Robust interpretation in the HIGGINS spoken dialogue system

Gabriel Skantze & Jens Edlund

Centre of Speech Technology, KTH, Sweden
{gabriel,jens}@speech.kth.se

Abstract

This paper describes PICKERING, the semantic interpreter developed in the HIGGINS project – a research project on error handling in spoken dialogue systems. In the project, the initial efforts are centred on the input side of the system. The semantic interpreter combines a rich set of robustness techniques with the production of deep semantic structures. It allows insertions and non-agreement inside phrases, and combines partial results to return a limited list of semantically distinct solutions. A preliminary evaluation shows that the interpreter performs well under error conditions, and that the built-in robustness techniques contribute to this performance.

1. Introduction

This paper describes PICKERING, the semantic interpreter developd in the HIGGINS project, which was instigated in mid 2003. The project is aimed at designing, testing and evaluating error handling techniques for conversational spoken human-computer dialogue. The HIGGINS dialogue system [1], which is developed within the project, relies on experiences from our previous dialogue systems (e.g. Waxholm, August & AdApt, for an overview see [2]) and is intended to be a platform for empirical error handling research. The development of the interpreter is motivated by the general goals of the project, and these are outlined in this section, followed by a detailed description of the interpreter and an evaluation of the robustness techniques it employs.

The initial work within HIGGINS has been focused on enabling empirical tests of different error handling strategies. To this end, the dialogue system is distributed and platform, as well as programming language, independent, so that different modules can be readily tested. Both the inter-process communication and the domain knowledge are highly generalised and extensible. They are encoded in XML and specified by XML Schemas. A side effect of this is that visualisation of system progress and configuration can be handled by normal web browsers using standard methods such as XSL transformations and style sheets. This has proved highly helpful in a rapidly changing research environment.

One of the key ideas in the project is that to the extent possible, every module, or process, should be able to estimate its own current reliability, similar to the ASR confidence measures. The measures may be used for auto-corrections by the process itself, by some following process, or by the dialogue manager when deciding on feedback and grounding strategy. Once the challenging issue of how to calculate the reliability of the results of each process has been handled, empirical tests will provide information on which measures give the best improvement to the dialogue system as a whole, and ultimately to the dialogue quality and user experience. Another key concept is that the system should work incrementally, which allows it to respond more quickly and may be used for example to give feedback during processing. For a more elaborate motivation, see for example [3].

The initial domain for the HIGGINS dialogue system is pedestrian navigation, which is sufficiently complex to present new challenges whilst still being manageable. It bears similarity to the now classic MapTask domain [4], as well as to a number of guide systems, such as REAL [5]. The user gives the system a destination and the system guides the user by giving verbal route directions. The system does not have access to the user's position, but has to rely on the user's descriptions of landmarks. Users are encouraged to speak naturally and freely. A typical example utterance is shown in Figure 1, which also exemplifies the visualisations mentioned previously.

2. Semantic interpretation: PICKERING

PICKERING is designed to work with continuous incremental input from a probabilistic speech recogniser. The ASR used in HIGGINS is KTH LVCSR [6], a large vocabulary probabilistic recogniser developed in-house in a separate project. It supports incremental recognition, which allows the system to use semantic completeness as well as silence to decide when to respond, and confidence measures on word and utterance level, which can be used by the interpreter to incorporate in its reliability measure.

2.1. Requirements

The general project requirements state that the interpreter needs to have a platform and language independent API (XML), that it should be able to pass on meta-information provided in the input untouched, that it should be incremental. and that it should deliver some form of reliability measures. Further requirements were provided by analysis of a corpus from an experiment in a similar domain [7], which reveals many examples of expressions referring to several objects and relations between them, such as part-of and spatial relations. Morphological distinctions are meaningful in order to distinguish both number and definiteness, which may signal whether objects and properties are previously mentioned [8]. Hence, the interpreter should combine interpretation techniques that take syntactic constraints into account and produce deep structured semantics, while allowing for unrestricted natural language input (within the target domain) and ASR errors.

Although the combination of features included in PICKERING is unique, much work has been focussed at achieving robustness in parsing and semantic interpretation beyond keyword spotting. Examples include [9], which deals with insertions in chart parsing, and [10], in which partial results are combined. The way interpretation results may be used to search the domain knowledge directly in PICKERING is inspired by the AdApt interpreter [11].

2.2. Description

The semantic interpreter PICKERING (freely available at [12]) takes a context-free grammar (CFG) with extensions and produces arbitrarily deep, tree-structured semantics. It is designed to handle unexpected input robustly by relaxing the CFG rules.

