A Model for Human Readable Instruction Generation Using Level-Based Discourse Planning and Dynamic Inference of Attributes Disambiguation

Daniel Dionne, Salvador de la Puente, Carlos León, Raquel Hervás, Pablo Gervás
Universidad Complutense de Madrid
Madrid, Spain
{dionnegonzalez,neo.salvador}@gmail.com,
{cleon,raquelhb}@fdi.ucm.es,pgervas@sip.ucm.es

Abstract

This paper shows a model of automatic instruction giving for guiding human users in virtual 3D environments. A multilevel model for choosing what instruction to give in every state is presented, and so are the different modules that compose the whole generation system. How 3D information in the virtual world is used is explained, and the final order generation is detailed. This model has been implemented as a solution for the *GIVE Challenge*, an instruction generation challenge.

1 Introduction

Recent technology advances have made it possible to use handheld devices, like mobile phones or PDAs, to guide the user by issuing commands or descriptions about the world the user is perceiving in some sense (Muller, 2002). This possibility opens interesting avenues of research in the shape of Natural Language Generation (NLG) Systems that adapt to the user in order to provide him with the most accurate expression. However, fully operational systems applicable in real life situations are difficult and expensive to implement. Under these circumstances, virtual environments may be seen as an intermediate solution, suitable for fast prototyping of experimental solutions. Virtual environments permit experimenting in a reduced, closed world, where everything that is relevant for the purpose at hand is explicitly represented in a graphical model and under the direct control of the researcher. This allows fast set up of experimental situations where the topography, the position of landscape features, colour, light conditions and visibility factors can be modified and adapted to suit the best conditions for testing particular approaches (Blue et al., 2002) or challenges (such as guidance for disabled users with different disabilities, for instance). In view of these observations, our research is focused on developing an interactive *virtual guide (VG)*, based on NLG, to give to a human user the required set of instructions to complete a specific task.

Such a set of instructions is called a *plan*. Formally, a plan is a sorted-in-time list of instructions that the user must fulfill in order to reach some goal. There are many planning algorithms that, with the proper world representation and a list of goals, can return a list like this (LaValle, 2006). The VG can take this basic plan as the actual set of instructions to convert into natural language to explain what the user must do to complete the task. However, these instructions are usually exhaustive (step by step) and very simple because they are based on basic world representations (and interpretations) and are simple enough to perform computational operations on them. A VG that generates this kind of simple instructions, from the point of view of a human user, can be tedious, boring and a time wasting. Consider the discourse *"Turn right. Turn right. Go ahead. Turn left. Press button-1. Turn around. Go ahead. Go ahead. Take item-1..."* as an example. Instead, the VG should take advantage of the environmental knowledge of the user inferring higher level instructions (less detailed and more human-like) from the basic plan (something more along the lines of *"Go press the buton in the far wall, come back and take item-1"*). The difference is shown graphically for a simple example in Figure 1.

There are several aspects to be considered in achieving this goal. First, a human guide would phrase his or her instructions at different levels of abstraction, to optimise the communicative effect of his/her utterances in terms of striking balance between sufficient informative content and economy of expression. Second, a human guide may operate in a reactive manner, providing additional feedback whenever the user requests help. But

Proceedings of the 12th European Workshop on Natural Language Generation, pages 66–73,
Athens, Greece, 30 – 31 March 2009. ©2009 Association for Computational Linguistics

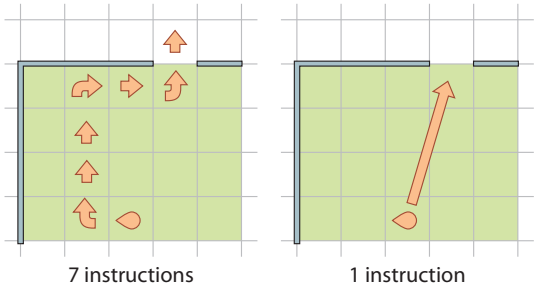

Figure 1: A comparison of a step by step plan versus a human readable plan like *"Walk out the door"*. Note the difference in the number of instructions given.

human guides are also likely to observe the person that is being guided, and be ready to intervene proactively if they notice the user seems lost or at risk. These two points are elaborated below.

