An Immersion Environment for Second Language Learning

Lottie Price

Supervisor: Gordon I. McCalla

Abstract

Although it is widely believed that the best way to learn a second language is by using it in an immersion environment, there are few computerized language tutors that provide such an environment. A computerized immersion environment that allows the user to enter text freely, instead of having the student choose from a menu or translate a specific sentence, needs to be able to perform a fair amount of natural language recognition and needs to be fairly robust in order to gracefully handle any ill-formed input generated by the student.

Natural language processing is, in general, a very difficult problem. However, if the situation in which the language is being used is restricted enough it may be possible to build a parsing mechanism sufficient to be able to understand the user's input.

This paper explores the design for a restricted-domain immersion environment where the student can enter free text. The domain chosen is the interaction between a customer and a waiter in a restaurant.

1 Introduction

1.1 Why an Immersion Environment?

It has been argued that the best way to learn a second language is in an immersion environment [6], yet there are relatively few computerized immersion environments for teaching second languages. Most computerized language tutors are based on simple word or sentence exercises. The FLUENT system [6] is one of the rare exceptions: it is based on a simple immersion environment where the student exercises her language skills in attempting to achieve goals. However, it is not clear from [6] how much conversational interaction actually takes place between the system and the student and how much of the learning takes place in monologues by either the student or the system.

In most tutors, exercises are formulated as either translations ([1], [9]), fill-in-the-blanks([2], [7]), or morphological form-changing exercises ([3], [5]).

Translation exercises at the word level are limited by the lack of context: the student learns the word in isolation and doesn't learn its full range of connotations. Often, the computer system accepts only a small number of translations as correct, despite the variety of answers that could conceivably be correct. In sentence translation exercises similar to those in [9], the exercise sentences are often worded in such a way as to make them easier to translate into the second language, as opposed to being written in a way that is natural in the first language. The student may feel frustrated when she attempts to translate her own sentences and ideas, which have not been so carefully worded.

Fill-in-the-blank exercises provide more context and avoid some of the difficulties of simple translation exercises, but also require a lot of forethought in specifying a full range of appropriate answers [10].

Morphological exercises, those based on changing the structure of a word to fit the context, are useful for practicing and emphasizing a part of the language. Their emphasis on a single aspect of the language at a time is what makes them effective tools for learning that aspect, but they do not build confidence in the student's ability to express themselves any more than the other styles of exercises.

Although tutors based on structured exercises have a role in teaching a second language, it is also important to give the student an opportunity to use the language spontaneously in order that they may feel confident in their ability to express themselves and be understood. Immersion environments can provide this opportunity.

1.2 The Restaurant Domain

The domain of a restaurant, besides being one of the topics that appears in every "learn to speak language X" phrase book, has some nice features from both a computational and a pedagogical perspective. The roles of the waiter and the customer are well defined and the interactions between them are fairly stereotypical, so they can be modelled in a straightforward manner. Pedagogically, the familiarity of the stereotypical roles allows them to concentrate on the language aspects of the interaction.

In a restaurant, the waiter is expected to lead the interactions, although the customer is not strictly constrained to answering the waiter's questions. Mixed-initiative dialogues, which are pedagogically desirable [12], are natural in this domain. As in a real restaurant, when the student/customer cannot be understood after a reasonable effort, the waiter can lead the conversation back to known territory and things do not stall out completely. Mixed-initiative dialogues are thus feasible because the waiter has a natural way to take control of the conversation when things get too far astray.

2 System Description

This section discusses the proposed architecture for a second language immersion environment embedded in a tutor. A view of the system from the student's perspective is presented. The interactions between the major components of the tutoring system as a whole are briefly discussed, followed by a more detailed presentation of the immersion environment. The section on the immersion environment focuses especially on two aspects of the environment, the waiter and the scenarios.

