An Overview of the RECON Controlled English Tool

Ed Barkmeyer, NIST, January 2013

The NIST Restricted English for Constructing Ontologies (RECON) language is intended as a close relative of English that has a well-defined interpretation in a formal logic language. The idea is that, with the help of a speaker of RECON English, a domain expert can formalize the concepts in his/her domain, and at least read the formalizations.

The RECON tool is designed to be a backend plug-in for an authoring tool that does not currently exist. It is currently run via a simple command-line interface program that invokes the RECON tool. The command line for the version 1 RECON tool has the form:

```
java –jar recon.jar
   -gpw -d<dictfile> -v<vocfile> -l<log> -u<uml> axfile [ ikl ]
```

where:

axfile	specifies the source file for RECON axioms to be processed. The axiom file may explicitly import other vocabulary and axiom files, using the $I (import) command, or include vocabulary declarations using the $V and $A commands to change input modes.
-d dictfile	specifies a file that supplies the spellings of all forms of words used in the vocabulary. (RECON uses a default file if this is not specified.)
-g	suppresses generation of the IKL logical form. Unless –g is specified, the IKL form is always generated.
ikl	specifies the file to which the IKL output is to be written. If this parameter is not specified, and IKL generation is not suppressed, the IKL file name will be constructed from the axfile name.
-l log	specifies an output file that will contain the message log (summaries, errors) for the invocation. If this option is not specified, the message log prints to the “error” console (System.err).
-p	causes RECON to prettyprint the parse of each successfully processed definition or axiom to the log file.
- u uml	is an output file that will contain a UML XMI form of the vocabulary. If this is not specified, no UML file is generated.
-v vocfile	specifies a file that supplies the vocabulary to be used for the axiom set provided. This is not required if the axfile contains or imports the vocabulary.
-w	causes RECON to prettyprint its word forms dictionary.

RECON Tool Structure

The RECON tool has five major functional areas:

- Dictionary management
- Vocabulary management
- Statement and definition parsing
- Formal logic generation
- UML generation

RECONManager provides the plug-in interface, initializes the tool, opens the Dictionary file and invokes the “keyword loader” (VocKey).

RECONmain provides the command line interface, and invokes RECONManager via the plug-in interface to process the named files.

RECONversion is a passive object that records the current version and build of the RECON software.

The RECON processing code is captured in a single Java package called RECON.Core. A handful of more general utility routines taken from other projects is in RECON.Utilities.

The RECON data structures are Java object modules created from a UML model using the Eclipse EMF facilities. The active elements of these modules do little more than record information, respond to type queries, and export themselves as strings. The generated module library is called NDVRlib.

Dictionary

Dictionary structure

The Dictionary is a set of words and forms of words. All of the forms of a word – singular and plural forms of a noun, various tenses and participles of a verb – are recognized as the same Word. The Dictionary stores all the WordForms, using a hash index scheme, and relates each to the Word it represents. Nouns have two WordForms – singular and plural. Verbs have five WordForms – infinitive/present plural, present singular, past, progressive participle, past participle. (Example: go, goes, went, going, gone.) Adjectives (currently) have only one WordForm. Other words, like prepositions, are called Adjuncts and have only one WordForm. If a given WordForm is a homograph that represents more than one kind of Word, the one with the most forms is used. The Dictionary software attaches no significance to the actual part of speech; it is only concerned with WordForms.

