OxiNet: A New Natural Language Processing Software

Ganapathy Mani, Stephen Kaisler
Department of Computer Science, George Washington University, Washington DC, USA
gans87@gwmail.gwu.edu, skaisler1@comcast.net

ABSTRACT

The idea of giving computers the ability to process human language is as old as the idea of computer themselves [1]. Nowadays, designing computer systems with full human intelligence capabilities has been the conspicuous focus of Artificial Intelligence researchers. They have made tremendous amount of efforts in designing programs that understand with deep knowledge of the semantic meaning of texts. In this paper, I am going to introduce an art called Natural Language Processing that makes all the Artificial Intelligence researchers to reach their goal. Through this paper, I will introduce the most important Natural Language Processing typical examples (UNO, Rule based, Statistical, Finite state machine, Logic programming etc.). Moreover, I will give a brief overview on the important tools used in the implementation of NLP applications. These tools include Lexicons such as Wordnet and syntactic parsers such as the Link Grammar. Finally I will provide a design schema that illustrates how lexicons and parsers are used in order to implement a simple NLP application. The application will take a user-defined sentence as input and it detects the main verb and outputs all the possible synonyms corresponding to that verb. This work presents a new Wordnet model called, Optimized Lexicon Wordnet (OxiNet) that is able to detect anomalies in sentences and words of a given text with close to 80% accuracy.
1. **INTRODUCTION:**

Natural Language Processing (NLP) can be defined, in a very general way, as the discipline having as its ultimate, very ambitious goal of that enabling people to interact with machines using their “natural” facilities and skills [3]. It means the computers should be capable of understanding spoken or written sentences constructed by humans. In this most ambitious form, this sort of endeavor is an extremely difficult one – a full, fluent and totally automated man/machine interaction surely beyond the possibilities of the current technology. According to Microsoft Research Group “The goal of the NLP is to build a computer system that will analyze, understand, process and generate languages that humans use naturally, so that eventually you can address your computer as you were addressing another person” [4].

As a simple example, an English speaker obviously understands a sentence like “Flying planes can be dangerous”, however, this sentence presents some difficulties to a software program that lacks both the human knowledge of the world and his/her experience with linguistic structures. “Is the best interpretation that the pilot is at risk, or that the danger is to people on the ground? How do "can" be analyzed as a verb or as a noun? Which of the many possible meanings of "plane" is relevant? Depending on context, "plane" could refer to, among other things, an airplane, a geometric object, or a woodworking tool.” [4]. The best solution to this problem is to train the NLP system to learn that a "plane" is a vehicle that flies. Nevertheless, in these last 20 years, some success in dealing in an automatic way with natural language have been achieved in several fields, and NLP techniques have gained considerably in popularity, attracting and increasing number of researchers and practitioners.
2. **HISTORY:**

We can trace the NLP’s appearance to the Second World War and the extensive utilization of statistical techniques for breaking enemy codes: They made use, in fact, of “linguistic” (Lexicographic) tools consisting of tables of letter frequencies, world frequencies, transition frequencies between successive letters etc. Starting in the late 1950s and the early 1960s, there has been, in fact, a huge computer-based production of lexicographic tools – as word indexes and concordances (indexes where each word is accompanied by a line of context) – intended to assist in the apprehension of the style and of the thoughts of authors like Livy, Thomas Aquinas, Dante, or Shakespeare [3].

The sentiment that the use of NLP techniques could go beyond the simple counting and rearranging of words did not take longer to appear. In 1946, Warren Weaver and A. Donald Booth began working on the idea that computers could be used to implement solution to the world wide problems concerning natural language processing into another. In 1949 “Weaver Memorandum” – which Weaver distributed to about 200 of his acquaintances and which marks the official beginning of the machine translation subfield – he writes: “*when I look an article written in Russian, I say, “this is really written in English, but it has been coded into some strange symbols. I will now proceed to decode”*. [3] This idea that the source and target texts say exactly the same thing influenced strongly early work on machine translation.

In 1954, a research group from IBM and Georgetown University set up a public demonstration of a system translating from Russian to English; [5] the system had no particular scientific value, but it encouraged the belief that machine translation was feasible and within
easy reach. The system was representative of the “direct translation” approach, a slight improvement on the early word-for-word techniques. The “direct” systems were monolithic pieces of software specifically designed, in all detail, for a particular pair of languages. This period is then remembered as the golden age of syntax, characterized by publication of important work by Y. Bar Hillel, Zellig Harris and, mainly Noam Chomsky. Chomsky’s “Syntactic Structure,” where “transformational grammars” were introduced was published in 1957. [3]

In parallel, the need for programming languages well – suited for manipulation of linguistic structures led to the creation of the first languages, COMIT and SNOBOL, geared more toward symbolic calculus than toward “number crunching” – this was not without influencing the creation, in 1960, of both ALGOL and LISP, characterized by the presence of new features such as lists and recursion. But semantics were lagging behind. In 1960, Bar-Hillel published a survey of machine translation where he expressed the opinion that machine translation can be improved by theoretical research. His famous example concerns two sentences “The pen in the box” and “The box in the pen,” and the impossibility for a machine to determine that the word “pen” in the second sentence means “an enclosure where small children can play” without having at its disposal some sort of universal encyclopedia. Not all researchers agreed with Bar-Hillel. However they officially endorsed by the publication in 1966, of the so-called ALPAC (Automatic Language Processing Advisory Committee) report. [6]
By the 1970s, William Wood developed LUNAR and Terry Winograd developed SHRDLU where both scientists attempting to deal in a comprehensive way with both syntax and semantics. The LUNAR has been used by NASA to find out the chemical components of rock samples brought back from the moon by Apollo 11 mission. [3] From there many developments and researches were carried out in speech understanding. Confronted with this situation, the Advanced Research Projects Agency of the U.S. Department of Defense (ARPA) promoted, starting in 1971, a 5 year program in speech understanding. The systems developed in APRA framework were HEARSAY – I and DRAGON in the first phase of the program (1971 – 1973), HARPY, and HEARSAY – II by 1976. The HEARSAY systems are particularly well known given that they were the first AI systems using “Blackboard” architecture.

The developments of NLP techniques in the 1960s and the 1970s are the beginnings of a field that is particularly important for setting up more comfortable, efficient, and natural languages between people and computer systems, i.e., the field of the “speech understanding systems”. In 1993, MUC – 5 conference gave the results of a study that conducted on the English microelectronics task that indicates the best machine system performs with an error rate of about 62%, a little less than twice the error rate of a highly skilled human analyst (33%). However very good performances (34%) attained have by the “best” system on the simple limited tasks. The results, even if they certainly show that machine performance is at least comparable with human performance, should however tone down any over-enthusiastic prediction about the appearance on the market as commercial products. [3]
3. **NLP IMPLEMENTATION MODEL:**

In this section we will be discussing some of the implementation approaches of the natural language processing.

3.1. **The Rule – Based approach:**

The rule based systems use many small silvers of knowledge organized into conditional If-Then rules. Inference engines for RBS (Rule Based Systems) are goal-driven, backward-chaining, or data-driven, forward-chaining, depending on the type of application and or generic task. Rules often represent heuristics – shortcuts or rules-of-thumb for solving problems. Regarding the amount of structure, RBS fall in between Case-Based Reasoning (CBR), and Model-Based Reasoning (MBR) – rules of thumb are abstracted and generalized from experience into small chunks of knowledge. Both CBR and RBS can be developed incrementally and can provide some value in an unfinished state. RBS can also be transformed into objects or frames using knowledge discovery and database mining techniques.

