IMPROVED HANDLING OF SMS MESSAGES WITH STATISTICAL NATURAL LANGUAGE PROCESSING TECHNIQUES

by

Ömer Yıldırım

B.S., Computer Engineering, Galatasaray University, 2000

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Systems and Control Engineering

Boğaziçi University

2005

ACKNOWLEDGEMENTS

I would like to thank my supervisors, Prof. Cem Say and Assoc.Prof. Levent Arslan, for pushing me back into the right direction when things got a bit out of hand. I would have been lost without their valuable comments.

ABSTRACT

IMPROVED HANDLING OF SMS MESSAGES WITH STATISTICAL NATURAL LANGUAGE PROCESSING TECHNIQUES

The Short Messaging Service (SMS) is built on the ability of mobile telephones to send and receive text messages. SMS based applications are increasing dramatically day by day in the telecommunications industry. The most common use of SMS is for notifying mobile phone users that they have new voice or fax mail messages waiting. Whenever a new message is dispatched into the mailbox, an alert by SMS informs the user of this fact. The Short Message Service can also be used to deliver a wide range of information to mobile phone users from share prices, match scores, weather, flight information, news headlines, lottery results, jokes. In general, user interaction based SMS services request some predefined keywords from the users and respond to them after processing their messages.

However, most users think that they are communicating not with a machine but with humans, so they compose misspelled and/or machine specific messages containing more than just the needed keywords. As a result, they receive error messages from the server and generally do not continue to use the software after trying two or three times by making same mistakes.

In this thesis, I introduce a new Short Message Service (SMS) parsing model using Statistical NLP Techniques, whose aim is to solve the existing SMS user subscription problem of a real software company. To do this, the N-Gram statistical approach will be used.

ÖZET

SMS MESAJLARININ İSTATİSTİKSEL DOĞAL DİL İŞLEME YÖNTEMLERİ KULLANILARAK ANLAMLANDIRILMASI

Günümüzde mobil telefonların metin tipindeki mesajları kabul edip gönderebilmelerini sağlayan SMS (Kısa Mesaj Servisi) son kullanıcılar arasında oldukça yoğun bir biçimde kullanılmaktadır. SMS protokolünün 160 karakterlik limiti (Unicode karakterler için bu limit mesaj başına 70 karaktere düşmektedir), HTML, XML gibi herhangi bir özel formatı olmadan sadece düz metinlerden kurulu olmasına rağmen günümüzde kısa mesaj servislerinin sayısı telekomünikasyon sektöründe her geçen gün artış göstermektedir.

Telekomünikasyon şirketlerinin spor, haber, hava durumu gibi çeşitli içerik hizmetlerinin sağlanmasında bu yönteme sıkça başvurdukları görülmektedir. Günümüzde bu çeşit kısa mesaj servisiyle verilen bir çok servis bulunmakta, bunların abonelik, iptal ve servis içeriğinin türüne göre gereken bazı parametreleri yine SMS protokolü ile son kullanıcılardan toplanmaktadır. Bu servislerin abonelik işlemlerinde kullanıcılardan daha önceden belirlenmiş bir anahtar kelime yada kelimeler istenmekte buna göre son kullanıcıların istekleri belirlenip arzu ettikleri hizmet kendilerine verilmektedir. Ancak kullanıcıların bir çoğu gönderdikleri mesajların karşıda bir insan tarafından okunduğunu düşünmekte ve çoğu zaman kendilerinden istenen örneğin önceden belirlenmiş "ABONE HABER NTV" yerine "ABONE HBR MTV" gibi mesajlar göndererek sadece gelen anahtar kelimeleri işlemeye göre programlanmış yazılımların hatalı yanıtlar vermesine yol açmaktadırlar. Üst üste başarısız bir iki denemeden sonra, bu tür yanıtlarla devamlı hata mesajını yanıt olarak alan son kullanıcılar da servis almaktan vazgeçmekte, bu da ilgili içerik sağlayıcının hem gelir kaybetmesine neden olmakta hem de müşteri memnuniyetini olumsuz yönde etkilemektedir.

Bu tezde istatistiksel doğal dil işleme yöntemlerinin başında gelen N-Gram yöntemiyle bu probleme bir çözüm yöntemi getirilmeye çalışılarak, yeni bir SMS işlemi modülü geliştirilecek ve bu modülün son kullanıcıları olan gerçek bir abonelik sistemi üzerinde çalıştırılmasıyla, yöntem ve sonuçları tartışılacaktır.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS / ABBREVIATIONS

Telco	Telecommunications company
SMS	Short Message Service
CSPS	Corporate Services Providing Subscriber System.
UMS	User Management System.
SIS	Subscription information system
PDO	Product data object
MLE	Maximum likelihood estimation
TH	Threshold value
UML	Unified modeling language
B2B	Business to business
B2C	Business to customer
GUI	Graphical user interface
RDBMS	Relational Data Base Management System
SMSC	Short Message Service Center.

1. INTRODUCTION

Besides being text based, the Short Messaging Service (SMS) has some well known limitations. The length limit of short messages is 160 characters when ASCII (e.g. English) alphabets are used and 70 characters when Unicode alphabets such as Turkish, Arabic and Chinese are used.

There are many elements that need to be taken into account when developing and deploying SMS. Essentially, any information that fits into a short message can be delivered by SMS. Therefore, the most difficult part of SMS based applications is the fact that customers can send any kind of messages without regard to some predefined key-based structures like "*ABONE HABER NTV*", "*IPTAL TRIBUN*", etc.

However, most users compose complicated messages containing more than just the needed keywords. As a result, they receive error messages from the server. There are also some errors that are not dealing with the users' faults like machine specific error that will be discussed in the following chapters.

In this thesis, a solution of this problem will be studied. To do this, a real world SMS based application server will be used as a platform. So our solution will be tested against the real persons not only in some simulation environment with hypothetically generated random text messages.

Firstly, this application server (Turtle SMS Handling System) will be studied with its infrastructures and some special features built on it. After that, some historical background of statistical natural language processing methods will be discussed. You can find some useful information about statistical methods especially N-Gram method which is used for the solution in chapter 2.

In chapter 3, our approach for this problem will be examined and its results, evaluations and conclusions will be presented in chapter 4.

Finally, you can find more details of the Turtle System and SMS Handling System in Appendix B.

2. BACKGROUND

Famous quotes:

- The notion “probability of a sentence" is an entirely useless one. . . [Noam Chomsky 1969] [1]

- Anytime a linguist leaves the group, the recognition rate goes up. [Fred Jelinek 1988] [1]

In fact, the use of probability in linguistic theory has been under discussion for many years.

There are two main approaches for Natural Language Processing: rule based and statistics-oriented approaches. In the rule-based approach, the expected input sentences are often modeled by a strict grammar. In this case, the user is only allowed to utter those sentences, which are explicitly covered by the (often hand-written) grammar [2].

Rule-based approaches, with rules induced by human experts, had been the dominant paradigm in the natural language processing community. Such approaches, however, suffer from serious difficulties in knowledge acquisition in terms of cost and consistency. Therefore, it is very difficult for such systems to be scaled up.

Statistics-oriented approaches are now dominant in natural language processing, and are gaining ground in theoretical linguistics. Since there is no robust “theory of everything", probabilities serve as a useful approximation of the world.

2.1. Overview of statistical NLP techniques

Abney [6] describes the ultimate goal of linguistics as understanding language. Traditional linguistics study firstly tries to describe a grammar for the language which will be studied. Thus, sentences that can be generated by the grammar are defined as grammatical. Other sentences are regarded as ungrammatical, that is, they are not acceptable according to the language's grammar.

This grammaticality of sentences is binary. Either a sentence is grammatically correct or not. Thus, the sentences which will be considered as grammatical are determined only according to whether they are well formed or not. This grammaticality however does not include the sentences which are semantically correct or the type of things people would practically say.

The traditional linguistics approach may work for 'simple' cases, but becomes harder for many real-world examples that are much more complex in structure. This kind of classification also does not provide any information about the frequency with which different sentence types and sentences are used. The structure and use of language also changes over time. For example some definitions of grammaticality that may be true at the time of study little by little become false over time.

Therefore, to help with this grammar categorization and changes in language, frequencies of use and statistical measures of words within a language can be obtained and analyzed. A major part of S-NLP (statistical NLP) is determining how to model the language by deriving good probability estimates for unseen events, such as new words appearing in previously unseen text. Although it may be harder to think about how semantics in S-NLP can be described, one way can be thinking about the distribution of contexts over which words are used.

For example, S-NLP disambiguation automatically learns lexical and structural knowledge from corpora (a collection of texts) by determining statistically which words have a tendency to group together.

Statistical NLP looks at common patterns in text using statistics (i.e. counting things) and probability (i.e. predicting things). Building a statistical model of the language can provide a solution for many natural language tasks, for example parsing texts, word-sense disambiguation and information retrieval.

Work in S-NLP comes from Shannon's ideas [3] of assigning probabilities to linguistic events. This is opposed to Chomsky's [3] formal language theory. Thus S-NLP approaches help enable linguists to say which sentences are 'usual' and 'unusual'.

2.1.1. Probability of a Word String $w_1 \ldots w_n$ and the Markov Assumption

If we consider each word occurring in its correct location as an independent event, we can represent this probability as follows: $P(w_1, w_2, w_3 ..., w_{n-1}, w_n)$

We can use the chain rule to decompose this probability:

$$P(w_1^n) = P(w_1)P(w_2 \mid w_1)P(w_3 \mid w_1^2)...P(w_n \mid w_1^{n-1}) = \prod_{k=1}^{n} P(w_k \mid w_1^{k-1}) \qquad (2.1)$$

The problem with the (Equation 2.1) is we do not know any easy way to compute the probabilities like $P(w_n \mid w_1^{n-1})$. For example, we can not just count the number of times every word occurs following every long string. We need a very big corpus for that.

This problem is solved by making a useful simplification: We can approximate the probability of a word given all the previous words. The approximation we will use is very simple: the probability of the word given the single previous word!

In other words, instead of computing the probability
P(Istanbul | ABONE HABER NTV) we approximate it with the probability
*P(Istanbul | NTV).*Thus our assumption can be formulated as (Equation 2.2).

$$P(w_n \mid w_1^{n-1}) = P(w_n \mid w_{n-1}) \tag{2.2}$$

So

$$P(w_1^n) = \prod_{k=1}^{n} P(w_k \mid w_{k-1}) \tag{2.3}$$

This assumption that the probability of a word depends only on the previous word is called a *Markov* assumption. Thus, with Markov models we can predict the probability of some future units without looking too far into the past.

2.2. The N–Gram approach

The N-gram method was first proposed by Markov (1913)[1] in his studies that are now called "Markov chains" (bigrams and trigrams) to predict whether an upcoming letter in Pushkin's Eugene Onegin would be a vowel or a consonant.

Markov classified 20,000 letters as Vowel or Consonant and computed the bigram and trigram probability that a given letter would be a vowel given the previous one or two letters. Shannon (1948)[1] applied N-grams to compute approximations to English word sequences. Based on Shannon's work, Markov Models were commonly used in modeling word sequences by the 1950s.

Today an N-Gram grammar is defined as a representation of an Nth order Markov language model in which the probability of occurrence of a symbol is conditioned upon the prior occurrence of N-1 other symbols.

N-Gram grammars are typically constructed from statistics obtained from a large corpus of text using the co-occurrences of words in the corpus to determine word sequence probabilities. N-Gram grammars have the advantage of being able to cover a much larger language than would normally be derived directly from a corpus. Open vocabulary applications are easily supported with N-Gram grammars.

2.2.1. Why N-Grams?

One of the main problems in the speech recognition, handwriting recognition, augmentative communication for the disabled and spelling error detection studies is finding the next word (or character) from given words (or characters). In such tasks, word-identification is difficult because the input is very noisy and ambiguous.

In simple speech recognition/speech understanding systems, the expected input sentences are often modeled by a strict grammar. In this case, the user is only allowed to complete those sentences, which are explicitly covered by the (often hand-written) grammar.

Experience shows that a context free grammar with reasonable complexity can never predict all the different sentence patterns that users come up with in spontaneous input. This approach is therefore not sufficient for robust speech recognition/understanding tasks or free text input applications such as dictation.

Thus, looking at previous words can give us an important indication of the next one that we are trying to guess. Imagine the given word of a sentence is like following: "*Bugün hava çok gezel.*"

The word "gezel" is definitely not a Turkish word. For a human being, it is easy to work out this problem. Because by our knowledge of word sequences in Turkish and by experience we can predict that correct form of this sentence should be "*Bugün hava çok güzel.*" Especially if we have known the context, it is much easier to guess the next word. One another interesting study is made by Russell and Norvig[1] in which they give an example from Woody Allen's "Take the Money and Run" movie. In the hold-up scene a bank teller interprets Woody Allen's sloppily written hold-up not as saying "I have a gub". A speech recognition system (and a person) can avoid this problem by saying that "a gub" is not an English word sequence and especially in the context of a hold-up, "I have a gun" will have a much higher probability than "I have a gub" or even "I have a gull".

Consider the problem [1] of detecting real-world spelling errors. These are spelling errors that result in real English words (although not the ones the writer intended) and so detecting them is difficult (we can not find them by just looking for words that are not in the dictionary). Table 2.1. contains some examples.

Table 2.1. Some attested real world spelling errors from Kukich (1992) [5]

Example phases
They are leaving in about fifteen *minuets* to go to her house.
The study was conducted mainly *be* John Black.
The design *an* construction of the system will take more than a year.
Hopefully, all *with* continue smoothly in my absence.
Can they *lave* him my messages?
I need to *notified* the bank of [this problem.]
He is trying to *fine* out.

For example, while the phrase *in about fifteen minuets* is perfectly grammatical English, it is a very unlikely combination of words. Spell checkers can look for low probability combinations like this.

N-Gram language models are traditionally used in large vocabulary speech recognition systems to provide the recognizer with an a-priori likelihood P(W) of a given word sequence W. The N-Gram language model is usually derived from large training texts that share the same language characteristics as expected input.

N-Gram language models rely on the likelihood of sequences of words, such as word pairs (in the case of bigrams) or word triples (in the case of trigrams) and are therefore less restrictive. The use of stochastic N-Gram models has a long and successful history in the research community and is now more and more effecting commercial systems, as the market asks for more robust and flexible solutions.

2.2.2. Simple (unsmoothed) N-Grams

N-Gram analysis is based on the probability formula:

$$P\ (W_n \mid W_1 .....W_{n-1}) = \frac{P(W_1 ....W_n)}{P(W_1 ....W_{n-1})} \tag{2.4}$$

where W is the word string. So $P\ (W_n \mid W_1 .....W_{n-1})$ is the probability of occurrence of W_n given that $W_1 .....W_{n-1}$ sequence has occurred.

N-Gram probabilities can be computed by simply counting in a corpus and normalizing (the Maximum Likelihood Estimate) or they can be computed by more sophisticated algorithms.

The most common N-Gram language models are “unigram” , ”bigram” and “trigram” models that depend on the Nth order Markov model where N=1,2 and 3, respectively.

2.2.2.1. Unigram language models

The simplest possible model of word sequences would simply let any word of the language follow any other word. In the probabilistic version of this theory, then, every word would have an equal probability of following every other word. If English had 100000 words, the probability of any word following any other word would be 1/100000 or 0.00001.

In a slightly more complex model of word sequences, any word could follow any other word, but the following word would appear with its normal frequency of occurrence. For example, the word *the* has a high relative frequency, it occurs 69971 times in the Brown corpus of 1.000.000 words (i.e. , 7 per cent of the words in this particular corpus are *the*). [1] By contrast the word *rabbit* occurs only 11 times in the Brown corpus.

We can use these relative frequencies to assign a probability distribution across following words. So if we have just seen the string *Anyhow*, we can use the probability 0.07 for *the* and 0.00001 for *rabbit* to guess the next word.

Probabilities from a corpus are calculated by counting words in a large corpus (body) of text. If C(w) is the number of times word w occurs in a corpus of N words, then we can simply use maximum likelihood estimation (MLE) to calculate P(w) as follows:

$$P(w) = C(w)/N$$

In terms of the N-Gram jargon, instead of word, we say unigram, meaning "word sequence of length 1".