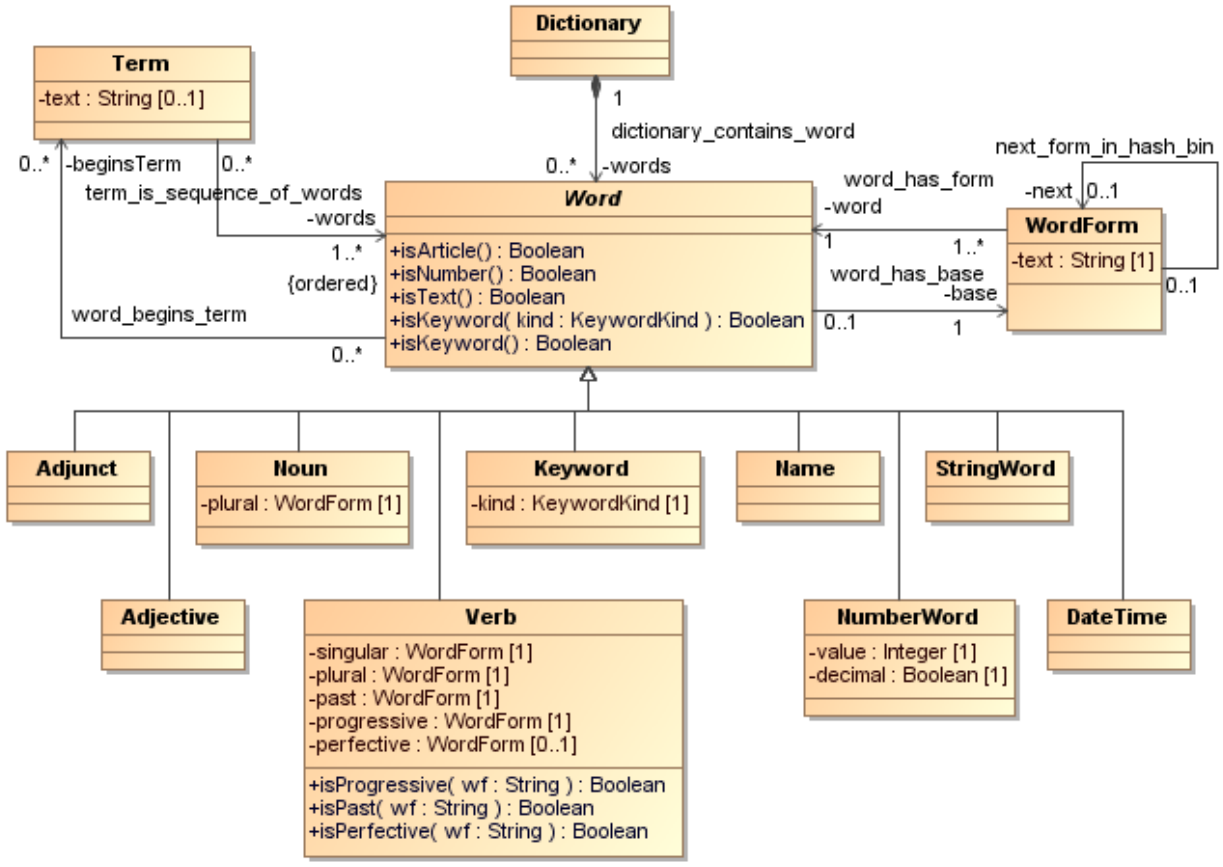

Figure 2: The Dictionary structure

Dictionary process

The Dictionary manager, such as it is, is currently contained in the VocLoad module, because Words and WordForms are currently loaded by declarations, just like terms. The intent is that this should be replaced by a mechanism that selectively constructs Words and WordForms from an English language repository, as the words appear in vocabulary terms.

The VocKey module adds the words and terms that are defined in the RECON grammar to the Dictionary (and to the Vocabulary, where appropriate). VocKey is a set of tables and a handful of routines that process them into the Dictionary and Vocabulary structures.

The Dictionary data structures are part of the NDVRlib.Vocabulary package.

Vocabulary

Vocabulary structure

The Vocabulary is concerned with Terms. A Term is one or more Words. Terms are parts of speech in the RECON language. There are five basic kinds of Terms, described in detail in [NISTIR].