2.1 Student's Perspective of the System

The student starts a session with the tutoring system by selecting a typical restaurant scenario from a list of scenarios chosen by the tutor as being appropriate. Some example scenarios might include ordering a drink, figuring out what to do or say when the waiter brings the wrong dish, or asking for a high chair for a young child. Each scenario contains an introductory section containing structured exercises to introduce the appropriate vocabulary and grammar structures. When the student has completed the introductory lessons, she enters the immersion environment. The student and the waiter converse approximately according to the details of the scenario. A sample conversation may be seen in Figure 1. When the scenario is over, the student receives feedback from the tutoring component or a real tutor regarding the conversation.

2.2 Major Components of the Tutoring System

A simplified diagram of the second language tutoring system is shown in Figure 2. The student interacts with the waiter in the immersion environment and with the tutoring component. Within the immersion environment, the scenario acts as a "script" [11] for the waiter. It contains a partially ordered list of "steps", or goals, relating to that scenario. The waiter uses a robust parser, his knowledge about restaurant conversations in general, and the goals or expectations from the current scenario to guide the understanding process. The restaurant fills the waiter's orders or informs the waiter when it

W: Can I get you something to drink?

C : Yes, do you had root beer?

W: No, sorry. We've got cola, orange, and ginger ale.

C : Maybe I have ginger ale then.

W: Okay.

Figure 1: Sample dialogue between the student and the system. The student plays the role of the Customer (C) and the system plays the role of the Waiter (W). Note that the student's input contains errors.

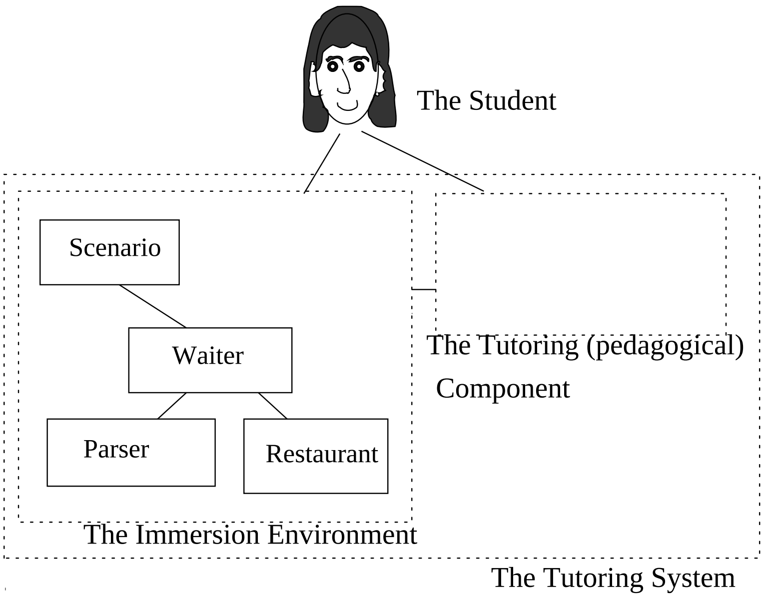

Figure 2: Architecture for the overall tutoring system

is not possible to fill the orders. At the end of the immersion simulation, the information about the conversation that took place can be passed to the tutoring component so it can give meaningful feedback to the student. Currently, the tutoring role would be filled by a human tutor; however, a computerized tutor is envisioned for the future. It is likely that a computerized tutor would need to have its own parser, since the goals of the waiter are to respond to the student in the most appropriate way, whereas the goals of the tutor are to correct the learner and comment upon the grammaticality and appropriateness of her utterances.

2.3 Inside a Scenario

A scenario contains the knowledge about how to proceed in the conversation. Figure 3 shows an expansion of the *drink not available* scenario. A scenario is comparable to a frame [8] or a script [11], only it has more than just slots and the slots are more than just blanks to fill. A scenario is made up of a collection of possibly ordered "steps", such as *set up the restaurant*, *say "you're welcome" when a customer thanks you*, *get a drink order*, or *serve drink*. Figure 3 shows just two of these steps, although the other two could also be a part of this scenario. Whether a step will be activated and when is determined dynamically based on the step's enabling condition.