So we constructed a unigram language model using maximum likelihood estimation. In Table 2.2., there is an example of unigram probabilities.

Table 2.2. Unigram example [3]

P(w)	**Value**
P(a)	0.0368
P(aardvak)	0.0001
P(aback)	0.0005
P(abacus)	0.0001
P(abandon)	0.0011
P(abide)	0.0003

2.2.2.2. Bigram language models

Bigram language models use conditional probabilities to predict what the next word will be, given the previous word. Consider the example given for the unigram language models. Suppose we have just seen the following string: "Bugün hava çok".

In this context, "*güzel*" seems like a more reasonable word to follow white than "*ve*" does. This suggests that instead of just looking at the individual relative frequencies of words, we should look at the conditional probability of a word given the previous words.

That is, the probability of seeing "*güzel*"given that we just saw "*çok*" (which will represent as P(*güzel*|çok)) is higher than the probability of "*güzel*" otherwise.

Example from the Berkeley Restaurant Project [1]:

The Berkeley Restaurant Project is a speech based restaurant consultant; the user asks questions about restaurant in Berkeley, California, and system displays appropriate information from a database of local restaurants (Jurafsky 1994)[1]. Here are some sample queries:

I'm looking for Cantonese food.
I'd like to eat dinner someplace nearby.
Tell me about Chez Panisse.
Can you give me a listing of the kinds of food that are available?
I'm looking for a good place to eat breakfast.
I definitely do not want to have cheap Chinese food.
When is Caffe Venezia open during the day?
I don't wanna walk more than ten minutes.

Table 2.3. Berkeley restaurant project

P(w)	**Value**	**P(w)**	**Value**
eat on	0.16	eat Thai	0.03
eat some	0.06	eat Breakfast	0.03
eat lunch	0.06	eat in	0.02
eat dinner	0.05	eat Chinese	0.02
eat at	0.04	eat Mexican	0.02
eat at	0.04	eat tomorrow	0.01
eat Indian	0.04	eat dessert	0.007
eat today	0.03	eat British	0.001

In Table 2.3. you can see a fragment of a bigram grammar from the Berkeley restaurant project showing the most likely words to follow "*eat*".

For example, P (on | eat) = 0.16
P (British | eat) = 0.001

2.2.2.3. N-Gram language models

As mentioned previously, an n-gram language model uses the previous n - 1 words to predict the next one. The number of probability numbers (parameters) required for an N-Gram model increases exponentially with n, so in practice “n” never goes beyond trigram models (n=3).

For example, assume a 20,000-word vocabulary

- A unigram model requires calculating 20,000 numbers
- A bigram model requires 20.000x20.000 = 400 million numbers
- A trigram model requires 20.000x20.000x20.000 = 8 trillion numbers
- A 4-gram model requires $1.6x10^{17}$ and so on…

Computing these parameters for a particular corpus is called training the language model on that corpus.To clarify this, let us calculate the Probability of “I want to eat British food" [1] from Berkeley restaurant project that we mentioned before.

Table 2.4. Bigram values of restaurant project

P(w)	Value	P(w)	Value
eat on	0.16	eat Thai	0.03
eat some	0.06	eat Breakfast	0.03
eat dinner	0.05	eat Chinese	0.02
eat at	0.04	eat Mexican	0.02
eat at	0.04	eat tomorrow	0.01
eat Indian	0.04	eat dessert	0.007
eat today	0.03	eat British	0.001

Assume that in addition to the probabilities in Table 2.4. , our grammar also includes the bigram probabilities in Table 2.5. with “<s>” special word meaning “Start of sentence”.

Table 2.5. More fragments from the bigram grammar from the restaurant project

P(w)	Value	P(w)	Value	P(w)	Value	P(w)	Value	P(w)	Value
<s>I	0.25	I want	0.32	want to	0.65	to eat	0.26	British food	0.60
<s>I'd	0.06	I would	0.29	want a	0.05	to have	0.14	British restaurant	0.15
<s>Tell	00.04	I don't	0.08	want some	0.04	to spend	0.09	British cuisine	0.01
<s>I'm	0.02	I have	0.04	want Thai	0.01	to be	0.02	British lunch	0.01

Now we can calculate probabilities of sentences like “I want to eat British food” or “I want to eat Chinese food” by simply multiplying the appropriate bigram probabilities together, as follows:

P(I want to eat British food) =P(I | <s>) x P(want | I) x P(to | want) P(eat | to)x
P(British | eat) x P(food | British)
=0.25 x 0.32 x 0.65 x 0.26 x 0.001 x 0.60
= 0.0000081

2.2.2.4. Smoothing

In the N-gram approach, some word sequence probabilities could be zero. This is too strict, because there are many perfectly good n-grams that just happen not to be in the corpus. If this occurs, then a smoothing algorithm must be used to correct these cases.

“Smoothing” is assigning new (small but non-zero) probability values to the cases which seem to have zero probability. Smoothing is also called discounting because the probabilities of the non-zero-probability n-grams are discounted a certain amount, and this amount is redistributed among the zero probability ones.

2.2.2.5. Add-one smoothing

Add-one (Laplace) smoothing adds 1 to each count, then normalizes by adding the vocabulary size V to the denominator

For unigrams:

$$P^{'}(w) = \frac{C(w)+1}{N+V} \tag{2.4}$$

For n-grams:

$$P^{'}(w_n \mid w_1 ... w_{n-1}) = \frac{C(w_1 ... w_n)+1}{C(w_1 ... w_{n-1})+V} \tag{2.5}$$

In order to make clear the (Equation 2.5), let us smooth Berkeley Restaurant Project bigram that we mentioned before.

Table 2.6. shows add one smoothed bigram counts for seven of the words (out of 1616 total word types) in Berkeley restaurant project corpus of ~10.000 sentences.

Table 2.6. Add one smoothed bigram

	I	**want**	**to**	**eat**	**Chinese**	**food**	**lunch**
I	9	1088	1	14	1	1	1
want	4	1	787	1	7	9	7
To	4	1	11	861	4	1	13
Eat	1	1	3	1	20	3	53
Chinese	3	1	1	1	1	121	2
Food	20	1	18	1	1	1	1
Lunch	5	1	1	1	1	2	1

Recall that normal bigram probabilities are computed by normalizing each row of counts by the unigram count:

$$P^{'}(w_n \mid w_{n-1}) = \frac{C(w_{n-1}w_n)}{C(w_{n-1})} \tag{2.6}$$

For add-one-smoothed bigram counts we need to first augment the unigram count by the number of total word types in the vocabulary V:

$$P^{'}(w_n \mid w_{n-1}) = \frac{C(w_{n-1}w_n)+1}{C(w_{n-1})+V} \quad (2.7)$$

We need to add V (that is, 1616) to each of the unigram counts. Add one smoothed unigram counts are showed in Table 2.7.

Table 2.7. Add one smoothed unigram counts

Word	**Counts**
I	3437+1616=5053
want	1215+1616=2931
to	3256+1616=2931
eat	938+1616=2554
Chinese	213+1616=1829
food	1506+1616=3122
lunch	459+1616=2075

Finally, the result is the smoothed bigram probabilities as shown in Table 2.8.

Table 2.8. Result of add one smoothed bigram

	I	**want**	**to**	**eat**	**Chinese**	**food**	**lunch**
I	0.0018	0.22	0.00020	0.0028	0.00020	0.00020	0.00020
want	0.0014	0.00035	0.28	0.00035	0.0025	0.0032	0.0025
to	0.00082	0.00021	0.0023	0.18	0.00082	0.00021	0.0027
eat	0.00039	0.00039	0.0012	0.00039	0.0078	0.0012	0.021
Chinese	0.0016	0.00055	0.00055	0.00055	0.00055	0.066	0.0011
food	0.0064	0.00032	0.0058	0.00032	0.00032	0.00032	0.00032
lunch	0.0024	0.00048	0.00048	0.00048	0.00048	0.00096	0.00048

3. OUR APPROACH

3.1. Turtle SMS Handling System

Turtle is designed especially for the Telecommunication industry. Turtle can be evaluated as a kind of portal system, which contains many modules including “SMS User Subscription System” that we will examine in this thesis. You can examine Appendix A to get more details about the Turtle system.

The SMS Handling System of Turtle is designed for the short message based user subscriptions. It has a message parsing logic working with simple and manually added “Variant Keys”. There is also special chapter about the SMS Handling system and its features in the Appendix.

For instance, we will consider Turtle as a black box and we will focus only “SMS Subscription System” of the Turtle platform.

3.1.1. Campaign Package Service Scenario for SMS Subscription Flow

SMS Subscription flow starts with the user sending an SMS message containing predefined and announced keywords to Subscription Service Short number. For instance, subscription operations start with an SMS of the form “*ABONE XXX*”.

The step-by-step subscription flow may be summarized as follows (Figure 3.1.):

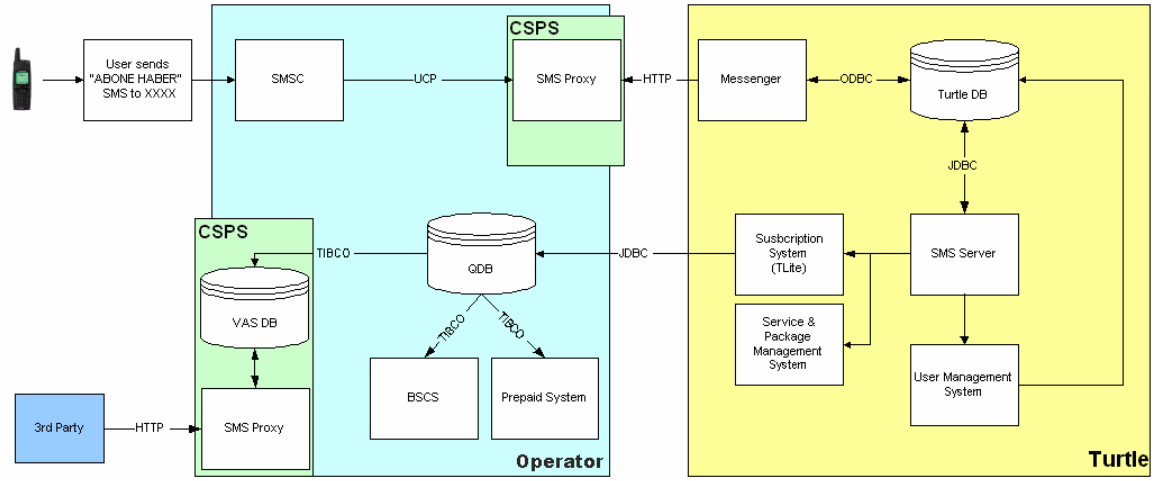

Figure 3.1. SMS subscription flow steps

- The user sends an SMS message to XXXX (specific short message number of the service) which has a content like “ABONE HABER”.

- The message goes from the phone to SMSC and then to Operator’s SMS Proxy Application that enables Turtle to receive and send the user’s message via http based protocol.

- The Turtle Messenger System continuously queries the SMS Proxy for incoming messages over http.

- Once Messenger receives the message, it is written to the Turtle DB to be read and interpreted by the servlet running on the SMS Server.

- The SMS Server polls the Turtle DB for incoming messages and retrieves the message. Then the SMS Server finds the related servlet that will interpret the incoming message.

- The Servlet checks the current subscription status of the user through the user management system (UMS) and the package / service subscription status through the subscription system.

- The Servlet writes the response SMS message to Turtle DB. The Messenger retrieves and delivers this message to the user through the SMS Proxy. There may be more than one response message.

- After the Servlet finishes collecting the necessary data, it initiates the subscription process through the Subscription System.

- The 3rd party, which is continuously polling the SMS Proxy about new service subscriptions, retrieves the new subscription information that has just been stored into the Turtle SIS DB.

- The 3rd party starts service to the new user.

In Figure 3.2. you can see the Subscription flow between Turtle and the End-user.

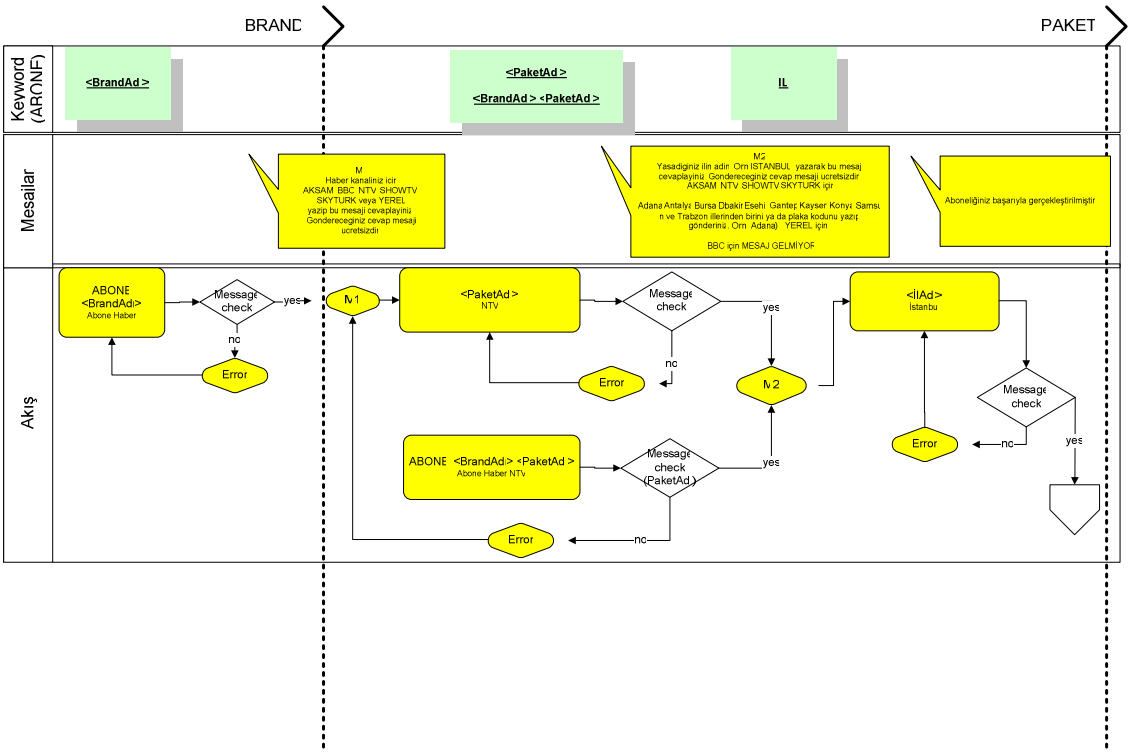

Figure 3.2. Subscription flow for package ex: “HABER” case

3.2. Data Collection

First, we need the data to process. To do this, we used old SMS messages that had already been sent by real customers. We collected more than 6 million SMS messages to analyze. We created a separate system with a new database and stored all these SMS messages in this database. To compute statistics, we added new fields into the original message specific data table.

In Table 3.1., the final SMS messages table design is showed with column definitions. Notice that the new columns marked as "*".

Table 3.1. SMS Message table design

COLUMN NAME	DEFINITION
MSGID	Message ID : primary key of the table
SERVICENO	Short number of the SMS message. Ex: 1234
MSISDN	GSM Number of the customer
MSGBODY	Message body : Message content
DATE	Received date of the message
STATUS	0: received but not processed 1:processing 2:processed
*SequenceID	Dialogue ID. It is common for every message sent in the dialogue
*HataTipi	The Error type of the message
*OlmasiGereken	The correct message that customer should send
*HangiPaketicin	Which package is the message sent for
*SonucaUlasmismi	Has the conversation ended successfully after this sequence ?
*KacAdimdaUlasmis	Number of steps to reach to the end
*HangiNoktada	The state of the conversation

3.2.1. The message structure

For our case, four levels are enough in order to categorize all SMS messages (Fig.3.3).

