# The GIVE-2.5 C Generation System

**David Nicolás Racca, Luciana Benotti** and **Pablo Duboue**
Universidad Nacional de Córdoba
Facultad de Matemática, Astronomía y Física
Córdoba, Argentina
{david.racca, luciana.benotti, pablo.duboue}@gmail.com

## Abstract

In this paper we describe the C generation system from the Universidad Nacional de Córdoba (Argentina) as embodied during the 2011 GIVE 2.5 challenge. The C system has two distinguishing characteristics. First, its navigation and referring strategies are based on the area visible to the player, making the system independent of GIVE's internal representation of areas (such as rooms). As a result, the system portability to other virtual environments is enhanced. Second, the system adapts classical grounding models to the task of instruction giving in virtual worlds. The simple grounding processes implemented (for referents, game concepts and game progress) seem to have an impact on the evaluation results.

## 1 Introduction

GIVE-2.5 is the third instance of the challenge on Generating Instructions in Virtual Environments (Byron et al., 2007). The GIVE Challenge is an NLG evaluation contest in which natural language generation systems help human players complete a treasure hunt in virtual 3D worlds.

In GIVE, the C system and the human Instruction Follower (IF)—the player—establish a dialogue situated in a virtual world. The C system verbalizes, in real time, instructions that the IF must follow in order to complete the game. Generating instructions involves the generation of referring expressions and navigation instructions. The C system was designed independently of GIVE world's internal concepts by using the IF's visibility information. As a result, the main algorithms of the system can be ported to different virtual environments.

To make the communication more effective, the C system implements a grounding model for referents based on Traum's grounding acts model (Traum and Allen, 1992; Traum, 1999). The system also implements a grounding process for unknown objects, such as alarms, describing them to the player and specifying their effects.

The paper is structured as follows. Section 3 describes the system's strategy based on player's visibility. Section 2 introduces the architecture design of the system. Section 4 explains C system's grounding model. Section 5 briefly analyzes the evaluation results and Section 6 concludes.

## 2 System Architecture

In the virtual worlds, the player has to press several buttons to accomplish the target goal. These buttons, when pressed, modify the state of the virtual world. C system's architecture is an adaptation of Reiter and Dale (2000) NLG architecture to GIVE's dynamic context. Figure 1 presents the architecture diagram of the C System. The arrows between the different modules represent how data flows through modules. Note that data flows through a cycle that starts with the player's actions information and finishes with C system's text instructions. On each iteration, C system checks what the player did or is doing at the moment and uses this information as well as the plan's information to create one or more instructions in response to the player's activities.

The Monitor module is responsible for checking player progress and status. It collects targeted



*Proceedings of the 13th European Workshop on Natural Language Generation (ENLG)*, pages 290–295,
Nancy, France, September 2011. 

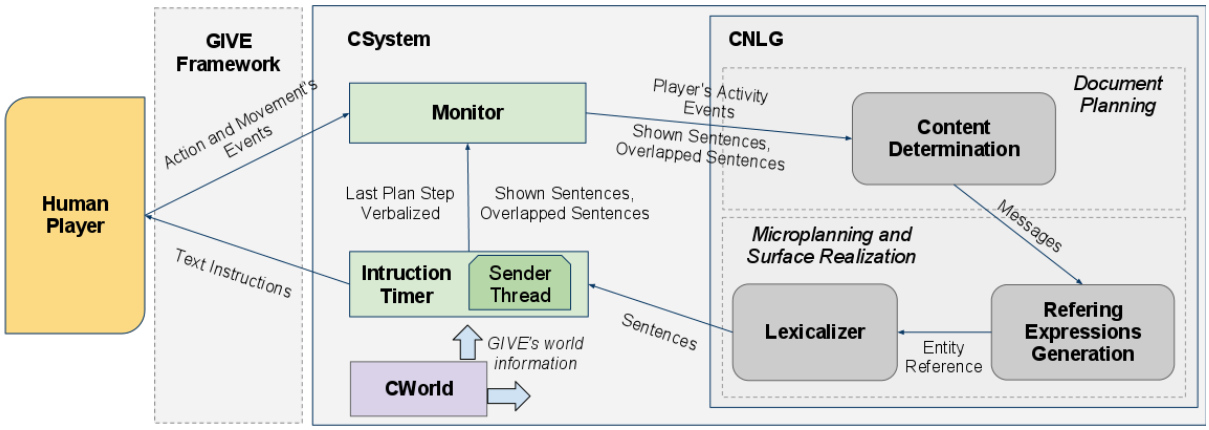


Figure 1: C system's architecture diagram.

player's actions which will be used later by the CNLG module to determine the content of the next instruction to be generated. Given a player's action, Monitor checks if the system's last verbalized plan step has now been accomplished by the player. That is, it verifies whether the player has performed the action previously indicated by the C system. This task is important for the grounding process implemented by the NLG as discussed in Section 4. The monitor also checks if the player is close to an alarm and whether an alarm is visible. In addition, it checks for player's inactivity using a set of timeouts which take into account the time the player is taking to perform the last issued instruction.

