ITS Tools for Natural Language Dialogue: A Domain-Independent Parser and Planner

Vali Georgescul

October 8, 2009
Outline
Introduction

Current situation

Overview of the ATLAS project

Atlas-Andes

Prototype implementation

APE

CARMEL

Questions
2. Introduction
2.1 Current situation
Research shows that students learn more effectively when they express their thinking in words.

When students are forced to construct knowledge for themselves, the learning process is more effective.
Research shows that students learn more effectively when they express their thinking in words.

When students are forced to construct knowledge for themselves, the learning process is more effective.

Without natural language dialogue, the best approach for tutoring systems consists of hints.
ITS landscape

- Hints provide a one way communication channel.
- No way for the student to take initiative and ask a question.
- The only way to lead the student through a multi-step directed line of reasoning or to ask the student a question is via a list of precoded answers.

On the other hand, ITSs with NLP can further benefit from common effective rhetorical methods such as analogies and reductio ad absurdum.
Hints provide a one way communication channel. No way for the student to take initiative and ask a question. The only way to lead the student through a multi-step directed line of reasoning or to ask the student a question is via a list of precoded answers. On the other hand, ITSs with NLP can further benefit of common effective rhetorical methods such as analogies and *reductio ad absurdum*. 
In spite of this, there are not many ITSs which support natural language input.
In spite of this, there are not many ITSs which support natural language input.

The reason behind this apparent paradox is the fact that natural language processing systems require tremendous amounts of time, effort and expertise.
2.2 Overview of the ATLAS project
Motivation

- Allow students to learn by conversing with a natural language based ITS.
- Do the hard work once, benefit many times.
Two components of the Atlas project will be presented here:

1. APE - Atlas Planning and Execution
2. CARMEL - Core component for Assessing the Meaning of Explanatory Language
Two components of the Atlas project will be presented here:

1. APE - Atlas Planning and Execution
2. CARMEL - Core component for Assessing the Meaning of Explanatory Language

APE + CARMEL - domain independent rule based subsystems.

Together, they provide a framework for extending an ITS with NLP capabilities.
To illustrate/explore the feasibility of the tool-set, a prototype ITS was built: Atlas-Andes
To illustrate/explore the feasibility of the tool-set, a prototype ITS was built: Atlas-Andes.

An existing physics ITS, Andes, was extended to include natural language capabilities.
3. Atlas-Andes
Andes is a first year college physics tutoring system. Students provide input to the system through the GUI. Feedback is provided through hints.
Sample problem

An elevator slows to a stop from an initial downward velocity of 10 ms in 2 sec. A passenger in the elevator is holding a 3 kg package by a string. What is the tension in the string?
Sample problem

An elevator slows to a stop from an initial downward velocity of 10 ms in 2 sec. A passenger in the elevator is holding a 3 kg package by a string. What is the tension in the string?
S: <draws acceleration vector in same direction as velocity>
T: What is the definition of acceleration?
S: I don’t have any clue.
T: OK, let’s try this. If a car was driving along east, which way would you have to push to make it stop?
S: West.
T: Exactly, the opposite direction. What does that tell you about the direction of acceleration?
S: It goes west too.
T: Right. The net force goes the opposite direction, and so does the acceleration. Try to draw the acceleration vector again now.
S: <draws acceleration vector correctly>
3.1 Prototype implementation
Architecture: Atlas-Andes

ATLAS - Andes Architecture

- User Interface Manager
- GUI Interpreter (Andes)
- Language Understanding (CARMEL)
- Host Tutor (Andes)
- Tutorial Planner (APE)

- User
- Grammar
- Lexicon
- Semantic Rules
- Tutorial Agenda
- Tutorial History
- Plan Library
To apply the Atlas tools to a new domain, two things are required:

1. a plan library, to guide the tutorial planner (APE)
2. semantic mapping rules, to guide the input understander (CARMEL)

A corpus of transcribed dialogues between 2 experienced tutors and 20 students trying to solve physics problems was used.
A plan library of around 100 plan operators.

An equivalent number of operators for:
- dialogue creation
- responding to specific user misconceptions
- handling domain independent issues
Implementation

3.1

A plan library of around 100 plan operators.

