Automatic Semantic Grouping in a Spoken Language User Interface Toolkit

Hassan Alam, Hua Cheng, Rachmat Hartono, Aman Kumar, Paul Llido, Crystal Nakatsu, Huy Nguyen, Fuad Rahman, Yuliya Tarnikova, Timotius Tjahjadi and Che Wilcox

BCL Technologies Inc.
Santa Clara, CA 95050 U.S.A.
fuad@bcltechnologies.com

Abstract

With the rapid growth of real application domains for NLP systems, there is a genuine demand for a general toolkit from which programmers with no linguistic knowledge can build specific NLP systems. Such a toolkit should provide an interface to accept sample sentences and convert them into semantic representations so as to allow programmers to map them to domain actions. In order to reduce the workload of managing a large number of semantic forms individually, the toolkit will perform what we call semantic grouping to organize the forms into meaningful groups. In this paper, we present three semantic grouping methods: similarity-based, verb-based and category-based grouping, and their implementation in the SLUI toolkit. We also discuss the pros and cons of each method and how they can be utilized according to the different domain needs.

1 Introduction and Motivation

With the improvement of natural language processing (NLP) and speech recognition techniques, spoken language will become the input of choice for software user interfaces, as it is the most natural way of communication. In the mean time, the domains for NLP systems, especially those handling speech input, have grown rapidly in recent years. However, most computer programmers do not have enough linguistic knowledge to develop an NLP system to handle speech input. There is a genuine demand for a general toolkit from which programmers with no linguistic knowledge can rapidly build speech based NLP systems to handle their domain specific problems more accurately (Alam, 2000). The toolkit will allow programmers to generate Spoken Language User Interface (SLUI) front ends for new and existing applications using, for example, a program-through-example method. In this methodology, the programmer will specify a set of sample input sentences or a domain corpus for each task. The toolkit will then organize the sentences by meaning and even generate a large set of syntactic variations for a given sentence. It will also generate the code that takes a user's spoken request and executes a command on an application. This methodology is similar to using a GUI toolkit to develop a graphical user interface so that programmers can develop GUI without learning graphics programming. Currently this is an active research area, and the present work is funded by the Advanced Technology Program (ATP) of the National Institute of Standards and Technology (NIST).

In the program-through-example approach, the toolkit should provide an interface for the programmers to input domain specific corpora and then process the sentences into semantic representations so as to capture the semantic meanings of the sentences. In a real world application, this process results in a large number of semantic forms. Since the programmers have to manually build the links between these forms and their specific domain actions, they are likely to be overwhelmed by the workload imposed by the large number of individual semantic forms. In order to significantly reduce this workload, we can organize these forms in such a way so that the programmers can manipulate them as groups rather than as individual items. This will speed up the generation process of the domain specific SLUI system. We call this process the *semantic grouping* process.

One straightforward way to group is to organize different syntactic forms expressing the same meaning together. For example,

(1.1) I want to buy this book online.
(1.2) Can I order this book online?
(1.3) How can I purchase this book online?
(1.4) What do I need to do to buy this book online?

The semantic forms of the above sentences may not be the same, but the action the programmer has in mind in an e-business domain is more or less the same: to actually buy the book online. In addition to the above sentences, there are many variations that an end-user might use. The embedded NLP system should be able to recognize the similarity among the variations so that the SLUI system can execute the same command upon receiving the different queries. This requires a group to contain only sentences with the same meaning. However in real applications, this might be difficult to achieve because user requests often have slight differences in meaning.

This difficulty motivates a different style for semantic grouping: organizing the semantic forms into groups so that those in the same group can be mapped roughly to the same action. The action can be either a command, e.g., *buy* something, or concerning an object, e.g., different ways of gathering information about an object. For example, sentence (1.5) would be grouped together with the above example sentences because it poses the same request: *buy books*; and sentences (1.6) to (1.8) would be in one group because they are all about price information.

(1.5) I want to buy the latest book about e-business.

(1.6) Please send me a price quote.
(1.7) What is the reseller price?
(1.8) Do you have any package pricing for purchasing multiple products at once?

This type of grouping is the focus of this paper. We propose three grouping methods: similarity-based grouping, verb-based grouping and category-based grouping. The process of grouping semantic forms is domain dependent and it is difficult to come up with a generally applicable standard to judge whether a grouping is appropriate or not. Different grouping techniques can give programmers different views of their data in order to satisfy different goals.