Rule-based systems should be used when: [7]

- Human experts think in terms of rules and heuristics
- Task involves decision-making, problem solving, heuristics or judgment
- Domain is complex and substantial expertise exists
- Knowledge is stable and is well - or semi-structured
- Expertise is primarily symbolic, not numeric
- Human experts are willing and available for knowledge citation
- Work performance and product quality are poor
• Employee turnover is high and training is expensive
• Impending loss of domain experts

The verification is based on a confidence value output by a rule mapping function. The mapping function uses some parameters that can be entered by the user. Mostly, the mapping function is a function of basic properties of the feature to be verified or detected. The NLP system takes action based on a calculated confidence value. A very important advantage of the rule-based paradigm is that rules are modular; they can be added or deleted without affecting the other parts of the system. Moreover, if there is a little training data, the user can use it to fine tune the parameters used in the rule mapping function. (This can improve the ability to produce a better confidence value and therefore, high quality output). [8]. However, The rule-based paradigm is more classical and is not robust since rule based systems can crash in the face of any unexpected input.

For example in Java, The rules are often described as a series of nested if-statement. When you have enough nesting, it gets really tough to see what is really going on in the code. And no least the difficulty to change the code when a new rule should be added or a rule should be changed, which requires a recompilation of the code. I have been one of those frustrated programmers, programming rules in Java that after some time looks like spaghetti code.

A typical Java program describing a few business rules could be like this: [9]

```java
if ((user.isMemberOf(AdministratorGroup) && user.isMemberOf(teleworkerGroup))
```
```
|| user.isSuperUser(){
    // more checks for specific cases
    if((expenseRequest.code().equals("B203")
    ||(expenseRequest.code().equals("A903")
    &&(totalExpenses<200)
    &&(bossSignOff> totalExpenses))
    &&(deptBudget.notExceeded)) {
    //issue payments
    } else if {
    //check lots of other conditions
    }
    } else {
    // even more business logic
    }
}
```

This above example shows that rules are hard to program and maintain in Java. Another problem is that of role. What if a businessperson wants to change the “B203” into a “B204”? Should the programmer be involved every time business rules changes? This is not sustainable. By having the rules separated from the source code, the businessperson can actually read and understand the rules, and change them to reflect changes in business. To explain this clearly I am going to use a use case assignment [10] called PetShop.
3.1.1 **PetShop:**

In this assignment I will look at a use case called PetShop, which is an on-line shop where you can buy Gold fish, fish tank and fish food. All the rules are fired when the customer checks out. We will use forward chaining. PetShop is an example application that is part of the Drools distribution. In the classroom presentation, I will demonstrate how easy it is to add your own rules. I have added a rule for discount on sales for a given month.

![The Interface to the PetShop](image)

*Figure: 1. The Interface to the PetShop*

The **PetShop** has the following business rules: [10]
Fish Food Sample

IF (shopping cart do not contain item “Fish Food”) AND (shopping cart do not contain “Fish Food Sample”) AND (item is a “Gold Fish”) THEN (add item “Fish Food Sample” to shopping cart)

Suggest Tank

IF (we have not suggested a “Fish Tank” before) AND (the shopping cart contains 5 or more “Gold Fish” items) AND (there is no “Fish Tank” in the shopping cart) THEN (suggest a “Fish Tank”)

Apply 5% Discount

IF (gross cost is greater or equals to 10.00) AND (gross cost is less than 20.00) THEN (add 5% discount to the shopping cart)

Apply 10% Discount

IF (gross cost is greater or equals to 20.00) THEN (add 10% discount to the shopping cart)

3.1.2. Drools – Rule-Based System:
The tool that I have looked at for this assignment is Drools [11].

Drools interference engine is based on the Rete [12] algorithm developed by Charles Forgy. The algorithm is not explained in this assignment, since it is not necessary to know about the Rete algorithm to understand the concept of Rule engines. Drools are a forward chaining inference expert system. Drools is programmed in Java, but support several other languages. Rules are described in XML files and Java (or Python, Groovy).

Example from the PetShop “Apply 5% Discount” rule: [10]

<!-- **************************************************** -->

<! -- Give 5% discount if gross cost is more than 20.00 -->

<! -- **************************************************** -->

<rule name="Apply 5% Discount">

<parameter identifier="cart">

<class>org.drools.examples.petstore.ShoppingCart</class>

</parameter>

<java:condition>cart.getGrossCost() &gt;= 10.00</java:condition>

<java:condition>cart.getGrossCost() &lt; 20.00</java:condition>

<java:condition>cart.getDiscount() &lt; 0.05</java:condition>

<java:consequence>

System.out.println("Applying 5% discount to cart");

cart.setDiscount( 0.05 );

</java:consequence>
The XML file can include many rules. The parameter tag `<parameter identifier="cart">` is a named reference to an actual Java class. The reference is used in the `<java:condition>` tag. The method `cart.getGrossCost()` is actually called by this condition. All conditions has to be met for the `<java:consequence>` to be executed.

This is how you load the rules:

```java
RuleBase ruleBase = RuleBaseLoader.loadFromReader(new FileReader("abc.drl"));
```

After the rules are loaded, we create a “WorkingMemory”. It is where all knowledge or “facts” is kept.

```java
WorkingMemory workingMemory = ruleBase.newWorkingMemory();
```

The WorkingMemory can be manipulated in various ways. You can add (assertObject), modify (modifyObject) and remove (retractObject) facts from the WorkingMemory. In this example we will only add facts to the WorkingMemory.

```java
workingMemory.createObject(cart);
```

We add the shopping cart as a fact, so that the engine is aware of it existence.

```java
workingMemory.assertObject(cart);
```

Now that we have loaded the rules and facts, let us fire all the rules.

```java
workingMemory.fireAllRules();
```
3.1.3. Resolution of Conflict:

A common problem is if several rules have been activated, what rule should be fired first. Drools offer several strategies [13] to solve this problem. I will not describe them all, but the on PetShop uses. The solution used in the PetShop is “Salience”. Salience is a kind of priority. You can specify a priority for each rule in the XML file. You add an attribute to the rule tag:

```xml
<rule name="Apply 2% Sales Discount" salience="-5">
```

It has to be an Integer value. The higher the number the higher priority the rule has. In this example, it is important that this rule fires last after all the other rules. So this rule has to have the lowest salience.

The following rule:

```xml
<rule name="Explode Cart" salience="20">
<parameter identifier="cart">
<class>org.drools.examples.petstore.ShoppingCart</class>
</parameter>

<java:condition>cart.getState("Exploded") == false</java:condition>
<java:consequence>
import java.util.Iterator;
System.out.println("Examining each item in the shopping cart.");
```
Iterator itemIter = cart.getItems().iterator();
while ( itemIter.hasNext() )
{
    drools.assertObject( itemIter.next() );
}
cart.setState( "Exploded", true);
drools.modifyObject( cart );
</rule>

It has the highest priority (20), since the other rules will fail if this rule has not been fired first.

3.2. THE FINITE STATE MACHINE MODEL:

(Motivated by: http://ai-depot.com/)

Here I will explain the implementation of finite state machines from boarder perspective. I will investigate how a finite state machine can be implemented, and do deep analysis at a framework that can facilitate multiple FSMs in a simulated real-time environment.

As single FSM is having a little use; therefore we need to do deep analysis of implementations from a broader view to understand where a single FSM plugs in. We will analyze portions of the finite state machine framework from the computer game Quake, in attempt to understand how to make use of the technique in a real world application. [16]
One possible way to implement finite state machines is to have a controller of some type which acts as switch box. When the thread of execution swings around to execute code of the FSM, it is pointed at the controller which evaluates or determines the current state, usually through the use of a switch case statement or if-then-else statements. Once the current state is determined, the code for that state is executed, actions performed and possibly state transitions for the next time the FSM is executed. The controller may be a simple switch statement evaluating an integer, but an implementation may see the controller performing some pre-processing of inputs and triggering of state transitions before-hand.