An equivalent number of operators for:

- dialogue creation
- responding to specific user misconceptions
- handling domain independent issues

21 semantic mapping rules.
3.2 APE
Key traits

▶ Reactive planning (chess game) vs planning based on finite state machines
Key traits

- Reactive planning (chess game) vs planning based on finite state machines
- Hierarchical structure of task oriented dialogues
Key traits

▶ Reactive planning (chess game) vs planning based on finite state machines
▶ Hierarchical structure of task oriented dialogues
▶ Local goals
The system stores its intentions (goals) in an agenda

Agenda - implemented like a stack with extra operations

Each entry
1. Goal
2. Plan operator
The system stores its intentions (goals) in an agenda. Agenda - implemented like a stack with extra operations. Each entry:

1. Goal
2. Plan operator

Operators are selected based on:

- historical information
- the current situation (current goal, students latest response)
- domain logic
(def-operator handle-same-direction
    :goal (...)
    :filter (...)
    :precond (...)
    ; We have asked a question about acceleration
    ; ... and the student has given an answer
    ; ... from which we can deduce that he/she thinks acceleration and velocity go in the same direction
    ; and we have not given the explanation below yet
    :recipe (...)
    ; Tell the student: "But if the acceleration went the same direction as the velocity, then the elevator would be speeding up."
    ; Mark that we are giving this explanation
    ; Tell the student that the tutor is requesting another answer
    ; ("Try again.")
    ; Edit the agenda so that tutor is ready to receive another answer
    :hiercx ()
Goal matching

- System searches the operator library to find all operators that match the goal on top of the agenda.
Goal matching

- System searches the operator library to find all operators that match the goal on top of the agenda.
- A goal is matched if the *preconditions* and *filter conditions* are satisfied.
Goal matching

- System searches the operator library to find all operators that match the goal on top of the agenda.
- A goal is matched if the preconditions and filter conditions are satisfied.
- Preconditions are used for characteristics that can change during the process.
- Filter conditions are used for static properties (e.g. properties of domain objects).
Goal matching

- System searches the operator library to find all operators that match the goal on top of the agenda.
- A goal is matched if the preconditions and filter conditions are satisfied.
- Preconditions are used for characteristics that can change during the process.
- Filter conditions are used for static properties (e.g. properties of domain objects).
- Goals are represented using first order logic (without quantifiers).
- Unification is used for matching.
- If more than one operators match, the last one found will be used.
Operators are represented as *multi-step recipes*

When a match is found, the matching goal is replaced by the operator (recipe)
Recipes

Operators are represented as *multi-step recipies*

When a match is found, the matching goal is replaced by the operator (recipe)

A recipe:

- **BEGIN** marker
- First step
- Second step
- ...
- **END** marker
Operators are represented as *multi-step recipies*

When a match is found, the matching goal is replaced by the operator (recipe)

A recipe:

- **BEGIN** marker
- First step
- Second step
- ...
- **END** marker

The end marker contains the goal that triggered the operator
Recipe types

Goal  Create subgoal

Primitive  Do an action (say something or graphical)

Interactive primitive  Say something and cease control for reply

Assert  Add a ground fact to the transient knowledge base

Fact  Evaluate a condition (if false, skip the rest of the recipe)
Recipe types

§3.2

Retract  Remove all matching facts from the transient knowledge base

Retry-at  If there are multiple ways of satisfying a goal, retry-at allows one to choose among them a second time if one’s first choice is later shown undesirable. (can be used to implement loops - along with fact)

Prune-replace  If the response to a particular question is unsatisfactory, one might want to start all over with a more refined goal/goals. This can also be done with retry-at.
Data storage

Two knowledge bases:
- Permanent information (domain facts)
- Transient facts (become true during program execution)
Data storage

- Two knowledge bases:
  - Permanent information (domain facts)
  - Transient facts (become true during program execution)
- Facts are added/deleted in the transient database with *assert* and *retract*
Data storage

3.2

- Two knowledge bases:
  - Permanent information (domain facts)
  - Transient facts (become true during program execution)

- Facts are added/deleted in the transient database with `assert` and `retract`

- The items from the current discourse context (`:hiercx`) are kept in the transient db as long as the operator which spawned them is on the agenda
Two knowledge bases:
- Permanent information (domain facts)
- Transient facts (become true during program execution)

Facts are added/deleted in the transient database with *assert* and *retract*

The items from the current discourse context (:hiercx) are kept in the transient db as long as the operator which spawned them is on the agenda

Communication with other agents (including UI) is done via the transient db
3.3 CARMEL
Provides the underlying linguistic knowledge that makes it possible to generalize the examples annotated by the authors into patterns that can match against unseen data.

