Designing Natural Language Objects

Koichi Takeda

Tokyo Research Laboratory, IBM Research
5-19 Sanban-cho, Chiyoda-ku, Tokyo 102, Japan
takeda@trlvm1.inu.isl.ibm.com

In this paper, we propose an architecture to handle natural language objects by which we represent meaning of sentences in a document. These objects and their class definitions can be a promising interface to natural language queries and text generation as well as advanced applications of textual databases including machine translation and text retrieval by content.

1. Introduction

Designing a natural language object has been a tough problem since the early days of business data processing. As domains of database applications extend to textual and multimedia information processing, the design problem has grown even harder although the recent object-oriented approach introduced a richer set of building blocks into the database arena[2].

Natural language (NL, for short) databases are particularly important among non-traditional database applications because NL appears so frequently in our everyday documents, mails, dialogues, etc. It seems quite reasonable that we keep such information as NL databases and everybody can ask/update information through an NL interface. That is, all three forms, data[15], query[6], and answer[17] can be expressed in NL. Since the expressive power of a natural language is far more than that of a relational language, an NL query has to be limited only to retrievals without deep understanding of either NL databases or the query in order to guarantee tractability of query processing. Meaning of sentences, however, can be symbolically stored and handled by a database system if we define a set of semantic concepts and mappings between NL syntax and semantic expressions. Thus, a variety of NL applications, machine translation, NL interface, text retrieval, etc. can be built upon such semantic NL concepts.

In this paper, we propose an architecture to handle NL databases. We define NL classes and NL objects as a set of semantic concepts to be stored in an object-oriented database system. Mapping rules between NL syntax and NL objects are then introduced. We will also discuss an application interface provided by the architecture. The NL classes are designed to satisfy the minimal requirement of our primary NL application, machine translation. The reader can find a massive knowledge base of human commonsense knowledge[7] as a possible extension of our NL classes for broader-range AI applications, where understanding and implications of sentences are incorporated.

2. Natural Language Classes

NL objects are instances of NL classes which define a set of conceptual word senses to represent meaning of sentences. The set of NL classes consists of two special classes, *TOP and *BOTTOM, and two disjoint subsets of classes, open classes and *closed classes. We have is-a relationships defined over NL classes. For any NL class *x, "x is-a "TOP" and "*BOTTOM is-a "x" hold. Open class objects correspond to entities which are expressed by nouns, verbs, adjectives, etc. Closed class objects are meant to be attribute values of some open class objects. (See[3] for the distinction of entities and attributes.) For example, aspects, tense and modals are closed class objects to modify actions while number, gender, and definiteness are closed class objects to modify nominals. Typically, closed class objects represent information conveyed by auxiliary verbs, determiners, prepositions, and inflections of verbs and nouns. The closed classes are stable, primitives classes and they are almost common to all domains and natural languages. The open classes are comprehensive and dynamic, but are dependent on both domains and languages. A class is defined in a frame[8] format as shown below. By convention, class names are preceded by an asterisk, user-defined slot (attribute) names are preceded by a colon, and system-provided slot names are not preceded by any special characters (comments are preceded by semi-colons).

(defun *remove ;; definition of a class *remove
(is-a (value *action)) ;; superclass is *action
 (:agent (sem *human *system))
 ;; agent must be *human or *system
 (:theme (sem *physical-object))
 ;; theme must be *physical-object
 (:source (sem *location *physical-object))
 ;; source must be *location or *physical-object
)

Here, the value facet of the is-a slot shows a literal filler (actual attribute value) of the slot. The sem facet of other slots shows semantic constraints on their potential fillers. Actual fillers of these slots, except is-a slot, are not given in the class definition. The is-a slot and part-of slot are the only system-defined slots for open classes, and all other slots are user-defined. The is-a slot defines generalization relationships among NL classes, which roughly correspond...
to a taxonomy of words. The part-of slot defines whole-part relationships among NL classes, which also frequently appear in the real world such as "a keyboard is part-of a personal computer" or "a nose is part-of a human face". A class can inherit each slot definition from its superclasses unless the slot is redefined in the class. Figure 1 shows a sample hierarchy of NL classes. A more complicated NL hierarchy is used for our machine translation project[14] but not presented here because most part of the hierarchy (open classes) is domain dependent and not related to the database issues.

