AUTOMATED NATURAL LANGUAGE EVALUATORS - (ANLE)

Khosrow Kaikhah
Assistant Professor
Department of Computer Science

Southwest Texas State University
601 University Dr.
San Marcos, Texas 78666

Final Report for:
Research Initiation Program
Rome Laboratory

Sponsored by:
Air Force Office of Scientific Research
Boling Air Force Base, Washington, D.C.

and

Southwest Texas State University

December 1993

AUTOMATED NATURAL LANGUAGE EVALUATORS - (ANLE)

Khosrow Kaikhah
Assistant Professor
Department of Computer Science
Southwest Texas State University

Abstract

By the turn of the century, it is expected that most computer applications will include a natural language processing component Both developers and consumers of NLP systems have expressed a genuine need for standard natural language system evaluators. Automated natural language evaluators appear to be the only logical solution to the overwhelming number of NLP systems that have been produced, are being produced, and will be produced.

The system developed here is based on the Benchmark Evaluation Tool [7] and is the first attempt to fully automate the evaluation process. This effort was accomplished in two phases. In phase one, we identified a subset of the Benchmark Evaluation Tool for each class of NLP systems. And in phase two, we designed and implemented a natural language generation system to generate non-causal semantically meaningful test sentences. The generation system can be queued for each class of NLP systems.

We followed an Object-Oriented Design (OOD) strategy. In this approach all concepts, including semantic and syntactic rules, are defined as objects. Each test sentence is generated as a chain of words satisfying a number of semantic, syntactic, pragmatic, and contextual constraints. The constraints imposed on the generation process increase dynamically while the sentence is being generated. This strategy guarantees semantic cohesiveness while maintaining syntactic integrity. In this approach, syntactic and semantic knowledge were utilized concurrently in word-objects. Each word-object is an independent knowledge source with local knowledge that can decide whether it can be a part of the sentence being generated, when called upon by the sentence-generator to join the chain.

Table of Contents

		Page #
	Abstract	2
1.0	Introduction	4
2.0	Discussion of the Problem	5
3.0	Methodology	7
3.1	Phase One	7
3.2	Phase Two	10
3.2.1	System Overview	10
3.2.2	Templates and Expansions	14
3.2.3	Lexicon	15
3.2.4	Constraints and CLOS	16
3.25	System Flow	19
3.2.6	Example 1	21
3.27	Example 2	28
4.0	Results	35
5.0	Conclusions	38
6.0	References	40

AUTOMATED NATURAL LANGUAGE EVALUATORS - (ANLE)

Khosrow Kaikhah

1.0 Introduction

The NLP community is rapidly growing, as a consequence, so is the number of NLP systems. However, evaluation of such systems has not received as much attention. In an effort to standardize the evaluation process, Calspan Corporation proposed and implemented the Benchmark Evaluation Tool for evaluating natural language processing systems [7]. The tool was designed to measure the linguistic capabilities of NLP systems regardless of the domain for which the NLP system was intended for.

The Benchmark Evaluation Tool was applied to PUNDIT during the summer of 1992 with the following conclusions [5]: A) The evaluation process must be customized for each class of natural language processing system, since NLP systems are designed for well-defined objectives and domains; and B) The evaluation process must be automated, since it is extremely long and time consuming. Generally, the NLP systems can be categorized into five classes: Database Management Systems, Command & Control Systems, Decision-Aiding Systems, Engineering Design Systems, and Diagnostic Systems. We attempted the automation process in two phases. In phase one, we identitied an applicable subset of the Benchmark Evaluation Tool for each of the five NLP system classes; and in phase two, we designed and implemented an object-oriented system for generating ***non-causal meaningful*** sentences. The system is implemented in CLOS ***(COMMON LISP Object System),*** an object-oriented extension of COMMON LISP.

This report will cover the object-oriented analysis, design, and implementation of the system. It will describe the objects that were used in generating a sentence and the way these objects interact to constrain the generation process. It highlights the major design and implementation issues and demonstrates the internal behavior of the system with two detailed examples. We conclude the report with sample sentences and a discussion of extension possibilities.

2.0 Discussion of the Problem

Both NLP developers and consumers have expressed a genuine need and desire for a standard evaluation procedure. NLP systems can be evaluated in several areas including: linguistic competence, end user issues such as reliability and likability, system development issues such as maintainability and portability, and intelligent behavior issues such as learning and cooperative dialogue. The underlying goal of the Benchmark Evaluation Tool was to create a product to test the linguistic capabilities of NLP systems, independent of the application under investigation. This feature is unique, in that, the tool is sensitive to each individual linguistic capability and not to each individual application.

The Benchmark Evaluation Tool [7] is composed of twelve independent sections which are designed to progressively test different linguistic features of an NLP system. The twelve sections are: 1) Basic Sentences, 2) Interrogative Sentences, 3) Noun Phrases, 4) Adverbials, 5) Verbs and Verb Phrases, 6) Quantifiers, 7) Comparatives, 8) Connectives, 9) Embedded Sentences, 10) Reference, 11) Ellipsis, 12) Semantics of Events. Each section is composed of a collection of evaluation procedures, each designed to test a single linguistic feature. These procedures include: an explanation of the feature being tested, a sentence template, example sentences, and criteria to use in evaluating the behavior of the NLP system.

After using the Benchmark Evaluation Tool to evaluate an NLP system, PUNDIT, we concluded that although the tool was extremely helpful in providing a guideline for testing the system, there were a number of clashes between the wide scope of the evaluation tool and the narrow application domain of the NLP system. Each class of NLP system possesses certain attributes that are unique. Each class has strengths and weaknesses which are directly associated with the goals and objectives of the system. Therefore, the evaluation procedure should place more emphasis on the class and objectives of the system and should be designed to test the applicable sentences to the system rather than the non-applicable sentences. For instance, if the NLP system is a Database Management System, the evaluation tool should place more emphasis on interrogative and basic sentences rather than ellipsis or quantifiers.

There are, naturally, sentences that are applicable to NLP systems and their domains. However, there are two completely distinct types of non-applicable sentences: A) non-applicable to the system, and B) non-applicable to the domain. Non-applicable to the system type sentences are those sentences where a meaningful sentence can not be formed with the suggested grammatical structure and available vocabulary. For instance, a Data Base Management system is not designed to handle a passive voice sentence. Non-applicable to the domain type sentences are those where a sentence can be constructed with the available vocabulary that can satisfy the suggested grammatical structure, but is not semantically meaningful. For instance, the sentence: 'Does the Atlanta to Atlanta flight have a stop in Atlanta?' should not be generated. Instead, the sentence: 'Does the Atlanta to Denver flight have a stop in Boston?' may be generated. The first phase of our project is designed to eliminate the non-applicable to the system sentences, while the second phase concentrates on the non-applicable to the domain sentences.

3.0 Methodology

A sentence at different levels is controlled by different forces. For instance, at the top level, a complete sentence is controlled by its main verb. The main verb determines the type, person, and in some cases the number of its subject and direct or indirect objects as well as their modifiers. Each phrase is also dominated by different forces. For instance, in a noun phrase, the noun determines the type and number of modifiers; in a prepositional phrase, the preposition is the driving force; and in a ,verb phrase, the verb is the dominating force. In our approach, a sentence is generated as a chain of words satisfying a number of semantic, syntactic, pragmatic, and contextual constraints concurrently. This chaining process ensures the compatibility of different components and results in a cohesive and unambiguous sentence.

Our goal was to design and develop a system that could generate semantically meaningful English sentences. The sentences were to be built according to a grammatical structure, or template, for a specified NLP system class, and were to be non-causal. In other words, there was no desired intention or content specified for the sentences. In our design we have utilized Object-Oriented Design (OOD) techniques [1] to develop a system where static objects of knowledge interact with dynamic elements of context in order to recursively build a sentence piece by piece. In this interactive mode, objects that have already been built are used to adjust the syntactic and semantic requirements of future sentence objects. We chose the object-oriented approach for the following two reasons: a) Knowledge can be divided into two general categories, static or functional. The object-oriented framework provides a convenient way to combine the two in the same data structure, an object; b) A large amount of data is used to acquire different types of knowledge. The uniform treatment of data and knowledge facilitates the maintenance of the system and promotes the reuse of system functionalities. This section describes, in detail, the design and implementation of our system.

3.1 Phase One

Although the idea of testing the sensitivity of individual linguistic capabilities of NLP systems rather than the sensitivity of the systems to individual applications is extremely attractive, it has nevertheless proved to be an ambitious task [5]. Most NLP systems are designed for well-defined domains and applications. Therefore, a general purpose evaluation tool may not be suitable for all classes of NLP systems. This was evident from our prior experience. In order for an automated evaluator to be successful, the evaluation should be performed within the scope of the NLP system. Therefore, there should be several different automated evaluators each specialized for a different class of NLP system. Each automated evaluator would have syntactic, semantic, and pragmatic knowledge of only one class of NLP system and would generate appropriate test sentences. In an effort to accomplish this goal, we attempted to identify a subset of the Benchmark Evaluation Tool for each class of NLP systems.