In order to build *more human* levels, a VG must consider the virtual environment in a manner as close as possible to the way a human being senses the real world. To model the different levels of abstraction employed by human guides, a good solution may be to model *the world as a hierarchy of spatial levels*. People tend to limit the area where they do certain activities by some kind of logical borders. Sometimes, these borders match physical borders such as the walls that define a room or a corridor, the outside perimeter of a building, the limits of a park, or a city boundary. In other cases, such as outdoor settings, borders can be more abstract, such as the line of horizon in all directions from the observer's current position. The areas defined by these borders may be contained inside one other, resulting in a tree-like structure from the smallest spaces to greater areas, i.e. from the room where the user is standing to the city he lives in. Of course, the areas are connected in a multigraph way where each edge is a connection like a door or a natural transition. To build a usable model of this type of cognitive representation of the world is far from trivial. We will describe how we faced this point in Section 3.1 (Constructing the World). Considering such a hierarchical view of the environment when generating instructions, results in more natural and human-friendly results. Instructing someone to *"exit the room"* works better than asking them to *"advance until passing through the door"*; *"leave the building using the main entrance"* is better than a set of instructions refering to more specific spaces like *"exit this room, now go down the stairs, now go to the elevator"* and so on. We return to this matter in Section 3.2 (Planning the Discourse).

The issue of abstractions in world modelling also affects a different language generation task: referring expression generation. In providing instructions, human guides often refer to abstract concepts such as corners or "the middle of the room". These are not usually represented explicitly in your run of the mill world representation, which usually prevents NLG systems from employing them as means of optimising references. In Section 3.4 (Hidden Reference Discovery), we will see how, besides visible information, a natural approach based on the inference of other "hidden" elements or references that can be extracted from the environment helps to reduce the length of the explanation needed, and to build better references. These elements are hidden because they are not visible or trivial, and they require a specific study and calculation.

The second point to consider is reactive versus proactive guidance. A reactive guidance system may rely on feedback from the user to decide when to intervene. Consider the following two representative examples: the user can say *"I did not undertand last instruction"* and the VG system can answer by repeating the instruction or building a new one phrased in a different way but with the same meaning; or the user can say *"I am lost"* and the VG will ask the planning software to recalculate the plan considering the new user's situation. However, there are situations where the user may not realize that he is lost or that he is about to perform a dangerous action (like walking on a slippery surface, pressing an incorrect button, going in the wrong direction or crossing a street when the traffic light is red). A good guide will warn the user before he does something wrong but it should not oppress the user each time he decides to explore another route to reach the goal. In other words, the VG must watch the user actions and take part when he is on the verge of commiting a serious mistake. We will discuss about how to warn the user in Section 3.3 (Warning the User).

2 Previous Work

Many NLG systems have considered generation of instructions in the past. A good review is provided in (Bourne, 1999). However, most existing instruction generating system focused on perform-

ing different types of static actions (actions that do not involve changes of location of the user). The present work is focused on the task of guiding the user through virtual environments.

The GIVE (**G**enerating **I**nstructions in **V**irtual **E**nvironments) Challenge (Byron et al., 2007) operates on a scenario where a user has to solve a particular task in a simulated 3D space. A generation module has to guide the human user using natural language instructions. A software architecture is provided that allows the generation module to abstract away from the rest of the system, while having access to world information from the 3D environment, user feedback from the client module, and plans generated by an off-the-shelf planner. The work presented in this paper arose from the author's participation in the GIVE Challenge, and relies on the software architecture provided for the challenge to implement all details of the system other than the NLG module.

A fundamental task to be solved for correct instruction generation is the construction of appropriate referring expressions. This task has been the object of many research efforts in the recent past. To construct a reference to a particular entity, the algorithm takes as input a symbol corresponding to the intended referent and a list of symbols corresponding to other entities in focus based the intended referent, known as the *contrast set*. The algorithm returns a list of attribute-value pairs that correspond to the semantic content of the referring expression to be realized. The algorithm operates by iterating over the list of available attributes, looking for one that is known to the user and rules out the largest number of elements of the contrast set that have not already been ruled out.

