Context Adaptive Interaction with an Automatically Created Spoken Interface for Intelligent Environments

Germán Montoro, Pablo A. Haya, and Xavier Alamán

Universidad Autónoma de Madrid,
Departamento de Ingeniería Informática,
Ctra. de Colmenar Km. 15. Madrid 28049, Spain
{German.Montoro, Pablo.Haya, Xavier.Alaman}@uam.es

Abstract. In this paper we present the interpretation and generation processes of a spoken dialogue interface for intelligent environments. The interface is automatically created for each specific environment and the interpretation and generation vary depending on the environment and its context. These processes rely on a dialogue tree structure. Several modules process the tree structure and the environment context information to produce specific dialogues for the current environment state. Dialogues are provided with clarification, error recovering, anaphora resolution and other capabilities. The interface is implemented in a real intelligent environment laboratory.

1 Introduction

Intelligent environments have appeared as a new research field in the user interface scientific community. They interact with users in a natural way and support them in their everyday life. Computers and computational devices are hidden from the users, and these get services from the system, for instance, by means of context sensitive spoken natural language interfaces.

The appearance of these environments makes necessary to build new interfaces that allow to interact with the users in a natural way. These interfaces have to be able to carry on conversations regarding the environment, its elements and the services that may be offered to the user. Spoken dialogue interfaces have to adapt to these systems, so that they can cope with the new challenges proposed by the intelligent environments.

We present a Spanish spoken dialogue interface for intelligent environments that adapts to every domain environment. Dialogues are automatically created and they allow to interact with the environment and control its devices by means of spoken natural language interaction. The paper focuses on the interpretation and generation processes and it just briefly explains how the system builds the plug and play dialogues for any given environment, which is thoroughly described in [1].

To develop our research we have built a real intelligent environment. This consists of a laboratory furnished as a living room, provided of a range of devices. These are lighting controls, a door opening mechanism, a presence detector, smart-cards, speakers, microphones, a TV set, an IP video-camera, etc.

A. Aagesen et al. (Eds.): INTELLCOMM 2004, LNCS 3283, pp. 120–127, 2004.

2 Environment and Dialogue Representation

The environment representation is written in an XML document. At startup, the system reads the information from this XML document and automatically builds:

- A blackboard [2], which works as an interaction layer between the physical world and the spoken dialogue interface.
- A spoken dialogue interface which, employing the blackboard, works as an interaction layer between the users and the environment.

The blackboard holds a representation of multiple characteristics of the environment. This blackboard layer isolates the applications from the real world. The details of the physical world entities are hidden from the clients [3].

Entities from the blackboard are associated to a type of entity. All the entities of the same type inherit the same general properties. Some of these properties are employed to create the spoken dialogue interface by means of linguistic parts. This information is composed of a verb part (the actions that can be taken with the entity), an object part (the name that it receives), a location part (where it is in the environment), etc [1].

The dialogue structure is based on a linguistic tree. Every set of linguistic parts is transformed in a tree path, with a node for each part.

As an example, let us suppose that the entity *light_1* has the following two sets of linguistic parts: {“turn_off”, “light”, “”, “ceiling above”, “”} and {“turn_off”, “fluorescent”, “”, “”, “”}, which correspond with three possible ways of interacting with it. In this case, the word part “turn off” is at the same level in both sets of parts so that only one “turn off” node is created, and “light” and “fluorescent” both hang from it. If now we have a new entity called *radio_1*, with this linguistic set of parts: {“turn_off”, “radio”, “”, “”, “”}, the system only has to append the name of the entity *radio_1* to the “turn off” node. Next it adds a “radio” node as its child, at the same level as “light” and “fluorescent”. Starting from an empty tree, the system would automatically create the linguistic tree showed in figure 1.