A step's enabling condition can depend on the student's input (for example, the step *ask about travellers' cheques* in Figure 4 gets invoked when the student asks whether the restaurant accepts travellers cheques as payment), internal ordering between the steps (for example, set up the restaurant

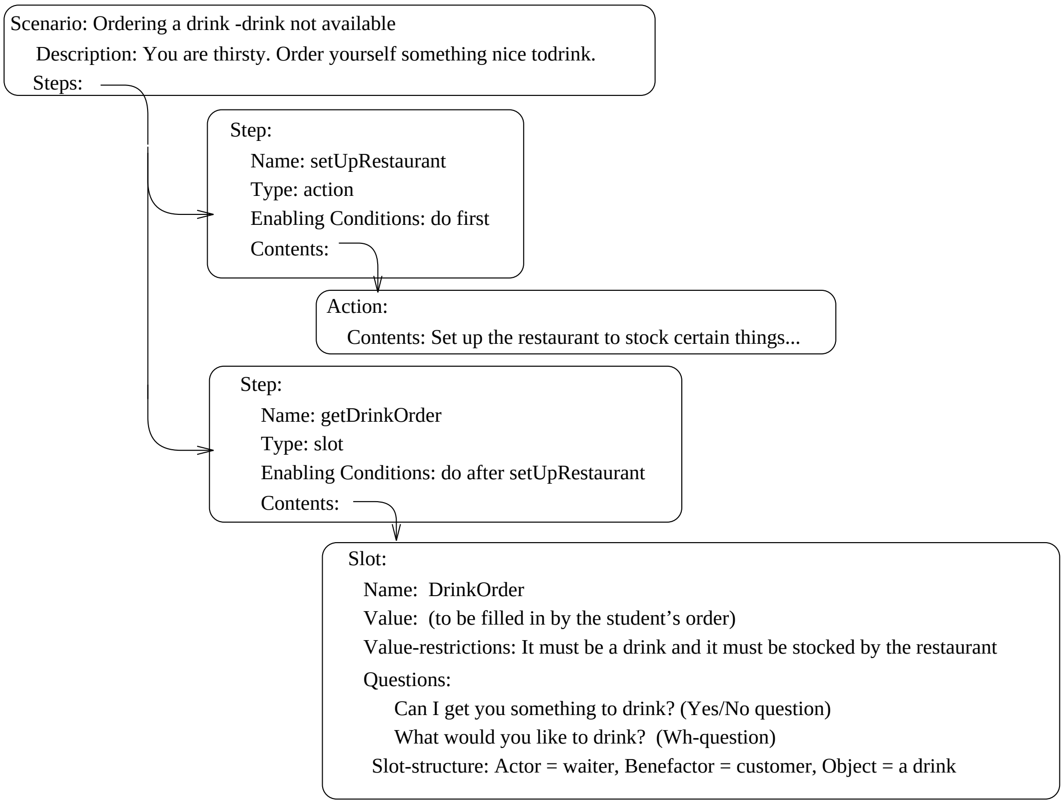

Figure 3: The *Drink not available* scenario

before getting the drink order), or the state of the restaurant (for example, if the kitchen is out of whatever the customer ordered, inform them and try to elicit another order). Steps act as a container for holding and ordering various activities one might want to do in a scenario. These activities can be used to categorize the steps into four types: slot steps (used to elicit information from the customer), protocol steps (when something needs to be said or done to satisfy some social protocol), action steps (used mainly to change the state of the restaurant), and scenario steps (used to embed simple scenarios into more complex ones).

2.3.1 Slot Steps

Slots are the most important kind of step. They require the waiter to elicit information from the customer; for example, to find out what the customer's food order will be. A given slot may be referred to by many steps in many different scenarios. For many slots, it is important to be able to access them independently of the scenarios or the steps they are implicated in because the customer may supply information for a different slot than is expected and the waiter should be able to process and record the information anyway. For example, the customer may order food and drink at the same time but the waiter may only be expecting a drink-order. Some slots should have a local scoping, however. For example, it makes sense that a slot referring to cream or sugar not get created until coffee has been ordered. These slots can be considered as part of another object, such as coffee-order, that do not get instantiated until the main object (coffee-order) is instantiated.