Figure: 2. FSM Implementation [14]

The implementation that the programmers at id Software have chosen could almost be considered to have Object-Oriented (OO) feel (though the implementation is not OO). As mentioned before, the game world is populated by entities, and as such a generic entity structure is used as their basis. Each entity in the collection of entities is provided with some execution time by calling its "think" function. The entity is executed by the game code in what could be described as a polymorphic manner. Entities have a common interface (because they are all the
same data structure), and this interface consists of function pointers which are used to execute entity specific and entity non-specific code as either action outputs or input events for the FSM.

An example; most entities are affected by damage. Damage can be inflicted by many things like a rocket projectile for example. When a damage trigger is transmitted to another entity, its pain function pointer is called, thus triggering a state transition of the effected entity into possibly a death or attack state. Key point: The damage inflicted is an input to the FSM, which may act as a trigger for a state transition.

In essence the same switch-box technique described in figure 2 is used, where the entities base data structure provides function pointers which act as the "switches". When an entity is given a chance to execute its state, its "think" function pointer is called. If previously a damage input was received, the entity may have had a state transition into its "die state". When the thread of execution runs, the objects "die state" code is then run (via a polymorphic call of the entities think function), removing the entity instance from the game world. Rather than further explain using theory, let’s take a look at a practical example from the game Quake. A dog is a simple monster that wishes’ to attack the player or any other monster that angers it, much like the previously seen Shambler monster.
Figure 3 shows the four main states of a Dog monster from Quake. Superimposed over that are the main actions performed in each of those states. Some map directly to functions, others map to chains of function calls. The representation clearly shows the five main input event actions with dark borders which are called as outputs of other finite state machines. There are two types of actions displayed, the gray background represents the dog monster specific code, and the white represent framework code used by all monsters.

The figure is a good diagrammatical tool for showing what monster specific code needs to be written and where it fits into the framework. It is not a complete representation, for
example, in regard to the code executed after spawning, there are other "swim" and "fly" start actions used by some monsters that can swim and fly as opposed to our walking dog. Also there is a third attack sub-state called "slide attack" which is not relevant to the dog monster.

It can be seen that although the design of a FSM on paper may be clear cut and seem easy to implement, when it comes to implementing a larger number of them, such as all the monsters in a computer game, it is easy and probably necessary to blur the boundaries of the states. In figure 3.2 we do not see well defined functions representing states as may have been expected, instead functions are shared by states and FSMs. In this example it is clear that core actions have been decomposed and their functionality abstracted to be incorporated into the framework to be used by all monsters of the game. It is a nice approach for speed of execution, maintainability and code reuse, but poor due to a moderate level of complexity (time must be invested to trace the execution path). [19].

The finite machine is provided with execution time through its think function. It evaluates inputs from its inputs from of the game world, but can directly receive specific events as input from the output of actions performed by other FSMs. These include a touch, use, pain and die events. These events can trigger state changes of the effected finite machine, for example, as seen in Figure 3.2 a touch input event is a collision determined by the game as it advances the game physics. When received in the above example, and if the dog monster has a valid "touch" sensor function specified (which may not always be the case), it will run the code in that function and possibly have a transition to its missile attack sub state or to its attack state for a revaluation of its attack sub-states. [18].
It is becoming clear that a FSM in this domain is very useful as a control mechanism and when used on a larger scale as seen, it is powerful. This example shows that a FSM framework can provide the ability for a simple multi-agent system, where each FSM system could be considered an agent (intelligent; uses Artificial Intelligent techniques, autonomous; acts independently). The FSMs have sensor functions implemented specifically to handle expected events, and also has effectors functions which are simply the actions performed in the game world. [15]

### 3.2.1 A Practical Analysis of FSM within the domain of first-person shooter (FPS) computer game

The intent of this section is to use a computer game to illustrate the conceptual workings of a FSM based on a practical rather than theoretical implementation. I will not attempt to provide insight into how the computer game works or even many specifics (code) of the implementation in the computer game. The FSMs discussed in this section have been coded, tested and released in production code, providing real world examples, within the domain of a first person computer game.

This section will provide two examples of finite state machines from a first-person computer game created by id Software [20] called Quake [21]. The reason I chose this product as the basis for the analysis was due the fact that game engine code has been released publicly under the GNU General Public License (GPL) [22]. The other reason for this decision was because at the time the game was released, it was cutting-edge and
had its code licensed by other companies that produced further highly-popular titles such as Unreal and Half-Life, thus proving the success of the original product.

We are going to start with the analysis of something very simple. We will see that even through mapping the states of a simple projectile like a rocket, we can learn more about the very nature of FSM. I make no claim at being an expert in regard to Quake game code (this was my first excursion through it), so forgive me if my interpretations based on code do not directly map. A rocket in Quake is a projectile fired from the Rocket Launcher weapon/item which may be possessed and operated by a human player. [23].

![State transition representation of a rocket projectile from Quake](image.png)

*Figure: 4. State transition representation of a rocket projectile from Quake [14].*
Rather than restricting me and confusing the reader by using a formal notation as mentioned in the first section of this document, the finite state machine has been represented using an approach very similar to a State Transition Diagram. The blue boxes are the states, the orange the triggers and the arrows are the state transitions. The black boxes show the entry point and exit points of the system.

The diagram shows the full life cycle of the rocket projectile within the game. It is interesting to note that the projectile is spawned into existence as the product of an action of another FSM, namely that of the "rocket launcher" from its "fire" action. When the projectile instance dies it is removed from the game, and no longer exists. [23].

This representation is a subjective interpretation of code. Another valid representation may break the "touch state" down further into touch and explode states. Personally I view explode as an action or effect performed by the rocket object in its touch state.

Another interesting note is that when the projectile is in its touch state, one of its effectors is to attempt to damage whatever it is touching. If it succeeds in damaging another entity in the game world, the damage action becomes an input which can trigger a state change of the effected entity into another state.

Quake makes extensive use of FSMs as a control mechanism governing the entities that exist in the game world. This has been provided by an interesting framework which is tightly related to the way FSM work in the computer game. For further discussion of this framework, please see section three.
Let’s take a look at a slightly more advanced FSM from Quake. A Shambler is a big bad monster entity from the single player component of Quake. Its mission in life is to kill the player, once it is aware of the player.

![State transition representation of a Shambler monster from Quake](image)

*Figure: 5. State transition representation of a Shambler monster from Quake*

Like the rocket projectile, this entity has an initial state (spawn state), and the system ends when the entity dies (die state). There are only four identified main states, but the Shambler is a good example that illustrates the ability to have a hierarchy of sub-states. Though the sub-states in this example could be considered actions of the "attack state", they are also sub-states because the monster can only perform one (or be in one) of them per execution of the attack state.
When in the attack state, the Shambler instance makes a decision based on evaluated inputs whether to perform a melee (close up) or missile (long distance) style of attack on its goal. Upon selecting a melee type attack (melee attack state), the inputs are further evaluated, along with a random number to select a melee attack type (smash with both arms, smash with left arm or smash with right arm).

The use of a random number in the selection of a melee attack sub-state adds a level of unpredictability to the selection. Each level in the hierarchy could be considered a sub-finite state machine of the greater monster entity, and in this case the sub-FSM of the melee attack state could be classified as non-deterministic.

It is important to understand the use of layered or hierarchical finite state machines, because when used as seen in the Shambler monster it allows far more complex behaviors. This technique is heavily used in Quake by all entities in the game world. Because of this, a lot of code has been abstracted to be easily be used in many different places, actions such as movement, visibility assessments and so on.