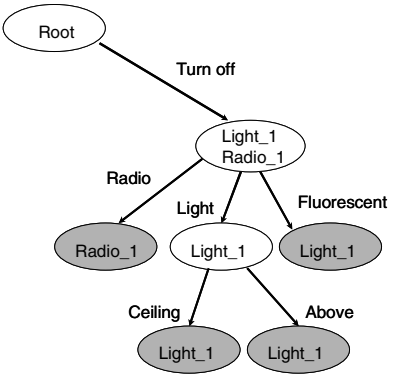

Fig. 1. Linguistic tree for light_1 and radio_1 entities

3 Interpretation and Generation

When the dialogue interface is created the system and the user may carry on conversations about the environment. The interface is managed by a dialogue supervisor, which is in charge of receiving the utterance from the speech recognizer, interpreting it and generating a result (a spoken answer or an action).

Initially, the system is asleep and does not recognize any utterance. If a user wants to initiate a conversation she has to wake it up by uttering the word *odyssey*. At that point, following the ideas defined by [4], the system considers that the goal of all the user utterances is to complete an action. If the user does not say anything in the next seven seconds (or seven seconds after the last dialogue interaction) the system returns to the sleeping mode.

3.1 Interpretation

When the system receives an interaction utterance from the recognizer the supervisor sends it to the utterance process module (UPM). The UPM checks for matches between the utterance and the children of the linguistic tree root (verb part nodes). If there is a match, the UPM goes down the tree to the matching node and checks for new matches with its children. This process continues until the UPM reaches an action node (the sentence was fully interpreted and the system may execute an action) or until there are not more matches (the system needs clarification).

Executing an Action. If the UPM reaches an action node it sends the node information to the node process module (NPM). The NPM gets the word related to the first node of the followed tree path (which corresponds with the verb part, that is, the action that the user has required to take). Then, the NPM orders the execution of the verb part by the action method associated to the reached action node. This execution usually implies a physical change in the environment, although it may also involve a system response. For instance, given the linguistic tree in figure 1, let us suppose that the user utters: *Please, could you turn off the ceiling light*. The UPM receives the sentence and checks for matches with the linguistic tree. *Turn off* matches with the "turn off" verb part node, *light* matches with the child of "turn off", and *ceiling* matches with the child of "light". The "ceiling" node is an action node. Then the NPM takes the control and gets that the action demanded by the user is *turn off*. As this node is associated to the *light_1* entity it executes the *light_1* action method, considering that the action requested by the user is *turn off*. Therefore, this action method can have two effects: it can turn off the *light_1* or it can inform the user about the fact that the *light_1* is already off. In any case, the UPM considers that the action was completed and goes back to its initial state, waiting for other utterances.

Clarification. If, after a user utterance the UPM does not reach an action node, somre clarification is needed. The NPM receives the node information from the UPM. This, again, gets the verb part (the action requested by the user) but does not execute any action. Now, the UPM sends the node information to the tree process module (TPM).

The TPM visits all the children of the current node, constructing an answer sentence to require more information from the user. This sentence is based on the

visited nodes, always considering the environment context represented in the blackboard.

The TPM is based on a recursive depth-first tree search [5]. Its procedure is:

- It visits the first child of the node and checks if the action requested by the user is different to the current physical state of any of the entities associated to the node, employing the blackboard. If so, it stores the name of the entities with a different state and their tree level. Besides, it gets the word related to the node, to build the answer sentence.
- After that it recursively continues with the first child of this processed node, following the same steps.
- The recursive process with the first child of a node is repeated until the TPM reaches an action node or until the node does not have any entity with a physical different state. If the TPM reaches an action node, and it corresponds to an entity that has a different state to the user request, it increments by one the number of actions that can be offered to the user and stores the name of the entity that produces the action. In any case, it goes up to the parent node to continue the recursive tree inspection with the following child of the node.
- When the TPM visits a node which is not a first child (a second or following one), it makes sure that the entities that it contains are not the same as the entities of other processed nodes of the same level. This makes possible that synonyms can be represented in the tree, but only the right one is considered in the interpretation and generation processes.
- This algorithm is repeated until the TPM has inspected all the child nodes of the root and, if necessary, their subsequent children.