Referring Expression Generation in physically situated environments has been studied in (Kelleher and Kruijff, 2005). The goal of this work is to develop embodied conversational robots that are capable of natural, fluent visually situated dialog with one or more interlocutors. In this kind of situation a very important aspect to take into account is how to refer to objects located in the physical environment. The authors present in the paper a computational framework for the generation of spatial locative expressions in such contexts, relying on the Reiter and Dale (Reiter and Dale, 1992) algorithm.

Another interesting work related to referring expression generation in spatial environments can be found in (Varges, 2005). The author uses the maps of the Map Task dialogue corpus as domain models, and treats spatial descriptions as referring expressions that distinguish particular points on the map from all other points (considered as distractors).

Related research can be found in (Stoia et al., 2006), where a study of how humans give orders in navigation environmnets and an algorithm implementing the observed behaviour is shown. There are many other approaches to instruction giving. Directly related with this work, it is worth mentioning `CORAL` (Dale and Geldof, 2003), which shows a full architecture for instruction giving, and `REAL` (Muller, 2002), which shows a multimodal system (graphics and text) for communicating with the user, adapting them to user behaviour.

3 A Functional Model of a Virtual Guide

The model of a virtual guide presented here addresses four specific issues: how to construct a representation of the world with higher levels of representation, how to generate higher instructions referring to the more abstract levels of representation, how the construction of references is implemented in terms of reference agents. A brief overview of the complete architecture of the module is also included.

3.1 Constructing the World

In GIVE, the world is discretized as a set of tiles. These tiles are the minimum portions of space and the user can move around from tile to tile. Orientations are discretized: the user can only face North, East, South or West. By default, the world consists of an infinite area of adjacent and accesible tiles. World representation assertations may state there is a wall between two adjacent tiles, blocking access from one to other. A 3D representation of this basic world gives the user an illusion of rooms but, from the point of view of the VG there is no data structure that reflects a hierarchy of rooms. This representation does not fit very well with the human sense of space, so a more abstract one had to be built to provide the abstract referents (rooms, corners, intersections, doors...) which we wanted our guide to use.

The first problem we had was defining a room. In architecture, a definition of room is *"any distinguishable space within a structure"*, but *distinguishable* is too vague to be of use. Figure 2 illus-

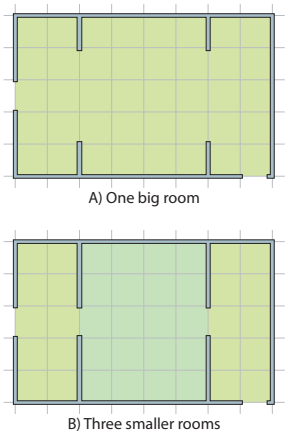

Figure 2: Defining a *distinguishable* space.

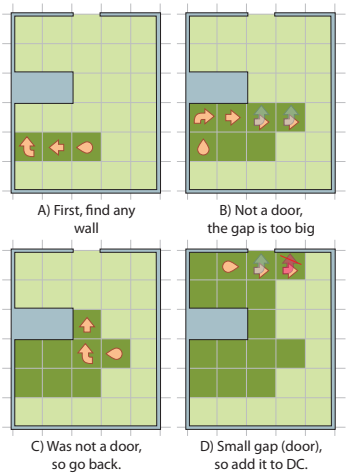

Figure 3: Looking for rooms.

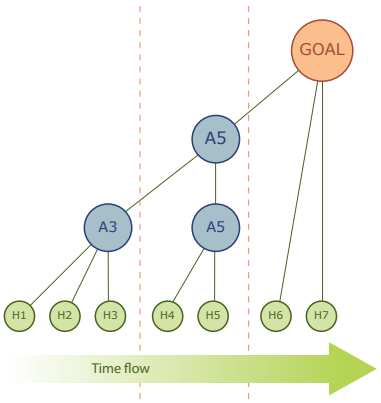

Figure 4: Tree representation of the plan at several levels.

trates the problem of defining when two spaces are distiguishable. Notice the only difference betwen A and B is the width of the gaps in relation to the size of the rooms. This problem has been extensively studied in robotics. An interesting example (Galindo et al., 2005) consists on identifying interconected "open spaces" in order to obtain an adjacency graph. From that graph, another graph can be calculated, grouping spaces to form rooms, corridors, etc.

For practical purposes, we have decided to consider that two spaces are distinguishable when the user has to go through a door to get from one to the other, with a door being a one-tile gap in a wall.