Slots may have restrictions on the types of values they will accept as valid. In the example in Figure 3, the *drink order* slot requires that the value be a drink and that the drink be stocked by the restaurant. Slots contain some questions that are useful for extracting the information they need to fill their value. They also contain some information about typical thematic roles [13] that conversations about the slot are likely to use, which is called the slot-structure in Figure 3. The *drink order* slot, for example, knows that the thematic role of *benefactor* will be played by the customer. The thematic roles in the slot-structure help the parser distinguish between things like "Could I have a coffee, please?" and "Do you have coffee?". Future work could involve generating the slot's questions directly from the knowledge of the slot-structure, but for this project the focus is on natural language understanding, so all output text is pre-formulated.

2.3.2 Protocol Steps

Steps can also be protocols. Protocols typically involve speech acts. They are things done and said not to elicit information but in order to fill linguistic or pragmatic goals of a conversation. For example, when a customer says "thank-you" after being served something, the waiter should say "you're welcome", even though no slots get filled and the environment does not change. Most protocols, such as the *thank-you* protocol, should be a part of the waiter's knowledge since they involve general knowledge about how to act the role of a waiter in a restaurant. There may also be some protocols that are limited to certain scenarios, such as the greeting protocol that is used when a customer first enters a restaurant.

2.3.3 Action Steps

The kind of action done in an action step is relatively arbitrary, though most will be messages to the restaurant to change somehow. Examples of actions include preparing the menu for the day, setting variables to determine how the restaurant will behave, or (perhaps) calculating the total on a customer's bill.

2.3.4 Scenario Steps

Scenario steps are simply steps that call up a new scenario. They can be used to build complex scenarios out of simpler ones. The *full restaurant enactment* scenario in Figure 4 is an example of a complex scenario construction.

2.4 Organization of Scenarios

Scenarios can be grouped to reflect similar or shared properties. In Section 2.3.4 we saw that they can be embedded in each other using the steps, but they can also be structured to inherit from each other. These two dimensions, aggregation and abstraction respectively, correspond approximately to the dimensions in SCENT's [4] ontology. The abstraction dimension allows steps that are common to many scenarios to be inherited. This is useful for digressions that could appear anywhere in a given part of a conversation, such as the user asking whether travellers cheques can be used as payment. The aggregation dimension allows complex scenarios to be built up from simpler scenarios. The aggregated scenarios appear as separate steps in a scenario. The aggregation dimension is useful for acquiring many steps at once in a pre-grouped order.

Figure 4 shows a small part of the scenario hierarchy. The *full restaurant enactment* scenario inherits various steps from *in a restaurant*, most of which would be activated by a certain input. It also has two sub-scenarios: *drink not available* (as expanded in Figure 3), which inherits from *order drink*, and *order meal*.

3 System Dynamics

An important aspect of the immersion environment that has not been discussed yet is the system dynamics: how the information in the various objects gets used to carry out a plausible dialogue with the student. The following section gives a brief discussion of one of the main algorithms for deciding what to do next, the slot-dialogue algorithm. The processing of the sample dialogue in Figure 1 is

