A Clarification Algorithm for Spoken Dialogue Systems

Charles Lewis *Giuseppe Di Fabbrizio*

AT&T Labs – Research
180 Park Ave – Florham Park, NJ 07932 - USA
{clewis,pino@research.att.com}

Abstract

This paper presents an algorithm for spoken dialogue systems that uses mixed initiative interactions and identification of multiple call-types to efficiently clarify the needs of the user. The representation of the application domain includes the relationships between key topics in the domain, the prompts used to discern between these topics, and the call-types associated with the topics. This representation is used by the algorithm to maintain the state of the conversation. By maintaining a picture of how all of the information conveyed by the user fits into this domain, regardless of whether it was information specifically requested by the system, the algorithm expedites the clarification process.

1. INTRODUCTION

Natural language spoken dialogue systems (SDS) are usually designed around a flowchart representation that guides the user through a series of subtasks, where each question permits a limited range of responses that are put together to discern the reason for the user's call. Although effective, the approach does not take advantage of information volunteered by the user which could speed up the process and improve the user experience.

In the approach presented here, a rooted tree data structure is used to represent the dialogue strategy and the relationships between topics. There are several spoken dialogue systems in the literature that use tree-based data structures to model the system knowledge and to disambiguate and clarify the user's inputs. In [1] and [6] an object oriented paradigm is used to represent the system task knowledge and to generate semantically consistent inputs. In this case a Boolean formula minimizes the number of rules needed to describe consistent inputs within the inheritance hierarchy. The application tree in [2] shows a similar approach applied to the DARPA Communicator system where nodes are also constrained by attribute values and rules to express relationship between nodes. Similarly the tree based dialogue in [3] that introduces concept ranking. Finally, the COLLAGEN task-model [4] adopts a general computational model that addresses semantic and referential ambiguities and some grade of dialogue repairs.

The algorithm presented here operates on a compact and effective representation of topic hierarchy or task ontology. The algorithm shortens the length of the dialogue by utilizing all the information offered by the user, including contradictory information. The representation is a concept hierarchy that includes descriptions of the semantic categories the user can reference. The user input is matched to these descriptions, and then one or more system questions are retrieved from the tree to further discern the user needs. The algorithm uses a semantic representation ofthe input to navigate down the tree from the node that suggests the most general possible question to one of the leaves of tree. Each leaf indicates a specific topic that the system can address.

This paper demonstrates the algorithm through an Internal Revenue Service (IRS) voice application that was built as a prototype. We'll use this prototype to demonstrate how the algorithm exploits this hierarchical data structure to interpret various types of input from the user. The algorithm has been implemented in the AT&T Spoken Dialogue System as part of the AT&T VoiceTone® service available to AT&T's business customers. This is part of the dialogue manager framework described in [7] that shows how different dialogue management strategies and algorithms can be mixed together using the concept of flow controllers (FC) and how they share dialogue state and execution history.

2. THE ALGORITHM

2.1. The SLU module

In our system, the user's input, in text form, is passed to a spoken language understanding (SLU) module. The SLU maps the input to a semantic representation that depends on the application domain and a collection of predefined call-types. Different technologies can be used to achieve this goal. Our system is based on an extended version of the boosting-style classification algorithm described in [5]. This classifier can be trained with both labeled data and hand-written rules. Consider as an example the utterance *I need tax information to file my tax returns* in the context of the automated IRS customer care system. Assuming that the utterance is recognized properly by the automatic speech recognizer (ASR) or typed in as simple text, the corresponding call-type should be *Request(Tax_Info),* which is used by the clarification algorithm to generate a subsequent clarification prompt: *I can help you with contact info, individual returns, business returns, or charitable organization returns*. Similarly, the utterance *Do you have any resources for small businesses?* corresponds to the two call-types *Business(Small)* and *Request(Resource).*

2.2. Application Representation

The central data structure used by this algorithm is a rooted tree. Each leaf of the tree represents a successful classification of the user's needs, and each internal node represents a category. A node is defined by a Boolean expression in terms of call-types, confidence assigned to the prediction of the used classifier, and references to the dialogue history. A similar Boolean expression is used to test for the "redo" condition (see section 3.3) for the entire tree. The application author creates this tree and the prompts played for the user as the tree is traversed. Each node in the tree can be described as either *lit* or *not lit*. There is a Boolean flag to describe this state. The algorithm also labels one node as the *focus*. The *focus* of the tree moves from node to node as the tree is traversed. Finally, each node has a set of prompts, used when that node is the focus. In summation, a node is composed of:

- A flag indicating lit status;
- Boolean expressions;
- Prompts;
- Pointers to child nodes, and the parent node.