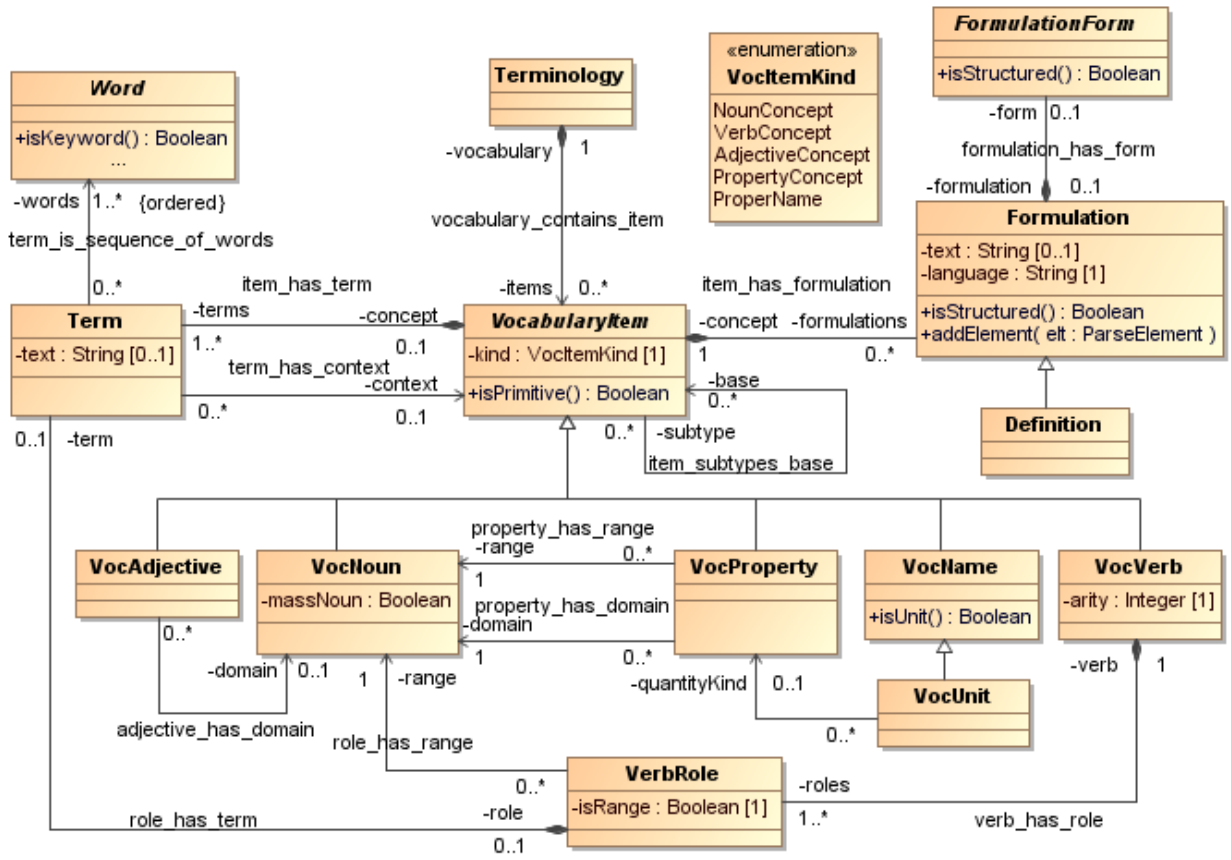

Figure 3: Vocabulary Structure

Noun terms, called VocNouns, are terms that refer to types of things. A special case of VocNoun is a 'mass noun' that refers to a substance.

Verb terms, called VocVerbs, are terms that refer to states and actions. The declaration of a verb term introduces a syntax for the use of the verb term, called a SyntaxForm. The SyntaxForm identifies the places where a noun phrase can appear as an operand in a use of the verb term. These places are referred to as VerbRoles. A VerbRole has a name and a nominal 'type', which specifies a VocNoun as the kind of thing that can play the VerbRole. The verb term is considered to be defined in the context of the VocNoun that is the type of the subject. If the same verb term is declared more than once, the result is multiple VocVerbs, where the type of the subject is used to distinguish them.

Adjective terms are called VocAdjectives. They are defined using a verb SyntaxForm of the form: <type> is <VocAdjective>. The VocAdjective per se is a sequence of words that can modify a noun in a RECON sentence. The verb form can be used as a verb form. In either case, the <type> is treated as the subject of the verb and is used to disambiguate multiple adjectives/verbs with the same spelling.

Property terms are a cross between nouns and verbs. They are declared with a verb SyntaxForm: <range> is <VocProperty> <subject>. But the <VocProperty><subject> part can be used as a noun. (Example: (person) is the owner () of (thing) declares the VocProperty "owner of", so that "the owner of

the vehicle" can be used as a noun phrase.) The verb form can be used directly as well, and the subject is used to disambiguate property terms with the same spelling.

Proper names, called VocNames, are terms that refer to individuals. They are treated much the same way as numbers and quoted character strings in RECON sentences. A special case of proper name is a measurement unit (VocUnit), which can be used in quantity expressions.

Any of these terms can have Synonyms or alternate forms, formal RECON definitions or natural language definitions.

Vocabulary management

The function of the Vocabulary management software is to parse declarations and manage the Vocabulary data structures.

The VocLoad module processes both Dictionary and Vocabulary declarations and builds the appropriate data structures. On encountering a formal RECON definition, it invokes the Parser module to interpret the formal definition. Similarly, on encountering an axiom/statement, it invokes the Parser module to interpret the axiom.

VocLoad processes directives to load additional vocabulary and axiom declaration files, and to change its mode of operation from input of vocabulary entries to input of axioms, rules and facts expressed in the RECON language. In that way it serves as the primary input processor for the whole RECON tool.

The VocFileIn module is used as an input filter to do the basic input character and line management for VocLoad.