There are some general challenges with robustness in an interpreter. Firstly, there is a risk that the interpreter will find too many interpretations covering different parts of the input without being semantically distinct. PICKERING utilises the semantic results to filter out solutions that are semantically equivalent or are a subset of another solution. Another potential problem is that the interpreter may find erroneous interpretations based on errors in the input. This is a serious problem for keyword-spotters, since virtually every erroneous content word will result in errors in the interpretation. For a full-fledged wall-to-wall parsing, this is not much of a problem – errors are likely to make parsing of the input impossible – but correct partial solutions are lost. PICKERING deals with this problem by searching for the best set of partial solutions. These may contain non-agreement and insertions into phrases. Finally, robustness can be inefficient if every interpretation is to be considered. The algorithm used in PICKERING is a kind of generate-and-filter technique that ensures that all interpretations are found. This can be costly, but the cost is balanced by the incremental processing – utterances are processed while the user is still speaking.

2.3. Grammar

The CFG is encoded in XML and consists of a rule-set, a collection of lexical entries, and an optional morphology, all of which may carry semantics. The following Swedish example shows a simple grammar:

```
<entry f:name="det">
  <match f:num="sing" f:info="given">den</match>
  <sem><w:object></sem>
</entry>
<entry f:name="attr">
  <match f:num="sing" f:info="given">stora</match>
  <sem><w:size>large</w:size></sem>
</entry>
<entry f:name="nom">
  <match f:num="sing" f:info="given">byggnaden</match>
  <sem><w:object xs:type="building"/></sem>
</entry>
<rule>
  <agreement features="num info gen"/>
  <match>
    <entry f:name="det"/>
    <entry f:name="attr" link="2"/>
    <entry f:name="nom" link="1"/>
  </match>
  <sem><unify copy="info num"/></sem>
</rule>
```

The example covers the noun phrase "the large building". There are three lexical entries and one rule. Both entries and rules have an associated list of features (in boldface), a `<match>` part that specifies what they match by assigning values to features, and a `<sem>` part which specifies the resulting semantics. Entries may match words, and rules may match words, entries or other rules. In Swedish, words in noun phrases must be congruent: they have to agree on gender, number and definiteness. Features that should agree are specified in the `<agreement>` element, which may contain any kind of features, be they syntactic or semantic.

2.4. Robust interpretation

In order to add robustness, the interpreter can relax the grammar rules in the following ways:

- Allowing insertion of words anywhere inside a phrase (e.g. "he hello what runs").
- Allowing non-agreement (e.g. "he run").
- Finding fragments and combining them to find the solution with the best coverage.

While the interpreter allows for these deviations from the grammar, they are taken into account when the solutions are scored. These techniques are described in section 2.6 and their usefulness is evaluated in section 3.

2.5. Semantic unification

Grammar rules also contain instructions for combining semantics from matching entries and rules. A common instruction, as seen in the grammar example, is `<unify>`, which is used to unify semantics in the order in which they are linked with the `link` attribute. This unification is not based on lambda expressions [13], but on a domain model specified in an XML-schema. Note that any domain model (i.e. any XML-schema) could be used by the interpreter. Since `<w:size>large</w:size>` cannot be directly unified with `<w:object xs:type="building">`, the interpreter will look into the domain model and find that the former should be placed under the latter, with the element `<w:properties>` placed in between. The `copy` attribute states that features resulting from agreement should be placed as extra semantic information under all included semantic nodes before they are combined. The resulting semantics for the example will be:

```
<w:object xs:type="building">
  <s:extra info="given" num="sing"/>
  <w:properties><w:size>
      large <s:extra info="given" num="sing"/>
   </w:size></w:properties>
</w:object>
```

The attributes on the `<s:extra>` element tells us that this is a reference to a single large building that ought to be identifiable to the addressee (both the object and its property are given), possibly because it has been mentioned previously in the discourse. If the sentence had been "the building is large", another rule would have been applied that would still have marked the object given, but the property new.

The semantics of an utterance can be used as a search expression by unifying it with the domain model. The domain model contains all objects and their relations and have the same structure as the semantic result. The result of searching the domain with the example would be all large buildings in the world, fully specified.