This paper is organized into 6 sections. In Section 2, we briefly describe the system for which the grouping algorithms are proposed and implemented. Section 3 presents the three grouping methods in detail. In Section 4, we describe how the algorithms are implemented in our system. We test the methods using a set a sentences from our corpus and discuss the pros and cons of each method in Section 5. Finally, in Section 6, we draw conclusions and propose some future work.

2 SLUITK

As mentioned in the previous section, the Spoken Language User Interface Toolkit (SLUITK) allows programmers with no linguistic knowledge to rapidly develop a spoken language user interface for their applications. The toolkit should incorporate the major components of an NLP front end, such as a spell checker, a parser and a semantic representation generator. Using the toolkit, a programmer will be able to create a system that incorporates complex NLP techniques such as syntactic parsing and semantic understanding.

2.1 The Work Flow

Using an Automatic Speech Recognition (ASR) system, the SLUITK connects user input to the application, allowing spoken language control of the application. The SLUITK generates semantic representations of each input sentence. We refer to each of these semantic representations as a *frame*, which is basically a predicate-argument representation of a sentence.

The SLUITK is implemented using the following steps:

1. SLUITK begins to create a SLUI by generating semantic representations of sample input sentences provided by the programmer.
2. These representations are expanded using synonym sets and other linguistic devices, and stored in a Semantic Frame Table (SFT). The SFT becomes a comprehensive database of all the possible commands a user could request a system to do. It has the same function as the database of parallel translations in an Example-based machine translation system (Sumita and Iida, 1991).
3. The toolkit then creates methods for attaching the SLUI to the back end applications.
4. When the SLUI enabled system is released, a user may enter an NL sentence, which is translated into a semantic frame by the system. The SFT is then searched for an equivalent frame. If a match is found, the action or command linked to this frame is executed.

In a real application, a large number of frames might be generated from a domain corpus. The semantic grouper takes the set of frames as the input and outputs the same frames organized in a logical manner.

2.2 The Corpus

We use a corpus of email messages from our customers for developing and testing the system. These email messages contain questions, comments and general inquiries regarding our document-conversion products. We modified the raw email programmatically to delete the attachments, HTML and other tags, headers and sender information. In addition, we manually deleted salutations, greetings and any information that was not directly related to customer support. The corpus contains around 34,640 lines and 170,000 words. We constantly update it with new email from our customers.

We randomly selected 150 sentential inquiries to motivate and test the semantic grouping methods discussed in this paper.

3 Semantic Grouping

We have mentioned in Section 1 that grouping semantic frames is domain dependent. Grouping depends on the nature of the application and also the needs of the domain programmer. Since this is a real world problem, we have to consider the efficiency of grouping. It is not acceptable to let the programmer wait for hours to group one set of semantic forms. The grouping should be fairly fast, even on thousands of frames.

These different considerations motivate several grouping methods: similarity-based grouping, verb-based grouping and category-based grouping. In this section, we describe each of these methods in detail.

3.1 Similarity-based Grouping

Similarity-based grouping gathers sentences with similar meanings together, e.g., sentences (1.1) to (1.4). There is a wide application for this method. For example, in open domain question-answering systems, questions need to be reformulated so that they will match previously posted questions and therefore use the cached answers to speed up the process (Harabagiu et al., 2000).

The question reformulation algorithm of Harabagiu *et al*. tries to capture the similarity of the meanings expressed by two sentences. For a given set of questions, the algorithm formulates a similarity matrix from which reformulation classes can be built. Each class represents a class of equivalent questions.

The algorithm for measuring the similarity between two questions tries to find lexical relationships between every two questions that do not contain stop words. The algorithm makes use of the WordNet concept hierarchy (Fellbaum, 1998) to find synonym and hypernym relations between words.

This algorithm does not infer information about the meanings of the questions, but rather uses some kind of similarity measurement in order to simulate the commonality in meaning. This is a simplified approach. Using different threshold, they can achieve different degrees of similarity, from almost identical to very different.

This method can be used for similarity-based grouping to capture the similarity in meanings expressed by different sentences.

3.2 Verb-based Grouping

Among the sentences normally used in the e-business domain, imperative sentences often appear in sub-domains dominated by command-and-control requests. In such an application, the verb expresses the command that the user wants to execute and therefore plays the most important role in the sentence. Based on this observation, a grouping can be based on the verb or verb class only. For example, sentences with *buy* or *purchase* etc. as the main verbs are classified into one group whereas those with *download* as the main verb are classified into a different group, even when the arguments of the verbs are the same.

This is similar to sorting frames by the verb, taking into account simple verb synonym information.