This example has been greatly simplified to make it more readable. An example of this is in the triggers which cause the state transitions. When a state transition occurs from "Attack State" to "Idle State", the trigger has been simplified as "lost goal". It is true that the transition occurs due to the loss of the goal entity, but a goal can be lost by the Shambler in a number of ways evaluated at different points in the code, including time-out and damage from another entity.
Another key point regarding this example is the use of goals as the primary motivator for the FSM. This technique has not been discussed, though it is an example of the power and flexibility of FSM as a control technique. As well as possessing a hierarchy of finite state machines, the high level FSM is driven by the entities desire to locate a goal, and attack its goal. The goal is usually a human player or even another monster in the game world. It should be noted that whilst the monster is in its idle state, as well as just standing around it is also looking for goals to walk to (for roaming).

Quake did not provide the best single player experience imaginable, though it was and still is fun and addictive, both key attributes of the games success. It is good example, and a good learning tool that can show the power of both very simple finite state machines such as the rocket projectile, and slightly more complex FSM made up of a hierarchy of FSM and motivated by goals, such as the Shambler monster. [23].

3.3. **THE STATISTICAL APPROACH:**

*(Motivated by: Prof. Kowalski, CS Dept, GWU, Information Retrieval Course and http://stp.lingfil.uu.se/~nivre/)*

The statistical natural language processing uses stochastic, probabilistic and statistical methods to resolve some of the difficulties discussed above, especially those which arise because longer sentences are highly ambiguous when processed with realistic grammars, yielding thousands or millions of possible analyses. [24].
3.3.1 **Statistical Models:**

There are two broad classes of mathematical models: deterministic and stochastic. A mathematical model is said to be deterministic if it does not involve the concept of probability; otherwise it is said to be stochastic. Furthermore, a stochastic model is said to be probabilistic or statistical, if its representation is from the theories of probability or statistics, respectively.

Although Edmundson applies the terms stochastic, probabilistic and statistical only to models, it is obvious that they can be used about methods as well. First of all, we have defined both application methods and acquisition methods in such a way that they crucially involve a (possibly parameterized) model. If this model is stochastic, then it is reasonable to call the whole method stochastic. Secondly, we shall see that also the algorithmic parts of application and acquisition methods can contain stochastic elements. Finally, it seems uncontroversial to apply the term statistical to evaluation methods that make use of descriptive and/or inferential statistics. [25].

In the taxonomy proposed by Edmundson, the most general concept is that of a stochastic model, with probabilistic and statistical models as special cases. Although this may be the mathematically correct way of using these terms, it does not seem to reflect current usage in the NLP community, where especially the term statistical is used in a wider sense more or less synonymous with stochastic in Edmundson’s sense. We will continue to follow current usage in this respect.
Thus, for the purpose of this paper, we will say that a model or method is statistical (or stochastic) if it involves the concept of probability (or related notions such as entropy and mutual information) or if it uses concepts of statistical theory (such as statistical estimation and hypothesis testing).

3.3.2. **Statistical Methods in NLP:**

1. **Application Methods:**

Most examples of statistical application methods in the literature are methods that make use of a stochastic model, but where the algorithm applied to this model is entirely deterministic. Typically, the abstract model problem computed by the algorithm is an optimization problem which consists in maximizing the probability of the output given the input. Here are some examples:

- Language modeling for automatic speech recognition using smoothed n-grams to find the most probable string of words $w_1; \ldots; w_n$ out of a set of candidate strings compatible with the acoustic data [26, 27].

- Part-of-speech tagging using hidden Markov models to find the most probable tag sequence $t_1; \ldots; t_n$ given a word sequence $w_1; \ldots; w_n$ [28, 29, 30].

- Syntactic parsing using probabilistic grammars to find the most probable parse tree $T$ given a word sequence $w_1; \ldots; w_n$ (or tag sequence $t_1; \ldots; t_n$) [31, 32, 33].
• Word sense disambiguation using Bayesian classifiers to find the most probable sense \( s \) for word \( w \) in context \( C \) [34, 35].

• Machine translation using probabilistic models to find the most probable target language sentence \( t \) for a given source language sentence \( s \) [34].

Many of the application methods listed above involve models that can be seen as instances of Shannon’s *noisy channel model* [36], which represents a Bayesian modeling approach. The essential components of this model are the following:

• The problem is to predict a hidden variable \( H \) from an observed variable \( O \), where \( O \) can be seen as the result of transmitting \( H \) over a noisy channel.

• The solution is to find that value \( h \) of \( H \) which maximizes the conditional probability \( P(h \mid o) \), for the observed value \( o \) of \( O \).

• The conditional probability \( P(h \mid o) \) is often difficult to estimate directly, because this requires control over the variable \( o \) whose value is probabilistically dependent on the noisy channel.

• Therefore, instead of maximizing \( P(h \mid o) \), we maximize the product \( P(h) P(o \mid h) \), where the factors can be estimated independently, given representative samples of \( H \) and \( (H;O) \), respectively.
Within the field of NLP, the noisy channel model was first applied with great success to the problem of speech recognition [26, 27]. As pointed out by [13], this inspired NLP researchers to apply the same basic model to a wide range of other NLP problems, where the original channel metaphor can sometimes be extremely far-fetched.

It should be noted that there is no conflict in principle between the use of stochastic models and the notion of linguistic rules. For example, probabilistic parsing often makes use of exactly the same kind of rules as traditional grammar-based parsing and produces exactly the same kind of parse trees. Thus, a stochastic context-free grammar is an ordinary context free grammar, where each production rule is associated with a probability (in such a way that probabilities sum to 1 for all rules with the same left-hand side); cf. also [31, 32, 33].

All of the examples discussed so far involve a stochastic model in combination with a deterministic algorithm. However, there are also application methods where not only the model but also the algorithm is stochastic in nature. A good example is the use of a Monte Carlo algorithm for parsing with the DOP model [37]. This is motivated by the fact that the abstract model problem, in this case the parsing problem for the DOP model, is intractable in principle and can only be solved efficiently by approximation.

2. Acquisition Methods:

Statistical acquisition methods are methods that rely on statistical inference to induce models (or model parameters) from empirical data, in particular corpus data, using either supervised or unsupervised learning algorithms. The model induced may or may not be a
stochastic model, which means that there are as many variations in this area as there are different NLP models. We will therefore limit ourselves to a few representative examples and observations, starting with acquisition methods for stochastic models.

Supervised learning of stochastic models is often based on maximum-likelihood estimation (MLE) using relative frequencies. Given a parameterized model $M_o$ with parameter $o$ and a sample of data $C$, a maximum likelihood estimation of $o$ is an estimate that maximizes the likelihood function $P(C \mid o)$. For example, if we want to estimate the category probabilities of a discrete variable $X$ with a finite number of possible values $x_1; \ldots; x_n$ given a sample $C$, then the MLE is obtained by letting $\hat{P}(x_i) = f(C)(x_i)(1 \leq i \leq n)$, where $f(C)(x_i)$ is the relative frequency of $x_i$ in $C$.

In actual practice, pure MLE is seldom satisfactory because of the so-called sparse data problem, which makes it necessary to smooth the probability distributions obtained by MLE. For example, hidden Markov models for part-of-speech tagging are often based on smoothed relative frequency estimates derived from a tagged corpus (see, e.g., [31, 23]).

Unsupervised learning of stochastic models requires a method for estimating model parameters from unanalyzed data, such as the Expectation-Maximization algorithm [31]. Let $M_o$ be a parameterized model with parameter $o$, let $H$ be the hidden (analysis) variable, and let $C$ be a data sample from the observable variable $O$. Then, as observed in [35], the EM algorithm can be seen as an iterative solution to the following circular statements:
- **Estimate**: If we knew the value of $o$, then we could compute the expected distribution of $H$ in $C$.