The CNLG module is the language generator of the system and is based on Reiter and Dale's architecture. The Content Determination unit uses the current plan and the monitor output to create a list of messages. Each message contains information corresponding to a CNLG's final utterance. Given a set of player's activity events such as a correct/incorrect object manipulation or player's inactivity, the Content Determination module selects from the plan the items that will form part of the next generated instructions. The Lexicalizer and Referring Expressions Generator (REG) modules convert the messages given by the Content Determinator module into a set of sentences objects (each representing a text utterance). The CWorld module provides information to all other modules about the current state of the GIVE World and player status.

Lastly, the Instruction Timer module sends the list of sentences to the human player ensuring that these are shown long enough to be considered completely read. It also determines which utterances will be shown using a priority hierarchy list of the sentence objects. Using this information, it classifies the sentences into shown and overlapped sentences.

## 3 Visibility-based Strategy

The main NLG task in GIVE is helping the human player by communicating a list of steps to reach the trophy. Thus, the NLG must communicate all the steps that compose the plan obtained using the GIVE framework planner. On this context there are three different types of plan steps (PS): movement plan steps (MoPS), object manipulation plan steps (MaPS) and object taking plan steps (TaPS). In GIVE, MaPS are related to pressing buttons where TaPS are actions that imply the possession of an object. MoPS are movement actions indicating the player must move from one region to another.

Since planners cannot handle continues environments, the virtual world has to be discretized. GIVE worlds are discretized into smaller rectangular regions such that, for all pairs of regions A and B, A is adjacent to B if and only if every point of A can be seen from B and every point of B can be seen from A. Two points in different regions can see each other if it is possible to draw a straight line between the two points without intersecting a wall. In GIVE, all rooms and hallways are rectangular and therefore so are all regions.

The strategy of the C system is based on player's

visibility. The C system chooses the next plan step to verbalize by checking whether the plan step's argument (e.g., button or target region) is visible by the player. The plan is a list composed of MaPS, MoPS and TaPS, sorted by the order in which these actions needs to be performed. A GIVE tipical plan has the following form:

$$\begin{pmatrix} MoPS_1^1, MoPS_2^1, ..., MaPS_1, \\ MoPS_1^2, MoPS_2^2, ..., MaPS_2, \\ ..., \\ MaPS_k, MoPS_1^{k+1}, MoPS_2^{k+1}, ..., TaPS \end{pmatrix}$$

Our algorithm selects one plan step at a time checking whether the argument of the first MaPS or TaPS is visible-360° by the player. An object is visible-360° by the player if she can visualize it directly by turning around 360°. If the first MaPS or TaPS do not satisfy this, then the C system takes the sublist $[MoPS_1^1, MoPS_2^1, ..., MaPS_1]$ and looks for the last MoPS that its "to" region is visible-360° by the player. A region is visible-360° when its center point is visible-360°. Therefore, the C system will first refer to the first object that the player has to manipulate if it is visible-360° and it will refer to the last visible region if that object is not visible. The principle of discretization given above ensures that such region exists if the plan is valid. The resulting behavior is to navigate the player referring to the furthest region until the first object to manipulate becomes visible. When giving MoPS instructions, the system replans only when the player has moved off the path enough to lose all MoPS region's visibility.

The C system replans if the player actions invalidate the current plan. This can happen if the player presses a button that was not the next button in the plan or when she goes so far away from the path established by plan that all the regions in the path are no longer visible-360°.

Object's visibility at 360° considers a circular visibility zone. This represents the points from which the player is able to visualize the object at 360°. The zone circle's radius determines the distance from which the system will consider that the player can see the object and it depends on the number of distractors that object has. This value is higher if there are few distractors and it is lower if there are many. By doing this, the C system forces the player to get closer to the target object if there are many distractors near, decreasing the number of visible distractors and thus, facilitating the generation of referring expressions. This also makes the C system capable of giving a button's reference from afar if the button is alone and then easily identifiable.


Figure 2: A referential expression from outside the button's room.