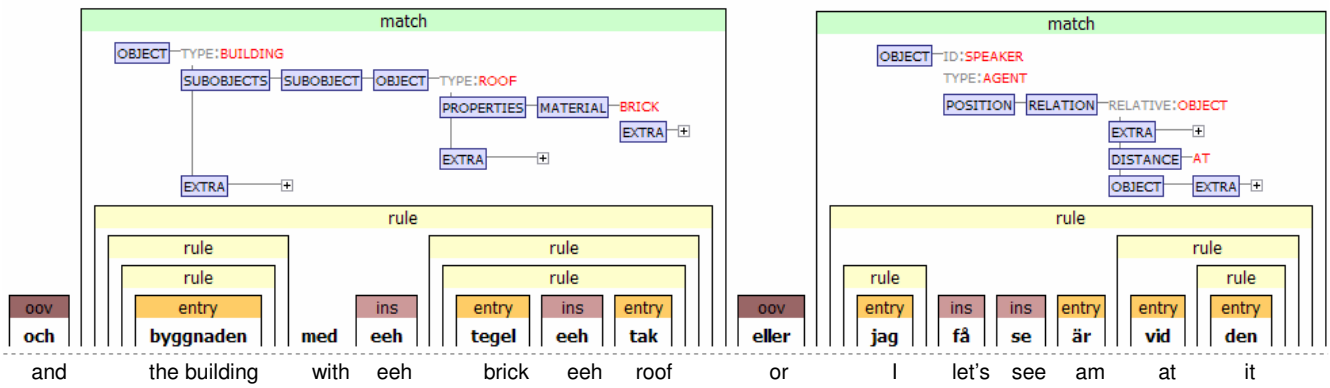

Figure 1: An example interpretation of an "ungrammatical" utterance (translation added). Swedish compounds are deliberately treated as sequences of separate words.

The information in <s:extra> is used for discourse modelling and is removed when searching the domain model.

2.6. Implementation

PICKERING is a modified chart parser [13] implemented in Oz [14]. Parsing is done in two steps: building the chart and searching the chart. To handle incrementality in an efficient way, the chart, with passive and active edges, is saved each time a word is added. The building of the chart then continues with the next word. When the chart has been built, it can be searched for the best combination of phrases (passive edges) that covers the input string – possibly with non-matching words between the phrases. This is done each time solutions are requested, which may be at any given time.

Every edge carries semantics and a set of features. When a passive edge is created, a semantic result for that edge is built based the corresponding grammar rule. Features that agree according to the <agreement> element are also copied to the parent from the constituents. If they do not agree, the majority class is copied (a random choice is used if there is a tie).

Every edge also accepts insertion of a limited number of subsequent words that do not match at every point. Since this procedure generates a very large number of edges in the chart, the goal for the search algorithm is to find a limited number of solutions meeting the following criteria: no two solutions should be semantically equivalent, no possible semantic interpretation should be lost, and the solutions should be ranked in order of correctness.

First, so-called *key edges* are extracted. An edge *E* is a key edge if it does not overlap with any other edge whose semantics is a super-set of the semantics of *E*. To make the comparison of semantic trees faster, the trees are flattened to lists and a sub-list comparison is done. The XML-schema used when writing the grammar and during unification ensures that the branches are flattened in the same order in different trees. Every edge, including lexical entries, is a candidate key edge.

The key edges are then searched to build a minimal list of combinations in which each key edge is represented at least once, ensuring that no unique semantic unit is lost. Since the key edges in each solution will not cover the whole string, the next step is to fill the gaps in between. If there are non-key edges that fit the gaps, these are selected. In cases where there are no fitting edges, the words in the gaps are marked with *nomatch*, or, in case the words are not represented in the lexicon at all, with *oov*.

The resulting list is then sorted in order to return the best solution first. Solutions are compared based on the following heuristics (listed in order of precedence):

1. Larger coverage is preferred (i.e. the number of words covered by the solution, excluding insertions and nomatches).
2. Fewer top phrases (i.e. the number of key edges and edges fitted into gaps) are preferred.
3. Fewer insertions are preferred.
4. Less non-agreement is preferred.

If a preceding criterion determines the best solution, the other criteria are ignored. Thus, if solution *A* has the same coverage as solution *B*, but fewer top phrases, solution *A* will be selected by criteria 2 and criteria 3 and 4 will not be used. The precedence of criteria 1 and 2 is necessitated by the interpreter design. 2-4 all state that less is better, so if 1 was not first, the smallest solution would always win. Similarly, without 2, no larger constructs would be searched for, causing the interpreter to act as a key-word spotter. The order of 3 and 4 is random. Future versions will allow probabilities to be added to criteria such as 3 and 4, allowing them to be combined with other data such as word confidence measures to form a combined score.