A tree consists of:

- A root node;
- A pointer to the focus;
- A 'redo' Boolean expression

The prototype system uses XML documents to store the information described above and a run-time interpreter to execute the algorithm that generates VoiceXML code for the VoiceXML interpreter which controls the implementation platform or interactive voice response system. An XPath language interpreter provides flexible ways to query the SLU output and allows the author to easily write complex logical expressions in the tree nodes [8] based on the user input. Figure 1 shows a snippet of XML code representing a node in the tree.

```
<actiondef name="individual"
  text="I have info about EFiling and filing if
you're self-employed.  Which would you like?"/>
...
<node name="individual" parent="intro">
  <conditions>
    <ucondoper="xpath" expr="//class
[@name='individual']"/>
  </conditions>
  <actions>
    <action>individual</action>
  </actions>
</node>
```

Figure 1. Example of node

In this example, XPath is used to extract a call-type from the SLU results, as described in [8]. The action associated with this node (named "individual") is a prompt that requests information to further narrow down the user's needs. The *actiondef* tag defines the prompt the system plays when the node is active. The attribute *parent* of the tag *node* (named "intro") is a unique identifier pointing to the parent node. In the definition, the lit flag and pointers to child nodes are not defined. The former is controlled by the algorithm, and the latter are derived from the parent node pointers.

3. ALGORITHM EXECUTION

3.1. Standard Execution

The algorithm has three steps. In the first step, input is solicited from the user. The prompt used here depends on the current focus node. Each node can have one or more prompts. Additional prompts allow the system to re-query the user (if needed) with different prompts.

Once the algorithm has received input, the other two steps are used to move the focus of the tree. They are repeated until the focus does not change, or a leaf is reached. If the focus does not change, the system prompts the user for more input. If a leaf is reached, the algorithm is complete.

The second step calculates the lit flag for unlit nodes in the sub-tree rooted at the focus node, excepting sub-trees that have been specifically eliminated. The lit flag for a node is true if the Boolean condition for the node is satisfied. These conditions depend on the classification call-types of the input, the confidence scores assigned to the input, and relevant dialogue history queries. This is the step that allows the algorithm to take advantage of user initiative. Every node that can be lit is lit. Also, once a node is lit, it stays lit; so previous utterances from the user are taken into account in the process.

In the third step, the lit nodes are used to move the focus of the tree. The goal is to move the focus away from the root of the tree until it reaches a leaf. The only exception to this is the when the "redo" condition of the tree is satisfied (see section 3.3).

The focus moves towards the leaves by traversing the tree to the lowest common ancestor of the lit nodes in the sub-tree below its current location. The new ancestor does not need to be a lit node. The other way that the focus moves towards the leaves of the tree is by jumping to a direct descendant which is lit. In this case, branches of the tree which contain lit nodes may be pruned off. The first case determines the most specific category that the user utterances have indicated, without eliminating any of the lit categories. The second case can eliminate lit nodes that are not immediately relevant to the current focus.

3.2. Competing Children

In the case where more than one direct descendant of the focus node is lit, the dialogue must revert to a directed dialogue to eliminate unwanted branches. One possible way to do this is to assign a *confirmation prompt* to each node below the root of the tree, which is a simple yes-no question to determine if the category is the one that the user wants. When more than one direct descendant of the focus is lit, the algorithm should iterate through them, asking the category prompt for each. If the reply comes back positive, the focus is moved to that node. If negative, that node and its descendants are trimmed from the tree, and removed from further consideration. This can be repeated until there is only one direct descendant of the focus that is lit.

3.3. The Redo Condition

When the redo condition is met, the algorithm will incrementally step back through the tree to allow the user to change their choices. This process supplants the normal steps for changing the focus.

If any direct descendents of the focus have been pruned (as in the "competing children" condition described in the previous subsection), they are all restored and the focus remains the same. Otherwise, the focus moves away from the leaves of the tree, back to the previous focus node. Any branches that have been pruned from the restored focus node will be restored.

Obviously, if the focus is already the root, and no children have been pruned, it cannot be rolled back further. One possible way of handling this circumstance is by failing out of the dialogue if this is attempted too many times.

4. AN EXAMPLE

To demonstrate this algorithm, we'll look at the IRS application prototype that we have developed. The purpose of this application is to provide information to the user about tax topics. The user may not know how to describe the information that they want, so the algorithm will be used to clarify their inquiries.