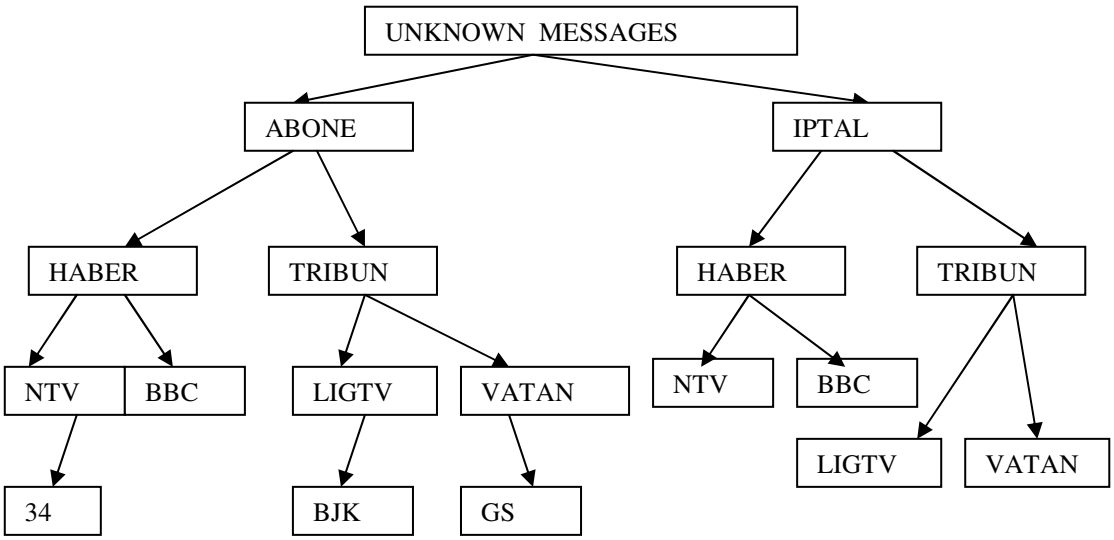

Figure 3.3. SMS messages map

Some possible subscription message combinations are:

- *ABONE HABER →NTV→34*: the case where the user subscribes to the package in three steps.
- *ABONE HABER NTV→34*: the subscription is made in two steps.
- *ABONE HABER NTV 34:* the subscription is made in one step.

Some possible cancellation message combinations are:

- *IPTAL→HABER→NTV:* in three steps.
- *IPTAL TRIBUN→LIGTV*: in two steps.
- *IPTAL HABER BBC:* in one step.

3.2.2. Tagging Process

In order to calculate N-Gram statistics, we needed to count unigram, bigram and trigram values of the SMS messages. To do this we needed to set every message's

SequenceID, HataTipi, OlmasiGereken, HangiPaketicin, SonucaUlasmismi, KacAdimdaUlasmis, HangiNoktada values of the data collection table.

We called this operation "Tagging" because we mark and regroup every message for extracting and representing the similarity of meaning of words. Because we needed to analyze more than 6 million SMS messages, a small utility program was written to make the tagging process faster. It simply gets data from the SMS message table and allows the users to select tagging parameters for the dialogue.

To make the tagging process as fast as possible, a simple graphical user interface was generated. This GUI allows selecting multiple messages from the same user, to help understand what the user is trying to achieve. A more detailed explanation of this program can be found in appendix C.

By using the tagging program, all the SMS data were reorganized in order to be used for the N-Gram process. To do this, 20 per cent of the tagged data were separated for testing the performance.

For the rest of the data, *SequenceID*'s were regrouped for the available six packages (*TRIBUN, FLORT, HABER, POP, FINANS*, and *GEZEGLEN*) and three commands (*ABONE* for subscription, *IPTAL* for cancellation *YARDIM* for help).

Therefore, with all the combinations there were 18 major groups. After evaluating the tagging results, the major error types in these messages can be regrouped as follows:

3.2.2.1. Typing errors

This kind of errors occurred mainly when the content of the messages are not correct due to mistyping. For example, when users type "*AB0NE*" or "*ABNE*" instead of "*ABONE*".

3.2.2.2. Mobile device specific problems

Some devices can produce unsupported characters without the user's intention. For example some Panasonic models add a pattern like "*<!XY>*" to every SMS message, so when the user sends "*ABONE HABER*", this message is received by the SMS handler system like "*ABONE HABER<!01>*".

3.2.2.3. Proper sentences

Some users send proper Turkish sentences instead of obeying the required format. For example: "*HABER PAKETIMIN IPTALINI ISTIYORUM*" instead of simply "*IPTAL HABER*".

3.2.2.4. Reflected messages

Some users can reply to the SMS handler System's questions within the template of the question messages. For example, when the SMS handler asks: "*ABONE OLMAK ISTEDIGINIZ PAKET ADINI ABONE BOSLUK PAKET ADI YAZARAK GIRINIZ*", the user replies to this message as follows: "*ABONE OLMAK ISTEDIGINIZ PAKET ADINI ABONE BOSLUK PAKET ADI YAZARAK GIRINIZ ABONE HABER NTV*"

3.2.2.5. Dialog based messages

Some users may think that they are in conversation with a real human. Therefore, they reply as if they are talking to a human. For example, a reply to the question "*ABONELIK ISLEMLERINIZE BASLAMAK ICIN BIR PAKET ISMI GIRINIZ*" could be

"*HABER PAKETINI ISTIYORUM SIZI AILECEK COK SEVIYOR VE ILGIYLE TAKIP EDIYORUZ*".

3.2.2.6. Inappropriate messages

Some users may send messages that contain some insult or inappropriate words or phrases.

3.2.2.7. Wrong services

Some users intend to get a Telco service other than Turtle. This could simply be due to mistyping the short service number. For example "*TV PRG SHOW*" which is similar to key based SMS command syntax, is correct call for another short number to demand a television program schedule of specific service but not valid for Turtle services. Some users can send these kinds of messages to Turtle by mistake.

3.2.2.8. Undefined messages

Some messages may not be possible to understand even for a human being.

For example: "*JGLK TYDDS '^''^+ 4r44 4344*"

After processing all data in the tagging process, the distribution of the SMS messages was obtained as in Table 3.2.

Table 3.2. Distribution of the error messages

MESSAGE TYPE	PER CENT
Valid	68.50
Invalid:	31.50
Total:	**100**

By examining the results we can say that 31.50 per cent of the messages are invalid, and not understood by the existing systems. So our scope is focused these 31.50 per cent of the wrong messages. Graphical representation of the error distribution is shown in Figure 3.4.

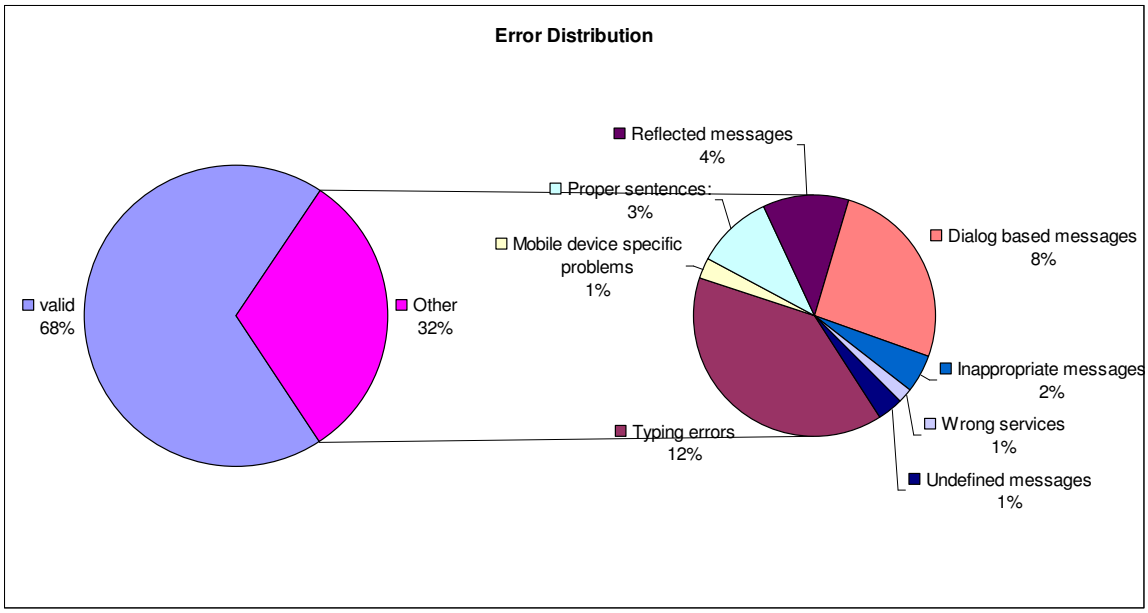

Figure 3.4. Error distribution

By evaluating these results, we can say that the most common error type is regrouped in the "Typing Error" category with 12 per cent value of the all SMS messages.

Table 3.3. Error categories

Category	Per cent
Typing errors	12.35
Mobile device specific problems	0.82
Proper sentences:	3.24
Reflected messages	3.75
Dialog based messages	8.06
Inappropriate messages	1.63
Wrong services	0.63
Undefined messages	1.02
TOTAL	**31.50**

The per cent values of the Error categories are shown in Table 3.3.After examining these statistics from our tagging process we can say that, 31.50 per cent of the all messages are incorrect; which means that the current system does not understand the content and set messages' status to invalid state. When we regroup these wrong messages within each others we obtained following results:

- 39 per cent of the invalid messages are in the "*Typing errors*" category
- 3 per cent of the invalid messages are in the "*Mobile device specific problems*" category.
- 10 per cent of the invalid messages are in the "*Proper sentences*" category.
- 12 per cent of the invalid messages are in the "*Reflected messages*" category.
- 26 per cent of the invalid messages are in the "*Dialog based messages*" category.
- 5 per cent of the invalid messages are in the "*Inappropriate messages*" category.
- 2 per cent of the invalid messages are in the "*Wrong services*" category.
- 3 per cent of the invalid messages are in the "*Undefined messages*" category.

These new percentages can be defined as error categories of the wrong SMS messages. Graphical representation of the error categories is shown in Figure 3.5.

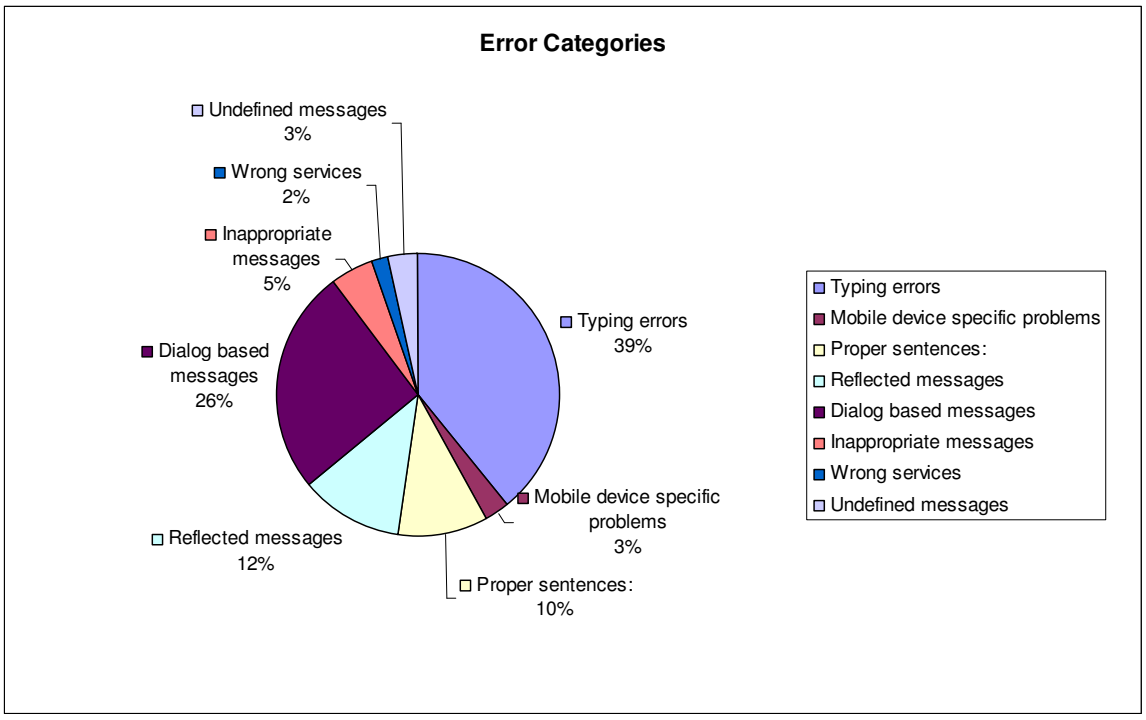

Figure 3.5. Error categories

3.3. Computation of the N-gram probabilities

By using the statistics from the tagging process, the following decisions were taken to implement the computation of N-Gram probabilities:

- Computation of the Unigram, Bigram and Trigram probabilities is enough to determine the context of the short messages sent by the users.

- Because of the 160 character limitation of the SMS messages, to cover up every possibility, instead of using word based approach we used characters for counting N-Gram values. For a character-based approach, possible N-Grams of the string *fork* are given as following: *the empty context*, *f, o, r, k, fo, or, rk, for, ork , fork*.

- 20 per cent of the tagging data has been reserved for evaluating of the performance of the N-Gram approach.
- Because the probabilities of the N-Gram values are too small to compute, logarithm probabilities were used to determine weight of the sequences.

- It is sufficient to use Bayesian Maximum Likelihood hypothesis formula which is mentioned in chapter 3 (Equation 3.9) in order to determine the most probable result. Because prior probability parts of the Bayesian formula P(h) are very similar

- In order to determine whether a given message belongs to a specific N-Gram table, it is necessary to give an acceptable weight limit (threshold value) for every N-Gram table.

Finally, our N-gram approach algorithm for the training data is shown in Figure 3.6.

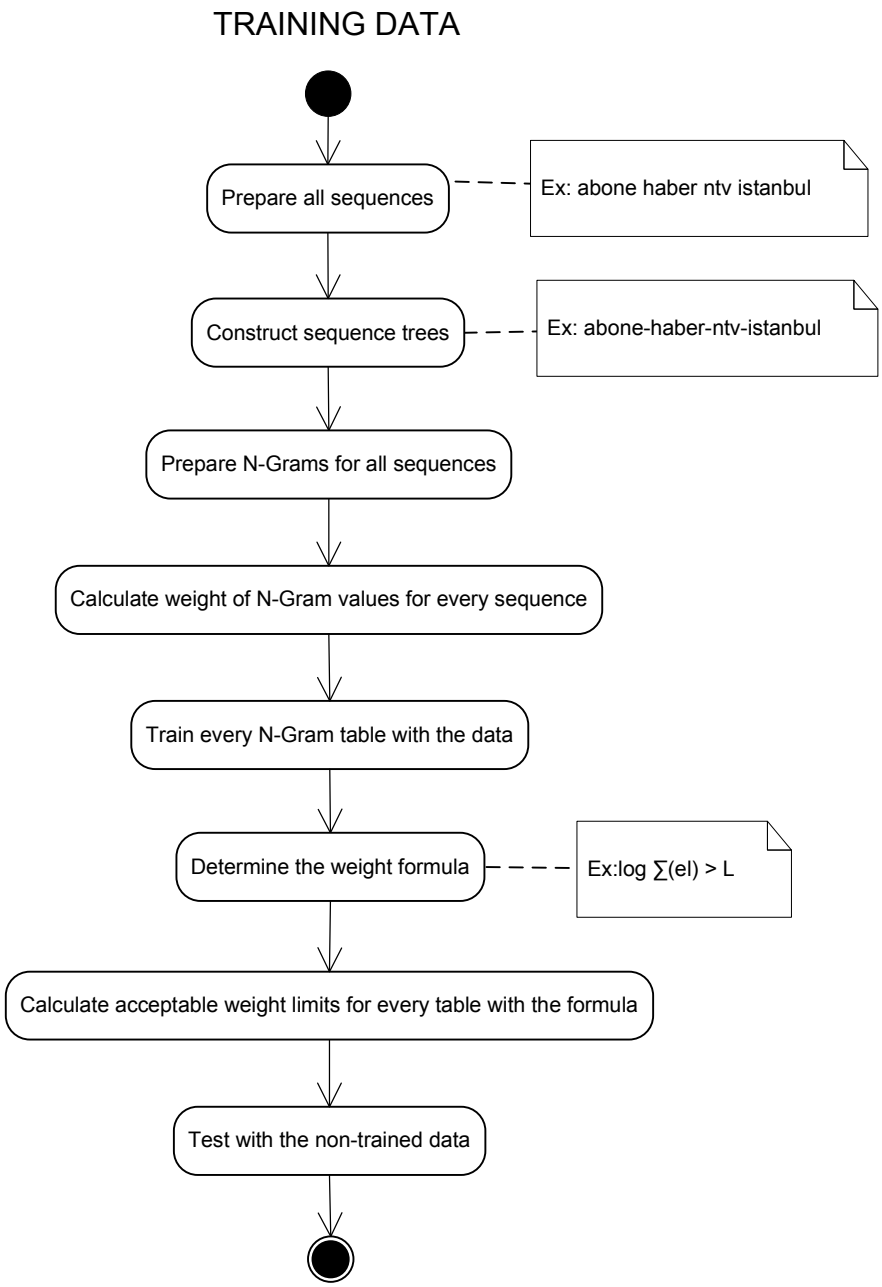

Figure 3.6. Training N-Gram data flow

The first step is preparing all the sequences and creating N-Gram tables. N-gram tables are simply a data structure that contains unigram, bigram, trigram statistics of the elements of the SMS message which is mentioned in section 3.1.1.