All of the Vocabulary elements are contained in NDVRlib.Vocabulary module.

Statement and Definition Parsing

The function of the Parsing software is to parse statements and definitions written in the RECON language and create the corresponding internal parse structures.

Sentence structures

RECON statements of facts, rules and axioms have one of a few forms defined by the RECON grammar [NISTIR]. The formal parse structures are depicted in Figure 4.

A SimpleForm statement is one of the SyntaxForms for a single Vocabulary verb, with each of the VerbRole slots replaced by a RolePhrase (see below).

A CompoundForm is multiple simple statements connected by and's and or's.

A Domain Form has the (traditional mathematics) form "For (some quantified type nouns), <statement>" or "There is/are (some quantified type nouns) such that <statement>", where <statement> can be any simple or compound statement or another DomainForm. A quantified type noun is a kind of RolePhrase and is discussed below.

An ImplicationForm has the form "If <statement1> then <statement", where <statement1> and <statement2> are any statements of the kinds above. It can also be followed by "else <statement3>" or "otherwise <statement3>", where <statement3> is any statement, including another ImplicationForm. There are also other forms of Implication using "only if" and "unless". <Statement1> if and only if <statement2> is actually stored as a CompoundForm.

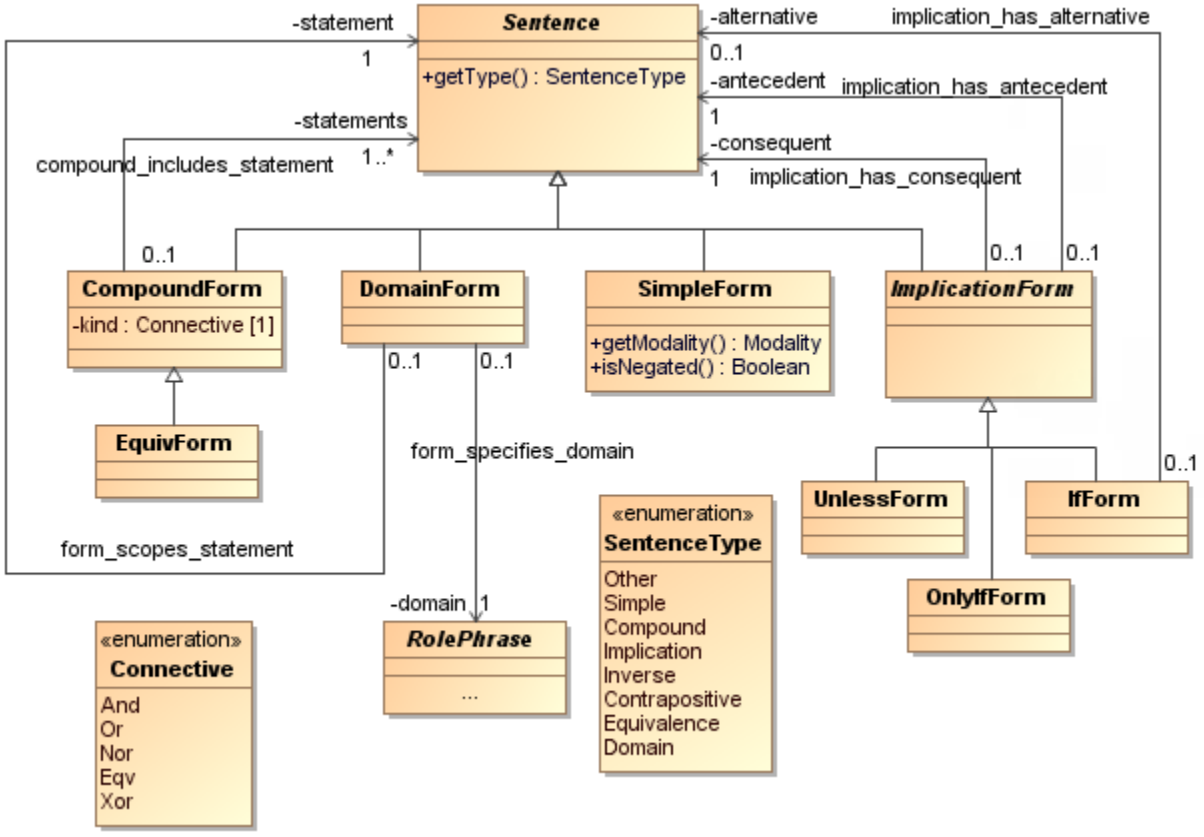

Figure 4: Statement types

Role phrase structures

A RolePhrase is a grammatical structure that identifies the thing or things that play some role in a verb. These can be quite complex and are discussed in detail in [NISTIR]. This is the richly expressive element in RECON. The supporting data structure is shown in Figure 5. There are four major kinds:

An Instance is the simplest RolePhrase. It is a term that refers to a thing, like a number or a proper name.