As described above, the first step is to author a category tree. The tree for this example is shown in Figure 2. Figure 3 is the legend for all tree diagrams in this paper.

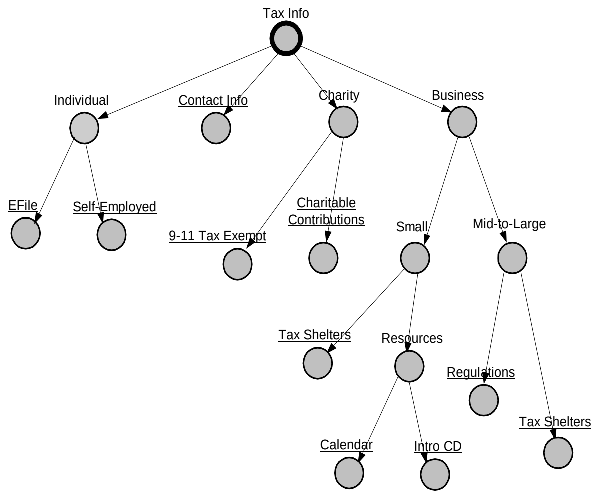

Figure 2: The category tree for the IRS example.

The sample tree has ten leaves. Each leaf represents information that the application can give to the user, such as how to electronically file (*EFile*) their taxes or how to get an informative compact disc (*Intro_CD*) about tax regulations. The intermediate nodes represent categories that can be used to navigate through the tree. Initially no nodes are lit, and the root is the focus.

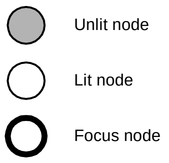

Figure 3: Tree legend

4.1. Example 1: System initiative

This simple example demonstrates the process of guiding the user to what they want through system initiative. Each node has one or more ways of prompting the user to choose one of its children. The "Tax Info" node is the focus when the exchange starts.

- *System*: Hello, this is the automated IRS customer service system. How may I help you?
- *User:* I need Tax Information.

The phrase "Tax Information" does not change the focus. The Tax Info node tries a different prompt.

- *System*: I can help you with contact info, individual returns, business returns, or charitable organization returns.
- *User:* Individual returns.

This lights the node labeled "Individual", and the focus goes there. The relevant part of the tree at this point is shown in Figure 4:

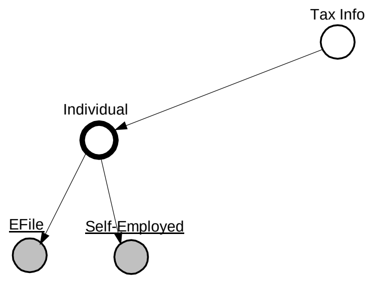

Figure 4: The category tree after the focus changes to the "Individual" node

- *System*: Would you like to hear about e-filing or self-employment?
- *User:* Electronic filing.
- *System:* Electronic payment options are convenient, safe and secure methods for paying income taxes. If you have a balance due, you can e-file and pay in a single step by authorizing an electronic funds withdrawal from your bank account. …

After receiving the user's reply to the new prompt, the node labeled *EFile* is lit and becomes the focus. This is a leaf node, so the algorithm has run its course. The user is given information about how to *EFile*.

4.2. Example 2: Category clarification after user initiative

This example shows how the algorithm requests category information from the user when more than one node is lit by input from the user.

- *System*: Hello, this is the automated IRS customer service system. How may I help you?
- *User:* I need to find out more about tax shelters.

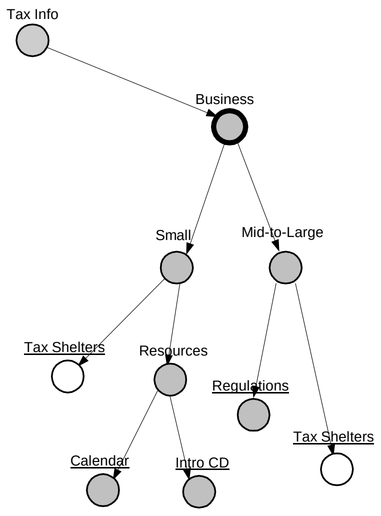

Figure 5: The category tree after the focus has moved to the "Business" node

This input satisfies the Boolean conditions of the two nodes labeled ‘Tax Shelter’. Their most specific common ancestor node, Business, becomes the new focus node (**Error! Reference source not found.**).

- *System*: Are you a small business, or a mid-to-large sized business?
- *User:* A small business.

This satisfies the Boolean condition for the node labeled “Small”. Since it is a direct child of the focus node, it becomes the new focus (Figure 6). This prunes off a branch that contains one of the lit nodes.