3. PICKERING evaluation

A robust interpreter may be too allowing and, as a result, find incorrect interpretations. Hence, it cannot be taken for granted that the techniques described above increase the performance of the interpreter under error conditions. To check for this, the interpreter was evaluated with respect to how it performs under different error conditions, and how its performance depends on the robustness techniques. The interpreter was tested against a corpus collected within the project. The evaluation included data with varying degrees of errors, to ensure that the interpreter works well under perfect conditions and degrades gracefully in the presence of errors.

3.1. Method and data

Eight subjects were given the task to move around in a virtual 3D-city (freely available at [12]) while they described their positions relative to objects in their surroundings. Their descriptions were transcribed and a PICKERING grammar was written to cover their utterances.

For evaluation, 16 new subjects were recorded and transcribed using the same procedure. Due to time limitations, a test set of 100 of these were manually encoded with semantics to create a gold standard for the evaluation. The average utterance length of the utterances was 11 words. The variation in error level required was simulated by using the KTH LVCSR speech recogniser [6] with a trigram language model on the utterances using different levels of pruning of the recognisers set of hypotheses. 18 pruning levels where used; the lowest pruning yielded 34.2% WER and the highest 95.3%.

Next, the transcriptions and the ASR results were processed by the interpreter and compared to the gold standard in order to study the effects of ASR errors. The following interpreter parameters were systematically combined:

- Agreement: weak (default; agreement preferred) or strong (mandatory agreement).
- Permitted insertions: 0, 1, 2 (default) or unlimited.

To simplify comparison, an approximation was made by flattening the semantic trees to form lists of minimal concepts (i.e. nodes in the trees). On average, there were 32 such concepts per utterance. The lists were compared to the gold standard using standard WER calculation to get a concept error rate (CER). The percentage of concept insertions and deletions were calculated separately. Substitutions were counted as a combination of a deletion and an insertion, and were added to both groups.

The lexicon of the interpreter was used to identify keywords in the utterances, so that the ASR performance for words that would have been significant for a keyword spotter could be calculated.

3.2. Results

For transcribed utterances (i.e. WER=0), the concept error rate was 20%, with 12.6% deletions and 9.9% insertions (using the default setting, arbitrarily chosen when the interpreter was implemented: 2 allowed insertions, weak agreement). Figure 2 shows how the performance for insertions and deletions (Y-axis) varies as WER increases. The X-axis represents the mean WER for the different beam pruning settings. The figure also includes the ASR insertion/deletion performance for keywords.

The figure shows that deletions rise steadily both for concepts and keywords, while insertions rise to a certain peak for keywords, but not for concepts. To test the significance of the differences between keywords and concepts, the areas between the lowest and the highest WERs under the curves shown in the figure were compared for all utterances. There were significant differences between keywords and concepts for both insertions and deletions (two-tailed paired t-tests; dF=106; $p<0.05$).

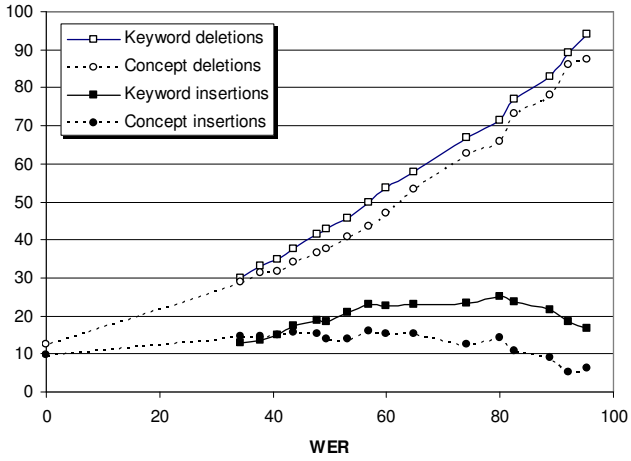

Figure 2: The percentage of insertions and deletions for keywords and concepts, depending on mean WER.

To compare the effects of different parameter settings on the general robustness of the interpreter, CER was calculated for each utterance under each parameter settings and for each pruning level. This CER was normalised using the WER for respective utterance and pruning level. The normalised CER for each utterance varies with the average WER. The area under the normalised CER was calculated between the lowest and the highest WERs and divided with the length of the WER span (95.3-34.2) to compute a mean score for the utterance over the WER span. The combined mean score for all utterances with each parameter setting is shown in Figure 3.

There were main effects for both strength of agreement and number of allowed insertions (two-way repeated measures ANOVA; dF=106; $p<0.05$). Post tests revealed that there were significant differences between 0, 1 and 2 insertions ($p<0.05$), but there were no improvements when more than 2 insertions were allowed. There were no interaction effects between the variables.