The first phase of our work involved building tables, similar to Table I, for each of the twelve sections outlined in the Benchmark Evaluation Tool [7]. We evaluated each syntactic feature, proposed for testing, for its applicability to each of the following NLP system classes: Database Management Systems, Command & Control Systems, Decision-Aiding Systems, Engineering Design Systems, and Diagnostic Systems. If the feature seemed applicable, it was marked with an "X", otherwise it was left blank. NA was entered in the columns, if the feature was not applicable to any class. For example, every template, except Passive Voice, was selected as applicable from the Basic Sentences section for Database Management Systems. From this effort we were able to build a profile of applicable sentences for each system class to use in the generation of sentences. These profiles[1] were used to limit the template choices available to the selected NLP system and allow access to only a subset of all the grammatical structures. Therefore, if the class of NLP system being evaluated is Command & Control, only those features applicable to its class will be examined.

[1] The complete set of profiles are available upon request.

	Database Management Systems	Command & Control Systems	Decision-Aiding Systems	Engineering Design Systems	Diagnostic Systems
I. Basic Sentences					
1. Basic Sentence Types					
1.1 Declarative Sentences	X	X	X	X	X
1.2 Imperative Sentences	X	X	X	X	X
1.3 Interrogative Sentences	X	X	X		X
2. Simple Determiners					
2.1 The Indefinite Article	X	X	X	X	X
2.2 The Definite Article	X	X	X	X	X
3. Simple Noun Phrases					
3.1 Count Nouns	X	X	X	X	X
3.2 Proper nouns	X	X	X	X	X
3.3 Mass Nouns	X	X	X	X	X
4. Simple Verb Phrases					
4.1 Copular Verb Phrases	X		X	X	
4.2 VP Involving the Verb Do					
4.2.1 DO used in Interr.	X	X	X		X
4.2.2 The Emphatic DO	X	X	X	X	X
4.3 Transitivity					
4.3.1 Simple Transitive VP	X	X	X	X	X
4.3.2 Simple Intransitive VP	X		X	X	
4.3.3 Simple Ditransitive VP	X	X	X	X	X
4.4 Voice					
4.4.1 Active Voice	X	X	X	X	X
4.4.2 Passive Voice	NA	NA	NA	NA	NA

Table I: Section I - Basic Sentences

3.2 Phase Two

Based on our previous experience, a human evaluator tests an NLP system by inputting a number of random sentences and observing the system's response [5]. These random sentences are not constructed to express a particular intention, but rather are constructed to be within the scope of the NLP system in terms of grammar and vocabulary. Hence they are non-causal. Our goal was to automate the generation process of these non-causal sentences. Therefore, we concentrated on generating semantically meaningful and syntactically correct non-causal sentences and ignored intention.

3.2.1 System Overview

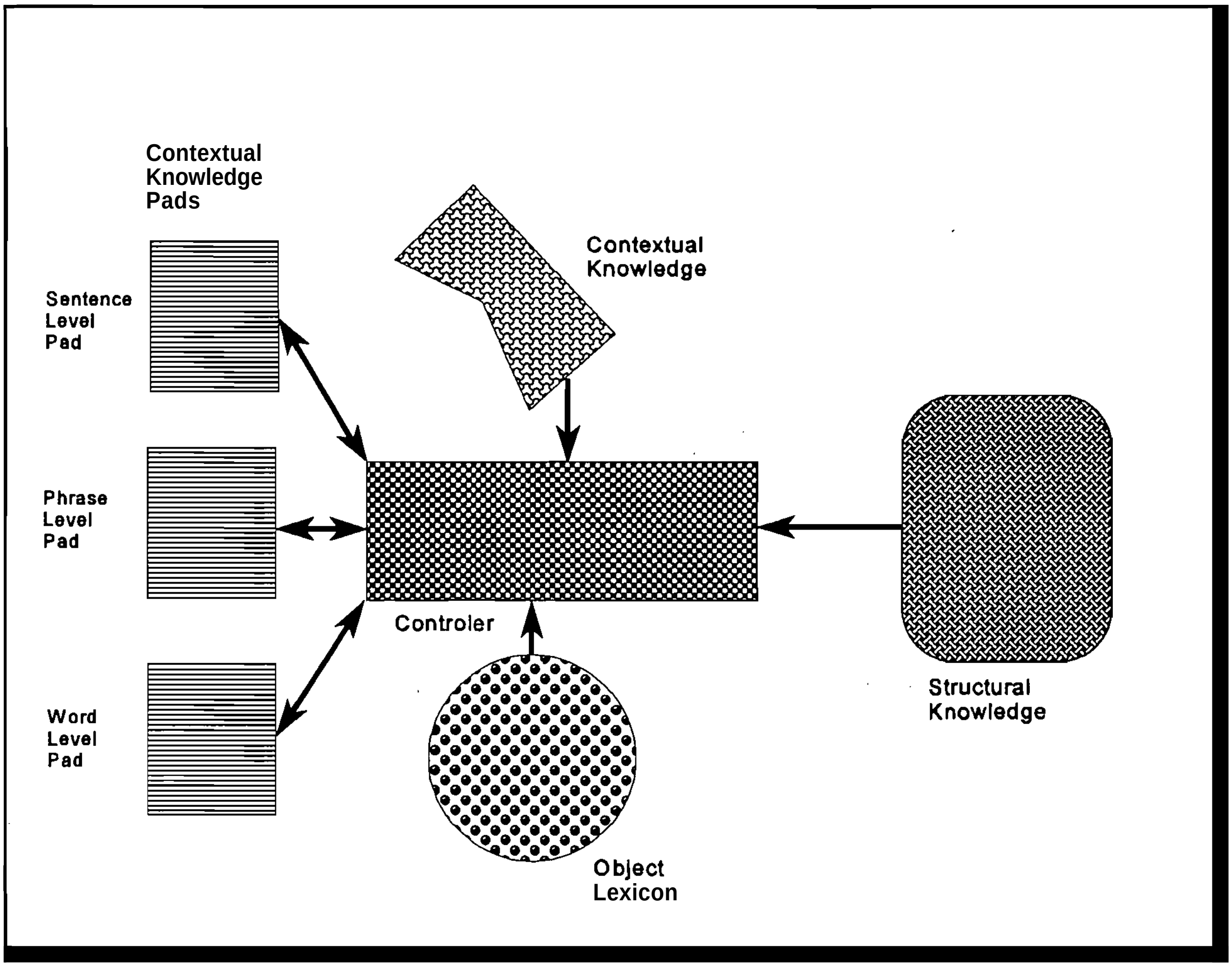


Figure I: **System Diagram**

Randomly generating a sentence from a template with a fixed vocabulary can yield many unusual and humorous combinations of words. Obviously, not all words can be combined to create a meaningful, sentence. The central idea of our system is that the decisions which are made to form one part of the template should limit the available choices for other parts of the template. We view the process of generating a sentence as a series of phrase generation sub-goals. Furthermore, each phrase generation is divided into word generation sub-goals. The generation is performed on a prioritized basis, so that the most important phrases within a sentence, and the most important parts within a phrase are completed fust. It is important to point out that most of the constraints used during the generation process are formed dynamically. Each phrase and each component of a phrase can be constructed in a number of ways. However, as parts of each phrase are being constructed, so are the constraints limiting the choices for the subsequent parts. The primary components (figure I) and the objects used to implement our ideas are described below.

System Components

Controller:

As its name implies this component is responsible for controlling the entire generation process. It creates and prioritizes new sentence, phrase, and word objects. Using the contextual knowledge and objects on the contextual knowledge pads the controller builds the constraints used in generation. Finally, it completes the process by forming the words, phrases, and sentences.

Obiect Lexicon:

This module contains the lexical objects and the knowledge about words that each object contains.

Structural Knowledge:

The structures used in building sentences and phrases are stored in the template and expansion objects in this module.

Contextual Knowledge:

This component is a collection of constraint generating rules.

Contextual Knowledge Pads:

The three areas that comprise the contextual knowledge pads are used to store objects that have already been created These dynamically created objects along with the contextual knowledge rules regulate the generation process of subsequent parts of the sentence.

System Objects

The system is implemented in an object-oriented environment [1]. The objects utilized in the system are described below:

Template Obiects:

The syntactic knowledge obtained from the Benchmark Evaluation Tool is stored in template objects. This knowledge is organized in a tree tuerarchy with branch nodes and leaf nodes. The leaf nodes contain the template string, an internal parsed representation of the template, selection flags, and sentence level constraints.