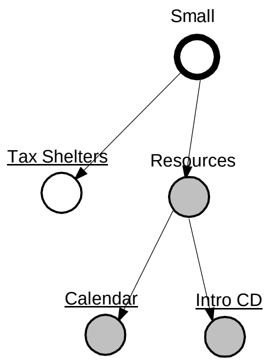

Figure 6: The category tree after the "Small" node becomes the focus.

When the algorithm is applied a second time (without requesting additional input from the user), “Tax Shelter” becomes the new focus because it is a direct child of ‘Small”. The algorithm is always applied until the focus node remains unchanged or a leaf node is reached. “Tax Shelter” is a leaf node so the disambiguation is complete.

Multiple nodes of different types can also be lit up by a single utterance. In the previous example, if the first user utterance had been *I need to find out more about small business tax shelters*, the clarification algorithm would have advanced the focus all the way to the leaf node (the focus would have gone from ‘business’ to ‘small’ without any further input, because small is a direct descendent that would have been lit by this utterance).

4.3. Redo Example

To demonstrate a “redo” action, let’s say that the “Resources” node is also lit in the last step of the previous example, shown in figure 6. This would prevent the focus from automatically moving to the “Tax Shelters” leaf after “Small” becomes the focus. Instead, the algorithm returns initiative to the user. If the user tries to back out of the system by saying, for example, “Go back,” and the “redo” condition for the tree matches the call-type generated by this phrase, then the algorithm will undo the changes made to the structure in the previous turn. In this example, when the “redo” condition is met the focus moves back to the node, “Business”, and the lit status of the node “Small” is changed back to not lit. This is the situation in Figure 5 (except in this example “Resources” is also lit). The user can now indicate that they have a mid-sized or large company and explore the other sub-tree under the “Business” node.

4.4. Mixed Initiative

The benefits of a mixed initiative strategy are leveraged here by allowing the user to express their interests as completely as they can before the system takes the initiative and presents questions to clarify the final topic. This can avoid a long series of unnecessary questions, or a prompt with a long list of possible options. When system initiative is used, the options presented by system prompts can be constrained to short lists to take advantage of the category tree structure. In the case where a user is unfamiliar with the options available, they can still cede initiative completely, and allow the system to guide them through the hierarchy of topics to their final choice.

5. CONCLUSIONS

The algorithm we’ve presented here uses a simple, hierarchical description of the application topic tasks to both guide and expedite the clarification process. Through this process, vague subject references are systematically clarified to arrive at defined subjects. The combination of mixed initiative and multiple call-types per utterance is key to this algorithm, which leverages everything that the user has said to expedite the clarification process. For users who take the initiative, this produces shorter dialogues, and dialogues that require less repetition than simple directed dialogue systems.

6. REFERENCES

[1] A. Abella and A.L. Gorin, “Generating Semantically Consistent Inputs to a Dialog Manager”, *Proc. Eurospeech*, Rhodes, Greece, Sep. 1997.

[2] E. Ammicht, E. Fosler-Lussier, and A. Potamianos, “System and method for representing and resolving ambiguity in spoken dialogue systems”, *Proceeding of Eurospeech*, vol. 3, pp. 2217-2220, Aalborg, Denmark, September 2001

[3] K. Macherey and H. Ney “Features for Tree-Based Dialogue Course Management”, In *Proc. European Conference on Speech Communication and Technology*, Vol. 1, pp. 601-604, Geneva, Switzerland, September 2003

[4] Rich, C.; Sidner, C.L.; Lesh, N.B., "COLLAGEN: Applying Collaborative Discourse Theory to Human-Computer Interaction", *Artificial Intelligence Magazine*, Winter 2001 (Vol 22, Issue 4, pps 15-25)

[5] R. E. Schapire and Y. Singer, "Boostexter: A boosting-based system for text categorization", *Machine Learning*, 39(2/3):135-168, 2000

[6] A. Abella and A. Gorin, “Construct Algebra: Analytical Dialog Management”, *Proc. ACL*, Washington D.C., June 1999.

[7] G. Di Fabbrizio and C. Lewis, “Florence: a Dialogue Manager Framework for Spoken Dialogue Systems”, *Proceedings of the 8th International Conference on Spoken Language Processing (ICSLP 2004)* , Jeju, Jeju Island, Korea, October 4-8, 2004.

[8] G. Di Fabbrizio and C. Lewis, “An XPath-based Discourse Analysis Module for Spoken Dialogue Systems”, *The Thirteenth International World Wide Web Conference*, New York NY, May 17-22, 2004.