To clarify this, for example imagine that our message map includes only following structure as shown in Figure 3.7.

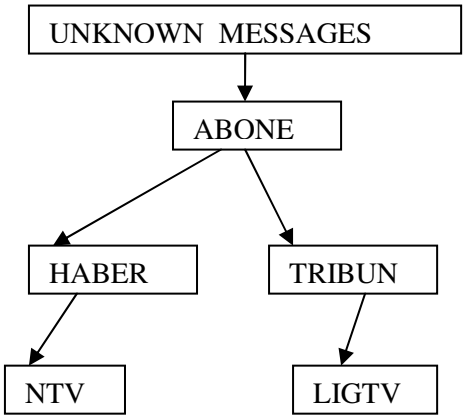

Figure 3.7. SMS message map for N-Gram table example

In this case we need to process given SMS messages for six alternative answers:

- ***"UNKNOWN":*** This node contains all the SMS messages in the system. For every SMS message we count its unigram, bigram, trigram statistics to be sure that at least one of the node of the SMS message map contain all SMS statistics. We can also say that these statistics are our corpus for the SMS messages of the Turtle.

- ***"ABONE":*** This node contains only unigram, bigram, trigram statistics for the SMS messages which have been marked as "ABONE" in the tagging step. For example all variations of the "ABONE" SMS messages (i.e. "ABN, abon, bone, ab0ne… etc") are processed in this node.

- ***"ABONE-HABER":*** This node contains N-Gram statistics for the SMS messages which have been marked as "ANONE HABER" in the tagging process. For example: "abone haber , ABone Hbr,Ab?n Haner … etc".

- ***"ABONE-HABER-NTV":*** This node contains N-Gram statistics for the SMS messages which have been marked as "ANONE HABER NTV" in the tagging process. For example: "abone haber MTV, ABone Hbr NTV,Haber NTV … etc".

- ***"ABONE-TRIBUN-":*** This node contains N-Gram statistics for the SMS messages which have been marked as "ANONE TRIBUN" in the tagging process. For example: "abone trbn, ABone türübün,Abone trubun… etc".

- ***"ABONE-TRIBUN-LIGTV":*** This node contains N-Gram statistics for the SMS messages which have been marked as "ANONE TRIBUN LIGTV" in the tagging process. For example: "abone trbn ligtivi, ABone türübün L1gtv,Abone trubun ligtv… etc".

Thus for this example we need to create six different N-Gram table data structures to compute N-Gram statistics of every element (node) of the SMS message map.

3.3.1. Computing Statistics within N-Gram Table Data Structure

As mentioned before N-gram tables are simply used for unigram, bigram and trigram statistics of the SMS message map nodes which are mentioned in section 3.1.1.

In Table 3.4. you can examine some part of the statistics for the "ABONE HABER" case. Notice that there are three different statistics computed for unigram , bigram and trigram cases stored in the "ABONE HABER_UNI","ABONE HABER_BI" and "ABONE HABER_TRI" columns of the table respectively.

Table 3.4. Example of N-Gram table data structure for “ABONE-HABER”

ABONE HABER_TRI	ABONE HABER_BI	ABONE HABER_UNI
_AB=0.036429872495	H=0.10904134484	=0.081174042650
ABO=0.072859744990	N=0.00090867787	_=0.02315982572
BON=0.072859744990	S=0.00136301681	A=0.1706030726
ONE=0.072859744990	T=0.00181735574	B=0.2309103416
NE =0.072859744990	_A=0.04543389368	E=0.13781242834
E H=0.072859744990	_I=0.000454338936	H=0.12015592753
HA=0.072859744990	AB=0.19990913221	I=0.002522357257
HAB=0.072859744990	AL=0.0009086778736	K=0.0006879156156
ABE=0.072859744990	B0=0.018173557473	L=0.00091722082091
BER=0.072859744990	BE=0.09086778736	N=0.09286860811740
ER =0.036794171220	BO=0.09086778736	NULL=0.00022930520
R N=0.019307832422	BR=0.01817355747	O=0.0924099977069
NT=0.019307832422	BU=0.001363016810	P=0.00091722082091
NTV=0.019307832422	E =0.045433893684	R=0.036000917220
R S=0.01020036429	ER=0.045888232621	S=0.0016051364365
SK=0.01020036429	HA=0.09086778736	T=0.004127493694
…	…	…

For example “ABO=0.072859744990” can be read as the probability of the trigram “ABO” is equal to 0.0728597449908925 for “ABONE HABER” cases. Notice that the value of the trigram “ABO” of the “ABONE HABER” (=0.07285974) is different than the value of the “ABONE HABER NTV” (=0.06039). So for every step of the SMS message map we use different N-Gram data table to compute N-gram statistics of this step.

Thus we can compute statistics with the N-Gram table data structure for every input SMS message. Let’s compute the N-gram statistics for the given message “ABONE”.

- **unigram:** P(ABONE)=P(A)*P(B)*P(O)*P(N)*(E)
  =0.170603073*0.230910342*0.092409998*0.092868608*
  0.137812428=4.65915E-05

Because we deal with very small numbers , we use log probabilities , so for this case result is log(P(ABONE))=log(4.65915E-05)= -14.38957428. Notice that we use logarithm base as 2 which is most commonly used in NLP [1] .

- **bigram:**P(ABONE)= P(_A)*P(AB)*P(BO)*P(ON)*P(NE)
  =0.045691906*0.199303742*0.089208007*0.09051349*
  0.092689295=6.81556E-06

Same as above, we use log probabilities so the result is -17.16273667. Notice that the character '_' is used to mark the beginning of the sentence as we mentioned in chapter 2.

- **trigram :**P(ABONE)= P(_AB)*P(ABO)*P(BON)*P(ONE)
  =0.087354409*0.085690516*0.084858569*0.088186356
  =5.60163E-05

And log probability for the trigram value of the input message "ABONE" in the "ABONE HABER" case is equal to -14.12379289.

As we mentioned before the given input SMS message can vary for every N-Gram data table structure. For example trigram value of the same SMS message "ABONE" for the "ABONE" N-Gram data structure is equal to -8.315805366. So we should consider every result only in its area which is N-Gram data structure for our case. We can not compare the probability obtained from one N-Gram data structure with the one another.

For our system, in the training process, all the N-Gram tables are created and computed by using tagging data. After that for every N-gram table a specific threshold value is determined to accept or reject a given string for this table.

Finally, the entire system is tested against non trained data from the tagging process that we reserved for this process. If the test results are not acceptable, the threshold values for every N-gram table are recalculated and the testing step is repeated until all the results are acceptable.

3.4. N-Gram Phase1

As mentioned in 3.2, after we created N-gram tables by our algorithm we have to calculate threshold values for every N-gram table until the results are acceptable in order to parse SMS messages correctly. Thus, we decided to start with making a simulation of the N-Gram flow to be sure that our system will work against 6 million of the data. Because it would take very long time if we directly use 6 million SMS messages to determine these threshold values.

Thus, in the simulation environment, we used only 100000 SMS messages in order to repeat the whole simulation easily and more than once a day. For the calculation of the weight formulas, we wrote a .NET assembly module to implement our approach.

After running the simulation for many times we made an assumption like following:

In order to calculate threshold values for every N-gram table , it is sufficient to use Zero probabilities from the smoothing algorithms and the length of the input SMS message. Consider the equation

$$TH(message) = \log(\prod_{\frac{L}{2}} Z) = \sum_{\frac{L}{2}} \log(Z) \qquad (3.1)$$

where TH is the threshold value for the specific N-gram table, Z is the zero probability of the N-Gram table and L is the length of the input message. We can say that if the combination of the unigram, bigram and trigram probabilities for the input message is lower than the unigram, bigram and trigram probabilities of the half of the length times of the zero probabilities, the input message will not be acceptable and will be filtered from the results. Or we can say simply that, at least half of the message character combinations should be similar to the N-gram table statistics to accept for computing the given message for the N-gram table. You can see the given example in the section “Processing Messages in the N-Gram Phase1 Module” to see how the threshold values are used in order to use for filtering barrier which determines minimum N-gram value to accept the given message by the table.

So our final implementation for Phase1 is depicted in Fig.3.8, where SMSHandler system is divided by four modules: SMSHandler, SMSHandlerAction, N-GramPhase1 and SMSHandlerError.

The SMSHandler module contains existing SMS parse logic. If the given SMS message is evaluated correctly by this module, it is sent to the SMSHandlerAction module in which there are some specific routines about the Turtle Subscription system.

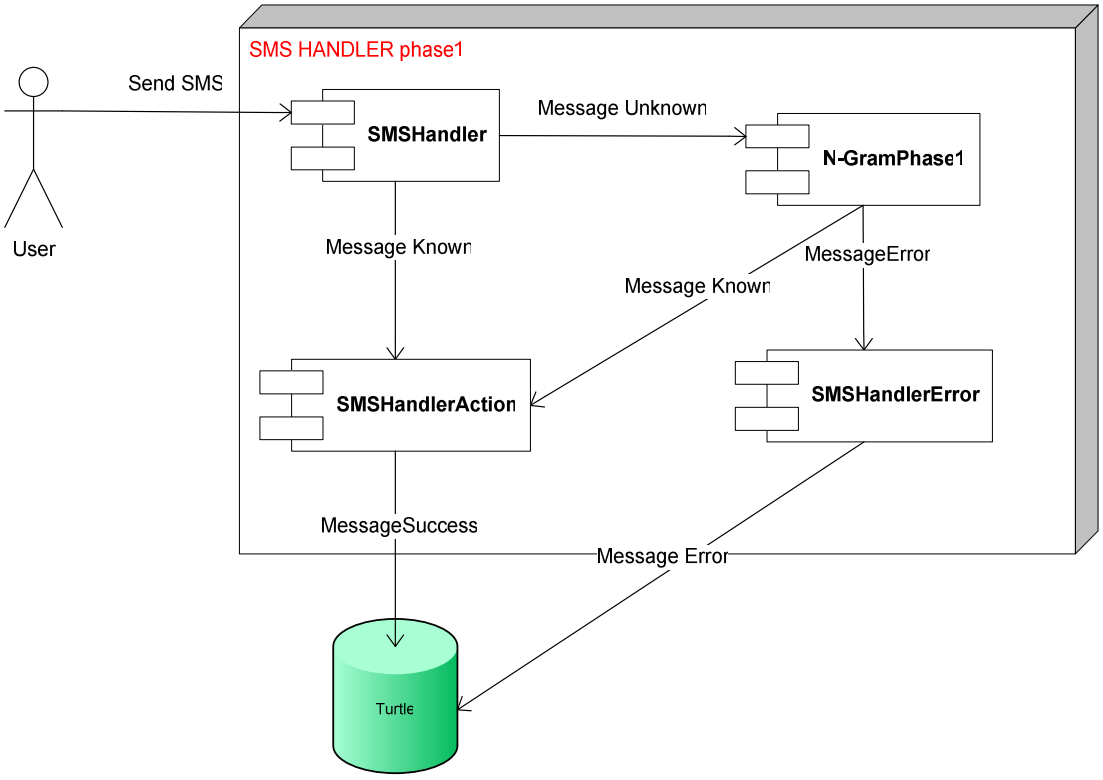

Figure 3.8. Implementation of N-Gram phase1

Sms messages are parsed in our N-Gram phase1 module only if they could not be evaluated by the existing SMSHandler module. If the given SMS corresponds to one of our N-Gram tables, it is sent to the SMSHandlerAction module for processing; otherwise it is sent to the SMSHandlerError module which is responsible for sending the User an appropriate Turtle Error Message.

After determining how to calculate threshold values in our simulation environment, we trained the whole system with 6 million messages, we deployed the SMSHandler into the real system and started to observe the performance of the new module, whose results can be checked in Chapter 4.

3.4.1. Processing Messages in the N-Gram Phase1 Module

You can see the UML sequence diagram for the N-Gram module phase1 in Figure 3.7. Thus, as mentioned before, when a new message comes into the system and SMSHandler can not directly evaluate the message, it calls N-Gram module's *evaluate* method which returns a message after parsed in the N-Gram module.

After that, if the returned message (which is shown as *Ng_msg* in Figure 3.9.) is equal to the input message, SMSHandler sends the message into the *Error* module. If the input message changed in the N-Gram module, it means that the N-Gram module could find a corresponding result for the given message, SMSHandler sends the message into the *Action* module.

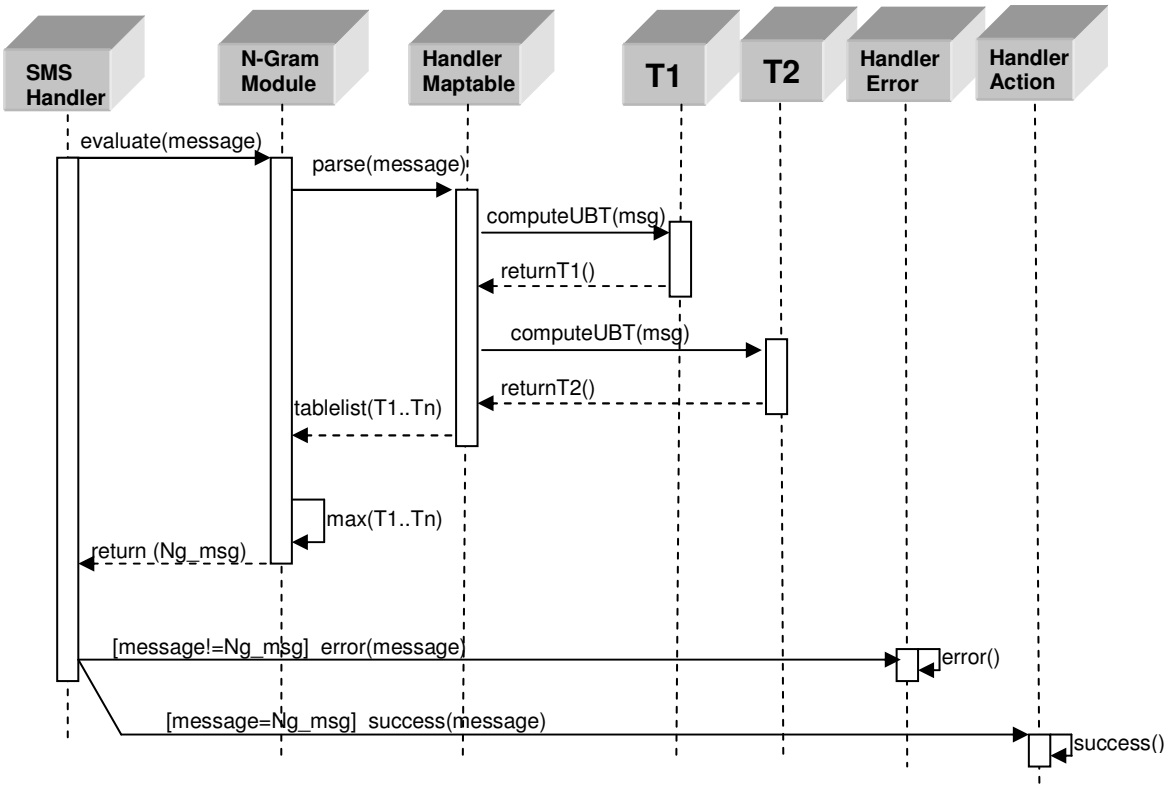

Figure 3.9. N-Gram module sequence diagram for phase1

In order to parse SMS messages, the N-Gram module uses *HandlerMaptable* which is a typical hash map of the all trained N-Gram tables (T1…Tn). The idea is to process given message's N-Gram values for every table which is defined in the *HandlerMaptable* and then to choose the best result that gives the message the highest probability.