Expansion Obiects:

A template is a collection of grammatical patterns, each pattern (such as [NP]) may have multiple ways that can be satisfied [2,10]. The expansion of the patterns found in the templates are stored in expansion objects. These object are also organized in a tree tuerarchy. The leaf nodes for these objects are specific parts of speech, for example count nouns or transitive verbs. The branch nodes, from the bottom of the tree up, have increasingly general patterns.

Lexical Obiects:

These objects are the words that make up the system's vocabulary. They store syntactic, semantic, and morphological information. The syntactic elements include parts of speech, and any syntactic limitations associated with a word The morphological information is used to form the word when it deviates fiom the normal rules of English orthography [3]. The semantic information is used to enforce semantic agreement, it contains concepts or qualities that the word possesses or requires.

Constraint Obiects:

Constraint objects are built for every phrase and for every word. They contain the information to be used to limit the generation. These objects have slots for the roof type, number, person, tense, form, case, gender, concept, quality, and quantifier [2,10].

Sentence Obiects:

The sentence objects contain the instructions for building the sentence and the final version of the sentence. Sentence object sub-classes include: declarative, imperative, interrogative, and clauses.

Phrase Obiects:

The phrase objects are the most complex objects in the system. Most phrase objects inherit fiom semantic, syntactic, and phrase-type super-classes [6]. For instance, the phrase object, agn-sub-np, is a combination of agent (semantic role), subject (syntactic role), and noun phrase (phrase type). The slots found in any one phrase object will vary, however, any information that could be needed later in the generation process is stored in a phrase object [4].

Word Obiects:

The word objects store the expansion pattern for the word, the lexical object selected, and the final version of the word. Each part of speech recognized in the system has a word object sub-class.

3.2.2 Templates and Expansions

The template and expansion objects store the structure upon which a sentence is constructed. They serve as a blueprint and a scaffold for the sentence. The template objects contain the instructions for building a sentence from phrases, while the expansion objects contain the instructions for building a phrase from words. The knowledge in both template and expansion objects is distributed in a tree hierarchy with more specific knowledge contained at the roots. This organization allows for a variety of general requests, while still maintaining the ability for specific requests .

The template objects consist of a textual template, the sentence type, selection flags for the NLP system classes, the template parse, and any sentence level constraints.

The template parse, which is stored in the template leaf-node object, is a generalized representation of the phrases within a template. In this representation, a sentence is viewed as a collection of phrases. Each phrase has semantic knowledge as well as the expansion objects which are used in generation process. The semantic knowledge describes the role each phrase plays within the sentence and is used to avoid structural ambiguity and to prioritize the phrase generation within a sentence. The possible semantic roles are ***agent*** (subject), ***action*** (verb), ***patient*** (direct-object), ***recipient*** (indirect-object), ***attribute*** (adjectival), ***manner*** (adverbial), ***auxiliary*** (auxiliary verb) and ***literal.*** For the modifiers (attribute and manner) additional information about what they modify and the type of phrase used for the modification is required.

The system provides a template to match various types of selection requests. If the NLP class is specified, the system will use the selection flags to prune the tree so that only those templates appropriate to the selected class will be available. A request can be made for any node on the tree. If the requested node is a leaf node the template information will be returned. However, if the requested node is a branch node, one of its children will be selected at random. This process will continue recursively until the selected child is a leaf node.

The expansion objects contain the grammatical patterns used to build phrases. The system fills an expansion request by returning a flat list of all possible expansions of the input. Each element of this list is a list of expansion leaf-node objects. The expansion of a branch-node proceeds by expanding each of its children nodes and then combines them for and branches or joining them for or branches [9]. The expansion process continues recursively until all branch-nodes are converted into lists of leaf-nodes.

3.2.3 Lexicon

The lexicon contains a collection of lexical objects which store syntactic, semantic, and morphological information. Furthermore, the objects in the lexicon are divided into subcategories based on their part of speech. The object categories include: noun, verb, adjective, adverb, preposition, pronoun, article, and conjunction.

The syntactic information includes the part of speech, the subtype of the word (such as count, mass, or proper for nouns), and any syntactic restrictions. For example, proper nouns are restricted to having a single number. The morphological information contains exceptions to the normal formation rules of the English language [3]. This includes tense formation for verbs, plural formation for nouns, comparative and superlative formation for adverbs and adjectives, and case and gender formation for pronouns. For example, the object for the verb "give" has a past tense form "gave" stored in it.

The lexical objects also store the semantic information that is used to enforce semantic agreement. This information varies for different parts of speech and is as follows:

Nouns & Pronouns:

The nouns and pronouns have two slots for concepts and qualities [8]. The concept slot contains a list of broad categories in which the noun or pronoun can be classified. The quality slot contains a list of attributes on which the noun or pronoun can be modified.

Verbs:

Verbs have the following additional slots: agent, patient, recipient, and quantifiers [8]. The agent, patient, and recipient slots contain semantic concepts that must be present in the candidate nouns. The quantdier slot contains a list of attributes on which the verb can be moddied.

Adjectives & Adverbs:

Adjectives and adverbs each have one additional slot that is used to ensure agreement with the object or action they mod@. The Adjectives have a qualifier-list slot which contains a list of attributes they can modify. Likewise, the adverbs have a quantifier-list slot which contains a list of attributes they can modify [8].

Prepositions:

Prepositions have one additional slot, concept-list, which contains concepts that must be present in the head noun of the preposition phrase [8]. Agreement between prepositions and the nouns or verbs they mod@ will be enforced through the preposition's subtype and the noun's quality slot or the verb's quantifier slot.

3.2.4 Constraints & CLOS

The constraints built during the formation of phrases and words are vital to the functionality of this system, for without them the system would be merely an over elaborate method of generating random sentences. This section describes the type of constraints and the method of their implementation.

Constraints, both static and dynamic arise from almost all components of the system. Static constraints come from template objects (in the form of sentence level restrictions), from the expansion objects (in the way they request specific words, word subtypes, a specific tense, or form), and from lexical objects (through syntactic restrictions). Constraints are also added dynamically during the interaction of sentence, phrase, and word objects according to the rules in the contextual knowledge component.

Before discussing the implementation of these dynamic constraints, we need to explain the behavior of CLOS (COMMON LISP Object System), an object-oriented extension of COMMON LISP. CLOS supports the object-oriented concept of polymorphism through a mechanism of methods. Generic functions are functions whose behavior can vary based on the objects they receive in their parameter list [6,11]. They have a single Primary-method which gets executed regardless of parameter values, and optional Before-methods and After-methods which are executed conditionally before and after the primary method when the parameter objects, or any superclasses of the objects, are of the type specified. For instance, the generate-word function in our system receives as parameters a phrase object, a word object, a word constraint object, and the phrase level knowledge pad This function has several Before-methods which execute depending on the type of objects received in the first two parameters.

As an example, lets look at the situation where the generate-word function is called with a phrase object of agn-sub-np and a word object of noun. This parameter combination triggers a Before-method that is looking for a noun-phrase object and a noun word object. Therefore, before the Primary-method can be executed, the Before-method must be executed. Notice that (Figure II) the phrase object, agn-sub-np, inherited the noun-phrase type fiom one of the superclasses fiom which it is composed Because of this inheritance some parameter combinations may cause multiple Before- or After-methods to be executed.

