. The last encountered object type is stored as a variable LastOT that is defined in a higher scope. The method calls LastOTFrom() and IsReading() call methods that are located in a manually coded extension of the generated parser. C# partial classes provide an elegant solution for extending the generated parser with method implementations.

Now consider the subtype derivation rule just below. Initially, we treated this as ambiguous, with the two different meanings shown in the subsequent rules. Detecting such cases requires checking whether a contracted QCRI is preceded by end text.

***Each** LazyDogOwner **is a** Person **who** owns **some** Dog **that** barks **and** is lazy.
***Each** LazyDogOwner **is a** Person **who** owns **some** Dog **that** barks **where that** Dog is lazy.
***Each** LazyDogOwner **is a** Person **who** owns **some** Dog **that** barks **where that** Person is lazy.

We are now considering allowing such expressions for input, by adopting this implicit previous subject rule: If a clause beginning with "**and**" or "**or**" immediately precedes predicate occurrence *R2*, and the previous predicate occurrence *R1* has no front text and is either a unary or an infix binary, then the subject of *R2* is the subject of *R1*.

For output verbalization, however, the expanded form (second formulation above) would be used to ensure users are aware of the interpretation taken.

6 Conclusion

This paper discussed the use of FORML 2 as a high level textual language that can be used with the NORMA tool for both input of ORM 2 models and output verbalization of ORM 2 models, including constraints and derivation rules. The combination of automatic transformation between textual and diagrammatic forms, and the wide range of rules covered, distinguishes our approach from many other approaches based on controlled natural languages. While the work on output FORML is relatively mature, the work on input FORML is still in its early stages, and further research is needed to exploit the full potential of this approach.

Future plans include a rich text editor for inputting front-end FORML rather than back-end FORML. The translation of derivation rules in input FORML to the role calculus form also requires more research. Use of a GLR parser generator could be considered as an alternative to the current LL(*) grammar. Instead of asking the user to disambiguate right away, a GLR parser tolerates ambiguity and generates a parse forest from which the intended parse tree can be chosen. Other plans include support for pluralization, dynamic rules [3], conceptual queries, and non-English languages.

References

1. Attempto project (Attempto Controlled English), http://attempto.ifi.uzh.ch/site/
2. Bakema, G., Zwart, J., van der Lek, H.: Fully Communication Oriented Information Modelling. Ten Hagen Stam (2000)
3. Balsters, H., 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)
4. Bovet, J., Parr, T.: ANTLRWorks: an ANTLR grammar development environment. In: Software: Practice and Experience, vol. 38(12), pp. 1305–1322. John Wiley, Chichester (2008)
5. 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)
6. Business Rules Solutions Website on RuleSpeak, http://www.rulespeak.com/en/
7. Chen, P.P.: 'The entity-relationship model—towards a unified view of data'. ACM Transactions on Database Systems 1(1), 9–36 (1976)
8. 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)
9. Halpin, T.: ORM 2. In: Meersman, R., Tari, Z., Herrero, P. (eds.) OTM-WS 2005. LNCS, vol. 3762, pp. 676–687. Springer, Heidelberg (2005)
10. Halpin, T.: ORM/NIAM Object-Role Modeling. In: Bernus, P., Mertins, K., Schmidt, G. (eds.) Handbook on Information Systems Architectures, 2nd edn., pp. 81–103. Springer, Heidelberg (2006)

11. Halpin, T.: Modality of Business Rules. In: Siau, K. (ed.) Research Issues in Systems Analysis and Design, Databases and Software Development, pp. 206–226. IGI Publishing, Hershey (2007)
12. 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)
13. Halpin, T.: A Comparison of Data Modeling in UML and ORM. In: Khosrow-Pour, M. (ed.) Encyclopedia of Information Science and Technology, 2nd edn., Information Science Reference, Hershey PA, USA, vol. II, pp. 613–618 (2008)
14. Halpin, T.: Object-Role Modeling. In: Liu, L., Tamer Ozsu, M. (eds.) Encyclopedia of Database Systems. Springer, Berlin (2009)
15. Halpin, T.: Predicate Reference and Navigation in ORM. In: Meersman, R., Herrero, P., Dillon, T. (eds.) OTM 2009 Workshops. LNCS, vol. 5872, pp. 723–734. Springer, Heidelberg (2009)
16. Halpin, T.: Object-Role Modeling: Principles and Benefits. International Journal of Information Systems Modeling and Design 1(1), 32–54 (2010)
17. Halpin, T., Morgan, T.: Information Modeling and Relational Databases, 2nd edn. Morgan Kaufmann, San Francisco (2008)
18. Heath, C.: The Constellation Query Language. In: Meersman, R., Herrero, P., Dillon, T. (eds.) OTM 2009 Workshops. LNCS, vol. 5872, pp. 682–691. Springer, Heidelberg (2009)
19. Heath, C.: ActiveFacts Website (2009), http://dataconstellation.com/ActiveFacts/
20. Hevner, A., March, S., Park, J., Ram, S.: Design Science in Information Systems Research. MIS Quarterly 28(1), 75–105 (2004)
21. 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)
22. Hoppenbrouwers, S., proper, H., van der Weide, T.: Fact Calculus: Using ORM and Lisa-D to Reason about Domains. In: Meersman, R., Tari, Z., Herrero, P. (eds.) OTM-WS 2005. LNCS, vol. 3762, pp. 720–729. Springer, Heidelberg (2005)
23. ISO 1996, Information technology – Syntactic metalanguage – Extended BNF, ISO/IEC 14977 (1966), http://www.iso.org/iso/catalogue_detail.htm?csnumber=26153
24. Meersman, R.: The RIDL Conceptual Language. Int. Centre for Information Analysis Services, Control Data Belgium, Brussels (1982)
25. Nijssen, M., Lemmens, I.: 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)
26. NORMA (Natural ORM Architect) tool download site for public-domain version, http://www.ormfoundation.org/files/folders/norma_the_software/default.aspx
27. Object Management Group 2003, UML 2.0 Superstructure Specification, http://www.omg.org/uml
28. Object Management Group 2005, UML OCL 2.0 Specification, http://www.omg.org/docs/ptc/05-06-06.pdf
29. Object Management Group 2008, Semantics of Business Vocabulary and Business Rules (SBVR), http://www.omg.org/spec/SBVR/1.0/
30. Parr, T.: The Definitive ANTLR Reference: Building Domain-Specific Langauges, 1st edn. Pragmatic Bookshelf, Raleigh (2007)

31. Pool, J.: Can Controlled Languages Scale to the Web? In: Proc. CLAW 2006: 5th International Workshop on Controlled Language Applications (2006), http://utilika.org/pubs/etc/ambigcl/clweb.html
32. Schwitter, R.: Controlled Natural Languages, http://sites.google.com/site/controllednaturallanguage/
33. Schwitter, R.: PENG (Processable English) (2007), http://web.science.mq.edu.au/~rolfs/peng/
34. Sowa, J.: Controlled English (2004), http://www.jfsowa.com/logic/ace.htm
35. Sowa, J.: Common Logic Controlled English (2004), http://www.jfsowa.com/clce/specs.htm
36. Warmer, J., Kleppe, A.: The Object Constraint Language, 2nd edn. Addison-Wesley, Reading (2003)