![Figure 1: Sample NL Class Hierarchy](image)

A class can have more than one immediate superclasses but we restrict its inheritance to be exclusive inheritance rather than general multiple inheritance. That is, an instance of a class can inherit slot definitions from only one of its superclasses. This is how we can realize certain identity of verbal and nominal word senses as well as consistency of semantic constraints. For example, most of verbs have their nominal counterparts and vice versa in natural language, say "remove" and "removal". Such a pair usually shares slot definitions (:agent, :theme, and :source) except that "remove" should inherit tense, aspect, and modals from its "verbal" superclass but not singularity (i.e., singular or plural) and definiteness (i.e., indefinite "a" or definite "the") from its "nominal" superclass, which "removal" should inherit. The exclusive inheritance solves this situation.

In fact, these two superclasses (verbal and nominal) should be defined only once in a top-level *action class as follows since this exclusive inheritance applies to each of the entire subclasses of the *action.

```lisp
(defun *physical-action
  (is-a (value *predicate)))
(defun *mental-action
  (is-a (value *object)))
(defun *action
  (is-a (value *physical-action *mental-action)))
```

There are three "meta" closed classes in NL classes. They are *var, *set, and *fun. The *var is a variable that ranges over NL classes which are constants. The *var corresponds to pronouns and question words in natural languages. The *set is a set constructor which can represent a coordinated structure in natural language. The *fun is a function from NL classes to NL classes. A function corresponds to so-called a semi-function word in natural language. For example, in some usages of "take", it does not really mean any specific action until it gets an argument "a walk", "a rest", "a shot", etc. We realize such a highly abstract word sense in terms of a function. Figures 3 and 4 give definitions and sample instances of these classes. These meta classes are system-defined and have no user-defined slots.

As we restricted the type of inheritance to be exclusive, a designer of NL classes may suffer from inability to organize classes from various viewpoints which ordinary multiple inheritance could support. Virtual classes are thus introduced to bundle a set of classes. For example,

```lisp
(defunv *land-vehicle
  (def (*car *bicycle)))
(defunv *male-thing
  (def (equal :sex *male)))
```

define virtual classes *land-vehicle and *male-thing, respectively. The *land-vehicle consists of *car and *bicycle. The *male-thing is a class whose instance has a :sex slot filled with *male. Therefore, an instance of whatever class can also be an instance of the *male-thing as long as its :sex slot has a value *male. The virtual class is similar to a view in relational databases[4] and generalization objects in IF0 model[1]. Some object-oriented programming languages also support such a virtual class[10]. The virtual classes can be defined from actual classes and other...
virtual classes but recursive definition is not allowed. Although the virtual classes cannot have their own slots, they are powerful enough to define a set of classes and instances which would be scattered among NL class hierarchies.

Choices and granularity of classes significantly differ depending on whether the NL classes are mono-lingual or multi-lingual. In a mono-lingual situation, say English, we can just assign a class for each English word stem. It might not be necessary to define specific classes which represent different word senses of an English word when there is no English word corresponding to such primitive word senses. In multi-lingual situation, however, the classes need to be a union of the word senses conveyed by natural languages. A designer of such NL classes has to pay more attention to taxonomy and synonym relationships as they are no longer available from mono-lingual dictionaries and thesauruses.

3. Syntax-Semantics Mapping for Natural Language Objects

NL classes are implementation of lexical word senses of natural languages. In this sense, they realize semantic dictionaries for natural languages. Sentences or phrases are more complex than simple words and they follow specific encoding scheme which we call NL syntactic rules or a grammar. A lot of efforts have been made to describe a comprehensive NL grammar under various formalisms[11]. Even though the coverage of existing grammars is not perfect enough to derive every sentence of any given document, NL grammars have become so practical that they support several applications with a help from users[3, 9]. We assume the availability of such a grammar. In particular, we employ a family of unification grammar formalism[12, 13] to describe NL grammar in the later discussion of syntax-semantics mapping. As words correspond to instances of NL classes, sentences and phrases correspond to expressions over NL instances. A mapping rule is associated with each NL class as shown below. For simplicity, we only give mapping rules for English, but a set of mapping rules can be defined for each natural language.