The facts of considering only those nodes which have entities with a different state to the state requested by the user and of processing only the first matching synonym make the tree search fully optimal. The TPM only has to follow the paths to the action nodes that can be processed in the current environment state, avoiding all the others.

Once the TPM has visited all the appropriate nodes, it has a full answer sentence (later explained in the generation section) and the number of actions that can be offered to the user. The UPM receives this information and, depending on the number of offered actions, it behaves as follows:

- If the number of actions is equal to zero, the UPM responds to the user that there is not any element in the environment that, in its current state, supports the requested action.
- If the number of actions is equal to one, the UPM directly executes the action method associated to the only entity that can produce an action. With this, we reduce the number of turns and assist the recognizer by employing the current environment context as one of the possible sources of information that improves the interpretation and understanding [6].
- If the number of actions is between one and three, the UPM utters the answer sentence built during the TPM tree search. This sentence presents all the possible entities that can perform the requested action, guiding the user through the dialogue [7].

– If the number of actions is higher than three, the UPM does not offer all the possible options. It utters a clarification question for the general user request. With this the system does not employ a sentence with too many options, which may be difficult to remember and tedious to hear, but still guides the user [8].

As we have seen, the interpretation varies depending on the current environment context. The same utterance may lead to different interpretations in different contexts. For dissimilar contexts the UPM can execute an action or consider that the user wants to interact with some environment entities.

As an example, let us suppose two different cases for the environment represented in the figure 1. They both share the same scenario, where the *light_1* and *radio_1* are both on:

1. In the first case the user utters *turn off the light*, so that the UPM stops at the "light" node. Then the NPM gets the requested action *turn off* and the TPM starts the clarification process. First, it goes down to the "ceiling" node and gets that the environment state of *light_1* (on) is different to the requested state (turn off), so it processes the node. Then, it adds the entity *light_1* to the list of entities visited under the "light" node, appends the word *ceiling* to the answer sentence and, given that it is an action node, increments the number of actions that can be offered and stores the name of the action entity *light_1*. After that it goes back to the "light" node and inspects the "above" node. Then it checks if the entities of this node are in the list of entities processed under the "light" node (this is, if it is a synonym of a previous processed node). As the "above" node belongs to the entity *light_1*, which is in the list of processed nodes for the "light" node, it does not process it. Finally it goes back to the original node, what concludes the tree search. The TPM returns the answer sentence, the number of offered actions and the list of action entities to the UPM. This verifies that there is only one possible action and executes it, that is, it turns off the ceiling light.
2. In the second case the user utters *turn off*, so that the UPM stops at the "turn off" node. Again, after the NPM gets that the requested action is *turn off* the TPM starts with the clarification procedure. First it gets that the state of *radio_1* is different to the requested state and that it is an action node. The TPM adds the entity *radio_1* to the list of processed entities under "turn off", increments the number of offered actions, appends the word *radio* to the answer sentence and adds the entity *radio_1* to the list of action entities. Next it goes back to the "turn off" node and after to the "light" node. Once there it works as explained in the previous case. Finally it goes back again to the "turn off" node and checks the "fluorescent" node. This node contains the same entity as a previously processed node under the "turn off" node (it is a synonym), so it is not considered. The UPM receives the answer sentence, the number of offered actions and the list of action entities. As the number of offered actions is equal to two it utters the clarification sentence: *do you want to turn off the radio or the ceiling light?* Notice that, as we have explained above, multiple synonyms are omitted in the clarification answer (this sentence only refers to the ceiling light and not to the fluorescent).

These very same sentences can produce different interpretations in a different context. Let us suppose a scenario where the *radio_1* is on and the *light_1* is off. If the user repeats the two previous sentences: (1) For the sentence *turn off the light*,

now the UPM will inform the user that all the lights are off. (2) For the sentence *turn off*, now the UPM will turn off the *radio_1*, because it is the only action entity with a different state to the user requested state.