That is, given a message M={T1,T2,…,Tk,…..,Tn}, where Tk is combined value of the different N-Gram orders (in our case these are unigram , bigram , trigram) by using the deleted interpolation algorithm[Jelinek and Mercer (1980)] [1]. As shown in Figure 4.7., for each table *HandlerMaptable* calls the *computeUBT* function which estimates the probability $P\ (w_n \mid w_{n-1}...w_{n-2})$ by mixing together the unigram, bigram, and trigram probabilities.

Each of these is weighted by a linear weight λ:

$$P^{'}(w_n \mid w_{n-1}...w_{n-2}) = \lambda_1 P\ (w_n \mid w_{n-1}...w_{n-2}) + \lambda_2 P\ (w_n \mid w_{n-1}) + \lambda_3 P\ (w_n) \quad (3.2)$$

such that the λ_s sum to 1:

$$\sum_i \lambda_i = 1 \quad (3.3)$$

Notice that, for the lambda of the trigram we give more weight in the interpolation than the one of the bigram. After repeating many times in the simulation environment we set the lambda values for every N-gram table as following:

- Unigram:0.1
- Bigram: 0.3
- Trigram:0.6

After calling *computeUBT* method of the each table in the hash map, *HandlerMaptable* obtains a list of the N-gram results. And then *HandlerMaptable* filters the results by using each of the threshold value of the N-Gram tables. If the result is smaller than the threshold value (Ln) obtained after training the N-Gram Table (Tn), *HandlerMaptable* remove it from the list.

Finally after calling the *parse(message)* method ,the N-Gram module receives a N-Gram table list (T1…Tn) filtered in the *HandlerMaptable*, and then maximizes the values of the list to select the Tn which gives the message the highest prior probability and returns the computed probability of the message (*Ng_message*) to the SMSHandler as a result.

In Figure 3.9., there are two N-Gram tables (T1, T2) used as example, imagine that both T1 and T2 are returned in the *tablelist* result of the *parse (message)* method of the HandlerMaptable if P(T1)>P(T2) N-Gram module returns *Ng_message* of the T1 as a to the SMSHandler module.

To clarify this, let us give an example: Imagine that given SMS string S=”ABONE HBR”. As mentioned before, in order to evaluate this message, the SMSHandler sends it to the N-Gram module. So it calls N-Gram module *evaluate (“ABONE HBR”).* After that N-Gram module calls *parse(“ABONE HBR”)* of the HandlerMaptable which calls directly *ComputeUBT(“ABONE HBR”*) methods for every N-Gram table (Tn).

As we mentioned in the Figure 3.1.1., for our case some of the N-Gram table list is given as following:

T1: UNKNOWN
T2:ABONE
T3:ABONE-HABER
T4:ABONE-HABER-NTV
T5:ABONE-HABER-BBC
T6:ABONE-TRIBUN
T7:IPTAL
T8:IPTAL-HABER
T9: IPTAL -HABER-NTV
T10:IPTAL-TRIBUN

Thus what we need is to compute N-Gram statistics via *computeUBT* method for every N-Gram data table (T1…Tn) for our case n is equal to 10.

ComputeUBT method just calculates unigram, bigram and trigram probabilities for the given string and then interpolates the tree results according to (Equation 4.1). Recall that we have already calculated unigram, bigram and trigram log probabilities for T3(“ABONE”) in 3.2.1.

The results were :

- **U(unigram):** -14.38957428
- **B(bigram):** -17.16273667
- **T(trigram) :** -14.12379289

Thus, T3→*ComputeUBT("ABONE")*= 0.6U+0.3B+0.1T= -15.06205416 where the multipliers 0.6 , 0.3 and 0.1 are the interpolation coefficients for the trigram, bigram and unigram respectively.

However, for our example we need to calculate T3→*ComputeUBT("ABONE HBR")*,so the same as before we can easily compute interpolated unigram , bigram and trigram statistics as following:

T3→U(ABONE HBR)=-log(P(A)*P(B)*P(O)*P(N)*(E)* (' ')*P(H)*P(A)*P(B)*(E)* (R))
=-log(P(A) -log(P(B) -log(P(O) -log(P(N) -log((E) -log((' ')-
log(P(H) -log(P(A) -log(P(B) -log((E) -log((R))
= **-33.30188495**

T3→B(ABONE HBR)=-log(P(_A)*P(AB)*P(BO)*P(ON)*(NE)* (E' ')*P(' 'H) *P(HB)
*P(BR))
=-log(P(_A)-log(P(AB)-log(P(BO) -log(P(ON) -log((NE) -log((E'
')-log(P(' 'H) -log(P(HB) -log(P(BR))
=**-34.82741728**

T3→T(ABONE HBR)= -log(P(_AB)*P(ABO)*P(BON)*P(ONE)*(NE' ')* (E' 'H)*
P(' 'HB)*P(HBR))
=-log(P(_AB) -log(P(ABO) -log(P(BON) -log(P(ONE) -log((NE'
')-log((E' 'H) -log(P(' 'HB) -log(P(HBR))
=-31.34550563

and T3->*ComputeUBT(ABONE HBR)*=0.6T+0.3B+0.1U= -32.4697309

In Table 3.5., all results of the N-Gram tables for the given string S="ABONE HBR" are shown.

Table 3.5. N-Gram table results for S=”ABONE HBR”

T:N-Gram table	T(ABONE HBR)
UNKNOWN	-37.21703869
ABONE	-51.29579233
ABONE-HABER	-32.4697309
ABONE-HABER-NTV	-35.70986535
ABONE-HABER-BBC	-34.60801572
ABONE-TRIBUN	-52.48618861
IPTAL	-89.27481631
IPTAL-HABER	-70.18319687
IPTAL-HABER-NTV	-66.6359062
IPTAL-TRIBUN	-84.03804728

As we showed in Figure 3.9., the N-Gram module receives all these N-gram table results and finds the maximum value for the T(ABONE HBR). In this specific example, the maximum result is reached only when N-Gram table value is equal to -32.4697309. Finally, the message S=”ABONE HBR” will be considered as ABONE-HABER in the Handler Action module.

3.5. N-Gram Phase2

The problem with the approach described in the previous section is that the N-Gram tables are constructed only after the tagging process, which needs to evaluate more than 6 million of the SMS messages .This process takes a very long time (for our case it takes 3 months with our tagging utility program) and needs a lot of manual work. It is not possible to repeat every step for each new package or SMS keyword that will be created in the future.

In order to see whether there is any way of automating the Phase1 steps without any manual work, we have tried two different approaches:

a-) Reconstructing N-gram tables by replacing and overriding existing tagged data

b-) Creating N-gram tables only by simple variants and reconstructing them at the runtime with predefined threshold values

3.5.1. Reconstructing N-Gram Tables by Using Existing Data

After reconsidering the construction of N-Gram Tables, we tried to reuse the N-Gram statistics of the existing tagged data for the new SMS scenarios (i.e. new packages, new campaigns ...etc).

For example, assume that a new package names *"SKY"* is to be introduced for the category *"HABER*". For the construction of the N-Gram tables, the only difference between the *"NTV"* package and the *"SKY*" package is the package name.

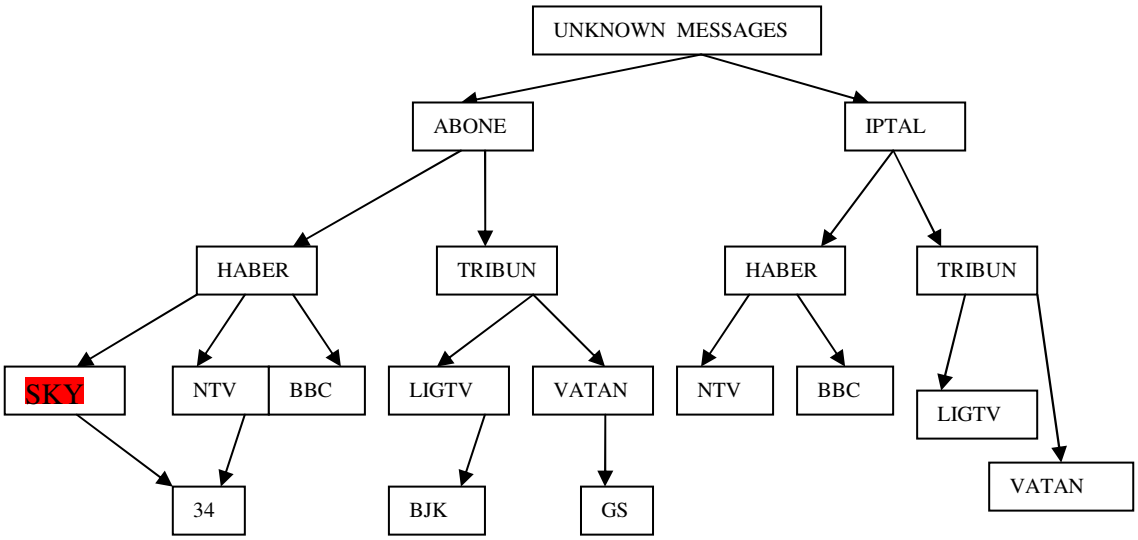

Figure 3.10. Adding New Package into the Message Map

If we continue our example, in order to consider N-Gram tables for the new packages, (in this case the "SKY" package) we have to create following N-Gram tables:

- "*ABONE*": This is already known by existing tagged data, there is no need to recalculate for the new package.
- "*ABONE HABER*": This is also known by previous statistics.

- "*ABONE HABER SKY*": The only difference is here, we can use existing "*ABONE HABER NTV*" statistics to create "*ABONE HABER SKY*" by using the following mask technique:

"*ABONE HABER SKY*"= "*ABONE HABER*"<"*ABONE HABER NTV*"> +" *SKY*"

Note that "X<Y>" means N-Gram statistics of the X part in Y context. In our case in order to create "*ABONE HABER SKY*" N-Gram table we use the same statistics for the "ABONE HABER NTV" N-Gram Table until the "*SKY*"(package name part) and after that part we replace "*NTV*" with the "*SKY*".

3.5.2. Creating N-Gram Tables from Scratch

In this case, instead of using tagged data, firstly we manually add some predefined obvious variants of the keys and we define a threshold value for every N-Gram tables as we did in phase1, but this case if the given message is accepted we add this message into the tables as a new variant and recalculate statistics in runtime.

For example, imagine a new package named "SHOW" will be created. The process of this approach will be as follows:

For the "ABONE HABER SHOW" the following N-gram tables should be created:

- "*ABONE*": This is already known by existing tagged data, there is no need to recalculate for the new package.

- "*ABONE HABER*": This is also known by previous statistics.

- "*ABONE HABER SHOW*": For this table we add manually some initial obvious variants like ABONE HABER SHOV", "ABONE HABER SOV", "ABONE HABER SHW", "ABONE HABER SOW"...etc. And for this table a threshold value for the acceptance will be defined. This could be parametric and it could be modified at runtime.

Finally, at runtime, when a new message is evaluated for this table, if the calculated value for the N-Gram evaluation is smaller than the threshold, the message will be rejected,

otherwise (calculated value is bigger than the threshold) the message will be accepted and the statistics of the table will be recalculated because the accepted message is inserted into the N-Gram Data table.

Thus in this case the N-Gram data table and its statistics are evaluated in runtime. The results for both of these cases are presented in the chapter 4.

4. RESULTS AND EVALUATION

In this chapter, the results and evaluations of the N-Gram techniques that we mentioned in the previous chapters will be presented. There are two big sections Phase1 and Phase2. Constructing N-Gram tables with the tagged data is evaluated in section 4.1. The rest of the work is evaluated in section 4.2, including runtime training of the N-Gram Phase1.

4.1. N-Gram Phase1

The implementation of this phase was finished in the spring of 2004. After that, the implementation was deployed in the production system and it ran against more than 10 million SMS messages until the spring of 2005.

In order to evaluate the results, the raw data and evaluated data by N-Gram were considered separately. Thus, we could calculate False Acceptance and False Reject values.

“False Accepted Message” signifies the messages that are accepted by the N-Gram approach but normally they are wrong or should not be accepted.

“False Rejected Message” signifies the messages that are rejected by N-Gram approach but normally they are correct and should be accepted by the system.

Therefore, in the evaluation process we also calculated *False Acceptance* and *False Reject* counts in order to measure N-Gram performance. These values are also used in order to set correctly threshold values of the N-Gram tables. By trying different values a better solution is obtained where “False Acceptance”= 2 per cent, “False Reject”=1 per cent and “Resolved Typing Errors”=10 per cent.

“Resolved Typing Errors” signify the messages that are correctly handled by the N-Gram system but caused errors in the previous system. By using this approach, 10 per cent of the Error messages are decreased and correctly handled.

N-Gram Phase1 results and their numerical representations are shown in Table 4.1. and Figure 4.1. respectively.

Table 4.1. Results of phase1

Result type	**Per cent**
Valid	67.52
Typing Errors	2.32
Mobile Device specific problems	0.29
Proper sentences:	3.17
Reflected messages	3.67
Dialog based messages	7.76
Inappropriate messages	1.42
Wrong services	0.59
Undefined messages	0.93
Resolved Typing Errors after Phase1	9.72
False Acceptance	2.07
False Reject	0.54

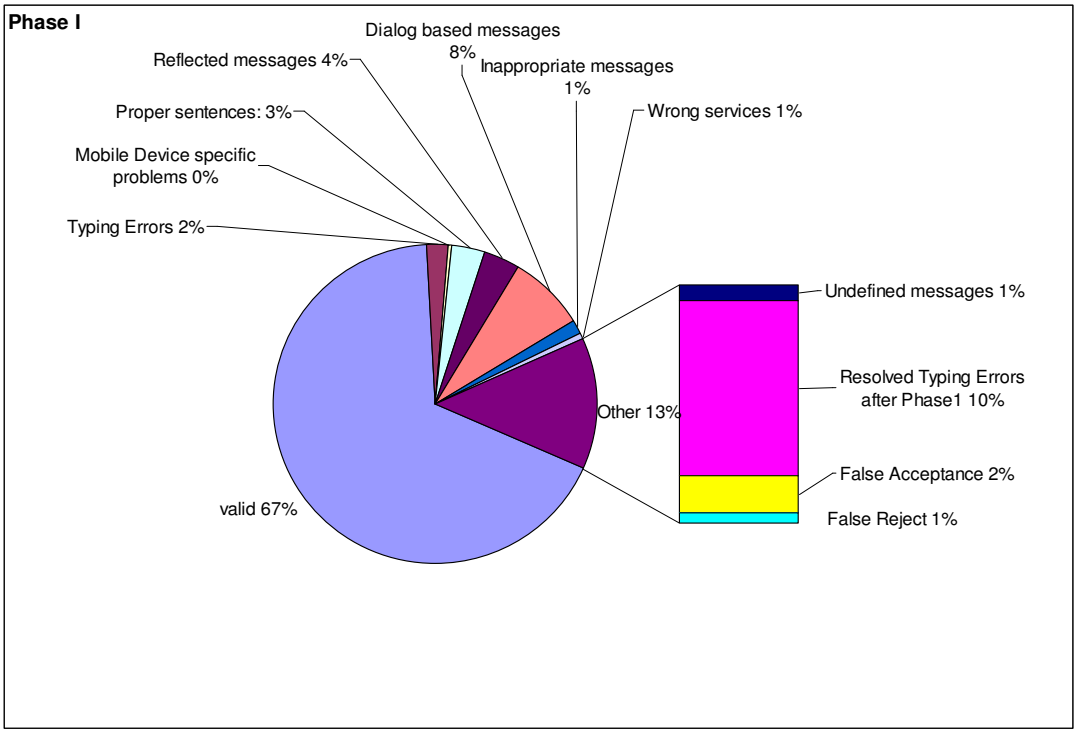

Figure 4.1. Results of phase1

In Figure 4.1., we see the results after evaluating N-Gram approach in the real system. With this approach mainly Typing errors are resolved. Typing errors are decreased from 12 per cent to 2 per cent.