Based on this definition, we have developed an algorithm to group adjacent tiles into rooms. The idea is to follow a wall around the room until the starting point is reached, thereby establishing the perimeter of the room, then establish the set of tiles corresponding to the room using a floodfill algorithm. Breaks in walls are handled by checking whether they are small enough to be considered doors into other rooms or not. If they are doors, they are noted as entrances to other rooms (which are stored in a *room list* for subsequent processing). If they are not, the wall beyond the gap is followed as part of the boundary of the current room. A small practical example of the algorithm in operation is shown in Figure 3.

Adjoining rooms stored in the `room list` are recursively processed. Each new room discovered is connected to its adjacent rooms to obtain a high level map of the available space. An analyzer is applied to each room to establish its type (room, hall, corridor, etc) and additional properties such as size or shape. This new world representation allows the VG to refer to doors and rooms.

3.2 Planning the Discourse

Discourse planning must take place at two different levels of detail. The VG must plan the discourse corresponding to the whole set of instructions to be imparted until the final goal is reached. But it also needs to plan how much of that is to be communicated to the user in the next turn of the dialogue. We solve the first issue by building a multi-level representation of the expected discourse for the whole of the plan to be carried out by the user. This representation is structured like a tree, with the set of low-level instructions as leafs, and subsequent nodes of the tree representing higher level instructions that group together the lower level instructions represented by their subtrees. The solution to the second issue is described below.

We define **action** as *anything the user can do*

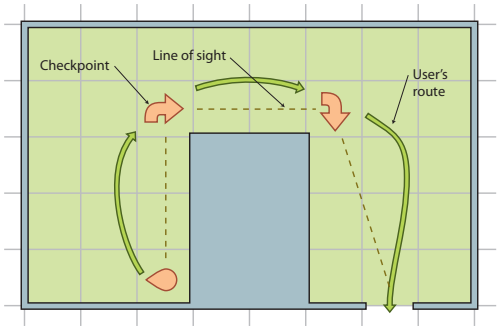

Figure 5: An n-shaped room does not let the user see the exit of the room so VG can guide the user from checkpoint to checkpoint.

that modifies the state of the world and **instruction** as *an action that the user should perform in order to advance in the plan*. Instructions are defined in terms of preconditions and postconditions. **Preconditions** are conditions that must be satisfied for the instruction to be performed, and **postconditions** are the conditions that must be satisfied to consider the instruction done. The *instruction tree* representation of the plan is built by grouping together sets of low-level instructions into a single high-level instruction. For instance, we group all tile-by-tile steps inside the same room to build a new instruction such as *"go from room1 to room2"*. We do not discard any low-level instruction, we just group them under the new high-level instruction, building a tree that represents the plan at different levels of abstraction (see Figure 4). This allows the user to fall back on low-level instructions at need (if, for instance, the light goes out and the VG has to guide him step by step).

An additional abstraction has been introduced to account for the tendency of humans to break the description of a complex path (where not all of the path is visible at the start) into segments made of the portions of the path that are visible at each particular point (see Figure 5). The concept of *checkpoint* is introduced for the end of each of these segments.

We have defined five types of high-level instructions: **MovementInstruction** (guides the user from tile to tile), **CheckPointInstruction** (guides the user from a his current position to a checkpoint), **Room2RoomInstruction** (guides the user from room to room), **ActionInstruction** (tells the user to interact with some element) and **GoalInstruction** (subtype of ActionInstruction concerned with achieving the final goal). Each of these high-level instructions has its own preconditions and postconditions.

The issue of how much of the instruction tree representation of the plan is addressed in terms of two conditions: how far in the original plan the user has advanced, and what level of abstraction is required for the next instruction. The first condition is easily checked over the state of the world, to establish what the current situation is. The second condition is determined by checking for satisfaction of preconditions and postconditions of the instructions at all possible levels that start from the current situation. The check starts at the highest possible level.

Instructions whose postconditions are already satisfied are pruned from the tree, as there is no longer any need to provide that instruction to the user. If preconditions are met but postconditions are not, the VG uses this instruction in the next turn, and then waits for a user action. If neither postconditions nor preconditions are satisfied for this instruction, the next (lower) level of instructions in the instruction tree is considered instead of this one. These decisions are handled by modules known as *Guide Agents*.