3.3 Category-based Grouping

Since SLUITK is a generic toolkit whereas the motivation for grouping is application dependent, we need to know how the programmer wants the groups to be organized. We randomly selected 100 sentences from our corpus and asked two software engineers to group them in a logical order. They came up with very different groups, but their thoughts behind the groups are more or less the same. This motivates the category-based grouping.

This grouping method puts less emphasis on each individual sentence, but tries to capture the general characteristics of a given corpus. For example, we want to group by the commands (e.g., *buy*) or objects (e.g., a software) the corpus is concerned with. If a keyword of a category appears in a given sentence, we infer that sentence belongs to the category. For example, sentences (1.6) to (1.8) will be grouped together because they all contain the keyword *price*.

These sentences will not be grouped together by the similarity-based method because their similarity is not high enough, nor by the verb-based method because the verbs are all different.

4 Grouping in SLUITK

Because we cannot foresee the domain needs of the programmer, we implemented all three methods in SLUITK so that the programmer can view their data in several different ways. The programmer is able to choose which type of grouping scheme to implement.

In the question reformulation algorithm of (Harabagiu, et al. 2000), all words are treated identically in the question similarity measurement. However, our intuition from observing the corpus is that the verb and the object are more important than other components of the sentence and therefore should be given more weight when measuring similarity. In Section 4.1, we describe our experiment with the grouping parameters to test our intuition.

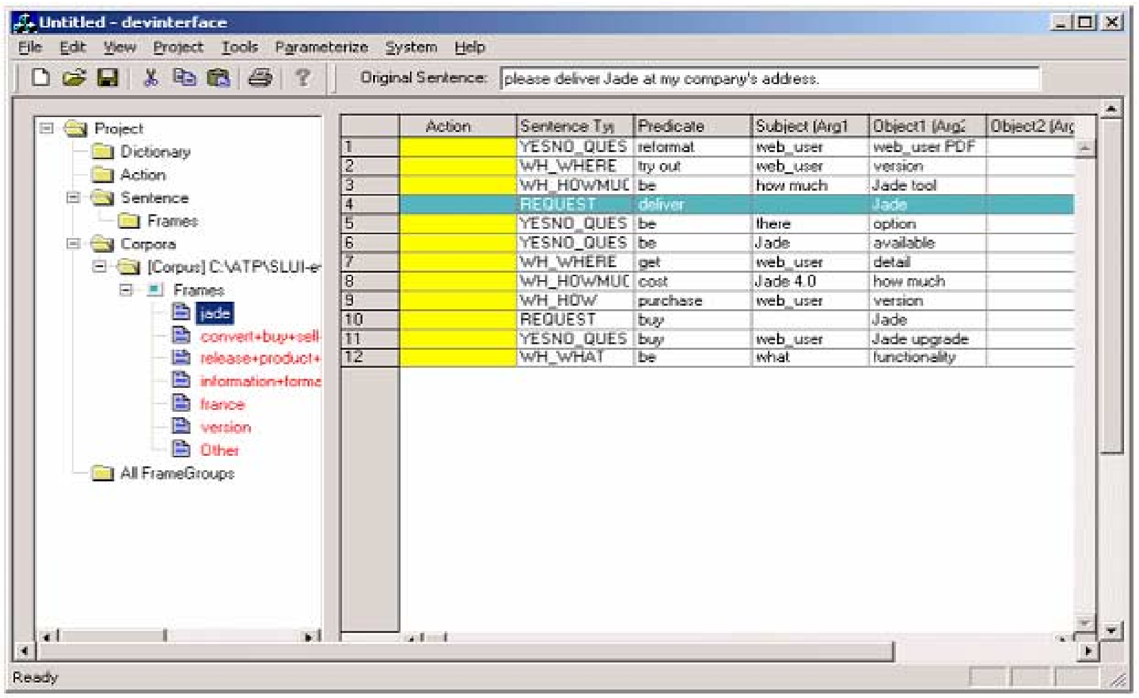
4.1 Experimenting with Parameters

We think that there are two main parameters affecting the grouping result: the weight of the syntactic components and the threshold for the similarity measurement in the similarity-based method. Using 100 sentences from our corpus, we tried four different types of weighting scheme and three thresholds with the category-based methods. Human judgment on the generated groups confirmed our intuition that the object plays the most important role in grouping and the verb is the second most important. The differences in threshold did not seem to have a significant effect on the similarity-based grouping. This is probably due to the strict similarity measurement.

This experiment gives us a relatively optimal weighting scheme and threshold for the similarity-based grouping.