A TypeNoun is a reference to things by their nature, like "2 or more people" or "a long job that requires an expert mechanic". TypeNouns therefore can have a simple structure – an article followed by a VocNoun – or a complex structure consisting of a quantifier, some adjectives, a VocNoun, and a set of qualifiers beginning with "that", or "who" or "which", and connected by and's and or's. A quantifier is an expression of quantity, like the "2 or more" above, or like "at least 100 metres of". Adjectives are terms that have been declared VocAdjectives. In the example above, "long" is an adjective, and

"expert" might be (or "expert mechanic" might be a term). A qualifier is a simple sentence in which one of the RolePhrases is a pronoun ("that", "who", "which") that refers to a potential instance of the VocNoun. In the example above, "that requires an expert mechanic" is a qualifier in which the subject is "that", referring to the "job".

A PropertyNoun is a reference to a VocProperty of some other thing, where the other thing is described by another RolePhrase. For example, we can refer to "the name of the owner of the vehicle", which involves two VocProperties "name of (person)" and "owner of (thing)."

A GroupPhrase is two or more RolePhrases connected by "both ... and ..." or "either ... or ..." or "neither ... nor ...". It can be as simple as "both Jack and Jill", or as complex as "either the title to the vehicle or a valid driver's license for each of the owners of the vehicle."