C system verbalizes MoPS referencing their region's center by using direction instructions such as —*Go straight*— or —*Move left*—. Also, while navigating, the system checks if there are alarms between the target region and player's location in a straight line and it warns the player about this. MaPS are verbalized using referential expressions for the target objects. To make this kind of references, the system uses object's type and color, its relative position with respect to others of the same type (e.g., first, second, in the middle) and its relative position with respect to player's location (e.g., on your left). It also implements visual focus and deduction by elimination types of references as —*That one*— or —*Not this one*—. Visual focus and deduction by elimination expressions are generated for trophy objects too, besides button's objects.

The C system visibility-based strategy is a general approach that allows to reference buttons as soon as they become visible (for example, the bottom green button in Figure 2). This strategy is, of course, not without its limitations. We can experience stability issues with respect to the generated descriptions for players that move abruptly (particularly if turning).

## 4 Grounding in Situated Dialogue

When people communicate, they constantly try to arrive to a state in which they believe to have understood, what has been said, well enough for current purposes. The process by which people arrive to this state is called *grounding*. The C system implements three different kinds of grounding.

First, the system grounds new virtual world objects such as alarms and safes. In this process what is grounded is the link between the graphical representation of alarms and safes inside GIVE with the role they play in the game. This grounding process is crucial for completing the GIVE task successfully since the player needs to identify the alarms in order not to lose the game, and she needs to identify the safe in order to win the game. The system implements this grounding process in two stages. The first time the player sees an alarm the system introduces the new object by first describing the object and prompting the player to pay attention to it—*Do you see that red region on the floor?*—then naming it—*That's an activated alarm* and finally describing its effects—*If you step over one of them, we'll lose the game*. In a second stage, every time the player gets too close to an alarm, the system will just present a warning—*There's an alarm, watch out*. The evidence that the alarms have been grounded is quite weak in the GIVE scenario since the player does not need to interact with them but to avoid them. However, we believe it had an impact in the number of lost games (see §5).

Secondly, the system grounds the state of completion of the task. In this process, what is grounded is the effect the player actions have on the state of the task. The C system implements this grounding process in order to minimize the amount of cancelled games following the hypothesis that the player will cancel less if she knows she is advancing in the task. This grounding process is implemented by indicating the effect of the player actions which advance the task—*We've opened one door. We need to open two more doors*—as well as those actions that were incorrect—*Wrong button! We've activated an alarm.* The evidence that the current state of the task has been grounded is non-existing in the GIVE scenario since the player does not react to it in any observable way. However, we believe it had an impact in the number of cancelled games and in the subjective metrics too (see §5).

Finally, the system needs to ground the buttons that the player has to manipulate in order to advance in the task. In this process, what is grounded is the identity of particular buttons that the player has to interact with. This is the grounding task which exhibits the strongest evidence, since the player interacting with the referred button is strong evidence that the intended referent was grounded. However, this is also the grounding task inside GIVE which is more complex since the GIVE worlds are designed such that the intended referents have many distractors (objects of similar characteristics). As a result, the best strategy to implement this grounding process is not to give a referring expression that uniquely identifies the referent but to implement it as a *collaborative* grounding process (as proven empirically by the GIVE-2 NA system (Denis et al., 2010)). The C system implements this grounding process adapting the model proposed by Traum (1992; 1999), which is a computational adaptation of the collaborative grounding model proposed by Clark and Schaefer (1989). In the rest of this section we explain how the C system adapts Traum's model to instruction giving in virtual environments.

Let's consider the following sample interaction with the system:

*IG(1): Press the left blue button*
*IF(2): [Stares at the button on the right]*
*IG(3): Not that button*
*IF(4): [Stares at the left button]*
*IG(5): Yep, that button*
*IF(6): [Pushes the left blue button]*

Traum models the grounding process as a finite state automata. Figure 3 illustrates the part of the automata of Traum's model that was implemented in the C system.

The arc (S,1) represents the contribution to be grounded— contribution (1) in our example. The remaining arcs represent contributions which do not need to be grounded because they are grounding acts—such as contributions (2) to (6)[1] in our example. Contributions (1) to (6) make the automata

[1]Notice that we consider that contributions are not only IG's utterances but also IF's actions.

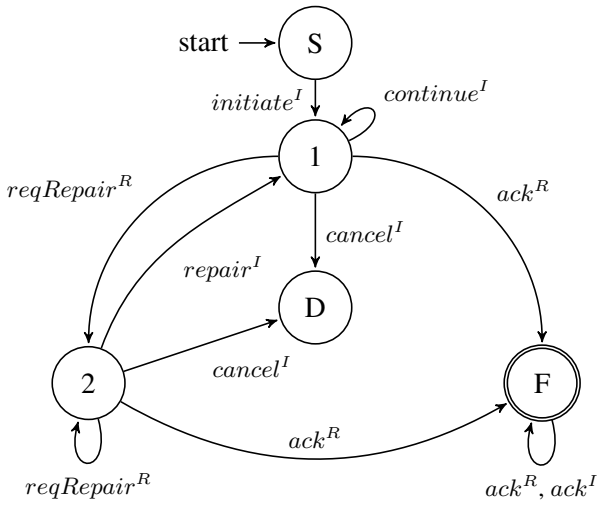


Figure 3: C system's grounding model.