4. Discussion and conclusions

To the extent that increased pruning produces errors similar to real ASR-errors, the techniques provide robustness: interpretation results degrade gracefully when ASR performance declines. The two techniques used to relax the CFG-constraints that were tested both improved performance. Allowing an unlimited number of insertions into syntactical structures caused, somewhat surprisingly, no decline in accuracy.

As a result of relaxing the CFG rules, the set of potential solutions becomes very large. The interpreter reduces this set by discarding solutions with identical semantics. The sorting algorithm used to pick the best solution penalises ungrammatical solutions, as well as solutions with low coverage, which is why solutions with many insertions are discarded. Although the sorting algorithm performs well, it would be interesting to add more knowledge (e.g. ASR word confidence scores) and train it using machine learning.

As mentioned previously, efficiency can be a problem for a robust interpreter. A comparison of the total speaking time of the utterances in the evaluation with the time it took to interpret them showed that the interpreter works approximately 20 times faster than real time on a normal desktop computer. Hence, we believe that efficiency will not be a problem if the interpreter is used with an incremental ASR.

We have presented a semantic interpreter that combines syntactic analysis with robustness techniques, incrementality and deep structured semantics. The preliminary evaluation indicates that the

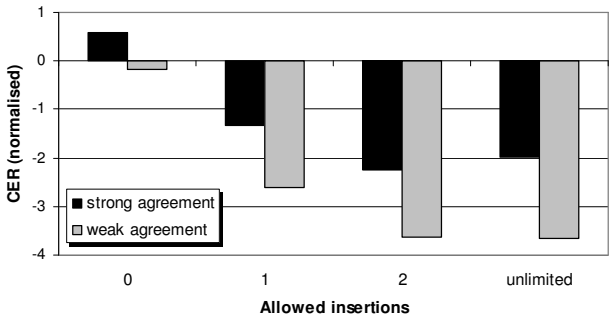

Figure 3: Mean interpreter CER (normalised against WER) depending on parameter settings.

set of techniques generalises well when applied to unseen utterances produced by new speakers. The interpreter meets the project requirements: handling the relevant constructions and their semantics, working with unseen constructions from the same domain, and performing robustly against ASR errors.

5. Acknowledgement

This research was carried out at the Centre for Speech Technology, a competence centre at KTH, supported by VINNOVA (The Swedish Agency for Innovation Systems), KTH and participating Swedish companies and organizations.

6. References

[1] Edlund, J., Skantze, G., & Carlson, R. (2004). Higgins – a spoken dialogue system for investigating error handling techniques. Submitted to *ICSLP 2004*.

[2] Gustafson, J. (2002). *Developing Multimodal Spoken Dialogue Systems - Empirical Studies of Spoken Human-Computer Interactions*. PhD Thesis, KTH, Stockholm.

[3] Allen, J., Ferguson, G., & Stent, A. (2001). An architecture for more realistic conversational systems. In *Proceedings of the 6th international conference on Intelligent user interfaces*.

[4] Anderson, A., Bader, M., Bard, E., Boyle, E., Doherty, G., Garrod, S., Isard, S., Kowtko, J., McAllister, J., Miller, J., Sotillo, C., Thompson, H., & Weinert, R. (1991). The HCRC Map Task Corpus. *Language and Speech*, 34(4), 351-366.

[5] Baus, J., Kray, C., Krüger, A., & Wahlster, V. (2001). A resource-adaptive mobile navigation system. In *Proceedings of the International Workshop on Information Presentation and Natural Multimodal Dialog*.

[6] Seward, A. (2003). *Efficient Methods for Automatic Speech Recognition*. PhD Thesis, KTH.

[7] Skantze, G. (2003). Exploring Human Error Handling Strategies: Implications for Spoken Dialogue Systems. In *Proceedings of the ISCA Workshop on Error Handling in Spoken Dialogue Systems*.

[8] Brown, G. & Yule, G. (1983). *Discourse Analysis*. Cambridge University Press, Cambridge.

[9] Mellish, C. (1989). Some chart-based techniques for parsing ill-formed input. In *Proceedings of ACL*.

[10] Kasper, W., Kiefer, B., Krieger, H., Rupp, C., & Worm, K. (1999). Charting the Depths of Robust Speech Parsing. In *Proceedings of ACL*.

[11] Boye, J. & Wirén, M. (2003). Negotiative Spoken-Dialogue Interfaces to Databases. In *Proceedings of DiaBruck.*

[12] http://www.speech.kth.se/higgins//

[13] Jurafsky, D. & Martin, J. (2000). *Speech and Language Processing*. Prentice Hall, Inc., Englewood, N.J.

[14] http://www.mozart-oz.org/