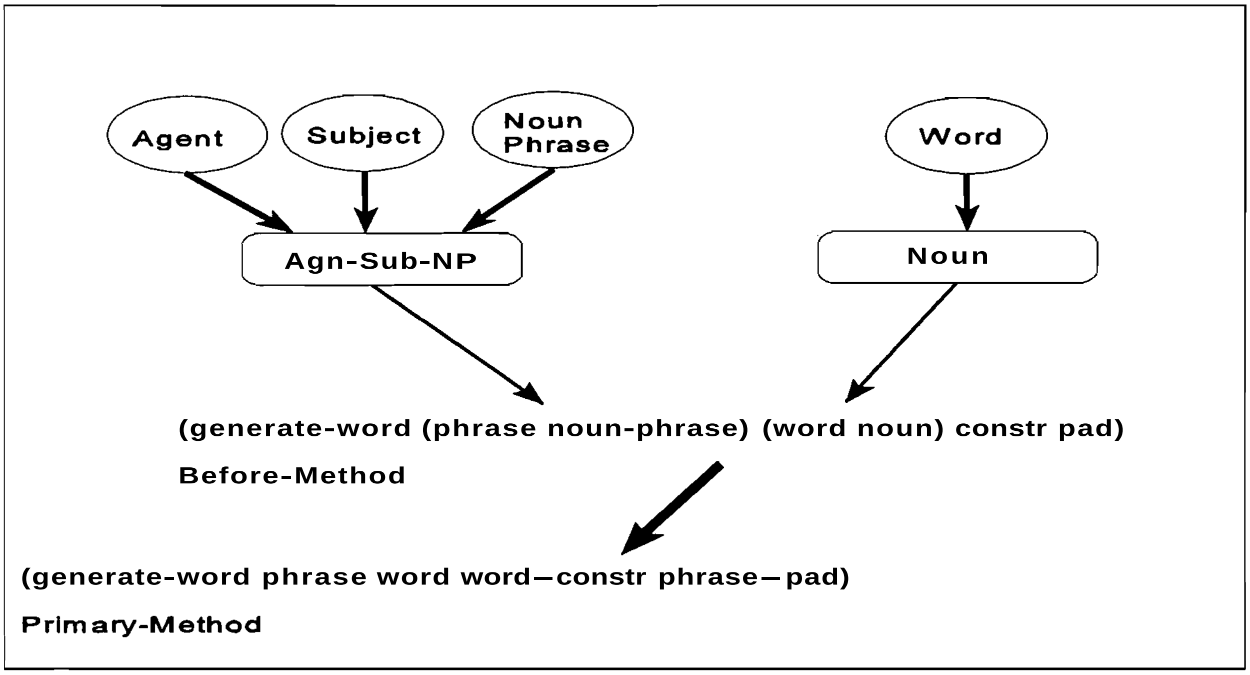

Agent
Subject
Noun Phrase
Word
Agn-Sub-NP
Noun
(generate-word (phrase noun-phrase) (word noun) constr pad)
Before-Method
(generate-word phrase word word—constr phrase—pad)
Primary-Method

Figure II - Before-Method for Agn-Sub-NP

The two main functions of our system, generate-phrase and generate-word, are both implemented as generic functions. It is through Before-methods and After-methods that the dynamic constraints are built. When these functions are first called the constraint objects in their parameter lists are empty. Before-methods examine other objects in the environment and build the constraints according to the needs of the objects that are being generated. For example, before a agn-sub-np phrase object is created the number, person, and concept slots of the phrase constraint are copied from the act-pred-vp object that had been previously created. Before-methods are also used to copy the static constraints to the constraint objects. After-methods are used to store information for future reference. For example, after the main verb has been selected the number, person, tense, agent, patient, and recipient concepts are stored in the phrase object for future use in constraint building.

3.2.5 System Flow

This section describes the function calls made by the controller in generating a sentence. The next section provides two detailed example to further explain the process.

Sentence Level

After a template has been selected the appropriate sentence type subclass is built using the textual template and the template parse from the template object This sentence is input to the function generate-sentence which returns the object with the sentence string slot filled in. The textual template and the sentence string are displayed as the system's output.

The generate-sentence function transforms the template parse into a list of phrase objects. These phrase objects are then sorted into a list of goals. Each subgoal phrase is formed by the function generate-phrase. After all the phrases have been generated, the form-sentence function builds the sentence string. The completed sentence object is then returned.

The order of the phrases is based on their semantic roles and is as follows: action, agent, patient, recipient, attribute, manner, auxiliary, literal.

Phrase Level

The build-phrasefunction takes the phrase information in the parsed template and builds the appropriate phrase object. This information includes the semantic role and an expansion object. For attribute or manner semantic object, it also includes the object being modified and the type of modifying phrase. During the construction of this phrase object, the expansion object is transformed into the list of possible expansions.

The **generate-phrase** function selects one of possible expansions randomly and **transforms** it into a list of word objects. These word objects are ultimately sorted into a list of goals, based on the type of phrase. Each word in each goal is formed through the function generate-word. After all the words have been generated, the **form-phrase** function builds the phrase **string**. The completed phrase object is then returned.

Word Level

Build-words creates a list of word objects by simply copying each expansion leaf-node object in the phrase's expansion list to a newly created word object.

The generate-word function receives the current phrase object, the current word object, a constraint object, and the phrase-level knowledge pad as input. Before any code is executed in the primary-method the multiple Before-methods are evoked to build the complete constraint object. Included in this process is the selection of those lexical entries which match the part of speech for the target word. Next, all the constraints are applied in order to select only those words that can pass the filter conditions from the selected lexical list. One of the candidates is then chosen at random, the word string is formed and the modified word object is returned.

The following rules **illustrate** how a regular verb is formed [2].

Present tense: 3rd Pars. Sing : root + "s" otherwise root

Past tense: root + "ed"

Future: **"will "** + root

Present Progressive: (appropriate present tense of "be") + " " + root + **"ing"**

Past Progressive: (appropriate past tense of "be") + " " + root + "ing"

Any verb which deviates from the above rules, all irregular verbs expect for the verb "to be", **will** have an entry in the morphology slot that will explicitly form the verb.

The verb "to be" is formed as follows [2]:

Present tense: 1st Pars. Sing. = "am" 3rd Pars. Sing. = "is" otherwise "are"

Past tense: 1st or 3rd Pars. Sing. = "was" otherwise "were"

Future tense: "will be"

3.2.6 Example 1

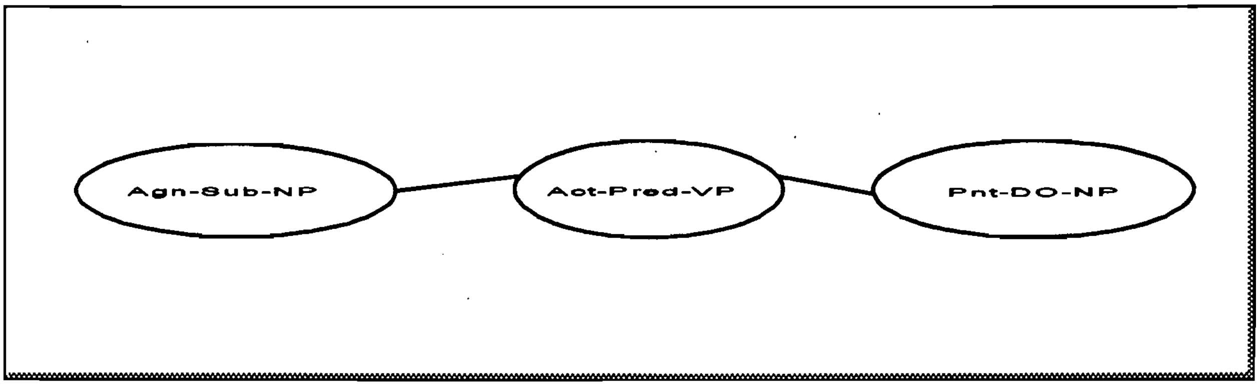
To see how a sentence is generated we will walk through the generation processes for the following sentence: "The supervisor hired a new employee". The template for this sentence is: [NP] [Verb] [Det] [Adj] [Noun]. Create sentence object from knowledge stored in template object. Generate a sentence from this object.

Generate Sentence:

Build phrase objects from the parse stored in the sentence object. For our sample sentence three objects are created: agn-sub-np (agent-subject-nounqhrase), act-pred-vp (action-predicate-verbqhrase), pnt-do-np (patient-direct–object-nounqhrase). Each object includes all expansion possibilities for that object.

Order phrases by priority based on their semantic roles. The ordered phrase list is act-pred-vp, agn-sub-np, pnt-do-np. Generate each phrase.

Form the sentence by combining the phrases.



Generate Act-Pred-VP:

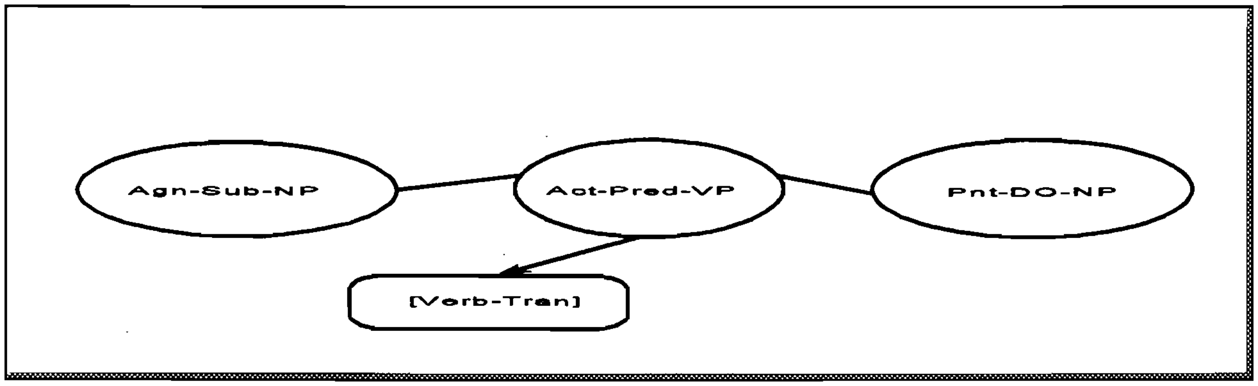
Before: No action, however, sentence-level constraints would be copied to the phrase constraint here.