However other erroneous messages like “Dialog based Messages”,” Reflected Messages” and “Phrased Messages” still can not be handled correctly.

Table 4.2. Error distribution after phase1

Result type	**Count**
valid	7724
Typing errors	232
Mobile device specific problems	29
Proper sentences:	317
Reflected messages	367
Dialog based messages	776
Inappropriate messages	142
Wrong services	59
Undefined messages	354

After the Phase1 evaluation overall system's error distribution graph is shown in Figure 4.2

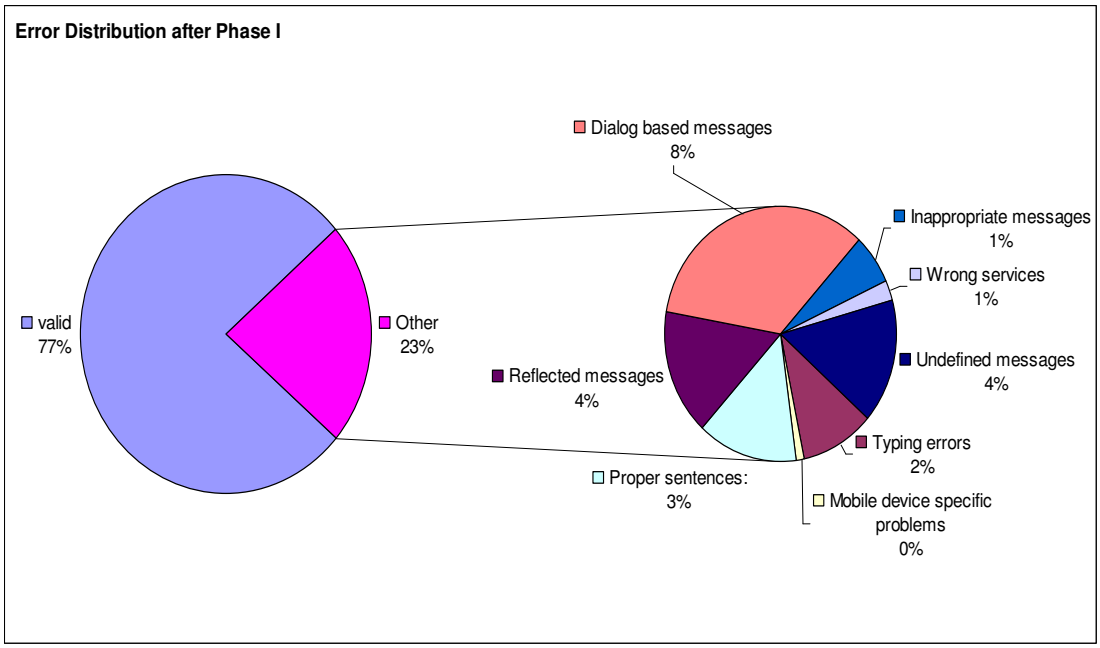

Figure 4.2. Error distribution after phase1

Total valid messages percentage has increased from 68 per cent to 77 per cent. This means that our approach increased the SMS handling performance significantly.

4.2. N-Gram Phase2

As mentioned before, Phase2 has two different approaches: "Reconstructing N-Gram Tables by Using Existing Data" and "Creating N-Gram Tables from Scratch". These approaches could not be run in the production system. Instead of this, they were run in the simulation environment against 100000 SMS messages for evaluation purposes.

4.2.1. Reconstructing N-Gram Tables By Using Existing Data

After the necessary configurations algorithm was deployed in the simulation environment and tested with the 100000 messages.

Table 4.3. Results graph by using existing data

Result type	Count
valid	6742
Typing Errors	587
Mobile Device specific problems	38
Proper sentences:	317
Reflected messages	358
Dialog based messages	759
Inappropriate messages	141
Wrong services	45
Undefined messages	89
Resolved Typing Errors after Phase1	537
False Acceptance	304
False Reject	83

Result graph is shown in Figure 4.3.

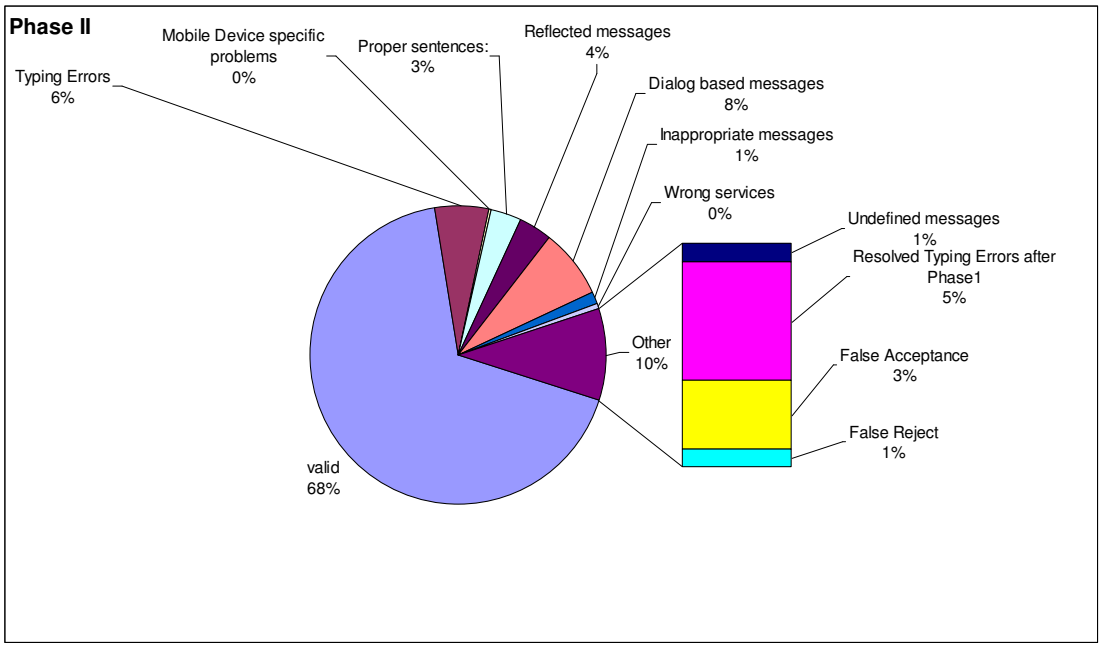

Figure 4.3. Results Graph by Using Existing Data

As shown in Fig. 4.3, the results are very similar to Phase1 . The main difference between Phase1 and this algorithm is the performance. The False Acceptance value is increased from 2 per cent to 3 per cent and “Resolved Typing Errors” is decreased from 10 per cent to 5 per cent.

Table 4.4. Error distribution by using existing data

Result type	Count
valid	7279
Typing errors	587
Mobile device specific problems	38
Proper sentences:	317
Reflected messages	358
Dialog based messages	759
Inappropriate messages	141
Wrong services	45
Undefined messages	476

The overall score is shown in the Table 4.4. and Figure 4.4. respectively. It means that by using this algorithm, handled (valid) SMS percentage is increased from 68 per cent to 73 per cent (It was 77 per cent in the Phase1).

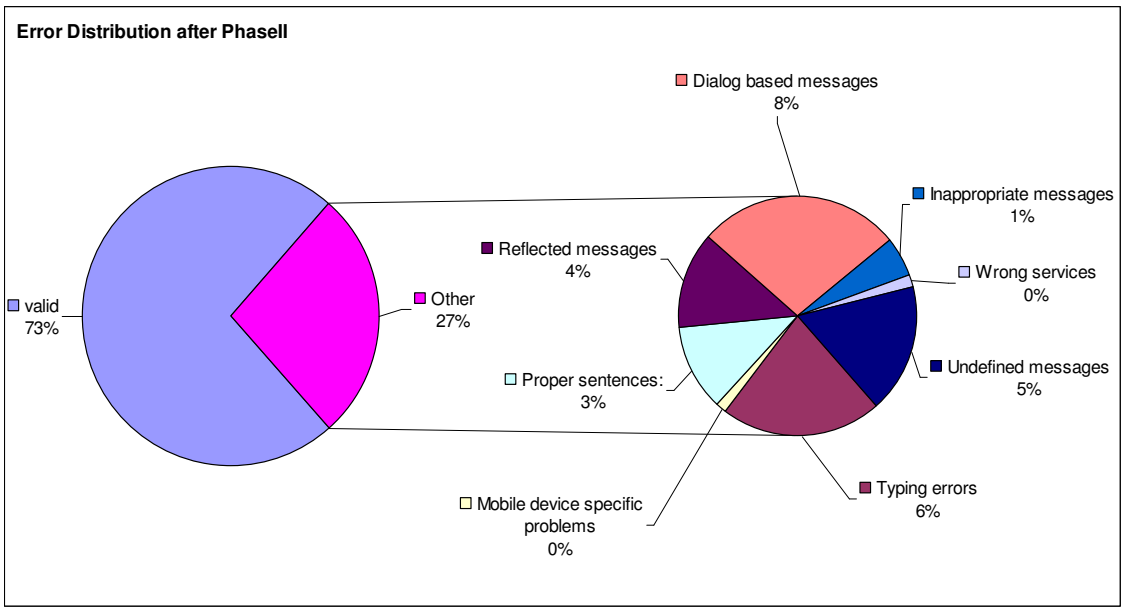

Figure 4.4. Error distributions by using existing data

However, these results are not as good as they seem, because they have been obtained by replacing keywords exactly same length of the old ones. For example, “ABONE HABER SKY” package is obtained by using the same statistics of the “ABONE HABER NTV”. When the new keyword length is different, the performance is decreased.

For example instead of “SKY” package if we want to use “ABONE HABER KANAL7” everything would be changed.

It means that this algorithm is very sensitive to the length of the difference. To make this issue clearer, we have prepared new simulations. If the same simulation runs with the difference length of the keyword, Figure 4.5. will be obtained:

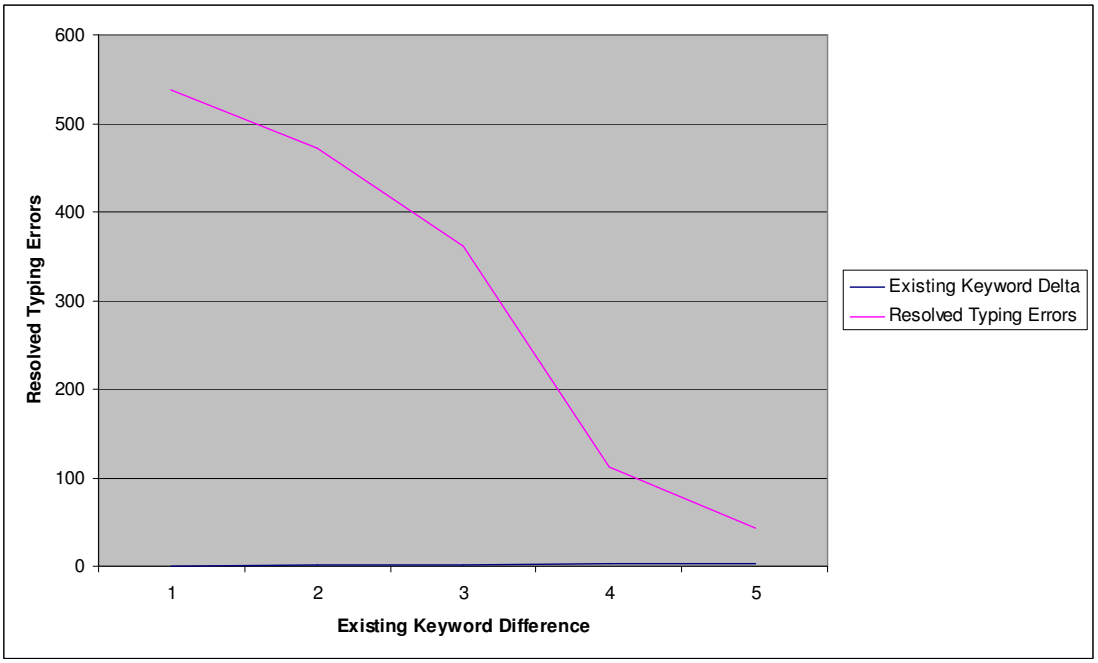

Figure 4.5. Effect of the existing keyword length difference

As shown in Figure 4.5., especially if the difference of the length is bigger than three characters, the resolved typing errors performance is decreasing dramatically.

4.2.2. Creating N-Gram Tables from Scratch

To do this firstly N-Gram tables are constructed manually with some obvious variants as mentioned before. After the simulation, best results are obtained as seen in Table 4.5.

Table 4.5. Result graphs for N-Gram tables from scratch

Result type	Count
Valid	6682
Typing Errors	709
Mobile Device specific problems	43
Proper sentences:	324
Reflected messages	365
Dialog based messages	763
Inappropriate messages	146
Wrong services	48
Undefined messages	93
Resolved Typing Errors after Phase1	437
False Acceptance	314
False Reject	76

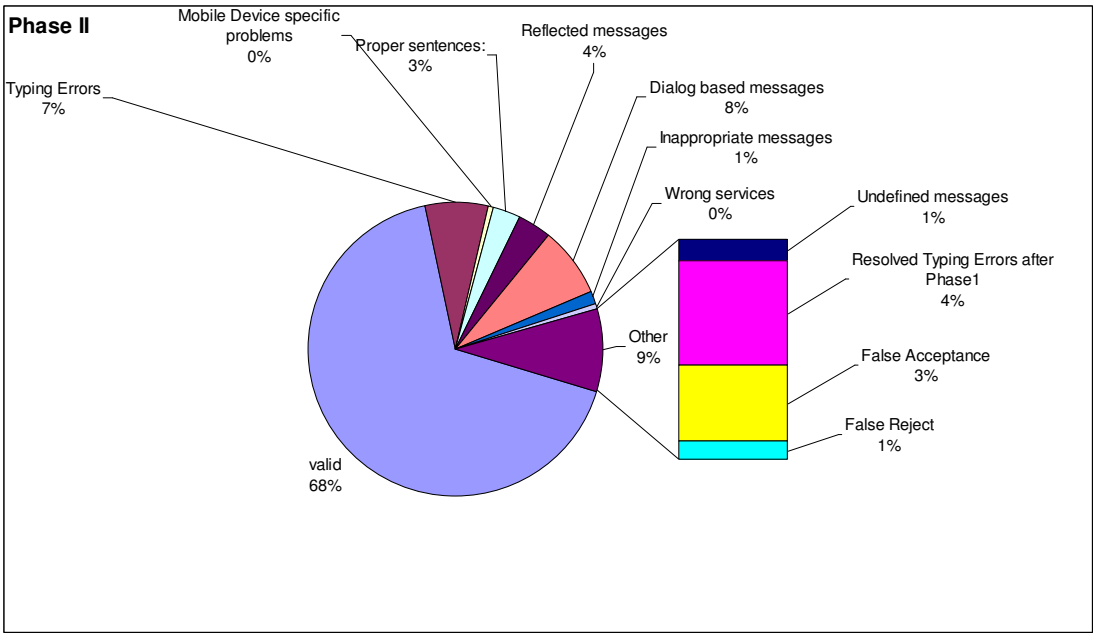

Figure 4.6. Result graphs for N-Gram tables from scratch

As shown in Figure 4.2.2. results are very similar to Phase1. The main difference between Phase1 and this algorithm is the performance. False Acceptance value is increased from 2 per cent to the 3 per cent and “Resolved Typing Errors” is decreased from 10 per cent to 4 per cent (ıt was 5 per cent for the first algorithm of the Phase2).

