The Heidelberg GIVE-2 System

Michael Roth, Michael Haas, Eric Hildebrand and **Eleftherios Matios**
Department of Computational Linguistics
Heidelberg University, Germany
{mroth,haas,hildebra,matios}@cl.uni-heidelberg.de

Abstract

This paper describes a NLG system, developed by students at Heidelberg University, that generates instructions that direct users through virtual environments. The NLG architecture is implemented as a modular pipeline consisting of elementary components. The system has been developed for participation in the 2nd installment of the Generation Instructions in Virtual Environment (GIVE) challenge.

1 Introduction

The purpose of generating instructions in the GIVE (Byron et al., 2009) framework is to inform a person of how to pick up a well-hidden trophy. Generating instructions for this task poses difficulties on various conceptual levels such as planning the sequence of actions to be executed, selecting landmarks (i.e., recognizable objects) in the environment, and splitting the task into appropriate single instructions that allow for successful navigation using the selected landmarks as reference points. The GIVE framework provided by the organizers already solves the first part of the challenge, computing a plan of actions. However, a full-fledged NLG system is still needed for the subsequent tasks.

The system described here uses a number of techniques proposed in previous research to generate instructions that efficiently guide users through the virtual environment of GIVE-2 (Koller et al., 2010). We designed a NLG architecture as proposed in the literature (Reiter and Dale, 2000) consisting of simple pipeline components. The resulting system can be considered as a sort of baseline implementation as we did not perform any evaluation on our own nor did we add any enhancements prior to the public evaluation by the GIVE organizers.

The remainder of this paper is structured as follows: In Section 2, we discuss the overall system architecture and the implementation of each component. Section 3 gives an overview of the system results achieved in the GIVE-2 challenge. Finally, we conclude and discuss future work in Section 4.

2 Architecture

The architecture chosen for our system is a standard pipeline infrastructure following the traditional approach described in the literature (Reiter and Dale, 2000). It consists of a *discourse planner*, which filters and combines messages, a *sentence planner*, which decides on the expressions to use for a given message, and a *surface realizer*, which generates grammatical and informative natural language sentences using these expressions. The following subsections describe each of the three components in more detail.

2.1 Discourse Planning

The discourse planning component takes as input an already computed *plan* from the GIVE API. The given plan consists of a list of actions that the user has to take in order to reach the trophy. Each of these actions is encoded in a so-called atom, which consists of a predicate (“move”, “manipulate”, “take”) and possibly multiple arguments (references to, e.g., where to move and what to push or take). In order to present an appropriate amount of relevant information to the user, the discourse planning component performs a filtering step and an aggregation step. Both methods reduce the number of messages in the given plan.

Filtering. In this step, the plan is reduced to ensure that each instruction only consists a limited number of actions. This is simply achieved by removing all atoms beyond the next “manipulate” predicate.

Aggregation. This method combines several "move" predicates based on a visibility analysis, i.e., if the designated goals of two predicates are in the same line-of-sight, the predicates will be combined to one new "move" predicate.

2.2 Sentence Planning

Given the filtered and aggregated plan from our discourse planning component, the sentence planning component computes the linguistic expressions necessary to generate a natural language instruction. At first, a pattern-based turning direction is computed to make sure the user faces the goal of the instruction. Depending on whether the goal is a button that needs to be pushed or a certain point in a room, either a unique referring expression is generated or a lexeme is simply chosen from a list of synonyms.

Turning. The sentence planning component computes a turning direction in order to make sure that the goal of the instruction is visible to the user. The following 9 directions are used depending on the angle between the object and the orientation of the user: right, left, half right, half left, slightly left, slightly right, left around, right around, and around.

Referring Expression Generation. We implemented a variation of Krahmer et al.'s graph algorithm (2003) to generate referring expressions. Due to time constraints, the component is simplified in that it does not perform an exhaustive check for isomorphic graphs. To still avoid overly ambiguous expressions, type and color information are added to the referring expression by default.

Lexicalization. In case the goal of an instruction is an object or region that can already be uniquely identified by its type, the lexicalization component is used to find an appropriate lexeme. The component is a hard-coded data structure that maps semantic concepts to lexemes selected from corresponding WordNet (Fellbaum, 1998) *synsets*.

2.3 Surface Realization

Given the input from the sentence planning component, our surface realization generates a variation of natural language sentences via the realization engine *SimpleNLG* (Gatt and Reiter, 2009). The component further adds variation to the generation process by adding modal verbs and using different sentence patterns.

SimpleNLG. SimpleNLG is a simple Java library that can be used to generate grammatical sentences from a syntactic input structure. In our work, we used version 3.8 from Ehud Reiter's website[1].

Syntactic Input. We compute a syntactic input structure for SimpleNLG by combining the linguistic expressions from the sentence planning component. Typically, each computed expression is used as the object of a sentence with the sentence predicates being equal to the predicates in the plan. Most instructions are imperatives but some natural variation is added by using modal verbs such as "should", "must" and "have to" as well as passive and active constructs. Here are some resulting outputs:

- "Go to the green button and push it."
- "You should press the yellow button."
- "Walk to the door and go ahead."
- "Turn slightly left and walk to the red button. It should be pressed."

3 Results

The organizers of GIVE-2 compiled a number of evaluation results including objective and subjective measures (cf. Koller et al. (2010) for an extensive overview). All participants were additionally provided with log files of played games and filled-out questionnaires. The following sub-sections give an overview of the results of our system including an interpretation of these results and an error analysis on reviewed games.