(map *remove <=l=> remove ((cat v)(subcat trans))
;; a class *remove is associated with a transitive verb "remove"
(role =sem (*physical-action))
;; *physical-action is its exclusive superclass
(agent =syn (subj))
;; :agent filler corresponds to a syntactic subj
(theme =syn (obj))
;; :theme filler corresponds to a syntactic obj
(source =syn (ppadjunct ((prep from))))
;; :source filler corresponds to a prepositional
"from" phrase
(map *physical-action <=s=>
;; *physical-action has no association with a specific word
(:mood =syn (mood))
;; :mood filler corresponds to a syntactic mood
(:time =syn (tense))
;; :time filler corresponds to a syntactic tense

Figure 3: Three Meta NL Classes

Figure 4: Sample Instances of the Meta NL Classes
The first rule is called a lexical mapping rule because an instance of the class is associated with a syntactic word. The second rule is called a structural mapping rule because there is no corresponding syntactic word for the class and only the mapping definitions between syntactic and semantic slots are inherited by its subclasses. The first rule defines that an instance of "remove" is created to represent the meaning of the transitive verb "remove", its exclusive subclass is "physical-object", and each of its semantic slot fillers is associated with a specific syntactic filler of the verb. The \( \Rightarrow \) operator defines a lexical mapping rule, the \( \Rightarrow \) operator defines an assignment of a filler to a semantic slot of an instance, and \( \Rightarrow \) operator defines a mapping between a semantic slot filler and a syntactic slot filler, which is recursively associated with another instance of an NL class. In the second rule, the \( \Rightarrow \) operator defines a structural mapping and two semantic slots are given their syntactic counterparts. The syntactic structures, for which the mapping rules are defined, are called feature structures and are constructed by applying grammar rules to sentences. In a word, a feature structure is a constant, or a set of feature name and feature structure pairs. Typical grammar rules for our running example will look like

1. \( \langle V \rangle \leftarrow \rightarrow (remove) \)
   
   ;; A lexical rule for the transitive verb "remove"
   
   \((\{x_0 \text{ cat} = v\} ;; \text{syntactic category is VERB}) \)
   
   \((\{x_0 \text{ subcat} = \text{trans} \} ;; \text{valency is TRANSITIVE}) \)
   
   \((\{x_0 \text{ root} = \text{remove}\}) ;; \text{root is REMOVE}) \)

2. \( \langle N \rangle \leftarrow \rightarrow (diskette) \)
   
   ;; A lexical rule for a noun "diskette"
   
   \((\{x_0 \text{ cat} = n\} ;; \text{syntactic category is NOUN}) \)
   
   \((\{x_0 \text{ root} = \text{diskette}\}) ;; \text{root is DISKETTE}) \)

3. \( \langle \text{IMP} \rangle \leftarrow \rightarrow (\langle V \rangle \langle N \rangle) \)
   
   ;; Phrase structure rule for an IMPERATIVE
   
   ;; sentence
   
   \((\{x_0 = x_1\}) \)
   
   ;; feature structures of \( \langle \text{IMP} \rangle \)
   
   ;; and \( \langle V \rangle \) are identical
   
   \((\{x_0 \text{ obj} = x_2\}) \)
   
   ;; \( \langle \text{IMP} \rangle \)'s obj filler is identical to
   
   ;; \( \langle V \rangle \)'s feature structure
   
   \((\{x_0 \text{ mood root} = \text{imperative}\}) \)
   
   ;; \( \langle \text{IMP} \rangle \)'s mood root is IMPERATIVE
   
   \((\{x_0 \text{ tenser root} = \text{present}\}) \)
   
   ;; \( \langle \text{IMP} \rangle \)'s tenser root is PRESENT

In the above rules, \( x_0 \) denotes a feature structure of a left-hand-side syntactic constituent, \( x_i \) denote feature structures of an \( i \)-th right-hand-side syntactic constituent \((i=1,2,...)\). Feature structures are nicely illustrated by rooted directed acyclic graphs(DAGs) such that their nodes represent a constant or a root of a feature structure and their arcs represent feature names that appear in the grammar rules. The equation \( \Rightarrow \) above means a unification operation of feature structures in the left and right hand sides. The syntactic-semantic mapping rules are then applied to the feature structures. By assuming that we have similar lexical mapping rules for "diskette", "imperative", "present", we get the feature structures and NL objects shown in Figure 5. The root values "remove", "diskette", "imperative", "present", which are recursively associated with another instance of an NL class. In the second rule, the \( \Rightarrow \) operator defines a structural mapping and two semantic slots are given their syntactic counterparts. The syntactic structures, for which the mapping rules are defined, are called feature structures and are constructed by applying grammar rules to sentences. In a word, a feature structure is a constant, or a set of feature name and feature structure pairs. Typical grammar rules for our running example will look like

Figure 5: Feature Structures and NL objects for Syntactic Constituents

"imperative", and "present" in the feature structures are keys to be matched in the lexical mapping rules. Fragments of feature structures given in the lexical mapping rules, \((\{cat v\} (\text{subcat trans}) (\text{root remove}))\) for example, specify further restrictions on feature structures to be matched. As a result, we can associate a complex NL object "remove" with a sentence "Remove diskette" through the NL grammar and mapping rules. This association is bi-directional. That is, if a sentence \( s \) is parsed with the NL grammar and mapped to an NL object \( o \), then there is a generation procedure that maps \( o \) into \( s \). Details are discussed in [13] and [16]. Note that there may not exist a sentence that can be associated with an arbitrary complex NL object. There may also be several sentences which map to the same complex NL object. These sentences are said to be equivalent with respect to the meaning. Moreover, we can define equivalence relation and subsumption relation over complex NL objects, which allows us a way to paraphrase a given sentence semantically [14].