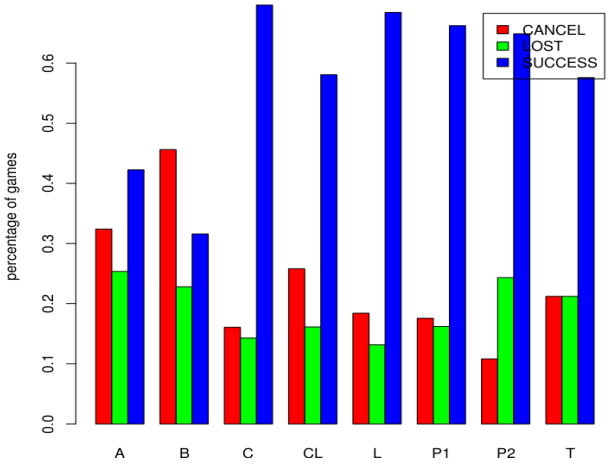


Figure 4: GIVE-2.5 results. Percentage of success, lost and cancelled games by system.

go through the states $\langle S, 1, 2, 1, 2, F\rangle$. That is, (2) and (4)—focusing a possible target and waiting—are treated as request repairs by the receiver R (the IG in this exchange). While (3) and (5) are treated as repairs by the initiator I (the IF in this exchange). Finally, (6) is modelled as an acknowledgement by the IF which grounds the instruction *Press the left blue button*; in the state F the contribution is considered grounded. If the IF would have pressed the correct button right after utterance (1) then the sequence followed would have been $\langle S, F\rangle$. While if the IF would have pressed the wrong button right after utterance (1) then the sequence followed would have been $\langle S, D\rangle$. The state D is a dead state, the contribution is considered ungroundable; after pressing a wrong button the system needs to find a new plan since the ongoing one may no longer be valid.

The C system implements only a part of Traum's grounding model because Traum's model includes the treatment of repairs contributed by the receiver. This is not possible in the GIVE scenario since the IF does not have enough information in order to correct the IG, the IG is the only one that is supposed to have knowledge of the task.

## 5 Evaluation Results

Figure 4 depicts the percentages for successful, lost and cancelled games of the results of the GIVE 2.5. C system's values for cancelled (16%) and lost(14%) games are lower than the observed on the other systems. We think this is a consequence of the grounding strategy for alarm objects and progress information used by C system.

The instructions about progress and effect's descriptions messages also enhanced the system's subjective metrics. For instance, most players thought that C system gave them useful feedback about their progress and most people considered they could trust on C's instructions.

## 6 Conclusions

In this work we have described the C natural language generation system for the GIVE-2.5 challenge. Our system classical grounding models (such as the ones from Traum (1992; 1999)) the process of giving instructions in virtual worlds. The simple grounding process for buttons, alarms, safes and game progress described in Section 4 had a positive impact on the evaluation metrics, as discussed in Section 5.

Moreover, the C system navigation and referring strategy (discussed in Section 3) is based on the area visible to the player. We believe this player-centric approach creates more natural-sounding instructions and reduces the chances for the player getting lost. The fact that these strategies make the C system also independent of the GIVE framework internal representations of concepts has portability implications we seek to explore in further work.

## References


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 *Workshop on Shared Tasks and Comparative Evaluation in Natural Language Generation*.

Herbert H. Clark and Edward F. Schaefer. 1989. Contributing to discourse. *Cognitive Science*, 13:259–294.

Alexandre Denis, Marilisa Amoia, Luciana Benotti, Laura Perez-Beltrachini, Claire Gardent, and Tarik Osswald. 2010. The GIVE-2 Nancy Generation Systems NA and NM. Technical report, Loria/INRIA, France.

E Reiter and R Dale. 2000. *Building natural language generation systems*. Cambridge University Press.

David R. Traum and James F. Allen. 1992. A "speech acts" approach to grounding in conversation. In *ICSLP*. ISCA.

David R Traum. 1999. Computational Models of Grounding in Collaborative Systems. In *working notes of AAAI Fall Symposium on Psychological Models of Communication*, pages 124–131, November.