3.3 Warning the User

If the user is going to cross a street when the traffic light is red, the VG will have to warn him about it. If the warning information is more important than the guiding, the VG will have to delay instruction giving, and warn the user first. To decide about the importance of the warning part of the discourse, we defined *agents* as entities in charge of watching for special situations. Each agent takes care of a specific kind of situation that may imply some sort of hazardous or bad result. They are all independent, and may differ depending on the kind of environment, goals or even the kind of user.

Each agent has a weight that reflects its priority when being considered. An agent always evaluates its situation and returns a value in the $[0, 1]$ interval. A near zero value means there are low probabilities for the situation to happen and a near to one value means the situation is on the verge to happening. All agents that exceed a threshold value will be considered as contributors to the discourse. We sort them in descending order based on the result of multiplying each return value by the weight of the agent. If an agent is considered

as a contributor, its warning is introduced in the discourse.

We defined three types of agents: **information agents** watch for interesting hotspots in an area, **status agents** watch over the user's status, and **area agents** watch over special areas, including dangerous areas.

In our entry for the GIVE challenge there was a status agent that checked how much time had passed since the last user action to identify when the user might be lost. There was one agent that checked for booby traps the user might step on (some of them resulted in loosing the game inmediately). Another one ensured the user remained within a *security area* that abstracted all possible common routes to reach the intended destination. If a user leaves the security area, he is going in the wrong direction.This security area is dynamicaly updated attending to the current user's position. Finally, **alarm agents** watch for wrong actions, controlling if user is on the verge of pressing the wrong button or leaving the room using a wrong exit. We implemented no information agents, but they would be interesting in real situations.

3.4 Hidden Reference Discovery

The center spot in a room is not a visible or tangible object, and finding it requires a non-trivial calculation of the room's shape. Adding it to the references container can help creating simpler and richer sentences. A reference like "the table across the room" can be generated when the listener and the target are in line with the center spot of the room, on opposite sides, independently of where the user is facing. In an indoor environment, architectural elements usually make many inferences possible. Two hallways that intersect make an intersection, two walls make a corner, etc. and though these elements might not be referenced as they are in the given environment, they should be taken into account. In a similar way, hidden relations discovery can be accomplished. Object alignments or arrangements can be revealed and used for the same purpose. Sentences like "the car in line with these pillars" can be generated. All of these additional high-level concepts and relations between them and low-level world entities are obtained by abstraction over the available representation. We create a family of *reference agents*, each one specialized in identifying candidate disambiguating properties of a different kind. Some of these properties are already explicit in the world representation (colour) and some require a process of abstraction (relations to corners, for instance). Once obtained, they become available as additional properties that may be used to disambiguate references.

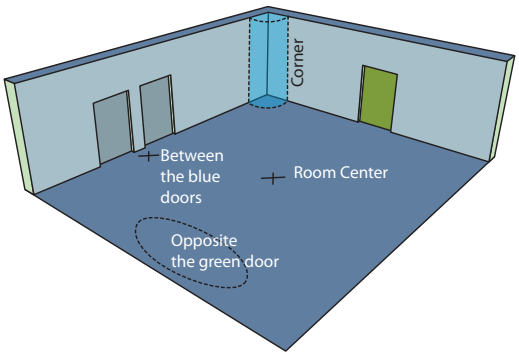

Figure 6: Hidden references in a room.

The goal of our design is to leverage the system's ability to express itself using different combinations of the complete set of disambiguating properties made available in this manner. This gives system designers a choice between having many simple agents or fewer more expressive, complex agents. This choice should be considered in terms of particular implementation details.

Reference agents rely on the Reiter and Dale algorithm (Reiter and Dale, 1992). Considering a list of distractors and the reference object, the goal is to filter the distractors list, building a reference that takes out all the distractors, so that the reference is good, not ambiguous. Each reference agent has the ability of taking out a different set of distractors, using different properties that are trivial or hidden, as explained above. Combining these agents in different ways generates different reference sentences, some of them longer but more specific, others shorter but ambiguous. What we tried to achieve is to find the right combination of reference agents that create the shortest non-ambiguous sentence. This is not a natural approach, as someone could prefer to have an ambiguous (but more human) spatial relation (Viethen and Dale, 2008) in a reference sentence. Or for example, someone could prefer having a longer reference like "the big red box that's on the third shelf from the bottom" than a perfectly specific (but not natural) reference like "the 3 kg box".