Primary: Select one of the possible expansions at random. For this template [verb-tran] is the only choice.

Build word **object(s)** for each expansion element.

Order words by priority based on part of speech. Generate each word.

After: No action taken.



Generate Verb-Tran:

Before: Build word constraint object:

Word List = Verbs

Type = Transitive

Primary: Apply constraints to reduce list of verbs.

Choose word randomly. "hire" is chosen.

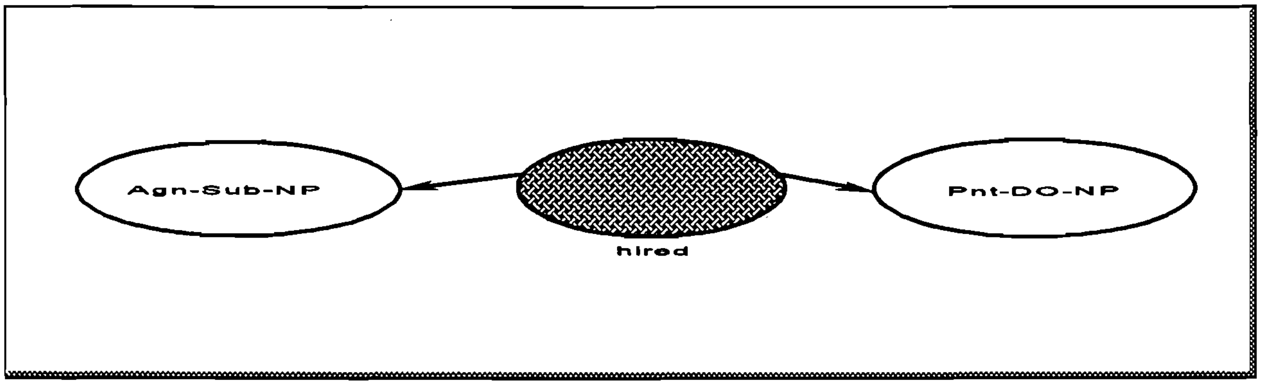
Form word and store in word object

Choose number, person and tense at random. Singular, **third,** past are chosen.

Use standard formation rules or lexical object morphology to build correct form of the word The string "hired" is built.

After: Copy predicate, number, person, tense, agent-concepts, patient-concepts and quantifier-list to act-pred-vp object.

The phrase "hued" is formed.



Generate Agn-Sub-NP:

Before: Build Phrase Constraint:

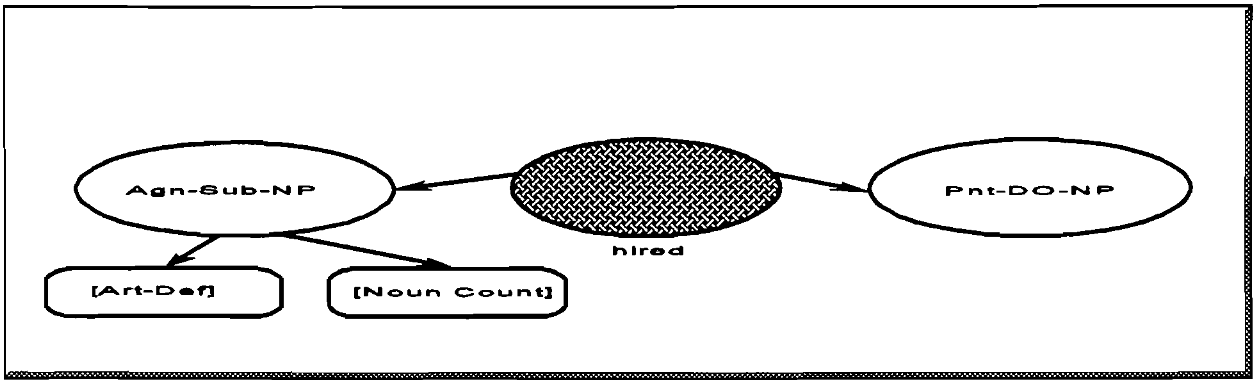
Copy number, person, and agent-concepts **from** the act-pred-vp object.

Primary: Select one of the possible expansions at random. **[art-def]** [noun-count] is chosen.

Build word **object(s)** for each expansion element.

Order words by priority based on part of speech. Generate each word.

After: No action taken.



Generate Noun-Count:

Before: Build word constraint object:

Word List = Nouns

Type = Count

Number = Singular

Person = Third

Concept-List = (Individual Management Personal)

Primary: Apply constraints to reduce list of nouns.

Choose word randomly. "supervisor" is chosen.

Form word and store in word object

Use standard formation rules or lexical object morphology to build correct form of the word. The string "supervisor" is built.

After: Copy subject, number, person, agent-concepts and quality-list to agn-sub-np object.

Generate Art-Def:

Before: Build word constraint object:

Word List = Articles

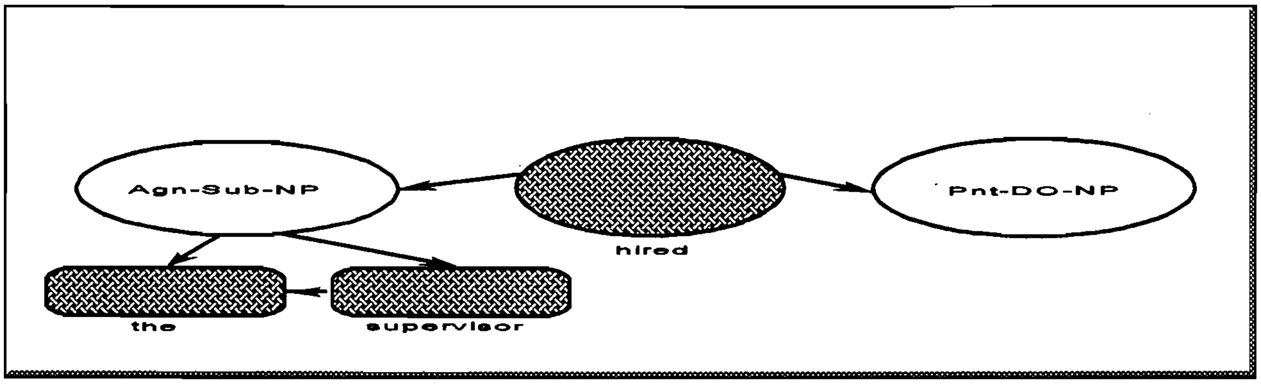
Type = Definite

Primary: Apply constraints to reduce list of articles.

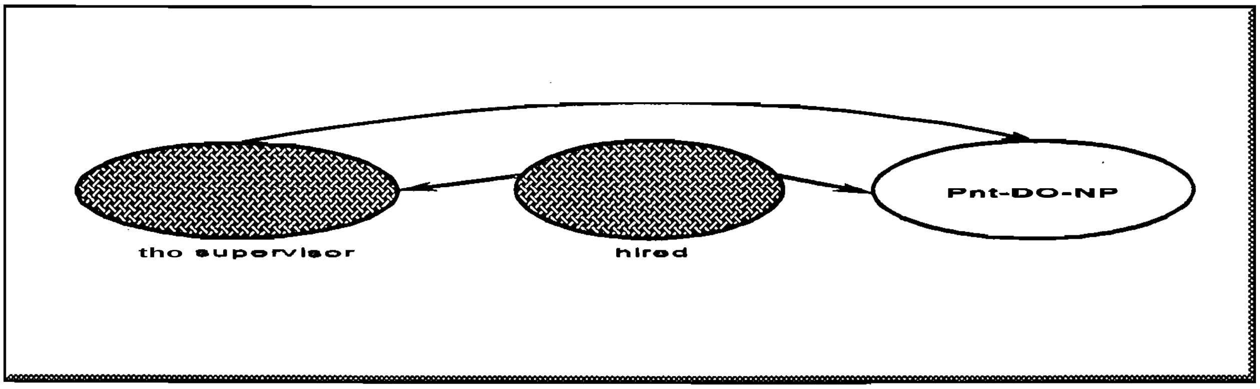
Only the word "the" meets the constraints.

Store "the" in word object

After: No action taken.



The phrase "the supervisor" is formed.



Generate Pnt-DO-NP:

Before: Build Phrase Constraint:

Copy patient-concepts from the act-pred-vp object.