3.1 Statistical Evaluation

Out of 365 valid games, the users only solved the actual task in 11% of the cases, putting the system into the lower midfield in comparison to other challenge participants. Each given instruction contained an average of 11 words compared to 6-18 for other systems. For all other objective measures (duration/distance needed to solve the task and number of actions/instructions for task completion), no significant difference can be observed between our system and better performing ones.

In addition to these objective measures, the organizers asked users to fill out a questionnaire for

[1] http://www.csd.abdn.ac.uk/~ereiter/simplenlg/

subjective evaluation of each system. Here we performed among the worst, both regarding quality and emotional effect. On average, users rated our system and its instructions as the most annoying, robotic and unfriendly among all participating systems. The ratings further indicate a noticeable lack of good timing and clarity.

3.2 Error Analysis

We manually reviewed 106 game replays to identify frequent error sources and to analyze major flaws in our system. 51 out of 106 (48%) analyzed games were lost because users stepped on so-called alarm tiles in the virtual world, 43 games were cancelled (41%) and 12 games were successfully finished.

Lost games. Probably the most common problem is that our system only produces straightforward instructions without providing the user with thorough background information. We found that in 24 cases (47%), users lost the game because they were not aware of the effect of alarm tiles at all. In 10 additional cases (20%), users were told to go to alarm tiles but it was not clear to them that the system actually meant a deactivated alarm tile. In 7 cases (14%), users stepped on alarm tiles because the system did not produce an sufficient instruction or the instruction was badly timed. In the remaining cases (19%), user were too imprecise in their movements and triggered the alarm by accidently stepping on alarm tiles.

Cancelled games. For many cases, it is difficult to say why exactly users cancelled a particular game. We analyzed the free text feedback in the questionnaire to find out some of the reasons. Often users seemed to be frustrated with the fact that the system has not always produced distinct referring expressions. In one of the three evaluation worlds, this problem made it practically impossible for users to make it past the first room. In other cases, the user did not follow the instructions of the system or did not move at all. A reason for this might have been that instructions were not displayed long enough, a problem also mentioned in the users' feedback. We further noticed one case in which no instruction was generated and one case in which the user cancelled after permanently confusing left and right.

3.3 Discussion

As indicated in the error analysis, the two most severe problems of our system were the lacking warning of alarm tiles and ambiguous referring expressions. During development and testing, it was not clear to us that these points would become such big issues in the actual evaluation. We believe that minor enhancements in this direction could already increase the task success rate achieved with our system considerably.

Still, the subjective evaluation measures show that users are also not very satisfied with the overall quality and friendliness of our system. We were able to identify some reasons for this from the free text feedback. Apart from some timing and clarity problems with the currently given instructions, many users seem to want much more background information on the actual task. A step-by-step generation of instructions might not be enough to satisfy all users. Hence a much more "natural interaction" with the system is needed.

4 Conclusions

We presented the Heidelberg GIVE-2 system, a straight-forward pipeline implementation for generating natural language instructions. Though our system performed reasonably well in comparison to other systems in terms of objective evaluation measures, it is one of the worst performing systems in regards to subjective evaluation measures. Users rated the overall quality and emotional effect of the system as rather poorly. A more sophisticated approach is needed to not only guide the user more successfully to his goal but also to prevent him from becoming frustrated and giving up.

For the next installment of the GIVE challenge, we will improve our system with regards to the identified problems. We will implement a better component for referring expression generation and extend the overall architecture to give more information to the user throughout the game. One example for this, inspired from previous work by Dionne et al. (2009), would be to provide additional feedback whenever a user comes close to an alarm tile.

Acknowledgments

We would like to thank the organizers of the GIVE-2 challenge for providing the framework and opportunity to participate. We are particularly grateful to Alexander Koller and Konstantina Garoufi for kindly answering all our questions and providing additional information. Thanks also to Lucie Martinet for reviewing the game logs and to all participants of the seminar "Natural Language Generation for Virtual Environments" for fruitful discussions.

References

Donna Byron, Alexander Koller, Kristina Striegnitz, Justine Cassell, Robert Dale, Johanna Moore, and Jon Oberlander. 2009. Report on the First NLG Challenge on Generating Instructions in Virtual Environments (GIVE). In *Proceedings of the 12th European Workshop on Natural Language Generation (ENLG 2009),* Athens, Greece, 30-31 March 2009, pages 165–173.

Daniel Dionne, Salvador de la Puente, Carlos León, Pablo Gervás, and Raquel Hervás. 2009. A model for human readable instruction generation using level-based discourse planning and dynamic inference of attributes. In *Proceedings of the 12th European Workshop on Natural Language Generation (ENLG 2009)*, pages 66–73, Athens, Greece, March. Association for Computational Linguistics.

Christiane Fellbaum, editor. 1998. *WordNet: An Electronic Lexical Database*. MIT Press, Cambridge, Mass.

Albert Gatt and Ehud Reiter. 2009. SimpleNLG: A realisation engine for practical applications. In *Proceedings of the 12th European Workshop on Natural Language Generation (ENLG 2009)*, pages 90–93, Athens, Greece, March. Association for Computational Linguistics.

Alexander Koller, Kristina Striegnitz, Andrew Gargett, Donna Byron, Justine Cassell, Robert Dale, Johanna Moore, and Jon Oberlander. 2010. Report on the Second NLG Challenge on Generating Instructions in Virtual Environments (GIVE-2). In *Proceedings of the 6th International Natural Language Generation Conference (INLG 2010)*, Dublin, Ireland, July.

Emiel Krahmer, Sebastiaan van Erk, and André Verleg. 2003. Graph-based generation of referring expressions. *Computational Linguistics*, 29(1):53–72.

Ehud Reiter and Robert Dale. 2000. *Building Natural Language Generation Systems*. Cambridge, U.K.: Cambridge University Press.