One relevant issue concerns the simplification of the semantic frames. For a sentence with multiple verbs, we can simplify the frame based on the verbs used in the sentence. The idea is that some verbs such as action verbs are more interesting in the e-business domain than others, e.g., *be* and *have*. If we can identify such differences in the verb usage, we can simplify the semantic frames by only keeping the interesting verb frames. For example, in the following sentences, the verb *buy* is more interesting than *be* and *want*, and the generated semantic frames should contain only the frame for *buy*.

(4.1) Is it possible to <u>buy</u> this software online?
(4.2) I want to <u>buy</u> this software online.



Original Sentence: please deliver Jade at my company's address.

	Action	Sentence Ty	Predicate	Subject (Arg1	Object1 (Arg2	Object2 (Arg
1		YESNO_QUES	reformat	web_user	web_user PDF	
2		WH_WHERE	try out	web_user	version	
3		WH_HOWMUC	be	how much	Jade tool	
4		REQUEST	deliver		Jade	
5		YESNO_QUES	be	there	option	
6		YESNO_QUES	be	Jade	available	
7		WH_WHERE	get	web_user	detail	
8		WH_HOWMUC	cost	Jade 4.0	how much	
9		WH_HOW	purchase	web_user	version	
10		REQUEST	buy		Jade	
11		YESNO_QUES	buy	web_user	Jade upgrade	
12		WH_WHAT	be	what	functionality	

Figure 1: A screen shot of SLUITK

We make use of a list of stop-words from (Frakes, 1992) in order to distinguish between interesting and uninteresting verbs. We look for frames headed by stop-words and follow some heuristics to remove the sub-frame of the stop-word. For example, if there is at least one verb that is not a stop-word, we remove all other stop-words from the frame. In the sentence *[Is it possible to] buy the software in Germany?*, *be* is a stop-word, so only the frame for *buy* is kept. This process removes the redundant part of a frame so that the grouping algorithm only considers the most important part of a frame.

4.2 Implementation in SLUITK

Figure 1 shows a screen shot of the interface of the SLUITK, which shows several grouped semantic frames. In this section, we give more detail about the implementation of the three grouping methods used in SLUITK.

Similarity-based grouping

Similar to (Harabagiu, et al. 2001), our similarity-based grouping algorithm calculates the similarity between every two frames in the input collection. If the similarity is above a certain threshold, the two frames are considered similar and therefore should be grouped together. If two frames in two different groups are similar, then the two groups should be combined to a single group. The central issue here is how to measure the similarity between two frames.

Since we have found that some syntactic components are more important to grouping than others, we use a weighted scheme to measure similarity. For each frame, all words (except for stop-words) are extracted and used for similarity calculation. We give different weights to different sentence components. Since in an e-business domain, the verb and the object of a sentence are usually more important than other components because they express the actions that the programmers want to execute, or the objects for which they want to get more information, the similarity of these components are emphasized through the weighting scheme. The similarity score of two frames is the summation of the weights of the matched words.

There is a match between two words when we find a lexical relationship between them. We extend the method of (Harabagiu, et al. 2000) and define a lexical relationship between two words W1 and W2 as in the following:

	Similarity-based	*Verb-based*	*Category-based*
Group Size	small	small	large
Number of Groups	large	large	variable
Speed	slow on large corpus	fast	slow on large corpus
Application	general	command-and-control only	general

Table 1 : Comparison of grouping methods

1. If W1 and W2 have a common morphological root. Various stemming packages can be used for this purpose, for example, Porter Stemmer (Porter, 1997).
2. If W1 and W2 are synonyms, i.e., W2 is in the WordNet synset of W1.
3. If the more abstract word is a WordNet hypernym of the other.
4. If one word is the WordNet holonym of the other (signaling *part of*, *member of* and *substance of* relations);
5. If W1 is the WordNet antonym of W2.

Domain specific heuristics can also be used to connect words. For example, in the e-business domain, *you* and *I* can be treated as antonyms in the following sentences:

(4.3) Can I buy this software?
(4.4) Do you sell this software?

When none of the above is true, there is no lexical relation between two given words.

Because the similarity-based grouping needs to consult WordNet frequently for lexical relations, it becomes very slow for even a few hundred frames. We have to change the algorithm to speed up the process, as it is too slow for real world applications.

Instead of comparing every two frames, we put all the words from an existing group together. When a new frame is introduced, we compare the words in this new frame with the word collection of each group. The similarity scores are added up as before, but it needs to be normalized over the number of words in the collection. When the similarity is above a certain threshold, the new frame is classified as a member of the group. This significantly reduces the comparison needed for classifying a frame, and therefore reduces the number of times WordNet needs to be consulted.