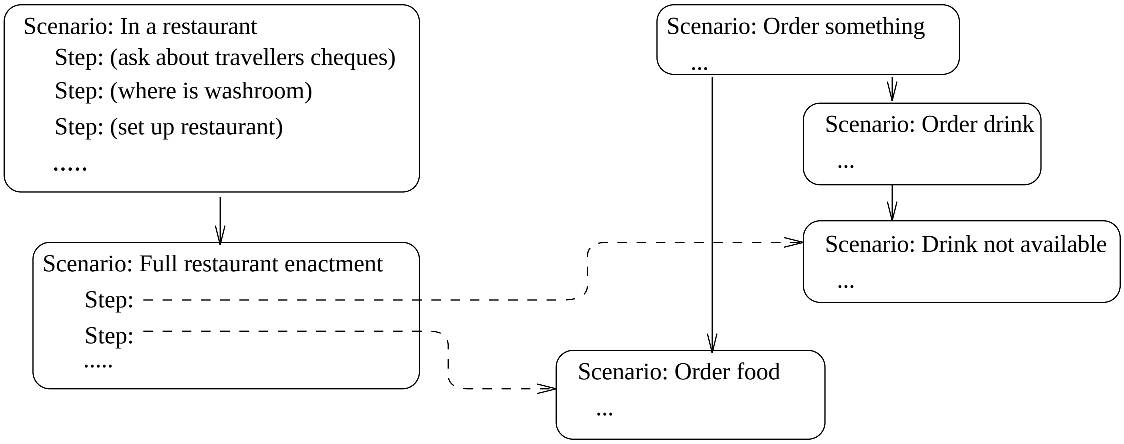

Figure 4: Hierarchical organization of scenarios. A dotted line indicates a relation along the aggregation dimension; a solid line, the abstraction dimension. Note that steps are like subgoals in that there may or may not be a fixed order between them.

demonstrated, and finally there is a discussion some of the techniques used in parsing the student's input.

The high-level algorithm is centred around the scenario. The scenario checks the enabling conditions on its steps and activates the most applicable step, if more than one step applies. Steps have priorities that help determine which is most applicable. The high-level scenario algorithm exits when there are no appropriate steps left.

Activating an action step is trivial, since the action is a chunk of code. Protocol steps are similarly simple, since they involve the waiter saying something in response to the student. Scenario steps get expanded and a high-level algorithm is started centred around that scenario. Slot steps are not so simple. They require a more complex algorithm, the slot-dialogue algorithm, explained in the next section.

3.1 Slot-dialogues

When the waiter is trying to elicit a value from the customer to fill a slot, it is not enough to ask a question and expect the student's reply to be the answer to the question. The student may volunteer the information needed to fill the slot before a question is asked; they may ask what kind of values are valid for a slot; they may check if a certain value they have in mind is valid; or they may provide an answer which is not valid, to name a few possible student responses. These different student responses are called response types in this paper. Flexibility is required in order to deal with different response types. The slot-dialogue algorithm should also be robust enough to be able to get the conversation back on track if a student response is not understood.

The proposed slot-dialogue algorithm looks something like this: first, check if there is input to parse. If there is not, then it is appropriate to generate a question about the slot to inform the student as to which slot is the current focus of attention. When the student's input is received it is sent to a parser. The parser picks out the main words and also attempts to classify the utterance into one of several response types, mentioned above. The waiter and student stay within the slot- dialogue until the slot has been filled or until the slot-dialogue is pre-empted by another dialogue.

A slot gets filled by creating an appropriate object or value to fill it. If the object has slots of its own that need discussion, the new slots get added to a stack or queue of slots pending further discussion, and a new slot-dialogue will be carried out for each one until the queue is empty. When the slot-dialogues are finished, the slot-dialogue algorithm exits to the main scenario algorithm, which can proceed to a new step.

Response types are not sentence types in the sense of imperative versus declarative versus interrogative. They are more closely related to the pragmatics of the situation. For the drink-order slot I have identified five major response types: *slot answer* ("Could I have a coffee, please?"), *list values* ("What kinds of soft drinks do you serve?"), *check validity* ("Do you serve coffee here?"), *explain item* ("What is coffee?"), and *confirmation*("Sure."). Each response type requires a different kind of response from the waiter, so there is a specialized routine for each. A description of the major response types is contained in the next few paragraphs.