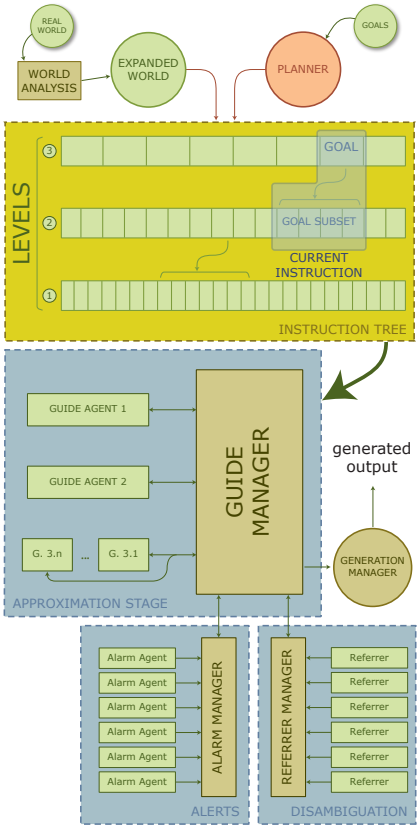

Figure 7: General design.

3.5 Guide architecture

The architecture design can be divided into two main parts. The *instruction tree*, shown as three interconnected lists in Figure 7, that contains all the generated levels of instructions as explained in section 3.2, and a set of components that perform the different guiding tasks. One input for the system is the "Real World", as opposed to the *Expanded World* that is generated after the analysis, as explained in sections 3.1 and 3.4. The second input is the set of goals to be achieved. After the basic instruction set is generated by the planner from the given set of goals, the *instruction tree* is generated, level by level.

Figure 7 represents a state of the guiding process where the user is trying to achieve some intermediate GOAL. The *current instruction* marker represents the location of the instruction that is to be given to the user to achieve the current GOAL (the one on the upper level). Since at this point the system has determined that *level 2* instructions should be used, the level 2 subset of instructions are represented here as part of the *current instruction*. As explained in section 3.2, the algorithm chooses what level should be used at each moment.

The *Guide Manager* makes use of the *Alarm Manager* and *Referrer Manager* to create the proper output. As explained in 3.3, the Alarm Agents examine the environment, and tell the *Guide Manager* if the user should be warned about any hazardous situation. The *Referrers* help building the proper reference sentences, as explained in sections 3.2 and 3.4, finally the different *Guide* help building the proper guiding sentences. The *Guide Manager* sends the output to the *Generation Manager*, which is in charge of generating the final output.

4 Discussion

The layered, multilevel hierarchy tries to imitate the way humans think about local plans, and the agent based view attemps to make instruction giving proactive rather than reactive. The algorithm first gives generalistic, global orders to get the user near the particular objective. Then, once the irrelevant information has been removed from the user point of view and it can not confuse the user, more specific orders are given. In this way, the algorithm decides what to say the "human way". Although the "human" generation of instructions could have been obtained with different algorithms, doing it the same way creates a more maintainable, natural form of expressing the operation. It would be interesting to input real human data, as done in (Stoia et al., 2006), in order to guarantee this objective.

Traditionally, planning systems have certain world representation based on discrete states which are more or less useful for finding a good solution (Chih-Wei Hsu and Chen, 2006). However, this representation is not necessarily useful for creating a natural language representation of each planning operator. For a good instruction to be generated, plain operators like "turn right" usually do not contain much information. Instruction generation systems have to find a compromise between planning efficiency and natural language content. Creating the instruction tree depends directly on figuring out what elements to include in the discourse.

The architecture shown in Section 3 has been designed with adaptability in mind, following the architecture presented in (Dale and Geldof, 2003). This shows a module layout where the text plan-

ner and the surface realizer are independently connected in the generation pipeline.

5 Conclusions and Future Work

The decisions to consider higher level of abstraction for both the representation of the world and the granularity of instructions, and the introduction of alarms have shown very satisfactory results over informal tests with users. Further evaluation is in process as part of the GIVE Challenge (Koller et al., 2007)[1]. The decisions presented in this paper should be revised in view of these results. The definition of a security area enables the system to provide suitable warning when the user really goes out of the way, but makes the system robust with respect to minor variations with respect to the literal plan provided by the planner.