So the overall Score is shown Table 4.6. as follows:

Table 4.6. Error distribution for N-Gram tables from scratch

Result type	Count
Valid	7119
Typing errors	709
Mobile device specific problems	43
Proper sentences:	324
Reflected messages	365
Dialog based messages	763
Inappropriate messages	146
Wrong services	48
Undefined messages	483

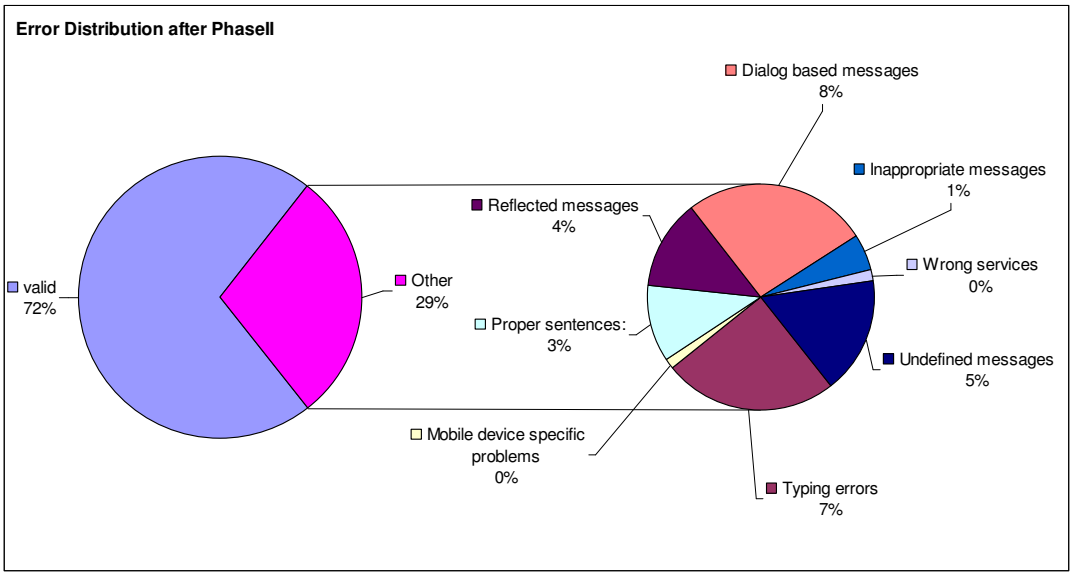

Figure 4.7. Error distributions for N-Gram table from scratch

As shown in Figure 4.7., the handled SMS percentage is increased from 68 per cent to 72 per cent with this algorithm (It was 77 per cent in Phase1 and 73 in the first Algorithm of the Phase2).

As mentioned before, these results are the best ones for this approach. There are also some other problems with this algorithm. This algorithm tries to fill N-Gram tables in runtime by using some threshold values. So after a certain time algorithm does not work because “False Acceptance” value increase dramatically. It means that it accepts every word as valid SMS.

It can be easily seen (in Figure 4.8.) that after the algorithm accepts 200 distinct new keywords overall performance is starting to decrease to zero.

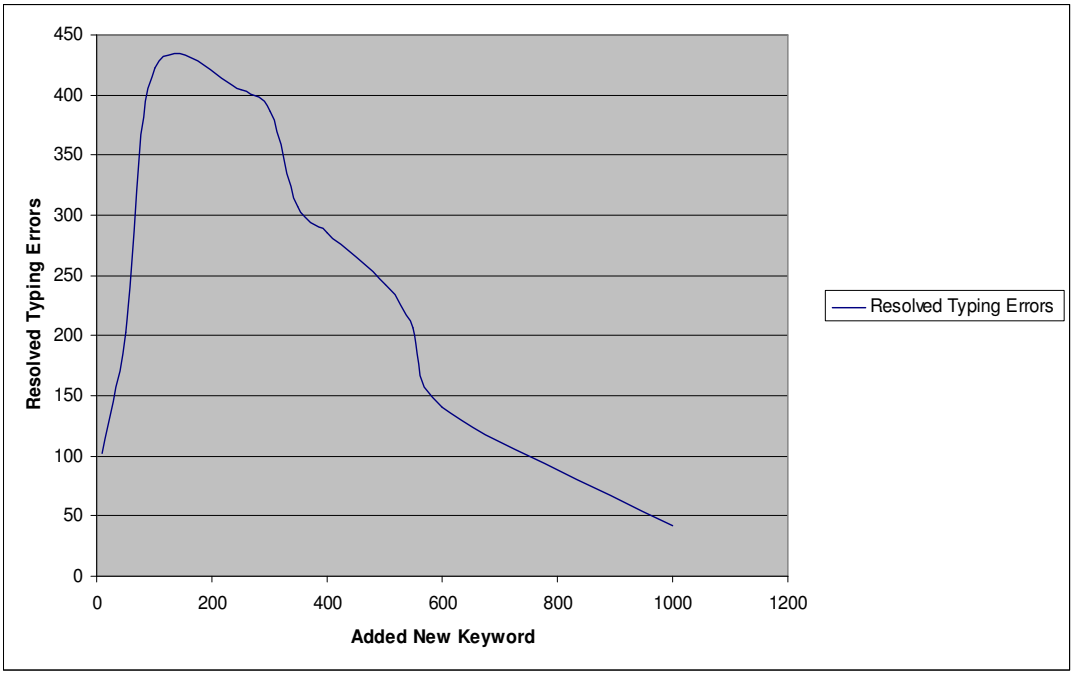

Figure 4.8. Effect of adding new keywords

To resolve this problem, we changed the algorithm as follows: After it accepts and adds new distinct keywords, a new keyword count control is added to the algorithm. If the different keyword count is achieved, no new keyword is added into the N-Gram table even if the other conditions are satisfied. So we obtained every time similar results that we mentioned 4.2.2.

5. CONCLUSION AND FUTURE WORK

The list of future improvements is quite long. In SNLP world there are many approaches to handle our problem. This study only introduced basic concepts of the N-Gram approach and its example of the use in order to increase SMS parsing performance.

It is necessary to say that N-Gram Approach is not very helpful for the Dialog based Error types. It is very sensible to the length of the sending message. When the length of the message increases, the N-Gram approach will lead to wrong results.

For the future, it would be very helpful to add some new algorithms in order to resolve long messages like the following:

"*LUTFEN BENIM TRIBUN PAKETIMI IPTAL EDERMISINIZ*"

One implementation could be to concentrate on dialog based conversation techniques in order to handle these types of errors.

There would be also some other improvements for our approach. For example, recall that we have used only first part of the Bayesian formula (Equation 2.9), so for the future work, using Bayesian prior information could be very helpful in order to increase SMS parsing performance.

However our approach is very useful to resolve the small typing problems. In fact most of the cases the customers make this kind of mistakes. As mentioned in Chapter 4, customers' typing problems are automatically resolved by this approach.

Finally, overall results can be seen as satisfactory, because the total valid messages percentage has increased from 68 per cent to 77 per cent. This means that company's successful customer subscription ratio has increased 13 per cent, which is directly related to the profit of the Company and the customer satisfaction.

APPENDIX A: TURTLE SYSTEM

A.1. Overview of the Turtle System

Turtle is mainly an integration project including B2B and B2C features. One of its main requirements is to integrate different parties using different technologies, with minimum set of requirements.

In fact, it consists of a technical architecture that bridges the gap between the external systems and the Telco infrastructure, foundation components, modules, and guidelines to achieve the synchronization between each other. You can see black box diagram of the Turtle system in Figure A.1.

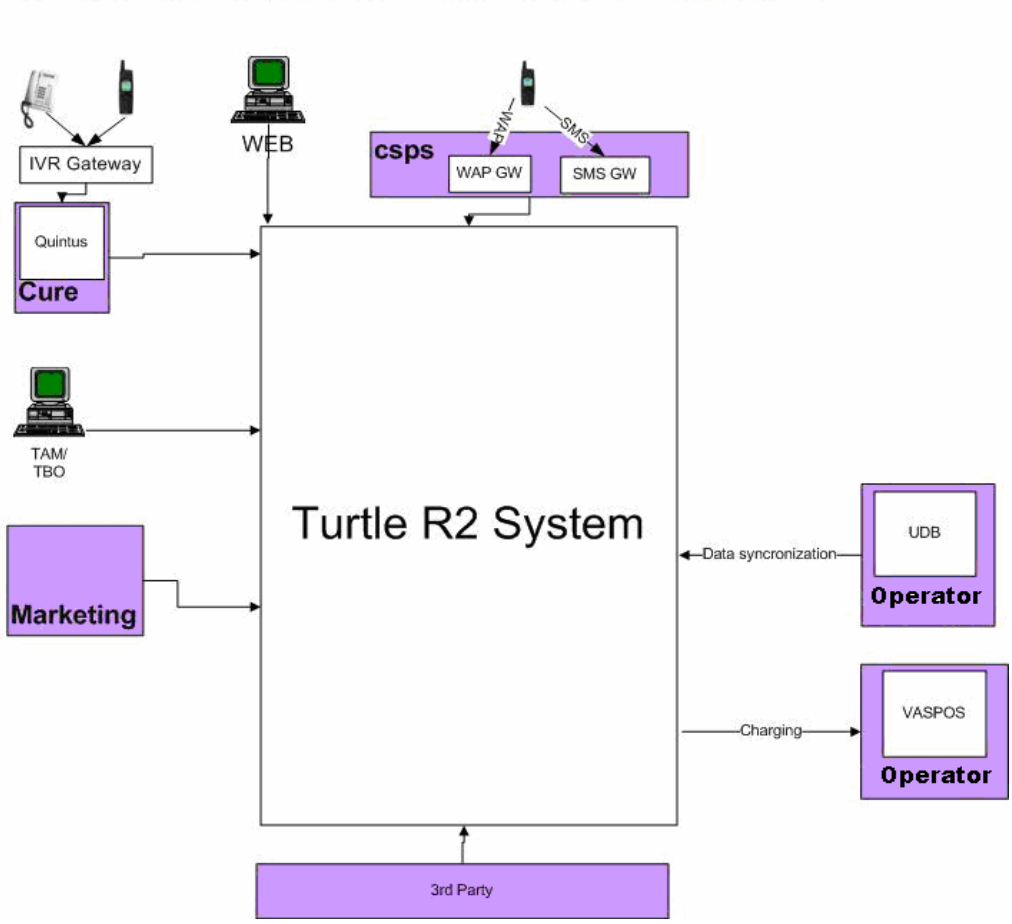

Figure A.1. Black box diagram of the Turtle system

The Turtle system architecture can be decomposed (Figure A.2.) into four main parts:

- Presentation Layer
- Business Logic Layer
- Data Model Layer
- Communication Layer

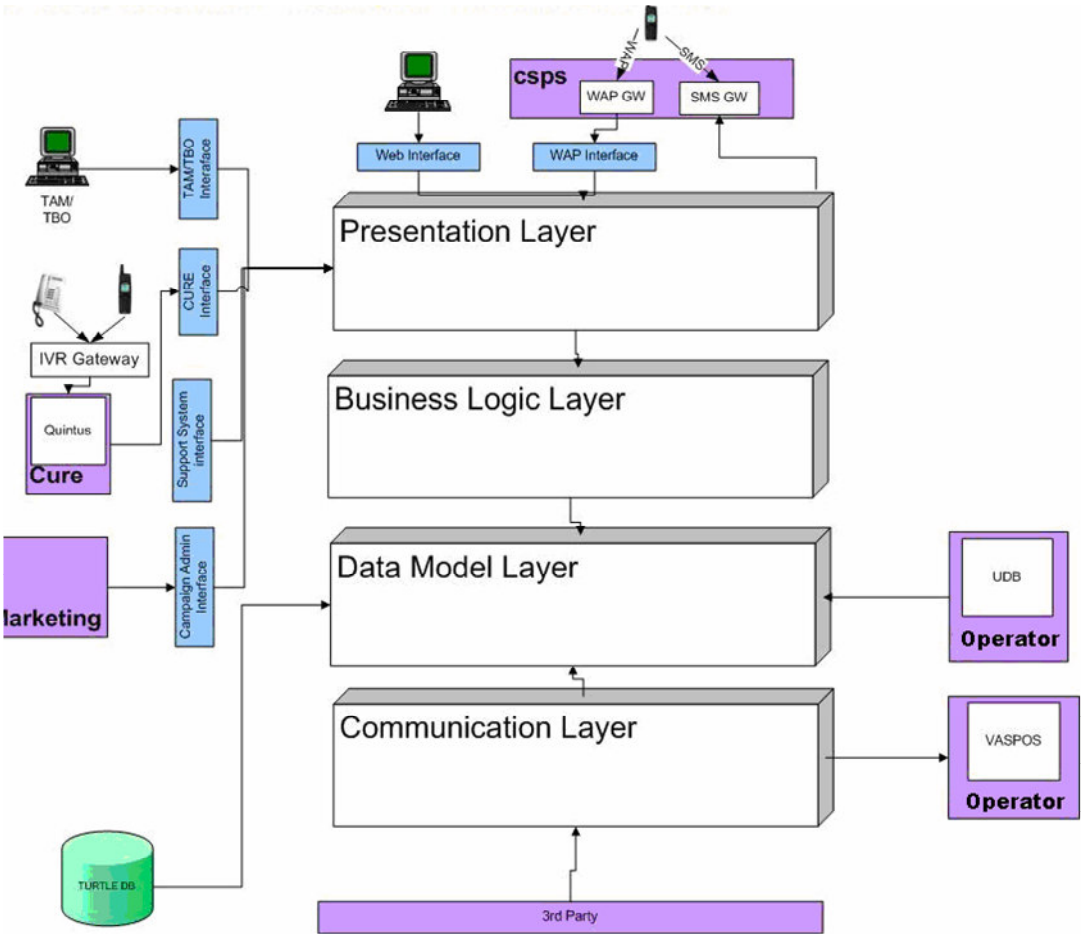

Figure A.2. Turtle n-tier architecture diagram

A.1.1. Presentation Layer

Turtle Release 2.0's SMS/WEB/WAP User interfaces, TAM/TBO, CURE, Support, and Marketing Operation interfaces are in the Presentation Layer.

As seen in Figure A.2., SMS is one of the channels for the Presentation Layer. In fact, 80 per cent of Turtle user subscriptions are made via the SMS channel. Thus, customer satisfaction mainly has to do with SMS operations.

A.1.2. Business Logic Layer

Turtle Release 2.0 has many features, such as campaign administration, charging, user subscription, etc. All of these features' logic and rules are in this layer.

A.1.3. Data Model Layer

In order to keep Campaign, Packets, Service, Parameter Definitions, User subscription and Charging Data, 3rd party's synchronization Data Turtle Release 2.0 needs a powerful Data Model to handle all the new and old features of the system.

The business logic layer uses and stores Turtle data in the data model layer. Data are stored in two main formats: XML and RDBMS.

XML is used mostly for the data representation for example Campaign list in the Web interface is made by using XSLT transformation of the Campaign XML.

User's product information is stored in the Database; Oracle 9i is used as RDBMS for the turtle.

A.1.4. Communication Layer

This layer is responsible for communication with external systems such as Telco's billing interface or third Parties subscription Information system interfaces.

The communication protocol depends on the system, for example, billing interface is based on HTTP proxy, while third party integration system is made with Web Services technology.

APPENDIX B: SMS HANDLING SYSTEM

SMS Handling Subsystem is responsible for parsing Users' Short Messages from SMS Gateway. In fact our approach is working behind the SMS Handling system.