When a slot answer has been recognized, the *slot answer* procedure is invoked. If it received an object, it then creates an instance of that class, checks to make sure the instance is properly detailed (it may require more information to be elicited), and declares that object to be the value of the slot. Suppose, for example, that the customer ordered coffee. An instance of coffee would be created. Coffee is not properly detailed: we need to know if the person wants cream or sugar with it. The *slot answer* procedure ensures that these questions will get asked by putting the slots on a queue and declares that coffee is the value for the *drink-order* slot. For some slots, a simple value may be sufficient and it may not be necessary to create an object. For example, the *sugar* slot for coffee may accept only "yes" or "no" values, which need not be full objects.

The *list values* procedure lists a set of possible values for the slot. For example, the student may have asked "I would like a hot drink of some kind, please.". The list of hot drinks served by the restaurant would be presented to the student. Sometimes the list of values to present to the customer must be limited. For example, if the student had said "What kind of drinks do you serve?" it is unrealistic for the waiter to enumerate them all, so the subclasses would be enumerated instead: "We have a variety of hot drinks, soft drinks, juices, beers, wines and other alcoholic beverages."

The *check validity* procedure is invoked when a specific object is asked about, if the *slot answer* hasn't been invoked. "Do you serve coffee here?" would invoke the *check validity* procedure.

The *explain item* procedure takes an item and asks the item to describe itself. All objects have a fixed paragraph to describe them. For example, if the student asked, "What is coffee?", a description of coffee would be presented to the user.

The *confirmation* procedure requires access to the previous utterance. If the confirmation is about a potential order, the waiter takes the value in the previous utterance and forwards it to the *slot answer* procedure, assuming it is a positive confirmation. If there is nothing in the previous utterance to fill a slot with, then the confirmation was most likely just at the syntactic level, and further student input should clarify what the *slot answer* should be. For example, if the waiter asked "Are you ready to order?" and the student replied "Yes, I'll have a coffee.", the "yes" is mainly at the syntactic level and the slot answer follows. It is more difficult to specify what to do when the confirmation is negative, but it relies on the wording of the previous utterance. For example, if the question was "Are you ready to order now?" then an appropriate reaction on the waiter's behalf is to come back in a bit and ask again. On the other hand, if the question was "Would you like to order something from our menu tonight?" then an appropriate reaction to a "No." would be to terminate the slot discussion with a nil value filled in.

3.2 Example: a dialogue with the system

What follows is a walkthrough of a conversation between the waiter and the student to give some idea of how the system works. The example is based on the dialogue from Figure 1, involving the *drink not available* scenario, as seen in Figure 3. This scenario gives the student valuable practice in handling a situation not normally detailed in standard phrase books.

The student will attempt to order a drink but that drink will not be available (either the drink is not sold by the restaurant or the restaurant is out of stock). *Drink not available* is a subclass of *drink order*, from which it inherits the step about the drink-order slot, and of *general scenario*, from which it inherits a protocol step that informs the waiter to say "okay" after a slot has been filled. It also has a slot of its own, the action slot that tells the waiter and the restaurant to ensure that the student's first drink order will not be available.

Starting in the *drink not available* scenario, the main scenario algorithm gets any steps that are enabled. The action step is the only one that triggers, so the action is performed. The restaurant gets set up to make sure the student can't have her first choice of drink.

The scenario gets the next step. Only the step involving the drink-order slot triggers. A slot-dialogue is initiated around this slot.

The slot-dialogue begins by checking if there is any input to read yet that could satisfy the drink-order slot's value position. There is not, so a question is generated about the slot by picking one of the four questions listed in the slot. The waiter says "Can I get you something to drink?".

The student's response is "Yes, do you had root beer?". The parser analyses the student's response, taking into consideration the current slot, drink-order. The results are as follows:

Results $\rightarrow$
  result
    type: confirmation
    relevantSlots: none
    value: yes
    relatedWords: none
    parseTree: (S-adv yes)
  result
    type: validityCheck
    relevantSlots: drink-order
    value: root-beer
    relatedWords: none
    parseTree: (y/n question (subject: waiter) (verb: have) (object: root-beer))

The first parse-result is a *confirmation* with a value of "yes". This is a linguistic response to the waiter's yes/no question There are more parse-results following, so we can go on to the second parse-result in the hope that the slot-answer is there.