Another possible situation is produced when there is more than one entity of the same type in the environment (for instance, two or more lights). In this case for the user utterance *turn off the light* there can be three possible situations: (1) If all the lights are off the UPM will inform the user about that respect. (2) If only one light is on the UPM will turn off that light. (3) If more than one light is on the UPM will utter a clarification question. This clarification sentence will refer only to those lights that are on, omitting the lights that are already off.

As it has been illustrated in these examples, the interpretation varies depending on the current environment state and the active entities in the environment. New entities can appear or disappear, active entities can change their state or there can be multiple entities of the same type. The dialogue interface adapts to these situations, automatically altering its structure and behavior.

Finally, after a system clarification request the UPM interpretation process suffers a modification in the following interactions. As usually, it checks for matches with the linguistic tree nodes, starting from the root. Nevertheless, after a clarification request it also checks for matches starting from the node where it stopped in the previous interaction. From both tree searches, the UPM selects the node at a lower level or the node that corresponds with an action node. With this, the UPM allows either to continue with a previous interaction (after a clarification answer) or to initiate a new dialogue (leaving behind the clarification dialogue).

Error Recovering. The recognition, interpretation and clarification processes may lead, in some cases, to misrecognitions and misinterpretations. Additionally, users may not provide enough information to process an utterance. To recover of these problems, the system supports some specific features.

As we have seen above, after a clarification answer the UPM permits either to continue with a previous dialogue path or to initiate a new one. This was designed to allow to recover from system misinterpretations. If the clarification answer provided by the system does not correspond with one of the user goals he can start a new dialogue from the beginning, instead of continuing an erroneous dialogue path.

As an additional feature, the UPM does not only check the root of the linguistic tree and the node where it stopped in the previous interaction. If there is not any match for these two nodes, it will also check their children for matches. With this, the system may either recover from speech recognizer misrecognitions or accurately interpret sentences where the user only provided part of the information. For instance if for the scenario represented in the figure 1 the recognizer returns the sentence …*the ceiling light* the UPM will not get any match for the root node. Then it checks the children of the root node, this is, the children of the "turn off" node, and so it gets a match for the "light" node and for its child, the "ceiling" node. It has recovered from recognition noise and thanks to the use of the environment context it may correctly interpret [9] that the original user utterance was *turn off the ceiling light*.

Anaphora Resolution. The spoken dialogue interface supports the resolution of pronominal anaphora to refer to the last mentioned entity.

To allow the use of anaphora the tree is modified to hold anaphora resolution nodes, one for each verb. These nodes are composed by a verb and a pronoun. Besides, after the UPM reaches an action node and executes its corresponding action, it stores the name of the referred object and the followed tree path.

When the UPM reaches an anaphora resolution node (the user has employed a pronoun to refer to an object) it goes to the tree node corresponding to that verb. Once there it goes down through the stored tree path until it reaches an action node and executes its associated action. Let us suppose that a user utters: *could you turn on the radio*. The UPM reaches the "radio" action node, turns the radio on and stores the full tree path of this action. Then the user utters *please, turn it down*. Now the UPM reaches the "turn it down" anaphora resolution node so it goes to the "turn down" tree node. Once there it follows the stored path, this is, it goes down to the "radio" node. As this is an action node, it turns the radio down.

3.2 Generation

As we have seen above, the generation process is carried out at the same time as the clarification process. The answer sentence is formed by words from nodes with entities that have a different state to the user request.