- **Maximize**: If we knew the distribution of $H$ in $C$, then we could compute the MLE of $o$.

The circularity is broken by starting with a guess for $o$ and iterating back and forth between an *expectation step* and a *maximization step* until the process converges, which means that a local maximum for the likelihood function has been found. This general idea is instantiated in a number of different algorithms that provide acquisition methods for different stochastic models. Here are some examples, taken from [27]:

- The Baum-Welch or forward-backward algorithm for hidden Markov models [29].
- The inside-outside algorithm for inducing stochastic context-free grammars [36].
- The unsupervised word sense disambiguation algorithm of [37].

It is important to note that, although statistical acquisition methods may be more prominent in relation to stochastic models, they can in principle be used to induce any kind of model from empirical data, given suitable constraints on the model itself. In particular, statistical methods can be used to induce models involving linguistic rules of various kinds, such as rewrite rules for part-of-speech tagging [33] or constraint grammar rules [27].

Finally, we note that the use of stochastic or randomized algorithms can be found in acquisition methods as well as application methods. Thus, in [26] a Monte-Carlo algorithm is
used to improve the efficiency of transformation-based learning [28] when applied to dialogue act tagging.

3. Evaluation Methods:

As noted earlier, evaluation of NLP systems can have different purposes and consider many different dimensions of a system. Consequently, there are a wide variety of methods that can be used for evaluation. Many of these methods involve empirical experiments or quasi experiments in which the system is applied to a representative sample of data in order to provide quantitative measures of aspects such as efficiency, accuracy and robustness. These evaluation methods can make use of statistics in at least three different ways:

- Descriptive Statistics
- Estimation
- Hypothesis Testing

Before exemplifying the use of descriptive statistics, estimation and hypothesis testing in natural language processing, it is worth pointing out that these methods can be applied to any kind of NLP system, regardless of whether the system itself makes use of statistical methods. It is also worth remembering that evaluation methods are used not only to evaluate complete systems but also to provide iterative feedback during acquisition.

Descriptive statistics is often used to provide the quantitative measurements of a particular quality such as accuracy or robustness, as exemplified in the following list:

- Word error rate, usually defined as the number of deletions, insertions and substitutions divided by the number of words in the test sample, is the standard measure of accuracy for automatic speech recognition systems (see, e.g., [31]).
• Accuracy rate (or percent correct), defined as the number of correct cases divided by the total number of cases, is commonly used as a measure of accuracy for part-of-speech tagging and word sense disambiguation (see, e.g., [31]).

• Recall and precision, often defined as the number of true positives divided by, respectively, the sum of true positives and false negatives (recall) and the sum of true positives and false positives (precision), are used as measures of accuracy for a wide range of applications including part-of-speech tagging, syntactic parsing and information retrieval (see, e.g., [31]).

Statistical estimation becomes relevant when we want to generalize the experimental results obtained for a particular test sample. For example, suppose that a particular system s obtains accuracy rate r when applied to a particular test corpus. How much confidence should we place on r as an estimate of the true accuracy rate p of system s? According to statistical theory, the answer depends on a number of factors such as the amount of variation and the size of the test sample. The standard method for dealing with this problem is to compute a confidence interval i, which allows us to say that the real accuracy rate p lies in the interval \([r - i; r + i]\) with probability p. Commonly used values of p are 0.95 and 0.99. Statistical hypothesis testing is crucial when we want to compare the experimental results of different systems applied to the same test sample. For example, suppose that two systems s1 and s2 obtain an error rate of r1 and r2 when measured with respect to a particular test corpus, and suppose furthermore that r1 < r2. Can we draw the conclusion that s1 has higher accuracy than s2 in general? Again, statistical theory tells us that the answer depends on a number of factors including the size of the
difference $r_2 - r_1$, the amount of variation, and the size of the test sample. And again, there are standard tests available for testing whether a difference is statistically significant, i.e. whether the probability $p$ that there is no difference between $p_1$ and $p_2$ is smaller than particular threshold $p$. Standard tests of statistical significance for this kind of situation include the paired t-test, Wilcoxon’s signed ranks test, and McNamara’s test. Commonly used values of $p$ are 0.05 and 0.01.

4. Semantic Indexing:

(Motivated by: Prof. Gerald Kowalski, CS Dept. GWU, Information Retrieval)

The usage of word senses in the process of document indexing is a pretty much debated field of discussions. The basic idea is to index word meanings, rather than words taken as lexical strings. A survey of the efforts of incorporating WSD (Word Sense Disambiguation) into Information Retrieval (IR) is presented in (Sanderson, 2000). Experiments performed by different researchers led to various, sometime contradicting results. Nevertheless, the conclusion which can be drawn from all these experiments is that a highly accurate Word Sense Disambiguation algorithm is needed in order to obtain an increase in the performance of IR systems. [38]

One of the largest studies regarding the applicability of word semantics to IR is reported by Krovetz (Krovetz and Croft, 1993), (Krovetz, 1997). When talking about word ambiguity, he collapses both the morphological and semantic aspects of ambiguity, and refers them as polysemy and homonymy. He shows that word senses should be used in addition to word based indexing, rather than indexing on word senses alone, basically because of the uncertainty involved in sense disambiguation.
He had extensively studied the effect of lexical ambiguity over the experiments described provide a clear indication that word meanings can improve the performance of a retrieval system.

(Gonzalo et al., 1998) performed experiments in sense based indexing: they used the SMART retrieval system and a manually disambiguated collection (Semcor). It turned out that indexing by synsets can increase recall up to 29% respect to word based indexing. Part of their experiments was the simulation of a WSD algorithm with error rates of 5%, 10%, 20%, 30% and 60%; they found that error rates of up to 10% do not substantially affect precision, and a system with WSD errors below 30% still perform better than a standard run. The results of their experiments are encouraging, and proved that an accurate WSD algorithm can significantly help IR systems. [38]

We propose here a system which tries to combine the benefits of word-based and synset-based indexing. Both words and synsets are indexed in the input text, and the retrieval is then performed using either one or both these sources of information. The key to our system is a WSD method for open text.

3.4. **THE OBJECT ORIENTED APPROACH:**

Morphological analysis, as well as Natural Language Processing, raises the problem of ambiguities and the results multiple solutions. Approaches based on sequential phases have shown their limitations [40].This is due to the lack of a real exchange of linguistic information between different phases that are needed to reduce ambiguities. In this section, an
An object-oriented algorithm for morphological analysis is presented. This algorithm accesses a lexicon to check the existence of word stems and to obtain related linguistic information. This lexicon is built as a hierarchy of the classes. Affixes are divided into three subclasses: prefix, infix, and suffix. Each class of these subclasses is subdivided into subclasses based on the size of the affix (number of letters). Instances of these affixes are represented by two levels. The first level contains the regular forms these affixes. The second level contains their irregular forms. Examples of the former (three letters) are "ing", "ful", and "est". Examples of the irregular forms of the prefix "in" are "ir", "il", and "im". This representation provides the following advantages:

• The classification of stems and affixes semantically [41].
• The linguistic information is distributed over several levels. Subclasses inherit the common attributes and override specific ones.
• The morphological analysis routines are distributed over the levels as methods.