The second parse-result is a check-validity response type. The person is not sure whether the thing they are ordering is available. The *check validity* procedure is invoked, which checks whether root-beer is valid. In this case, the restaurant has been set up so that root-beer is not valid for the drink-slot, i.e. the restaurant does not serve root-beer. The waiter responds with "No, sorry." and then goes to the *list values* procedure, which retrieves from the restaurant a list of values. Note that the choice of valid values to be presented should be scoped to include only those that are closely related to the slot value that is being checked. In our example, since root-beer is a soft drink, other available soft-drinks should be listed. The waiter outputs "We've got coke, sprite, ginger ale and orange.".

We go back to the slot-dialogue algorithm. The student has responded with "Maybe I have ginger ale then.", so there is input to parse and we do not need to generate a question. The parse returns the following results:

results $\rightarrow$
  result
    type: slotAnswer
    relevantSlots: drink-order
    value: ginger-ale
    relatedWords: none
    parseTree: (statement (subject: customer) (verb: have)(object: sprite))

The result is a slot answer. We go to the *slot answer* procedure to fill in the slot. First we check if the filler is appropriate to the current slot. We are still in the drink-order slot and ginger-ale is indeed an appropriate filler. We now create an object of type ginger-ale and insert it into the blank in the slot. Ginger-ale creates no new slots. The slot step is now complete.

We go up to the level of the (main scenario) algorithm again and get the next step. This time, a protocol step from a superscenario fires: the protocol that requires the waiter to acknowledge receipt of the order. The waiter says "Okay.".

Once again we update the list of candidate steps. This time it is empty, so we are done. The system exits out of the simulation to the system level and asks the student if they would like feedback.

3.3 The Parser

The parser is the other major component used in handling the communication between the waiter and the customer. The parser provides the input to the slot-dialogue algorithm that allows it to determine what to do next. The output of the parser contains information about which slot(s) is relevant to the student's input, what value might fill that slot if the student mentioned an appropriate value, and the response type of the input, along with a few other minor things.

The parser starts by looking up every word in the student's input in a dictionary. The dictionary contains the following information about each word: which slot or slots it is most likely to apply to (if the word is more likely to occur in conjunction with some slots than with others), the part of speech of the word (noun, verb, adjective...), and whether the word belongs to a fixed phrase or collocation that has a meaning independent of the word. If the word is a noun, then the dictionary tells what class of object it is, if it is a known object. This is needed so that type matching can be done on the slot's value. If the word is a verb, then there is information about the argument structure of the verb. This is useful in comparing against the role structure of the slot and helps to determine the type of utterance it is.

The difficult task for the parser is determining the response type of the student's input. In order to do this, the parser does some syntactic parsing to determine the mode of the sentence (imperative, declarative, etc) and makes use of which roles are present in the sentence and which are not. These two pieces of information along with the information gleaned from the dictionary seem to be enough to distinguish most utterance types. For example, declarative sentences are almost always slot-answers. Yes/No questions can be slot-answers ("Could I have a coffee?") or validity-checks ("Do you have coffee?") but if the actor = waiter role is present then it is more likely to be a validity-check, and if the benefactor = customer role is present it is more likely to be a slot-answer. (In this case if it is not clear it's better to assume it's a validity-check, since there is an assumption that the person is checking the validity of ordering the item because they are interested in ordering it.) Similarly, wh-questions point to either explain-item or list-values, but the class of the word that fills the value role of the slot can be used to distinguish these: if the value role is a leaf-class then it is probably an explain-item ("What is coffee?") whereas if the value role is a superclass then the type is probably list-values ("What do you have for soft drinks?"). Choosing the response type based on distinctions such as these may not be generalizable; however, I think there will be at least some re-use of response-type distinctions across the slots.