Initially the answer sentence is empty. Before the clarification process, it is filled with the question sentence *do you want to* and the words presented in the tree path that goes from the root node to the node where the UPM stopped. Words are added to the answer sentence by an addition word module (AWM). The AWM gets the number and gender of the word and appends its right form to the answer sentence, preceded by the appropriate article (in Spanish, nouns, adjectives and articles have number and gender). For instance, if in the figure 1 the UPM stops at the "turn off" node, the answer sentence will be initially formed by *do you want to turn off*. After that, the TPM appends the other appropriate tree words that it gets during the clarification process. Words are added as it was explained in the clarification section. Additionally, if a previous word at the same level was already added, before attaching the new word it appends the word *or*, in order to show alternatives. Furthermore, in Spanish it is necessary to append the preposition *from* before the modifier and location parts and the word *to* before the indirect object part. Following with the previous example, the TPM uses the AWM to append the words *the radio*. Next, given that "light" is at the same level as the "radio" node, it appends the word *or* and the words *the light*. After that, as *ceiling* is a location word, it appends the preposition *from* followed by the words *the ceiling*. The final answer sentence is (omitting some words, for a better translation to English): *do you want to turn off the radio or the ceiling light?*.

As it can be seen, the generation process also employs and adapts to the current environment context, automatically obtaining natural and suitable sentences for the given situation.

Additionally, the interface does not only generate answer sentences but also lightweight audio signs [10]. These audio signs are environmental sounds that try to provide information in a non-intrusive way, avoiding to disturb users if it is not necessary. Currently, environmental sounds are employed in two situations:

1. An audio sign (similar to a yawn) is reproduced when the recognizer returns to the sleeping state. If this happened because the user finished the interaction with the environment, it is not necessary to distract her by informing about this subject. If not, the user is still paying attention to the conversation, and an audio sign is enough to let her know about the new recognition state.
2. A different audio sign (similar to an interjection) is used when the UPM does not get any match at all. This sign is repeated in a next unsuccessful interpretation and only after a third consecutive failure, the system informs about the subject and requires the user to change his kind of sentence. We introduced this sign after checking with users that, in a few cases, the recognizer produced substitution or insertion errors, that is, it provided a different sentence to the uttered sentence or it interpreted noise (not an utterance) as a user sentence [11]. In the first case, we verified that it is faster and more efficient to reproduce an audio sign than to utter a whole sentence. In the second case, a recognizer error does not distract unnecessarily the user.

Acknowledgments

This work has been sponsored by the Spanish Ministry of Science and Education, project number TIN2004-03140.

References

1. Montoro, G., Alamán, X. and Haya, P.A. A plug and play spoken dialogue interface for smart environments. In Proceedings of CICLing'04. Seoul, Korea. February 15-21, 2004.
2. Engelmore, R. And Mogan, T. Blackboard Systems. Addison-Wesley, 1988.
3. Salber, D. and Abowd, G.D. The design and use of a generic context server. In Proceedings of Perceptual User Interfaces (PUI'98), 1998.
4. Searle, J. Speech Acts. Cambridge University Press. London, 1969.
5. Cormen, T.H., Leiserson, C. E., and Rivest, R. L.. Introduction to Algorithms. The MIT Press. Cambridge, Massachusetts. London, England. 2001.
6. Ward, K. and Novick, D.G. Integrating multiple cues for spoken language understanding. In Proceedings of CHI'95, Denver, May 7-11, 1995.
7. Yankelovich, N. "How do users know what to say?" ACM Interactions, 3, 6, December, 1996
8. Marx, M. and Schmandt, C. MailCall: Message presentation and navigation in a nonvisual environment. In Proceedings of CHI'96 (Vancouver, April 13-18), 1996.
9. Nagao, K. and Rekimoto J. Ubiquitous talker: Spoken language interaction with real world objects. In Proceedings of IJCAI-95, Vol. 2, 1284-1290, 1995.
10. Mynatt, E.D.; Back, M.; Want, R. and Frederick, R. Audio Aura: Light-weight audio augmented reality. In Proceedings of ACM UIST'97 (Banff, Canada), 211-212, 1997.
11. Schmandt, C. and Negroponte, N. Voice communication with computers: conversational systems. Van Nostrand Reinhold, New York. 1994.