The object oriented approach deals with building a base system (framework) that can be reused by any NLP application programmer by plugging other components in the base system. The system should run on a distributed and concurrent environment. Two core architectures can be used as a base for any NLP system especially text processing systems. There are two important natural language processing factors considered as the driving forces of the two architectures presented later in this section. These are the text presentation and the algorithms that process that text. These factors originate from two NLP architectural paradigms used to develop the base system. The two paradigms are: A processing paradigm in which the process is the driving force. In this approach, a process accepts the text and processes it. In text paradigm the text is the driving force.
The text model is a set of words organized hierarchically (Text, chapter, paragraph, sentence). A text model can contain text data views since a processing routine can’t process the whole model or hierarchy. Therefore, a text data view (TDV) can be used to let the routine process the necessary text only. The changes made to the TDV propagate to the hierarchy used by other routines. The TDV is used to access only a particular subset of the text.

3.5. **THE PROCESSING MODEL:**

Also called **Pipeline and Dynamic Behavior**, the processing paradigm originates from the traditional way of procedural thinking. According to this approach, the text should be processed through several processing steps in order to obtain the desired result.
This way of working can be seen as sending a text through a pipeline of processing steps. A pipeline is modeled using different pipeline processes that can be connected to each other according to a specific configuration. An example of a pipeline is illustrated in Figure 7.

![Figure 7. A pipeline example in NLP applications [39]](image)

Figure 7 illustrates a set of possible processing steps in a translation system. A text is submitted to the first processing step. When the step is finished the text is sent to the next processing step. When the last step exits, the result is returned to the caller of the pipeline. For a framework this model is not flexible enough. Often another pipeline setup will be needed when certain conditions are met. Therefore, in order to be able to change the pipeline, we introduce a pipeline monitor. When a text needs to be processed it is submitted to the pipeline through the pipeline monitor.
Figure 8 above shows the way the PM interacts with the actual pipeline. The pipeline monitor first selects a first step in the pipeline and sends the text to this step. The particular processing step accepts the text, handles it and then returns the resulting text back to the pipeline monitor. The pipeline monitor then selects the following processing step, submits the text, etc. When certain conditions are met the pipeline monitor can change the pipeline by inserting processing steps or taking alternative processing steps as can be seen on the figure.
A typical example of the processing paradigm is the machine translation example shown in figure 6 below. In this example, the text is submitted to a text pipeline monitor. The text pipeline monitor submits the text to an ML process that does a morphological lookup on the text. The translation process (TL) decomposes the text in sentences and sends it to the Sentence pipeline monitor. The Sentence Pipeline Monitor then does the actual translation of each sentence by calling the Parsing (PA) and transfer (TR) processes. The parsing process creates representations of the first language sentences. The transfer process transfers each word in the representation to its translation in terms of the second language. Finally, the translated sentence is passed to the caller of the pipeline, which is the translation process (TL). After the text is translated, a new generation process (GE) starts. The generation process processes the representation returned by the translation process and generates the translated sentence in the second language.
3.6. **THE TEXT APPROACH:**

The text paradigm originates from a more object-oriented background. Instead of having an algorithm that processes a text, the text will process itself. This means that next to the data representation, the text encapsulates information on how to process itself. The text model we use is the same as presented in figure 6. The processing components are encapsulated in *text visitors*. Text visitors are a set of methods that the text uses to process itself.
In the text paradigm, a supervising unit, called the processor selects several text visitors in some order, hands them to the text and asks the text to process itself using these text visitors. Generally, the processor knows what needs to be done, step-by-step. The text visitors are the steps that know how to do it, as long as the processor sends them to the text in the right order.

Figure 10 clarifies this approach with an example. The example is about the implementation of the translation system using the same text data view used in figure 3. First of all, the processor chooses a morphological lookup (ML) component and asks the text to accept the component. After acceptance, the text starts processing itself by calling the `handleText()` method. `handleText()` takes the title of the text, which can be retrieved from the root text component of the text structure, and starts looking up the words from the title. When `handleText()` finishes looking up the words, `handleText()` starts iterating over the children of the Text object (chapters) by sending the ML component to these subcomponents. The chapter objects also accept, but this time `handleChapter()` is called. The words in the title of the chapter are looked up, and then `handleChapter()` iterates again over the paragraphs in the text. This continues until the leaves in the text structure are looked up. The ML component finishes and the processor selects the following component, such as a parsing component (PA). The parser is sent over the text structure in the same way, parsing the text title, chapter title and so on.
The two approaches can be seen as opposite point of views. The process point of view deals with sending a text through several processes to obtain a result while the text point of view deals with sending several processes through a text. Although the two models can be seen as opposites, they have a lot in common too. They share the same text model. They both offer the possibility to take run-time processing decisions based on text characteristics. This is obtained by using pipeline monitors in the process point of view and processors in the text point of view.
4. **THE MAIN NATURAL LANGUAGE PROCESSING TOOLS:**

4.1. **The syntactic parsers (Link Grammer):**

Syntactic parsers are programs that receive inputs from interactive online commands and break them into parts that can be managed by other programs. A very good and robust parser is the *Link Grammar* parser. Given a sentence, the Link Grammar assigns to it a syntactic structure, which consists of a set of labeled links connecting pairs of words. “The parser is robust; it is able to skip over portions of the sentence that it cannot understand, and assign some structure to the rest of the sentence. It is able to handle unknown vocabulary, and make intelligent guesses from context and spelling about the syntactic categories of unknown words. It has knowledge of capitalization, numerical expressions, and a variety of punctuation symbols.” [42]. The Link Grammar parser can be integrated in NLP applications through its API. The Link Grammar API defines five basic structures. In order to parse a sentence and extract information from it, the user

Overview of these five data structures:

- **Dictionary**: A Dictionary defines the set of word definitions that defines the grammar. A user creates a Dictionary and passes it to the various parsing routines.
- **Sentence**: The input string tokenized and interpreted according to a specific Dictionary.
- **Parse Options**: Parse_Options specify the different parameters that are used to parse sentences. Examples of the kinds of things that are controlled by Parse_Options include maximum parsing time.
- **Linkage:** This is the API's representation of a parse. A Linkage can be constructed after a sentence has been parsed, and can be thought of as a Sentence together with a collection of links.

- **Postprocessor:** A PostProcessor is associated with each Dictionary, and automatically applied after parsing each Sentence constructed using that dictionary. Individual linkages can be post-processed with different sets of context-sensitive post-processing rules. The API enables this by letting the user open up a set of rules and passes it around as a PostProcessor.

### 4.1.1. Link Grammar Example: [42]

The following example is about a program that opens up a dictionary and then parses two sentences, graphically displaying a linkage for each.

```c
#include "link-includes.h"

int main() {
    Dictionary dict;
    Parse_Options opts;
    Sentence sent;
    Linkage linkage;
    char * diagram;
    int i, num_linkages;
    char * input_string[] = {
```
"Grammar is useless because there is nothing to say -- Gertrude Stein.",
"Computers are useless; they can only give you answers -- Pablo Picasso.");

opts = parse_options_create();
dict = dictionary_create("4.0.dict", "4.0.knowledge", NULL, "4.0.affix");
for (i=0; i<2; ++i) {
    sent = sentence_create(input_string[i], dict);
    num_linkages = sentence_parse(sent, opts);
    if (num_linkages > 0) {
        linkage = linkage_create(0, sent, opts);
        printf("%s\n", diagram = linkage_print_diagram(linkage));
        string_delete(diagram);
        linkage_delete(linkage);
    }
    sentence_delete(sent);
}
dictionary_delete(dict);
parse_options_delete(opts);
return 0;
}
The statements :

opts = parse_options_create();
dict = dictionary_create("4.0.dict", "4.0.knowledge", NULL, "4.0.affix");
Create Parse_Options and a Dictionary to be used in processing sentences. In order to create a dictionary, the program looks in the current directory for the files 4.0.dict and 4.0.knowledge.