SMS Handling subsystem has the architecture, shown in fig B.1

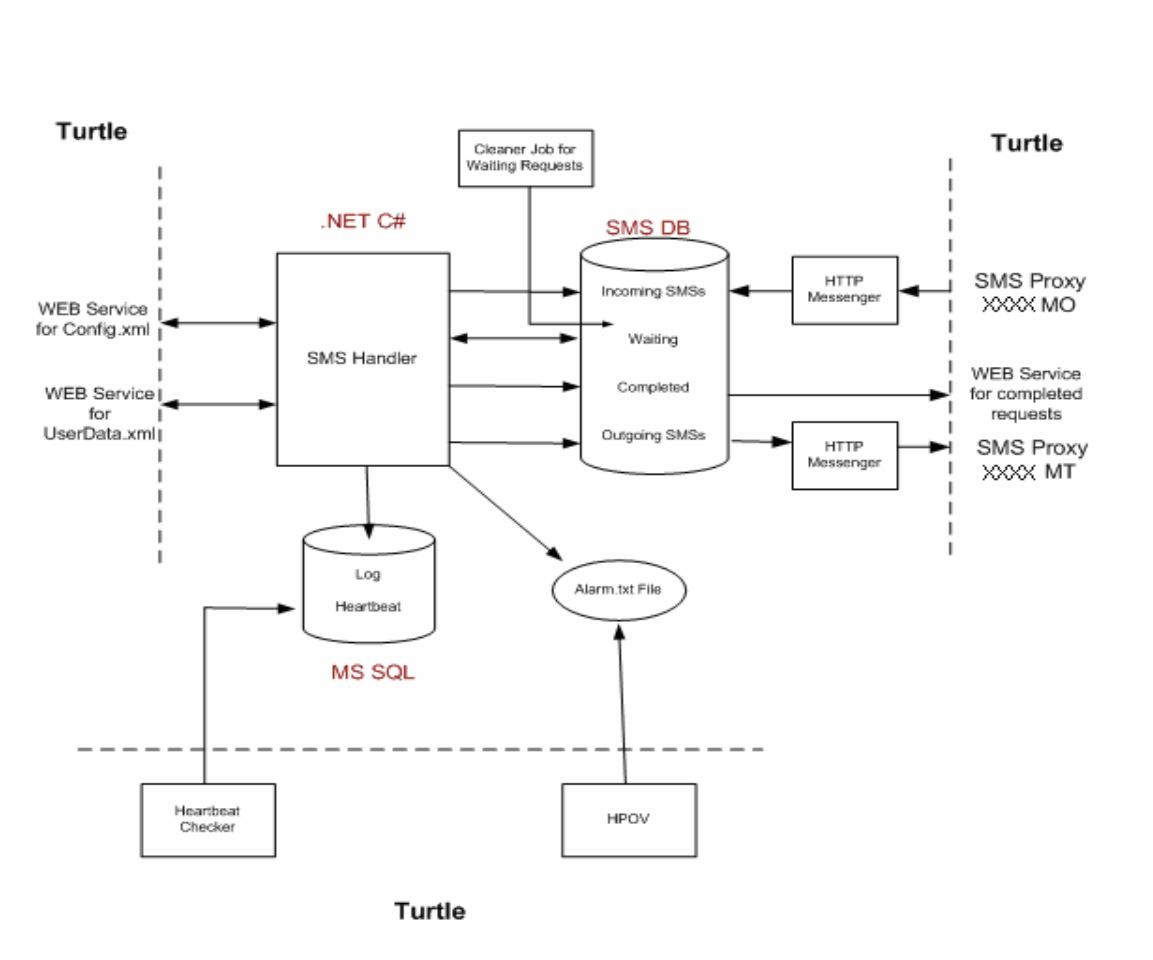

Figure B.1. SMS handling system.

B.1. Some benefits of the SMS Handling Subsystem

In the current System of Turtle, the SMS parsing operation is not flexible. It uses restricted format.

In the SMSHandling subsystem, the following benefits will be provided:

- Faster subscription process.

- Package and service parameters will have an option which will specify whether a given parameters can be taken from the user profile; this parameter will specify the field of the profile to be used.

- Easier to use: The user will be allowed to enter the parameters in any order. A user will be able to subscribe in one SMS by sending the subscription keyword and as many parameters as he can. The system should look at parameters if any and then ask for the missing ones only. The user will skip package name if the brand name is sufficient. If a given brand name is (ABONE SHOWTV), if a given brand name maps to more than one package the system will prompt the user with the package first. If there are garbage keywords in the SMS they will be ignored. In case of a keyword conflict the user will be asked to specify the action he wants to perform. When the permissible value of a parameter is one of value of a finite list of discrete value, the user will be able to select the index of the value instead of entering the value.

B.1.1. Additional features

Support for a new keyword to continue a service without entering any new parameter. For example suppose that we accept the keyword “Devam”, on receipt of that keyword the system will look at packages/services which are marked to accept such a command and renew the subscription if appropriate (need to be defined a bit more.)

Query of the user’s status at Turtle.

The messages which are part of the subscription flow will be stored in a user-readable format that can be changed with any text editor. Performance monitor and integration to HPOV.

APPENDIX C: IMPLEMENTATION DETAILS

C.1. Microsoft .NET Overview

All business logic in the Turtle system has been implemented in Microsoft .NET Framework version 1.1. For Application server Microsoft .NET Framework 1.1 used as an Integrated Development environment. In order to handle Service, Package, Campaign structures, SMS Handler will use Web service technology provided by Service, Package, and Campaign system.

Because of this, the code for this study is also implemented in .NET language for integration purposes.

.NET is defined [4] as the Microsoft Web services strategy to connect information, people, systems, and devices through software.

Integrated across the Microsoft platform, .NET technology provides the ability to quickly build, deploy, manage, and use connected, security-enhanced solutions with Web services. .NET-connected solutions enable businesses to integrate their systems more rapidly and in a more agile manner and help they realize the promise of information anytime, anywhere, on any device.

C.2. Tagging Process

As mentioned in the thesis, Tagging process was one of the main steps of the evaluation part. There were more than 6 million SMS messages to evaluate in order to regroup them to calculate statistics values. Therefore, it was necessary to write a utility program built in via graphical user interface to process all data in the limited time. To do this, .NET Framework libraries was used. A simple form based data entry tool was created. The form screenshot is in Fig.C.1.

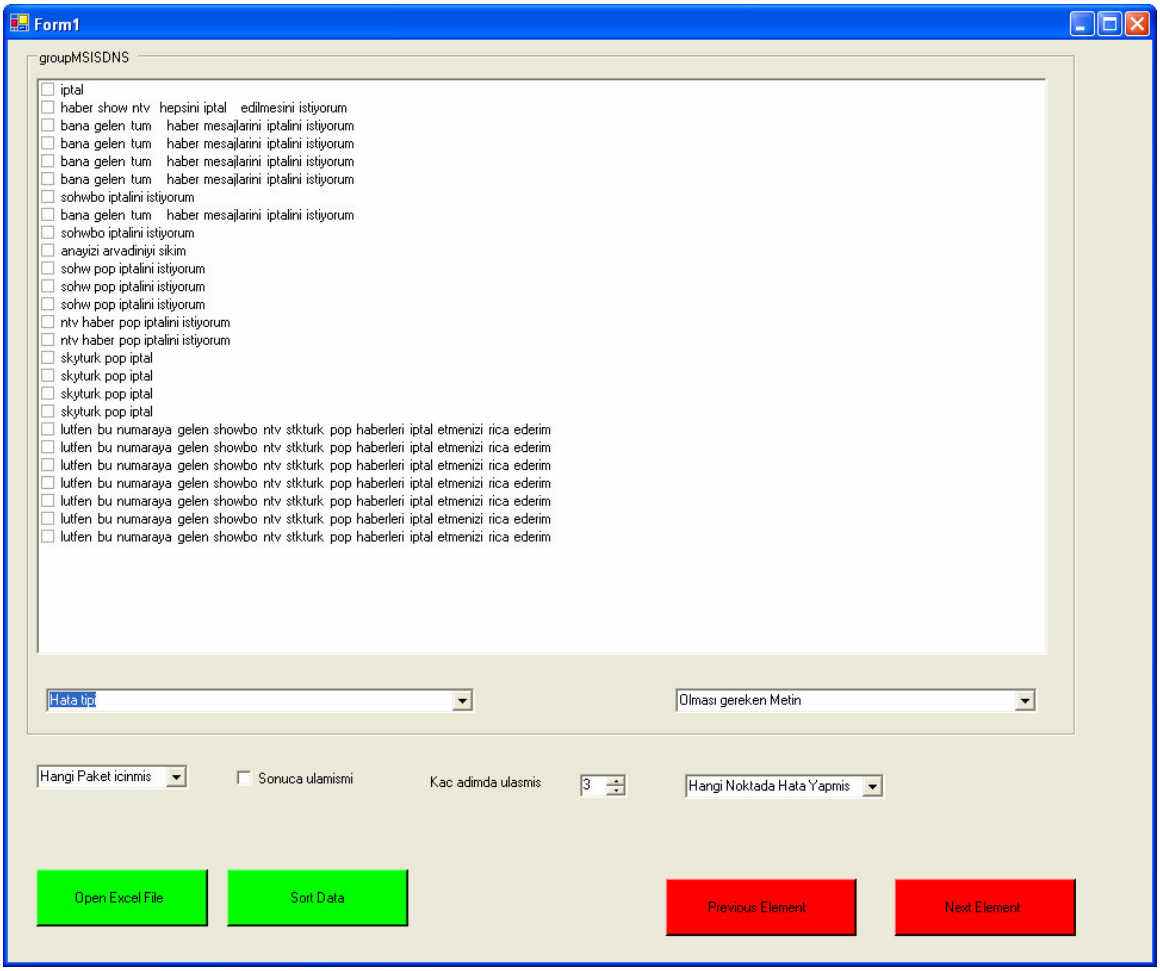

Figure C.1. Tagging utility screen shot

C.2.1. Tagging Strategies and Rules

In order to process large amount of data in a very limited time some strategies were employed in the Tagging process. Firstly messages sent by the same user within a same day are regrouped and shown at the same time in the GUI screen. These regrouped messages are called “SEQUENCE”, Every subscription message step is in fact an element of the “SEQUENCE” So, seeing all the messages at the same time gives more clear information to us about to learn users’ aims.

There are also some other elements to make programs more user friendly such as” possible correct text” area,” Error type”,” Package identifier of the sequence”.

Finally, all error messages are categorized like following:

- **Typing errors:** Errors occurred due to pressing wrong button combinations. These are generally simple errors, which changes one or two letters in whole message.

  Examples:

  - ABONE IPTAlL
  - ABONE TRIBN
  - Iptal haber skytrl

- **Mobile device specific problems:** Some of the mobile devices add special characters to end of the messages, or some cannot send words with Turkish characters, so that devices change some characters.

  Examples:

  - 0PTAL (means İPTAL)
  - ^HUBO (means ŞHUBUO)
  - abone fl□rt (means abone flört)

- **Proper sentences:** This error type occurs when individuals do not use keywords. Except keywords, they write sentences to express their request. Examples:

  - Kontorumu istyorummmm
  - Gunluk burc iptal edilsin
  - Pop Muzik Abonemi iptal etmek istiyorum

- **Reflected messages:** This case is generated, if user forwards the message, which the system sent, back to the system.

  Examples:

  - Flort Paketi aboneligin bulunmamaktadir. Sen de Flort Paketine uye olmak istiyorsan ABONE FLORT yaz XXXXye gonder.

  - KampusCelliye 2 mujdemiz var! 30Nisana kadar aboneligini yenile 50kampusici SMS kazan. Ustelik 6Nisandan itibaren grupici dk.si 1kontor/8,5Ykra konus
  - SuperSifreniz:473485. Guvenliginiz icin sifrenizi kimse ile paylasmayiniz. Bu mesaj icin ucret alinmamaktadir.

- Dialog based messages: user interacts with system, as there is a human-being reading and replying messages.

  Examples:

  - YOK KALSIN AMA DUR DUR KABUL ED0YOM NASIL YAPCAM
  - NE ZAMAN IPTAL OLACAK ASTROLOJI
  - Sen iptal ettir kayit oldugu yerleri...

- Inappropriate messages: the words are legal and exist in lexicon, but total message does not satisfy meaningful sentence.

  Examples:

  - ABONE IPTAL
  - HABER FLORT
  - IPTAL EVET

- Wrong services: messages that was sent to wrong service number.

  Examples:

  - Avanskontur
  - KONTORBIZDEN
  - WAP AYAR NOKIA

- Undefined messages: the messages which does not mean anything and/or irrelevant with the system.

  Examples:

  - Bentrkiyedeyasamucgeniolusturdum

- Ama seni seviyorum

After selecting the error category of the messages, if we can predict what user requests then, we enter this to the meaningful term to the system and save messages to DB.

After tagging messages with this GUI data in the table looks like Figure C.2.1. as follows:

MSGID	MESSAGEBODY	MEAN
25475686	ABONE	ABONE
25475687	IPTAL SAMSUNSPOR	IPTAL SAMSUNSPOR
25475688	BLOKE 8090	
25475689	AJUDJGJM	
25475690	NASSSIN	
25475692	KB	
25475693	Tribun.Paket	
25475694	IPTAL HABER NTV	IPTAL HABER NTV
25475695		
25475696	Gagaga	
25475699	null	
25475701	A,dpmpmpma1jrlar	
25475704	ABONE	ABONE
25475706	iptal	IPTAL
25475707	null	
25475708	ABONE POPMUZIK	ABONE POPMUZIK
25475709	0ptal aa	IPTAL AA
25475710	IPTAL	IPTAL
25475711	ABONE POPMUZIK	ABONE POPMUZIK
25475713	IPTAL	IPTAL
25475714	IPTAL	IPTAL
25475715	IPTAL HABER SHOWTV	IPTAL HABER SHOWTV
25475716	ABONE POPMUZIK	ABONE POPMUZIK
25475717	IPTAL	IPTAL
25475718	ARA	
25475719	ABONE HABER NTV	ABONE HABER NTV
25475720	Iptal.Haber	
25475721	Paket	PAKET
25475723	null	
25475724	IPTAL	IPTAL
25475725	Cevap: EVET ---Orijinal mesaj--- TAHMIN MACKOLIK paket	PAKET MACKOLIK 24
25475726	Ororpu	
25475727	BLOKE 8O9O	
25475728	Abone Hava 34	ABONE 34
25475729	Haber	HABER
25475730	ABONE POPMUZIK	ABONE POPMUZIK
25475731	Kontrbizden	
25475732	null	

Figure C.2. Tagged data in the database

APPENDIX D: RUNTIME ENVIRONMENT

D.1. Platform

Because of the confidentiality and the company policies, we cannot give all hardware infrastructure information here.

However, to give some idea, we can cite some information about the runtime environment. In the runtime environment there are Windows and Unix systems working together.

Their Versions and capacities are as follows:

Unix :
Sun Solaris 8
Sun Cluster 3.0
BEA Weblogic 8.1 SP1
Oracle 9.2.0.4
Apache Webserver 1.3.2.2.

Microsoft :
.NET Framework 1.1
Window Server 2003 (Enterprise)
MS SQL Server 2000

REFERENCES

1. Jurafsky, D. and J. H. Martin, *Speech and Language processing*, Prentice Hall (ISBN: 0-13-095069-6), 2000.

2. Brown, M. K. and A. Kellner, *Stochastic Language Models (N-Gram) Specification*, http://www.w3.org/TR/2001/WD-ngram-spec-20010103/, 2001.

3. Keh-Yih Su, K.Y., T. H. Chiang and J. S. Chang, "An overview of Corpus-Based Statistics oriented (CBSO) Techniques for Natural Language Processing", *Computational Linguistic and Chinese Language Processing*, Vol.1, no.1, August 1996.

4. Microsoft, "What is .NET?", *Basics of .NET*, http://www.microsoft.com/Net/Basics.aspx, 2005.

5. Kukich, K., *Techniques for automatically correcting words in text*., Comput. Surv., 24, 4, 377-439., 1992.

6. Abney, S., *Encyclopedia of cognitive science*, Macmillan Reference, 2000.

REFERENCES NOT CITED

Apache Software foundation, *Apache Lucene Project 1.4.3*, http://lucene.apache.org/java/ docs/, 2005.

Hatcher, E. and O. Gospodnetic, *Lucene in Action*, Manning Publications (ISBN: 1932394281), 2004.

Keselj, V., *Text::Ngrams - Flexible Ngram analysis (for characters, words, and more),* http://www.ai.mit.edu/courses/6.863/Ngrams.html, 2004.

Stutz, J., W. Taylor and P. Cheeseman, *AutoClass C - General Information*, NASA, Ames Research Center:, http://ic-www.arc.nasa.gov/ic/projects/bayes-group/ autoclass/autoclass-c-program.html, 1988.

Pearl, J., *Probabilistic Reasoning in intelligent Systems: Networks of Plausible Inference*, Morgan Kauffman Publishers (ISBN: 1558604790), 1988.

Heckerman, D. and E. Horvitz, *Inferring Informational Goals from Free-Text Queries: A Bayesian Approach*, Decision Theory & Adaptive Systems Group, Microsoft Research, http://research.microsoft.com/research/dtg/horvitz/aw.htm, 1998.