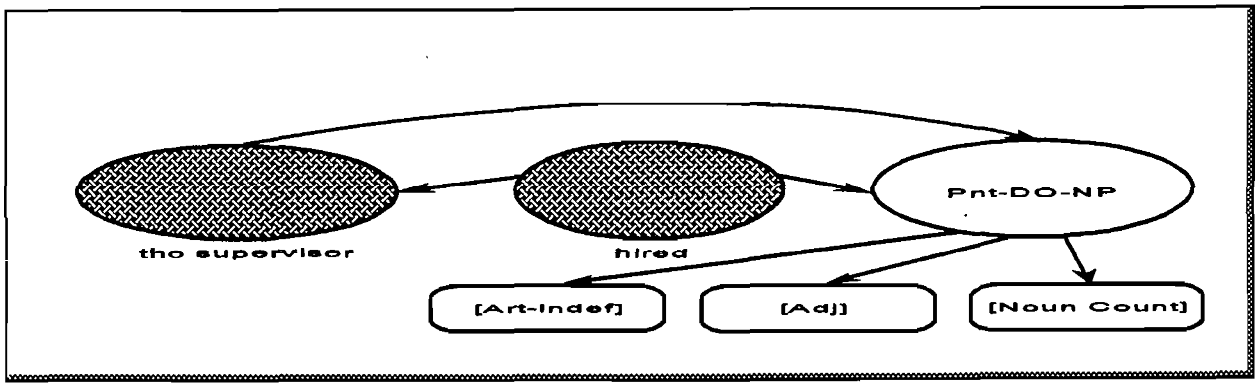
Copy subject (so subject and object aren't the same) from the **agn-sub-np** object.

Primary: Select one of the possible expansions at random. **[art-indef]** [adj] [noun-count] is chosen.

Build word **object(s)** for each expansion element.

Order words by priority based on part of speech. Generate each word.

After: No action taken.



Generate Noun-Count:

Before: Build word constraint object:

Word List = Nouns

Type = Count

Concept-List = (Individual Labor Personal)

Root <> "supervisor"

Primary: Apply constraints to reduce list of nouns.

Choose word randomly. "employee" is chosen.

Form word and store in word object

Choose number, person at random. Singular, third are chosen.

Use standard formation rules or lexical object morphology to build correct form of the word. The string "employee" is built.

After: Copy direct object, number, person, patient-concepts and quality-list to agn-sub-np object.

Generate Adj:

Before: Build word constraint object:

Word List = Adjectives

Type = Qual

Quality-List = (Age Intellect Size Marital-Status Sex Work-Hist)

Primary: Apply constraints to reduce list of adjectives.

Choose word randomly. "new" is chosen.

Store "new" in word object.

After: No Action taken.

Generate Art-Indef:

Before: Build word constraint object:

Word List = Articles

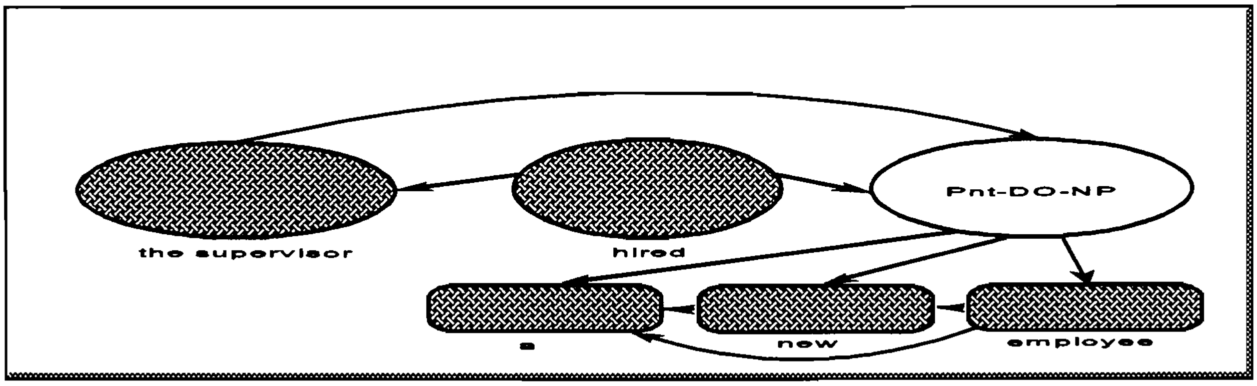
Type = Indefinite

Primary: Apply constraints to reduce list of articles.

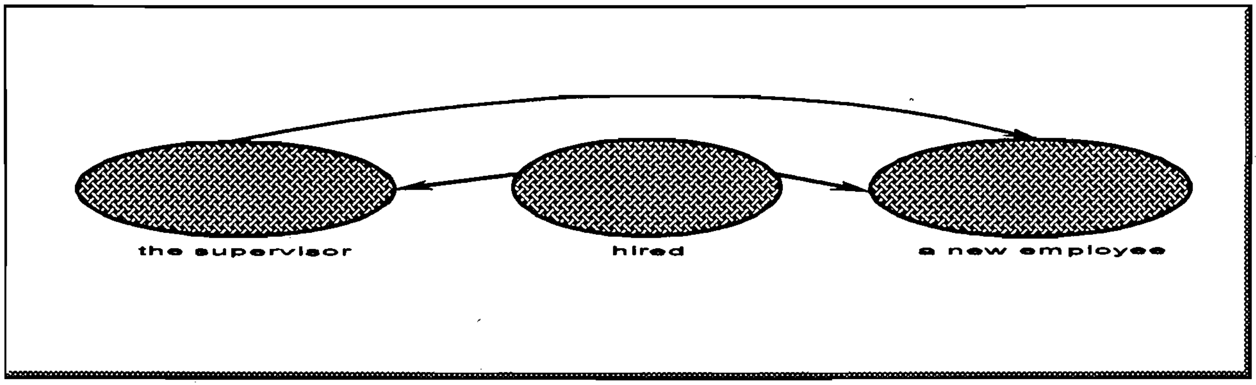
Only the word "a" meets the constraints.

Store "a" in word object

After: No action taken



The phrase "a new employee" is formed.



Finally, the complete sentence "The supervisor hired a new employee." is generated.

3.2.7 Example 2

Let's now look at the generation of the following sentence: "Who did the supervisor hire?". The template for this sentence is Who **[DO-Verb] [NP] [VP]**.

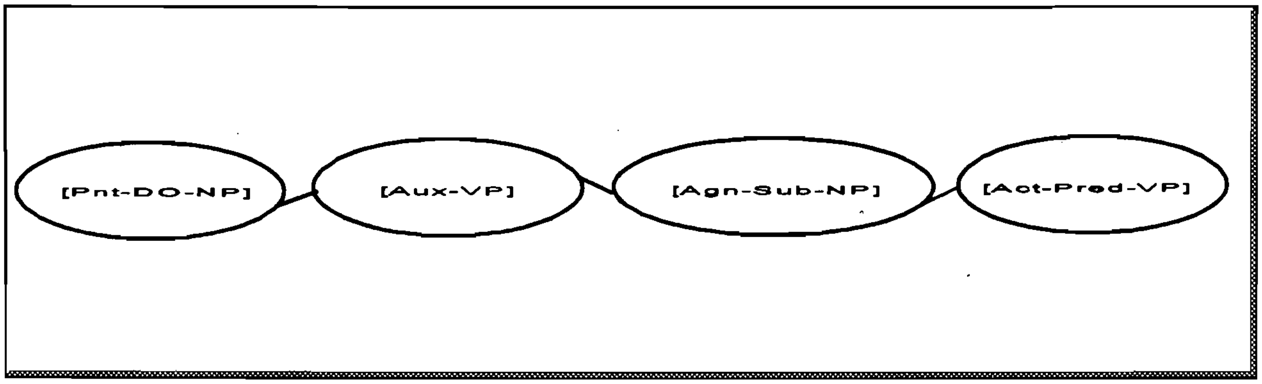
Create sentence object from information stored in template object. Generate a sentence from this object.

Generate Sentence:

Build phrase objects from the parse stored in the sentence object.. For this sentence four objects are created: pnt-do-np, aux-vp, agn-sub-np, and act-pred-vp.

Order phrases by priority based on semantic role. The ordered phrase list is act-pred-vp, agn-sub-np, pnt-do-np, aux-vp. Generate each phrase.

Form the sentence by combining the phrases.



Generate Act-Pred-VP:

Before:	No action taken.
Primary:	Select one of the possible expansions at random. For this template [verb-root] is the only choice.
	Build word object(s) for each expansion element.
	Order words by priority based on part of speech. Generate each word.
After:	No action taken.