`sent = sentence_create(input_string[i], dict);`

takes the sentence as input and uses the Dictionary that was created earlier to tokenize and create word definitions such as (noun, verb, object…etc). The statement

`num_linkages = sentence_parse(sent, opts);`

passes the sentence, along with the Parse_Options, to the function `sentence_parse`, which returns the number of all possible linkages. The following statements

`linkage = linkage_create(0, sent, opts);`

`printf("%s\n", diagram = linkage_print_diagram(linkage));`

`string_delete(diagram);`

`linkage_delete(linkage);`

Extract the first linkage indexed by 0 in the list, prints the linkage diagram and then deletes linkage and the string allocated for the diagram. After each of the input strings is processed, and for memory management purposes, the program deletes the Dictionary and Parse_Options with the statements:

`dictionary_delete(dict); parse_options_delete(opts);`

The program output is illustrated in the following figure 11.
Figure: 11. Link Grammar Program Output

Two main Link Grammar API functions are:

char ** linkage_get_words(Linkage linkage)

char * linkage_get_word(Linkage linkage, int w)

cchar ** linkage_get_words(Linkage linkage); takes a linkage as an argument and returns the array of word spellings of the current linkage. Ex. Computers.n, are.v…etc.

char * linkage_get_word(Linkage linkage, int w); returns one linkage word at a time based on the index w. These words can be used by other NLP applications components such as lexicons.

4.2. Parser evaluation methods:

Parsers play a very crucial role in Natural Language processing applications performance. Therefore, smart developers should always evaluate and test different kinds of parsers before integrating them into their applications. Parsers are either dependency based or constituency-based systems. Constituency based parsers produce a hierarchically organized constituent structure while dependency based parsers produce linkages.
between words of sentences [sri]. Parsers are potentially important in every natural
language processing system. The parser quality mostly reflects the system performance.
Parsers and choosing the right one depends on the evaluation method chosen. According
to [sri], three evaluation methods are available.

☐ Intrinsic

☐ Extrinsic

☐ Comparative

**Intrinsic evaluation:**

It measures the performance of a parser in the context of the framework in which it is
developed [sri]. This approach can be used for both grammar based and statistical parsers. In
grammar-based parsers, intrinsic evaluation checks the grammar vulnerabilities. In statistical
parsers, the underlying statistical model performance is measured. Intrinsic evaluation is divided
into two approaches, test suite based and corpus based evaluations. Corpus based evaluation is
subdivided to annotated and un-annotated corpus based evaluation methods. Intrinsic evaluation
also uses the *test suite based* approach. It gives good direction on the improvement of statistical
parsers. This approach is not suitable for unrestricted text data.

Intrinsic method also uses two additional approaches: un-annotated corpus-based and
annotated corpus-based. The first approach uses unrestricted text data as corpora in order to
evaluate parsers. The sentences in corpora are not annotated with any linguistic information. The
annotated corpus-based approach uses unrestricted text data formed of sentences that are annotated with information such as part of speech tags, subject-verb-object triples…etc [43]. This methodology is very efficient in evaluating the parser tagging as well as parsing accuracies. One can measure the exact match between the structure output by the parser and that of the corpus annotation.

**Extrinsic evaluation:**

Meanly measures the parser adequacy. In other words, it is more suitable for developers who want to integrate a parser into their systems. By using the extrinsic evaluation, the developer can see how the parser contributes to the all system performance. This method provides an indirect parser comparison too.

**Comparative evaluation:**

Comparative evaluation is appropriate for comparing parsers that use different grammar or statistical models. This method uses the PARSEVAL metric that proved many limitations such as crossing brackets penalization to mis-attachments.

In the model proposed by Banglore, Sarkar, Doran and Hockey, a parser is viewed in multiple dimensions where each dimension represents a relation R. The performance of the parser is measured in terms of precision (P), recall (R) and F-measure.
- Let $x R y$ represent that $x$ is in a relation $R$ with respect to $y$.

- Let $S_{gold}$ be the relation, $R$, expressed in the key (annotated corpus).

- Let $S_{out}$ be the relation, $R$, expressed in the output of the parser.

- Recall $= |S_{gold} \cap S_{out}|/|S_{gold}|$

- Precision $= |S_{gold} \cap S_{out}|/|S_{out}|$

- $F$-Measure $= (\beta^2 + 1) \times P \times R/(\beta^2 \times P + R)$ where $\beta$ is the relative importance of Recall and Precision.

*Figure: 12. The relation model for parser evaluation summary [44]*

Other evaluation method, that we would consider, is summarized bellow [44]:

- **Listing Constructions:** Consists of simply listing covered constructions that may not be covered by the evaluated parser.

- **Coverage (Un-annotated corpus):** This includes the calculation of a percentage of examples from a given corpus, which are assigned one or more global analyses by a parser.
- **Average parse base (Unannotated corpus):** The evaluation is based on computing the geometric mean of the analyses divided by the number of input tokens in each parsed sentence. This produces the average number of parses for a sentence of n words.

- **Tagging Accuracy: (Annotated corpus):** The evaluation is done by calculating the accuracy with which a parser assigns the correct lexical syntactic category to a word running in text. Precision / recall are the best measure.

- **Tree similarity measure (annotated corpus):** Evaluation is performed through the calculation of the ratio of rules correctly deployed in a derivation to all rules in that derivation computed from an annotated corpus.

4.3. **Lexicons (OxiNet):**

Organized into synonym sets (synsets) such as {board, plank}. These synonym sets do not explain what the concepts are; they only explain that there is a concept that exists. According to [Miller], WordNet is organized by semantic relations. Since a semantic relation is a relation between meanings, and since meanings can be represented by synsets, it is natural to think of semantic relations as pointers between synsets. There are four main divisions in WordNet: Adverbs, nouns, verbs and adjectives. WordNet has many semantic relations that the developer should focus on when developing natural language processing systems. Each WordNet division has its own relations that relate the division corresponding synsets. Following are the main WordNet relations.
Synonymy

Two expressions are synonymous if the substitution of one for the other never changes the truth value of a sentence in which the substitution is made. For example, “the substitution of plank for board will seldom alter truth values in carpentry contexts.” [45].

Antonymy

The antonym of a word $x$ is sometimes $not$-$x$, but not always. Ex. Rich and poor are antonyms. But it’s not rich doesn’t mean poor. “Antonymy provides a central organizing principle for the adjectives and adverbs in WordNet, and the complications that arise from the fact that antonymy are a semantic relation between words is better discussed in that context.” [Miller].

Hyponymy/Hypernymy (IS-A relation)

It’s also named as subset or superset relation. For example, an apple is a hyponym of a tree. A hyponym always inherits the features of its hypernym

Meronymy/Holonomy

“A concept represented by the synset \{x, ..., \} is a meronym of a concept represented by the synset \{y, ..., \} if native speakers of English accept sentences constructed from such frames as A $y$ has an $x$ (as a part) or An $x$ is a part of $y$. The meronymic relation is transitive (with qualifications) and asymmetrical (Cruse, 1986), and can be used to construct a part hierarchy” [Miller].
The following figure illustrates how WordNet semantic relations are represented.

![Diagram of semantic relations among lexical concepts](image)

**Figure: 13. Representation of three semantic relations among a variety of lexical concepts**

**Entailment**

Entailment is a semantic relation that relates verb synsets. “A proposition \( P \) entails a proposition \( Q \) if and only if there is no conceivable state of affairs that could make \( P \) true and \( Q \) false.” [45]. An example of entailment is the following: *snore* lexically entails *sleep* because the sentence *He is snoring* entails *He is sleeping*; the second sentence necessarily holds if the first one does.
5. AN NLP APPLICATION OF OXINET:

5.1. OMTrans (English to Oriya Translation System): [47]

5.1.1. Architecture:

The architecture of OMTrans System is divided into various parts (Figure 2) such as Morphological Parser, POS Tagger, Translator, Disambiguator etc. and some Software tools used to see the result.