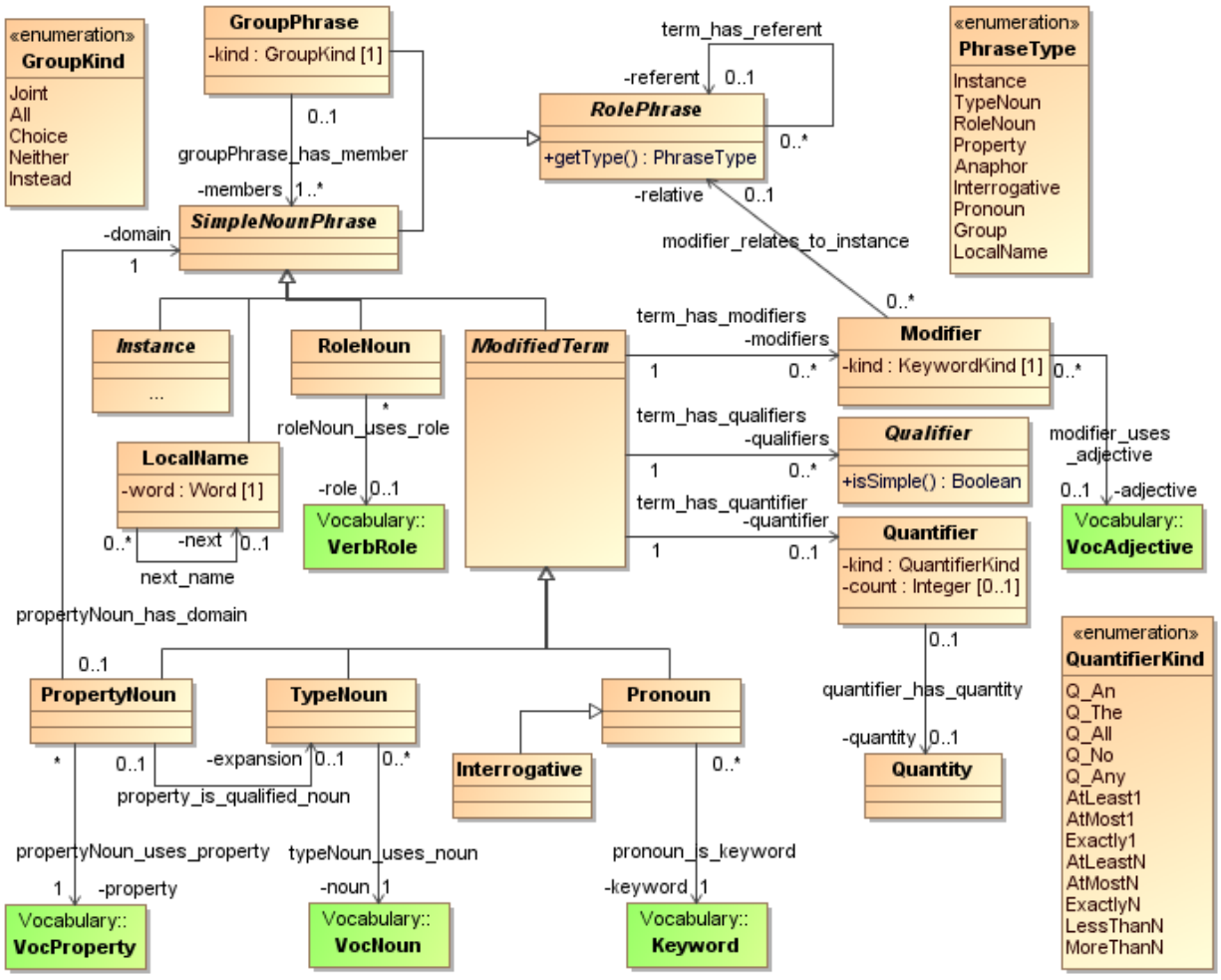

Figure 5: Role Phrases

Parse process

The parsing process is controlled by ParseManager. The parsing process occurs in three steps:

First, the text of the definition or statement is reduced to a sequence of Tokens, each of which is a WordForm (or number or quoted string) associated with a Word. The sequence is called a TokenStream.

Second, the sequence of Words is grouped (and re-grouped) into multiple possible sequences of Terms from the Vocabulary, called TermTokens. It is possible that one Term is the same as the beginning of another, or that two Terms are the same sequences of Words but are used with different subjects, as described above. In this process some Words that are grammatical keywords may be absorbed into Terms that include them.

Third, the most likely sequence of terms (using longest matches and some other heuristic rules) is submitted to the proper Parser entry. If the Parser succeeds, it produces a Formulation object, RECON assumes that is the meaning of the text, and exports the formal logic interpretation (see below). If the Parse fails, the detected Error is queued, and the process returns to the Second step, choosing the next best alternative Term sequence. The alternative to the last choice before the point of failure is taken, and every choice beyond that is reset. The Parser only fails completely if there are no choices left, and in that case the entire queue of error messages is written to the error log.

The Parser module implements the Grammar rules and parses the passed TermToken Stream into the corresponding ParseStructure. It has two entry points with different grammatical targets: the definitions of VocNouns and VocNames are RolePhrases; other definitions and all statements are Sentences. The Parser is a recursive descent parser. There is essentially one routine for each grammar rule, and the function of the routine is to recognize a sequence of terms and keywords that satisfies the grammar rule, and return the corresponding parse structure, or fail and return nothing. If the routine recognizes a part of the grammatical structure that could not be some other structure, it queues an error message before failing. Failing by recognizing nothing special is an expected occurrence. A grammar rule that has optional components is implemented by calling the recognizer for the first component, and if that fails, calling the recognizer for the next alternative.

When the Parser is successful, it returns a Formulation object containing the elements of the parse structure.

Logic Generation

RECON converts all formal definitions and statements to a formal logic model, using the elements depicted in Figure 6.