The GIVE challenge set up was a good starting point to begin our experiments, but we are considering more complex environments to test advanced features. Extensions that promise interesting challenges are: the consideration of a continuous world representation (rather than discretised in terms of tiles and four cardinal points), more realistic test maps to extend the level of hierarchy to buildings and urban areas, and new environments designed to experiment with distorted representations of the scenary in order to simulate physical impediments like blindness.

Acknowledgments

This research is funded by the Ministerio de Investigación, Ciencia e Innovación (GALANTE: TIN2006-14433-C02-01), and Universidad Complutense de Madrid and Comunidad de Madrid (MILU: CCG07-UCM/TIC 2803).

References

Russell S. Blue, Jeff Wampler, G. Bowden Wise, Louis J. Hoebel, Boris Yamrom, Christopher R. Volpe, Bruce Wilde, Pascale Rondot, Ann E. Kelly, Anne Gilman, Wesley Turner, Steve Linthicum, and George Ryon. 2002. An automated approach and virtual environment for generating maintenance instructions. In *CHI '02: CHI '02 extended abstracts on Human factors in computing systems*, pages 494–495, New York, NY, USA. ACM.

Juliet C. Bourne. 1999. *Generating Effective Natural Language Instructions based on Agent Expertise*. Ph.D. thesis, University of Pennsylvania.

[1]The results of this challenge will be made available as part of the ENLG 2009 Workshop.

Donna Byron, Alexander Koller, Jon Oberlander, Laura Stoia, and Kristina Striegnitz. 2007. Generating instructions in virtual environments (GIVE): A challenge and evaluation testbed for NLG. In *Proceedings of the Workshop on Shared Tasks and Comparative Evaluation in Natural Language Generation*, Arlington.

Ruoyun Huang Chih-Wei Hsu, Benjamin W. Wah and Yixin Chen. 2006. Handling soft constraints and goals preferences in SGPlan. In *ICAPS Workshop on Preferences and Soft Constraints in Planning*.

Robert Dale and Sabine Geldof. 2003. Coral: Using natural language generation for navigational assistance. In *Proceedings of the 26th Australasian Computer Science Conference*.

C. Galindo, A. Saffiotti, S. Coradeschi, P. Buschka, J.A. Fernandez-Madrigal, and J. Gonzalez. 2005. Multi-hierarchical semantic maps for mobile robotics. *Intelligent Robots and Systems, 2005. (IROS 2005). 2005 IEEE/RSJ International Conference on*, pages 2278–2283, Aug.

John D. Kelleher and Geert-Jan M. Kruijff. 2005. A context-dependent algorithm for generating locative expressions in physically situated environments. In *Proceedings of ENLG-05*, Aberdeen, Scotland.

Alexander Koller, Johanna Moore, Barbara di Eugenio, James Lester, Laura Stoia, Donna Byron, Jon Oberlander, and Kristina Striegnitz. 2007. Shared task proposal: Instruction giving in virtual worlds. In Michael White and Robert Dale, editors, *Working group reports of the Workshop on Shared Tasks and Comparative Evaluation in Natural Language Generation*.

S. M. LaValle. 2006. *Planning Algorithms*. Cambridge University Press, Cambridge, U.K. Available at http://planning.cs.uiuc.edu/.

Christian Muller. 2002. Multimodal dialog in a mobile pedestrian navigation system. *IDS-2002*.

E. Reiter and R. Dale. 1992. A fast algorithm for the generation of referring expressions. In *Proceedings of the 14th conference on Computational linguistics*, Nantes, France.

Laura Stoia, Donna Byron, Darla Shockley, and Eric Fosler-Lussier. 2006. Sentence planning for realtime navigational instructions. In *Proceedings of the Human Language Technology Conference of the North American Chapter of the ACL*.

Sebastian Varges. 2005. Spatial descriptions as referring expressions in the maptask domain. In *Proc. of the 10th European Workshop on Natural Language Generation*.

Jett Viethen and Robert Dale. 2008. The use of spatial relations in referring expression generation. In *Fifth International Natural Language Generation Conference*.