Figure: 14. Architecture of OMTrans System
The OMTTrans system is a collection of programs, which takes the help of other modules to perform the task of effective translation (i.e. English to Oriya). The heart of the OMTTrans system is the OMTTrans database (bilingual dictionary along with some ontological information). In this database we have stored different information, such as: English word, category, tense, Oriya meaning of the word etc. The Oriya meaning of the words has been stored in simple ISCII format. The system is developed using Java programming language and MySQL as the database.

5.1.2. Parser (POS Tagger):

The parser takes the English sentence as input from the OMTTrans system then each and every sentence is parsed according to various rules of parsing with the help of OMTTrans database. The rules are like following:

S -> NP VP

NP -> Art Noun

VP -> V NP

NP -> Noun etc,

During parsing in some cases it is taking the help of morphological analyzer (Ex: for words like boys morphological analyzer will convert it to boy). For the words having more than one category is tagged properly by the help of rules.

Example:

I have a book. (book will be tagged as noun)

I will book ticket for him. (book will be tagged as verb)

I will light the light. (first light will be tagged as verb & second light will be tagged as noun)
Finally the parsed structures of sentences are returned to the translator for translation. For this we have used finite-state POS tagger.

**Figure: 15. Typical FSA for POS tagging for words having more than one Category**

### 5.1.3. Translator:

The translator takes the parsed structure of sentences as input then it will perform the task of translation with the help of OMTrans database. This translator part is taking the help of tense analysis and verb tagging module.

Ex:

I go to school.

\(\text{cÊÝ dPaD LÊ aPf} \text{ÔĐf D} \text{ (retrieved from database). According to grammar it will be rearranged but to convert the verb dPaD to dĐH, which is appropriate as per tense analysis, for it verb tagging is done.}\)
5.1.4. **Word Sense Disambiguation:**

Word sense disambiguation is a major problem in MT. The disambiguator module is performing the task of disambiguation with the help of frequencies of bigrams, trigrams etc. obtained from Corpora (Oriya Corpora) by using the n-gram model [35].

Our method of disambiguation which is an unsupervised one depends on the value of log likelihood ratio.

Example:

\[
\text{Pr (Sense}_i \text{/ Collocation}_i, \text{Collocation}_j) \\
\text{Log} \\
\text{Pr (Sense}_j \text{/ Collocation}_i, \text{Collocation}_j)
\]

Where

Collocation \(_i\) = combination of nearby words along with the word having Sense \(_i\).

Collocation \(_j\) = combination of nearby words along with the word having Sense \(_j\).

We are getting the various meaning according to senses from the bilingual dictionary during translation. So, the sentences of the target language, which is not the final one, contain various meanings of the ambiguous words. The collocations are identified from the ambiguited translation in the target language. For calculating the probabilities the frequencies are derived from the target language corpus. If the log likelihood ratio is evaluated as positive Sense\(_i\) is preferred otherwise in case of negative Sense\(_j\) is preferred. In the worst case if it is evaluated as zero arbitrarily the sense has to be taken which will give the final translation.
5.1.5. **Object Oriented Design:**

The object-oriented paradigm is a set of theories, standards and methods that represent a way of organizing knowledge. Action takes place when a message is sent to an object; the object interprets the message and then performs some method in response to the message. An object is an instance of a class and if the same message passed to multiple objects from the same class, they perform the same task. This notion of objects and classes is the basic building block of object-oriented programming. With the brief discussion about the object oriented paradigm the following paragraph will describe an object oriented approach to OMTTrans system. The different classes used in the system are given below.

*OMTrans class*

*Splittext class*

*Databaselink class*

*Parse class*

*Morphology class*

*Tense class*

*Translation class*

*Verbtagging class*

*Disambiguation class etc.*

The OMTTrans class is the main class and it invokes the object of other classes to perform its task. The necessary methods with their arguments of various classes are designed according to
the rules of translation. The translation class with their own methods as well as methods of other class but used in it, description and example are shown in Table 1.

<table>
<thead>
<tr>
<th>Methods with Arguments</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>String findwcat (String word)</td>
<td>Takes an argument i.e. a word of a sentence and finds its category.</td>
<td>If the input is “go” output is verb.</td>
</tr>
<tr>
<td>String findoword (String word)</td>
<td>Takes an argument i.e. a word of a sentence and finds its corresponding Oriya meaning.</td>
<td>If the input is “go” output is dPaDa (JibA) verb.</td>
</tr>
<tr>
<td>String morphology (String word)</td>
<td>Takes an argument i.e. a word of a sentence and finds its root word.</td>
<td>If the input is “going” output is go.</td>
</tr>
<tr>
<td>String tense (String aux, String verb)</td>
<td>Takes two arguments (i.e. auxiliary verb and main verb) and finds the tense of the sentence.</td>
<td>If the input is null string (no auxiliary verb) and go then output is present indefinite.</td>
</tr>
<tr>
<td>String tagfeature (String word)</td>
<td>Takes an argument i.e. a word of a sentence and finds its tag attached to root word.</td>
<td>If the input is “going” then output is “ing”.</td>
</tr>
</tbody>
</table>

*Table: 1. Methods Used in Translation class*
### 5.1.6. Implementation:

We have implemented the OMTrans system for various types of simple as well as complex sentences. It has been designed in a fully extensible manner i.e. we can add/modify new rules for translation with the existing system [46]. In the first phase the system is tested with various types of sentences taken from schoolbooks. One of the vital features of our system is that, it is capable of translating the scanned files from the books as well as newspapers. The system has the capability of doing word sense disambiguation along with the help of frequencies extracted from corpora. For word sense disambiguation testing have been made for various words, it is giving very accurate result according to the contexts. In the translation browser the standard X-windows mouse function are used to open, close, move the window and change its size. The translation browser provides the facility that one can enter the text for translation in the browser or he can get the input text from the previously stored file by using the read file button. The translation browser along with the outputs is shown in Figure 16. The final output after disambiguation is shown in Figure 17.
Figure: 16. Output of Translation Browser
Figure 17. Disambiguated Output
6. **CONCLUSION:**

The impact of a breakthrough in computer use of natural languages will have as profound an effect on society as would breakthroughs in superconductors, inexpensive fusion, or genetic engineering. The impact of NLP by machine will be even greater than the impact of microprocessor technology in the last 20 years. The rationale is simple: natural language is fundamental to almost all business, military, and social activities; therefore, the applicability of NLP is almost limitless. NL analysis and generation could revolutionize our individual, institutional, and national ability to enter access, summarize, and translate textual reformation. It can make interaction with machines as easy as interaction between individuals. The computer's linguistic proficiency may never be as great as a human's. However, the existence and use of current NL products and the market projections cited suggest that investment in this technology should lead to useful spinoffs in the near term and mid-term. The technology stands at a turning point. New approaches offer opportunities for substantial progress in the next five years, and breakthroughs within 3 to 10 years.
7. **REFERENCES:**

[1] Speech and Language Processing: An introduction to NLP computational linguistics and speech recognition. 2nd edition by Daniel Jurafsky and James H. Martin


[6] Bar Hillel and Machine Translation: Then and Now by Sergei Nirenburg, Computing Research Laboratory, New Mexico State University


[10] Rule – Based expert systems – Artificial Intelligence and intelligent systems, November 2005, Prof. Henning Christiansen


[23]  Quake source code released under the GPL, links for engine and game code:


[38] Semantic Indexing using WordNet Senses, Rada Mihalcea and Dan Moldovan Department of Computer Science and Engineering Southern Methodist University