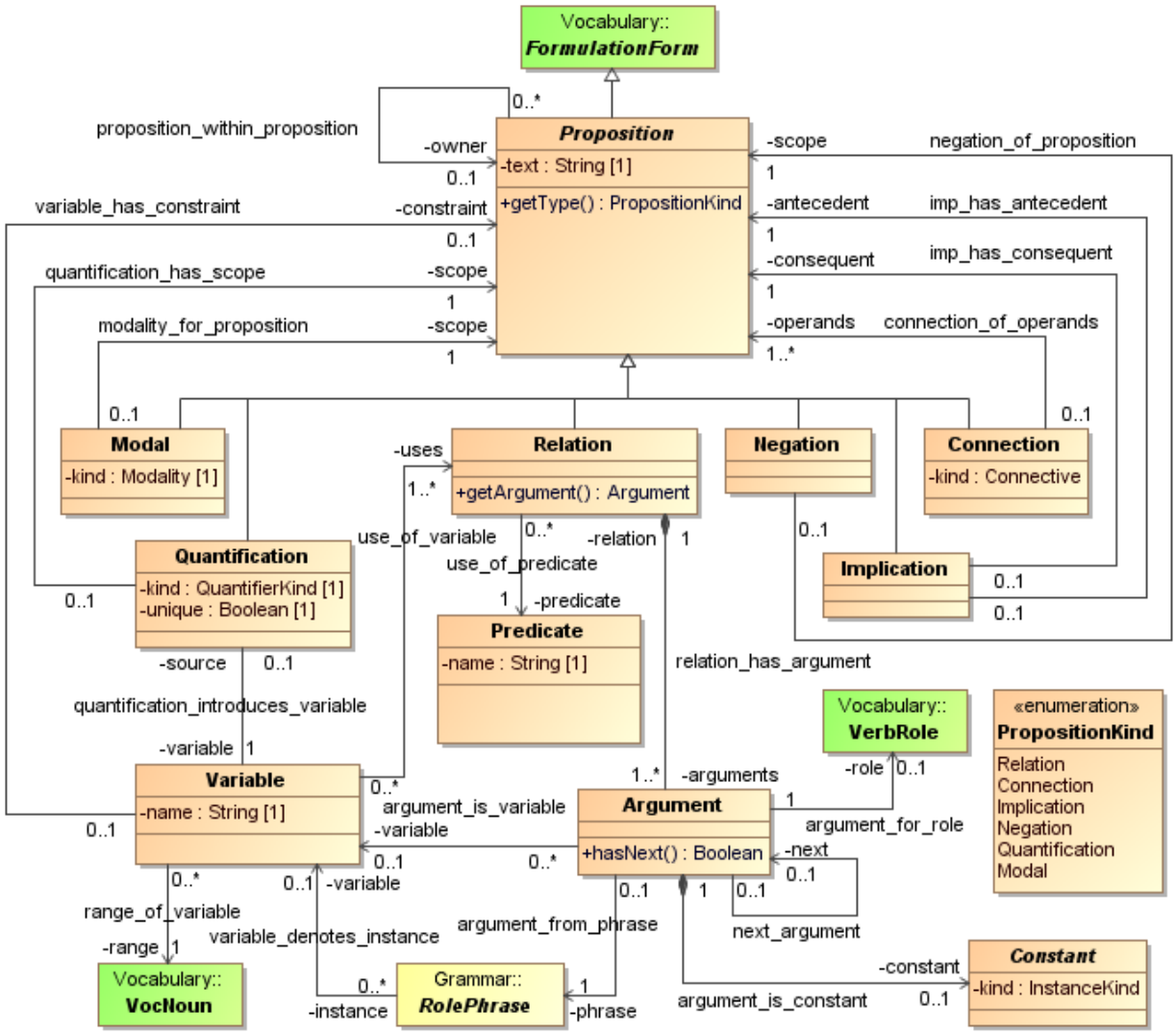

Figure 6: Logic structures

Logic structures

A Proposition is a sentence in the logic language. Every definition and statement is converted to a Proposition. There are several kinds of Proposition.

A Relation is a simple sentence, consisting of a Predicate (the verb, derived from a VocVerb) and its arguments – the things playing the VerbRoles. Since the VerbRoles are syntactically played by RolePhrases in the parse, the arguments are representatives of those RolePhrases. RolePhrases that are Instances (numbers, character strings, proper names, quantities) are represented directly as arguments. Any other RolePhrase is represented by a Variable that refers to the things that may play the VerbRole.

A Quantification creates a Variable and has a scope that is a (possibly complex) Proposition that includes all uses of the Variable. The Variable is derived from a RolePhrase that is a TypeNoun or PropertyNoun in the Parse Formulation, and it is used for all RolePhrases that refer back to that one. (That includes the that/which/who references in qualifiers, local names, pronouns, and 'back references', which are discussed in [NISTIR], but not here.) The Quantification also indicates that the variable corresponds to all instances that match the RolePhrase or some instance that matches the RolePhrase, according to the quantifier used in the RolePhrase.

A Negation has a scope that is a Proposition and means "not" that Proposition. It is derived from a SimpleForm VerbPhrase in the Parse that had a "not" or "never" in it. The scope of the Negation is usually either the Relation derived from the SimpleForm sentence, or some set of Quantifications that introduce Variables used in representing that SimpleForm sentence.

A Modal has a scope that is a Proposition and means that the Proposition is required to be true or required to be false by some rule. It is derived from a VerbPhrase that includes "must" or "must not", in much the same way as a Negation.

A Connection is an 'and' or 'or' of Propositions. Connections are derived from CompoundForms, but they can also be created by "rewriting" some of the original Parse (see below).

An Implication includes an 'antecedent' Proposition and a 'consequent' Proposition and means "If the antecedent is true, then the consequent is true." Implications are derived from ImplicationForms in the Parse Formulation, but they can also be created by "rewriting" some of the original Parse (see below).