4 Future Work

The example in section 3.2 demonstrated the feasibility of the model on a small sample dialogue; however, the model should be verified over a larger range of examples to ensure generality. Though the basic model appears sound, necessary additions to and limitations of the model will become apparent as it is tested further. The model would benefit in particular from a careful examination of the sufficiency of the response types and a more powerful parser that can handle ellipsis and other linguistic phenomena that would confuse the parser as it is currently designed. The tutoring component of the system should also be expanded and explored.

5 Contributions

This paper presents a model for an immersion environment in a second language tutor. Using a robust parser for the waiter and a unique ontology of response types when dealing with slots, a student's faulty input can be understood and responded to appropriately. Separating the work of understanding the student's intent from the work of verifying her syntax provides a robustness within the immersion environment that would otherwise not be possible.

Immersion environments in second language tutors are rare. Whereas most tutors focus on syntax or vocabulary, the restaurant immersion environment provides an opportunity for students to practise their language skills in a goal-directed environment which emphasizes the communication aspect of the language first and the grammatical details second. The restaurant domain is a familiar and useful one to students who wish to learn a second language and travel in a foreign country.

References

[1] Bartorillo, C. & Bartorillo, A. *Language Teacher.* Distributed by Micro Tutor Products, 1819 Millstream Drive, Frederick, Md. 21702-3017, 1992.

[2] Cerri, S.A., Cheli, E. & McIntyre, A. Nobile: Object- Based User Model Acquisition for Second Language Learning. In *Intelligent Tutoring Systems for Foreign Language Learning*, Swartz, M.L. & Yazdani, M. (eds), Springer-Verlag, 1990, pp.171-190.

[3] Frumkes, L.A. Silent Movies: A Digitized Video Approach to the Russian Verb. In *Proceedings of Ed-Media 94*, Ottmann, T. and Tomek, I. (eds), AACE, Charlottesville VA, pp. 223-228.

[4] Greer, J.E. & McCalla, G.I. A Computational Framework for Granularity and its Application to Educational Diagnosis. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI-89)*, Detroit, 1989, pp. 477-482.

[5] Guevenir, H.A. Drill and Practise for Turkish Grammar. In *Intelligent Tutoring Systems for Foreign Language Learning*, M.L. Swartz & M. Yazdani (eds), Springer-Verlag, 1990, pp.275-291.

[6] Hamburger, H. & Hashim, R. Foreign Language Tutoring and Learning Environment. In *Intelligent Tutoring Systems for Foreign Language Learning*, Swartz, M.L. & Yazdani, M. (eds), Springer-Verlag, 1990, pp.201-218.

[7] Handke,J. WIZDOM: A Multiple-Purpose Language Tutoring System Based on AI Techniques. In *Intelligent Tutoring Systems for Foreign Language Learning*, Swartz, M.L. & Yazdani, M. (eds), Springer-Verlag, 1990, pp.293-305.

[8] Jacobs, P.S. & Rau, L.F. Innovations in Text Interpretation. In *Natural Language Processing*, Pereira,F.C.N and Grosz, B.J. (eds), MIT press, Cambridge, MA, 1994, pp.143-191.

[9] Heift, T. & McFetridge, P. The Intelligent Workbook. In *Proceedings of Ed-Media 94*, Ottman, T. and Tomek, I. (eds), AACE, Charlottesville, VA., pp.263-268.

[10] Richmond, I.M. Doing it Backwards: Using Translation Software to Teach Target-Language Grammaticality. In *Computer Assisted Language Learning, Vol.7, No.1 (1994)*, Swets & Zeitlinger, Lisse, the Netherlands, pp.65-78.

[11] Riesbeck, C.K. & Schank, R.C. *Inside Case-Based Reasoning.* Lawrence Erlbaum Associates, Hillsdale, NJ, 1989.

[12] Wenger, E. *Artificial Intelligence and Tutoring Systems.* Morgan Kaufman Publishers, Inc., Los Altos, Calfornia, 1987.

[13] Winston, P.H. *Artificial Intelligence.* Addison-Wesley, Don Mills, Ontario, 1992.