We compared this improved algorithm with the original one on 30 handcrafted examples; the generated groups are very similar.

Verb-based grouping

The verb-based grouping implementation is fairly straightforward and has been described in Section 3.2.

Category-base grouping

For the category-based method, we first count all the non stop-words in a given corpus and retrieve a set of most frequent words and their corresponding word classes from the corpus. This process also makes use of the WordNet synonym, hypernym, holonym and antonym information. These word classes form the categories of each group. We then check the verbs and objects of each sentence to see if they match these words. That is, if a category word or a lexically related word appears as the verb or the object of a sentence, the sentence is classified as a member of that group. For example, we can pick the most frequent 20 words and divide the corpus into 21 groups, where the extra group contains all sentences that cannot be classified. The programmer can decide the number of groups they want. This gives the programmer more control over the grouping result.

5 Discussion

We tested the three methods on 100 sentences from our corpus. We had 5 people evaluate the generated groups. They all thought that grouping was a very useful feature of the toolkit. Based on their comments, we summarize the pros and cons of each method in Table 1.

The similarity-based grouping produces a large number of groups, most of which contain only one sentence. This is because there are usually several unrelated words in each sentence, which decreases the similarity scores. In addition, using WordNet we sometimes miss the connections between lexical items. The verb-based grouping produces slightly larger groups, but also produces many single sentence groups. Another problem is that when sentences contain only stop-word verbs, e.g., *be*, the group will look rather arbitrary. For example, a group of sentences with *be* as the main verb can express completely different semantic meanings. The small group size is a disadvantage of both methods. The number of groups of the category-based grouping can change according to the user specification. In general it produces less groups than the other methods and the group size is much larger, but the size becomes smaller for less frequent category words.

Both the similarity-based and category-based grouping methods are slow because they frequently need to use WordNet to identify lexical relationships. The verb-based method is much faster, which is the primary advantage of this method.

The verb-based method should be used in a command-and-control domain because it requires at least one non stop-word verb in the sentence. However, it will have a hard time in a domain that needs to handle questions. From the point of view of assigning a domain specific action to a group, this grouping is the best because each verb can be mapped to an action. Therefore, the programmer can link an action to each group rather than to each individual frame. When the group size is relatively large, this can greatly reduce the workload of the programmer.

The category-based method produces a better view of the data because the sentences in each group seem to be consistent with the keywords of the category. The disadvantage is that it is difficult to link a group to a single action, and the programmer might have to re-organize the groups during action assignment.

The similarity-based method did not perform well on the testing corpus, but it might work better on a corpus containing several different expressions of the same semantic information.

In summary, each method has its advantages and disadvantages. The decision of which one to choose depends mainly on the needs of the domain programmer and the composition of the input corpus.

6 Conclusions and Future Work

In this paper we propose semantic grouping as a way to solve the problem of manipulating semantic frames in developing a general Spoken Language User Interface Toolkit (SLUITK). We introduced three methods for grouping semantic frames generated by the NLP components of the toolkit. We tested the methods and discussed the advantages and disadvantages of each method. Since the judgment of the grouping result is application dependent, the methods co-exist in our SLUITK to suit the requirement of different applications.

Future work includes improving the efficiency and accuracy of the methods and testing them on a larger corpus.

References

Alam H. (2000) *Spoken Language Generic User Interface (SLGUI).* Technical Report AFRL-IF-RS-TR-2000-58, Air Force Research Laboratory, Rome.

Fellbaum C. (1998) *WordNet, An Electronic Lexical Database*, The MIT Press, Cambridge, Massachusetts.

Frakes W. and Baeza-Yates R. (1992) *Information Retrieval, Data Structures and Algorithms*, Prentice-Hall.

HaraBagiu S. and Moldovan D. and Pasca M. and Mihalcea R. and Surdeanu M. and Bunescu R. and Girju R. and Rus V. and Morarescu P. (2000) *FALCON: Boosting Knowledge for Answer Engines*, TREC 9.

Porter M. (1997) *An algorithm for suffix stripping*, in Readings in Information Retrieval, Karen Sparck Jones and Peter Willet (ed), San Francisco: Morgan Kaufmann.

Sumita E. and Iida H. (1991) *Experiments and Prospects of Example-Based Machine Translation*. In Proceedings of the Annual Meeting of the Association for Computational Linguistics, pp. 185-192.