Logic Process

The logic generation software consists of three steps: rewrite, interpret, export.

The LogicManager manages the generation process.

The Rewrite module rewrites the Parse graph into a form suitable for formal logic rendering. It has three primary functions:

(1) Rewrite converts adjectives and properties to qualifiers. That is, it replaces the existing parse structures with equivalent verb forms. For example, "any defective shipment from S" is rewritten to "any shipment that is defective and that is a shipment from S".
(2) Rewrite identifies TypeNouns (and rewritten PropertyNouns) that will require logical Variables, and rewrites the Sentence in which they occur into a DomainForm. For example, "if the customer receives any defective shipment, the customer must send a XXX message" is rewritten to "For each shipment that is defective, for each customer that receives the shipment, it is an obligation that there is a XXX message such that the customer sends the XXX message." The reason for this elaborate circumlocution is that it is the pattern for the formal logic expression. In the logic form, "for each shipment that is defective" will become a Quantification that introduces a Variable for the shipment, and each subsequent reference to "the shipment" will refer to that Variable.

(3) Rewrite converts quantifiers that involve quantity expressions to statements about quantities or collections of the TypeNoun . For example, “No container contains more than 50 kg of parts” becomes “No container contains a collection of parts such that the mass of the collection is greater than 50 kg.”

The Interpreter module transforms the rewritten parse graph into its formal logic equivalent, more or less 1-to-1.

An axiom sentence is converted directly to the corresponding Proposition.

A formal definition is converted to a logical equivalence – a Connection whose Connective is “EQV”. For VocNouns, the equivalence is stated as: For every x, x “is a <VocNoun>” if and only if <interpretation of the RolePhrase that is the definition>. Similarly, for VocNames, the equivalence is stated: For every x, x is equal to <VocName> if and only if <interpretation of definition>. For verbs, properties and adjectives, the primary verb form is defined as a Predicate with one variable argument for each VerbRole. So the definition looks like:
(<VocVerb> <role$_1$> ... <role$_n$>) if and only if (<Proposition derived from the definition sentence>).
Note that in Figure 5, one possibility (not discussed above) for a RolePhrase is a RoleNoun – an occurrence of the name of a VerbRole, which is only permitted in the definition of the verb. Interpreter treats those RolePhrases in the defining sentence as references to the <role$_i$> Variables.

A SimpleForm becomes a Relation in which the Predicate is generated from the primary term for the VocVerb and the Arguments are created from the RolePhrases in the SimpleForm in order of the VerbRoles in the VocVerb. Any complex RolePhrase has been reduced by Rewrite to a back reference to an expression in a DomainForm that now has a corresponding Variable.

A DomainForm becomes a Quantification and a corresponding Variable is created. Depending on the quantifier, the scope of the DomainForm is converted to an Implication (each) or an AND Connection (some) that relates any qualifiers on the “domain” of the DomainForm to the actual scope Sentence.

The introductory VocNoun in a DomainForm is treated as an instance of the “is a” verb. That is, “no container” is treated as “no thing that is a container”. The first qualifier on every Variable is therefore the statement that it is an instance of the VocNoun type. This is accomplished by turning the primary term for the VocNoun into a Predicate as well.

A CompoundForm becomes a Connection.

An ImplicationForm becomes an Implication, with suitable assignments of antecedent and consequent, and suitable uses of Negation. If an “else” is present, the ImplicationForm becomes an AND Connection of two Implications, where the first is the interpretation of the “if ... then ...” part, and the antecedent of the second Implication is the negation of the antecedent of the first, and the consequent is the interpretation of the Sentence following “else”.

After the transformation, Interpreter ensures that all of the references to a Variable are within the scope of its defining Quantification, by adjusting quantification scopes, using a minimum spanning tree algorithm.

Export: The final act of the Interpreter is to initiate the process of generating the output text in the IKL logic language [], which is an extension of ISO Common Logic []. Interpreter first produces "cl:comment" elements for the text of the definition or sentence being converted. Then it directs the Proposition object that represents the whole Sentence to export itself. Each element of the Logic structure has a method that exports itself and its components in the form of elements of the IKL.

IKL provides for the logical expression of "nominalizations" – use of a logical proposition as an operand. This is used in the formulation of references to statements, situations, obligations and prohibitions. As indicated above, an IKL vocabulary of "predicates" (the names of relations) are generated from the Vocabulary elements declared by the user. Some RECON-specific terms are added to the user vocabulary. (These deal primarily with quantities and collections.)