Generate Verb-Root:

Before: Build word constraint object:

Word List = Verbs

Type = Transitive

Tense = Root

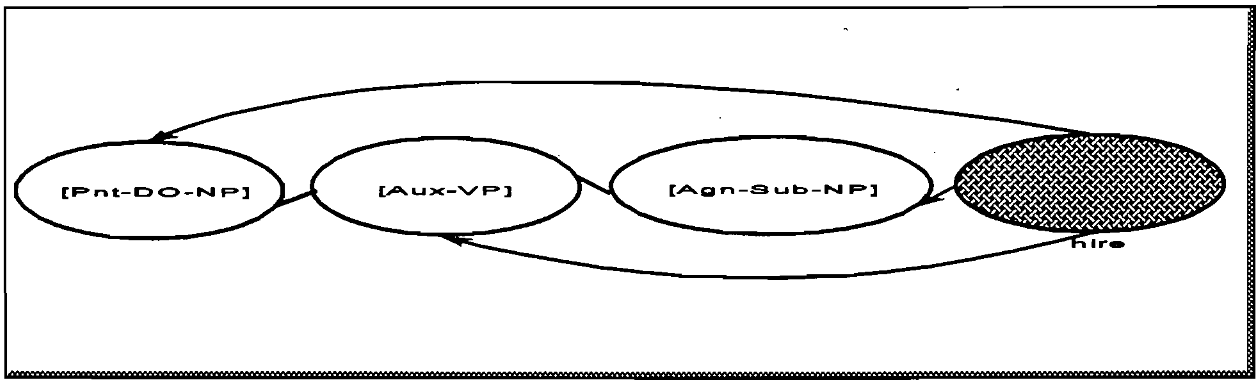
Primary: Apply constraints to reduce list of verbs.

Choose word randomly. "hire" is chosen.

Store root form "hire" in word object.

After: Copy predicate, agent-concepts, patient-concepts and quantifier-list to act-pred-vp object.

The phrase "hue" is formed.



Generate Agn-Sub-NP:

Before: Build Phrase Constraint:

Copy agent-concepts from the act-pred-vp object.

Primary: Select one of the possible expansions at random. [art-def] [noun-count] is chosen.

Build word object(s) for each expansion element

Order words by priority based on part of speech. Generate each word.

After: No action taken.

Generate Noun-Count:

Before: Build word constraint object:

Word List = Nouns

Type = Count

Concept-List = (Individual Management Personal)

Primary: Apply constraints to reduce list of nouns.

Choose word randomly. "supervisor" is chosen.

Form word and store in word object

Choose the number and person at random. Singular, thud is chosen.

Use standard formation rules or lexical object morphology to build correct form of the word. The string "supervisor" is built

After: Copy subject, number, person, agent-concepts and quality-list to agn-sub-np object.

Generate Art-Def:

Before: Build word constraint object:

Word List = Articles

Type =Definite

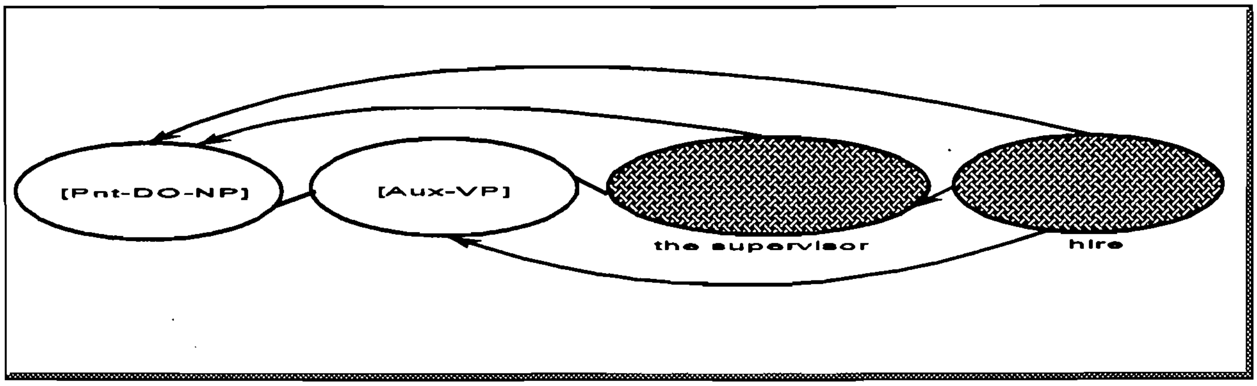
Primary: Apply constraints to reduce list of articles.

Only the word "the" meets the constraints.

Store "the" in word object

After: No action taken.

The phrase "the supervisor" is formed.



Generate Pnt-DO-NP:

Before: Build Phrase Constraint:

Copy patient-concepts from the act-pred-vp object.

Copy subject from the agn-sub-np object.

Primary: Select one of the possible expansions at random. For this template [who-pro] is the only choice.

Build word object(s) for each expansion element.

Order words by priority based on part of speech. Generate each word.

Form the phrase by combining the generated words. Phrase is "who".

After: No action taken.

Generate Who-Pro:

Before: Build word constraint object:

Word List = Pronouns

Root = "who"

Concept-List = (Individual Labor Personal)

Root <> "supervisor"

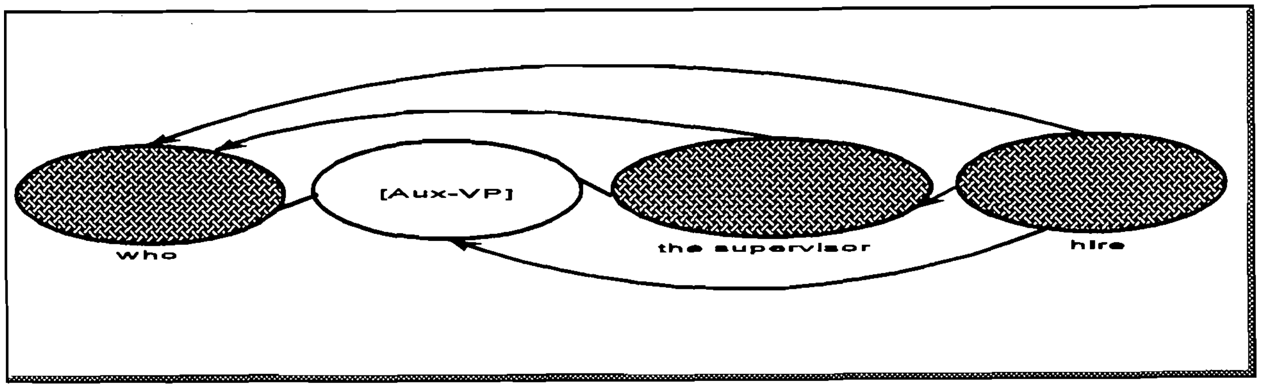
Primary: Apply constraints to reduce List of pronouns.

Only the word "who" meets the constraints.

Store "who" in word object

After: No action taken.

The phrase "who" is formed.



Generate Aux-VP:

Before: No action taken.

Primary: Select one of the possible expansions at random. For this template [DO-Verb] is the only choice.

Build word object(s) for each expansion element.

Order words by priority based on part of speech. Generate each word.

Form the phrase by combining the generated words. Phrase is "did".

After: No action taken.

Generate DO-Verb:

Before: Build word constraint object:

Word List = Verbs

Root = **"do"**

Primary: Apply constraints to reduce list of verbs.

Only the word "do" meets the constraints.

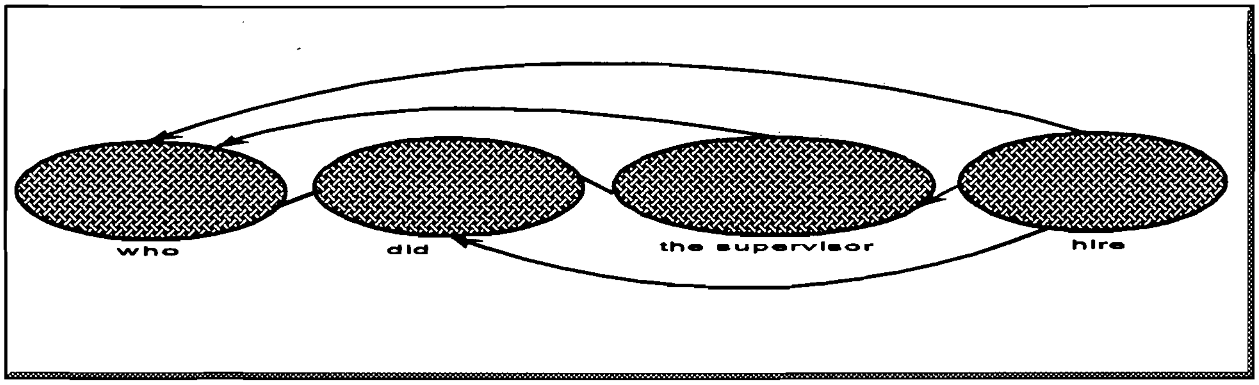
Form word and store in word object.