4. Application Interface

The NL class system together with the NL grammar and mapping rules realizes an architecture to store/handle NL sentences both syntactically and semantically. Such an architecture provides a useful interface to various NL applications.

A. Machine Translation

This is our on-going project built upon an NL class system. By developing a pair of NL grammar and a set of mapping rules for each natural language, we can translate a stored document (a set of complex NL objects) into sentences of a given language. This approach is similar to so-called a pivot approach but semantic paraphrasing, mentioned in the previous section, plays an important role in our approach because the sets of complex NL objects that can be mapped into sentences differ depending on the NL grammar and mapping rules. Semantic paraphrasing
thus handles subtle differences in natural languages that reflect in the meaning of sentences. Figure 6 shows an example of English to Japanese translation using the NL objects.

```
( Ident "remove diskette." )

**** English Feature Structure ****
(OBJ ((CASE ACC) (PERSON 3) (NUMBER SG))
 (ROOT DISKETTE) (COUNT +) (PRE-NOM +) )
(CAT N) (SEM DISKETTE-107) (PL-FORM $)
(NUM ((SEM *-SINGULAR-106) (ROOT $SINGULAR))))
(SUBCAT TRANS) (SEM REMOVE124)
(MOOD ((SEM *-IMPERATIVE-123) (ROOT $IMPERATIVE)))
(AUX ((SEM *-ASPECT-MODAL-121) (ROOT $AUX)))
(TENSE ((SEM *-PRESENT-120) (ROOT $PRESENT)))
(FORM INF) (MORPH-TYPE E) (ROOT REMOVE)
(CAT V) (A-AN A))

**** NL Objects ****
(REMOVE124
 (:DESCRIPTOR
 (*-ASPECT-MODAL-121 (:TENSE (*-PRESENT-120))))
 (:THEME (*DISKETTE-107 (:NUMBER (*-SINGULAR-106))))
 (:ROLE (*-PREDICATE-) (:MOOD (*-IMPERATIVE-123)))
)

**** Japanese Feature Structure ****
((ROOT TOINUZOKU) (CAT V) (SUBCAT TRANS)
 (AUX ((ROOT $AUX) :TMR (ROOT $FUTURE))))
 (:PASSIVE (ROOT $MINUS)))
(OBJ ((ROOT DEISUKETTO) (CAT N))
 (:MOOD (ROOT $IMP))))
```

Figure 6: NL Objects and Feature Structures for English to Japanese Translation

B. Text Retrieval and NL Query

Text retrieval is easily implemented by searching for specific NL objects that subsume an NL object representing a given NL query or a keyword. An advantage of this implementation is that a query and original text may be written in different natural languages. The material to be retrieved may be multimedia document, encyclopedia, dictionary, newspaper articles, etc. Processing of the answer (e.g., abstraction of text) can be defined in terms of NL object manipulation, not in terms of syntactic operations. Hence, this approach provides common methods to abstract meaning of text no matter what language it was originally written in.

C. Document Processing

Combining a set of methods for formatting a document, proofreading it, and sending and printing it with the NL class system, users can make use of existing document or store new materials in the database. This approach can include authoring tools for recent hypermedia databases. Some methods can be language independent while others, formatting and proofreading may require syntactic knowledge for each natural language.

5. Concluding Remarks

In this paper we discussed an object-oriented database for storing/ handling natural language documents. The architecture was motivated from a machine translation project which requires a large amount of knowledge sources. Without the meaning representation like NL objects, the translated material would just be thrown away and we have no way to organize dictionary, case-bases, and document in an integrated form. The NL class system allows us to develop and maintain a knowledge base and methods under a natural hierarchy of word senses. The translated material would be part of the NL object base and incremental nature of the system and case-base will contribute to accuracy of the translation. The NL object will be also utilized by other applications in the future. The remaining problems include automatic acquisition of word senses from existing dictionaries and implementation of equivalence and subsumption relationships among NL objects.

References