or in translated form.[1]

understands speech was (and still is) largely unknown, and because the available computer processing power

| User Speech | Computer Output | Commentary |
| --- | --- | --- |
| for statement | `for( ; ; ) f g` | Inserts a for statement |

was told that his partner was sitting at the keyboard, and the participant had to get the gist of the code

{ Too few or two many words

*Students didn't say most of the punctuation characters*. Semicolons were never spoken, open and close parens as part of method calls were never spoken. Sometimes the commas separating arguments to a method call were spoken, but this was inconsistent. Methods with more arguments tended to cause people to say the comma. Dots marking quali ed identi ers were sometimes spoken and sometimes skipped, though it was far more common to say them. Close

representation of the program that may be required by some text-based tools [65].

HARMONIA

had been identi ed by the successful active parsers.

Our GLR parser is slightly more elaborate than what I've described here because operates incrementally. If there are changes in the input token stream, at any place in the stream, the parser can reparse the changed tokens without deconstructing and reconstructing the entire parse forest. This incrementality is crucial to supporting vocal input in real-time.

incorporate these latter ideas into our prototype programming by voice editor. In addition, we will provide access to the standard speech recognition tools for improving spoken language recognition accuracy and for adding new words to the dictionary.

# 6 The Program Editor

If we can keep the size and complexity down through aggressive ambiguity resolution, we should be able to avoid performance issues.

## 6.2 Program Composition

Supported by the HARMONIA analysis framework, we can provide several forms of program composition that will better approximate how people speak code today. First, we propose to support a template instantiation feature. Each production of the grammar can be associated with a spoken keyword. When this keyword is spoken, an instance of the grammar production can be inserted into the parse tree at the current cursor location, where terminals are lled in by text, and non-terminals are lled in by place holders. For example, the user might say `if` and get `if (null_predicate) then` *f* `null_statement` *g*.

Another form of code entry is to support textual macro expansions. Cthe pars Td369(expai1ars -368(v)28(lace)pair

our automatic disambiguation techniques, we will add these as well, in order to spare the user a lot of needless dialogue from the editor about disambiguating parse trees.

Is it due to improper feedback from the editor, or is it due to inaccurate speech recognition? How many and what type of mistakes does the programmer make? How often does he not know what to say or not know how to verbalize something? This includes not knowing what to call a particular language construct (before we support pointing gestures), and also includes misuse of the programming langage grammar. Does the user experience voice strain? How much training in speech tools is necessary before the proper techniques to avoid RSI are learned? Finally, do people like programming by voice? What about it appeals to them? What tasks are annoying or di cult to accomplish?

# 9 Dissertation Contributions

[52] Gerd Szwillus and Lisa Neal. *Structure-Based Editors and Environments*. Academic Press, New York, NY, USA, 1996.