Choose tense at random. Past is chosen.

Use standard formation rules or lexical object morphology to build correct form of the word. The string "did" is built.

After: No action taken.

The phrase "did" is formed.



Finally, the complete sentence "Who did the supervisor hire?" is generated.

4.0 Results

The system can operate on one of five modes. Each mode represents a class of NLP system and is an input to the system. This input is utilized to limit the template choices at the top level and does not alter the functionality of the system in any way. The sentences are generated to satisfy the semantic, pragmatic, and contextual constraints. The semantic and pragmatic constraints are stored as static objects in contextual knowledge base as well as in object lexicon. The contextual constraints are also stored as objects in contextual knowledge pad but are generated dynamically during the generation of each sentences. These constraints are only kept temporarily during each sentence generation and are erased from the contextual knowledge pad once the sentence is completed.

This generation system is a tool to assist the human evaluator with the evaluation process. The human evaluator indicates the class of NLP system being evaluated and the system generates semantically meaningful applicable test sentences. The human evaluator can then score the NLP system's response. Our objective has been to increase the human evaluator's productivity by automating the test sentence generation phase of evaluation process. This way, the evaluator can spend more time analyzing and scoring the NLP system's response rather than forming sentences that are applicable to the system.

Due to our highly modular design which is based on an object-oriented framework, adding additional functionalities to the existing system is extremely simple. With relatively a few number of words, approximately 100, we were able to demonstrate the power of our system. Expanding the object lexicon, by adding new words, will dramatically enhance the strength and flexibility of the system. The system operates in real time. The average generation time for each test sentence, after all superclass objects have been instantiated, is approximately 1.3 seconds. The following is a sample set of test sentences along with their syntactic and semantic templates the system generated.

[PP] [NP] [VP]

(Mnr-Mod-PP) (Agn-Sub-NP) (Act-Pred-VP)

On Monday Mary Smith produced the computers.

[WH-Word] ([NP]) [Verb] [NP] [PP]

(Agn-Sub-NP) (Act-Pred-VP) (Pnt-DO-NP) **(Mnr-Mod-PP)**

Who sells the books to Texas?

[NP] [Verb] **([Det])** [Count Noun]

(Agn-Sub-NP) (Act-Pred-VP) **(Pnt-DO-NP)**

The directors hired a salesperson.

[Verb] **([Det])** [Count Noun]

(Act-Pred-VP) **(Pnt-DO-NP)**

Hire the salesperson.

[NP] [Verb] [NP] [PP]

(Agn-Sub-NP) (Act-Pred-VP) (Pnt-DO-NP) **(Mnr-Mod-PP)**

The salesperson sells the book on Monday.

[NP] [BE-Verb] [Adj]

(Agn-Sub-Np) (Act-Pred-VP) **(Atr-Mod-Adjp)**

Jane Doe is large.

Who [Verb] [NP]

(Agn-Sub-NP) (Act-Pred-VP) (Pnt-DO-NP)

Who will hire the engineers?

List [NP] **[Rel** Pronoun] [Verb] [NP]

(Act-Pred-VP) (Pnt-DO-NP) (Atr-Mod-RC)

List the presidents who hue the engineers.

[NP] [Verb] anyone **[PP]**

(Agn-Sub-NP) (Act-Pred-VP) **(Pnt-DO-NP) (Atr-Mod-PP)**

John Doe will **fire** anyone in Austin.

List **[NP] [Rel** Pronoun] [Verb **+pres-prog] [NP]**

(Act-Pred-VP) (Pnt-DO-NP) (Atr-Mod-RC)

List the directors who are hiring the engineers.

[DO-Verb] [NP] [VP]

(Aux-VP) (Agn-Sub-NP) (Act-Pred-VP) (Pnt-DO-NP)

Did the directors fire Mary Smith?

[Det] [Adj] [Noun] [VP]

(Agn-Sub-NP) (Act-Pred-VP) **(Pnt-DO-NP)**

The smart manager hires an engineer.

5.0 Conclusions

Evaluating an NLP system is a complicated task There are several areas in which NLP systems can be evaluated. They include: a) linguistic competence, b) end user issues such as reliability and likability, c) system development issues such as maintainability and portability, and d) intelligent behavior issues such as learning and cooperative dialogue. The Benchmark Evaluation Tool was very thorough in its coverage of the English language, but we feel that its expectations for linguistic competence and for domain independence are not attainable given the current state of technology. The task in the first phase of our project was to compile a collection of sentences from the Benchmark Evaluation Tool that were appropriate to each of five NLP systems classes. Based on our previous experience with the Benchmark Evaluation Tool limiting the scope of the evaluation appeared to be the most realistic and constructive approach.

Therefore, in the first phase of this project, we attempted to define boundaries for each of the five classes of NLP systems. In the second phase of our project we attempted to automate the sentence generation process. We designed and implemented an object-oriented framework for generating non-causal sentences. In this system, syntactic and semantic knowledge were embedded concurrently in our design. Each word-object, for instance, has local static knowledge which enables it to make decision. Hence, each word is an independent knowledge source. In addition, contextual constraints are generated dynamically which will limit the available choices for the subsequent parts of the sentence. This approach, which mimics the way human form sentences, worked well and showed much future promise. This is the only way to preserve the cohesiveness and to maintain the compatibility of parts of each sentence. The object-oriented viewpoint proved extremely beneficial in coding rules, in that the same object could be viewed in a variety of ways. The modular design allows for the addition of lexical entries and templates that can increase the expressive power of the system with little or no changes to the core code

Our objective here has been to automate the sentence generation process. Hence, we have concentrated on generating non-causal sentences. We believe that we have broken new ground. A natural extension of this work is to modify the system to generate causal sentences, sentences which express a desired meaning.

The proposed system would add another layer on top of the current system. This layer would receive as input a collection of objects, each object expressing a segment of the desired intention, and collectively expressing the complete theme or intention. This **information** is used to select the appropriate sentence structure and to construct the proper thematic constraints and would then serve as input to the current system to guide the generation process. For instance, the following represents a partial overview of three objects: ***(agent john) (action ptrans past) (recipient store (category old)).*** To express the desired intention, the following surface structure: **[NP]** [verb] **[PP]** along with the following semantic structure: **(Agn-Sub-NP)** (Act-Pred-VP) (Pnt-DO-NP) may be selected. These structures along with the three objects above guide the system to generate the sentence: 'John went to the old store.' This extension would require very little change in the **current** system. The mechanism for allowing sentence level constraints already exists in the current system. The thematic **constraints** would simply need to be expressed as sentence level constraints in order to be passed throughout the system.

The system developed here is the first step in completely automating the entire evaluation process. The entire evaluation process of an NLP system performed by a human evaluator requires three stages. In stage one, a test sentence is generated; in **stage** two, the NLP system's response is analyzed; and in stage three, the NLP system's response is scored. The current state of our generation system performs stage one of the evaluation process. We plan to continue this effort by extending the system to generate causal sentences and to perform stage two and three of the evaluation process.

6.0 References

[1] Booch G. 1991. *Object Oriented Design with Applications.* Benjamin/Cummings Publishmg Company.

[2] Fowler H., 1980. *The Little, Brown Handbook.* Little, Brown and Company.

[3] Gazdar G., and Mellish C., 1989. *Natural Language Processing in LISP.* Addison-Wesley Publishmg Company.

[4] Hirst G., 1987. Semantic *Interpretation and the Resolution of Ambiguity.* Cambridge University Press.

[5] Kaikhah, K., 1992. *An Investigation of the Benchmark Evaluation Tool.* Technical Report, *Air* Force Office of Scientific Research (AFOSR), Summer 1992.

[6] Keene S., 1989. *Object-Oriented Programming in Common LISP.* Addison-Wesley.

[7] Neal J., Feit E., Funke D., and Montgomery C. 1992. Benchmark Investigation/Identification Program. Technical Report, Calspan Advanced Technology Center.

[8] Sowa J., 1988. Using a Lexicon of Canonical Graphs in a Semantic Interpreter in *Relational Models of the Lexicon: Representing Knowledge in Semantic Networks.* ed Evans, M.. Cambridge University Press.

[9] Steele G., 1990. *Common LISP - Second Edition.* Digital Press.

[10] Thomson A, and Martinet A, 1986. *A Practical English Grammar - Fourth Edition.* Oxford University Press.

[11] Winston P., and Horn B., 1989. *LISP - Third Edition.* Addison-Wesley Publishmg Company.