Thus, used for processing student input.
Architecture: CARMEL

- Sentence
- Morphological Analysis
- Spelling Correction
- Lexical Lookup
- Lexical Input Graph
- Robust Parsing
- Parse Result
- Repair
- Parse Quality
- Final Result
- Partial parse
- Full Analysis
1. Lexical preprocessing

1. Morphological analysis - segmentation into morpheme

Morphology is the branch of linguistics that studies patterns of word formation within and across languages and attempts to formulate rules that model the knowledge of the speakers of those languages.
1. Lexical preprocessing

1. Morphological analysis - segmentation into morpheme

   Morphology is the branch of linguistics that studies patterns of word formation within and across languages and attempts to formulate rules that model the knowledge of the speakers of those languages.

2. Spelling correction - computes possible replacement words based on the form of the current, previous and next word.
1. Lexical preprocessing

1. Morphological analysis - segmentation into morpheme

   **Morphology** is the branch of linguistics that studies patterns of word formation within and across languages and attempts to formulate rules that model the knowledge of the speakers of those languages

2. Spelling correction - computes possible replacement words based on the form of the current, previous and next word

3. Lexical lookup - used for the retrieval of lexical entries
1. Lexical preprocessing

- Retrieves all lexical entries (from the lexicon) that match the root form of each word
- The parser will perform a syntactic and semantic analysis on these lexical entries
- Robustness - coordination of the 3 sub-stages
2. Robust Parsing

- LCFLEX

- Robust, flexible parser (introduces flexibility as needed)
  - Parameterized flexibility:
    - Skipping over words
    - Insertion of words
    - Relaxation of grammatical constraints
  - Ambiguity packing and pruning, statistical disambiguation
2. Robust Parsing

- LCFLEX
- Robust, flexible parser (introduces flexibility as needed)
- Parameterized flexibility:
  - Skipping over words
  - Insertion of words
  - Relaxation of grammatical constraints
- Ambiguity packing and pruning, statistical disambiguation
3. Repairing

3.3

- The allowed flexibility can be too high, deviating too much from the grammar’s coverage.
3. Repairing

3.3

- The allowed flexibility can be too high, deviating too much from the grammar’s coverage.

- A fragmentary analysis is passed over to the repair module, which assembles the fragments.
Meaning Representation Specification

§3.3

- Frame based DSL
CARMEL Performance

- CARMEL is fully implemented

100 spontaneous student utterances, ranging between 1 and 20 words long in the Newtonian physics domain.
87% with an average runtime of .1 seconds per utterance.
CARMEL's LCF LEX robust parser was tested extensively in the context of the JANUS large scale multi-lingual speech-to-speech system.
73.3% acceptable translation in an evaluation over a set of 500 randomly selected spontaneous utterances in the Appointment Scheduling domain.
CARMEL Performance

- CARMEL is fully implemented
- 100 spontaneous student utterances, ranging between 1 and 20 words long in the Newtonian physics domain.
- 87% with an average runtime of .1 seconds per utterance.

CARMEL’s LCF LEX robust parser was tested extensively in the context of the JANUS large scale multi-lingual speech-to-speech system.

73.3% acceptable translation in an evaluation over a set of 500 randomly selected spontaneous utterances in the Appointment Scheduling domain.
CARMEL Performance

- CARMEL is fully implemented
- 100 spontaneous student utterances, ranging between 1 and 20 words long in the Newtonian physics domain.
- 87% with an average runtime of .1 seconds per utterance.
- CARMEL’s LCFLEX robust parser was tested extensively in the context of the JANUS large scale multi-lingual speech-to-speech system.
- 73.3% acceptable translation in an evaluation over a set of 500 randomly selected spontaneous utterances in the Appointment Scheduling domain.
4. Questions
Questions

▶ How does this plan based approach compare to the strategies used by the system Johan is working on?
▶ Do you think it’s feasible/worth it to extend the strategy based approach with natural language processing capabilities?