

Improving the Authoring of Foreign Language Interactive Lessons in the Tactical Language Training System

Joram Meron, Andre Valente, W. Lewis Johnson

Alelo, INC.

11965 Venice Blvd, Suite 402, Los Angeles, CA 90066

(jmeron,avalente,ljohnson)tacticallanguage.com

Abstract

The Tactical Language and Culture Training System (TLCTS) teaches foreign language and culture using a task-based approach. Four trainers have been developed so far, for Iraqi Arabic, Pashto, French and Levantine Arabic. The Tactical Iraqi system has been used to train thousands of users in the U.S. military. In this paper, we describe recent work we undertook to improve the effectiveness and efficiency of the process of authoring new content for our system. The first improvement is the introduction of *utterance templates*, which combine two previously separate components of the system (ASR grammar specification, and user input mapping), and improve system flexibility. Secondly, we consolidated the knowledge into centrally managed object libraries in order to enable scaling up to more complex and extensive lesson content. Finally, we created dedicated object editors to facilitate lesson authoring and simplify the production of authored content into a running system.

Index Terms: Computer assisted foreign language learning, authoring tools

1. Introduction

Development of a commercial grade computer assisted foreign language learning system is a process which is interdisciplinary not only in terms of the technologies involved (speech and language processing, animation, AI, software engineering), but to an even higher degree in terms of the kinds of experts (language experts, education experts, script writers, Artificial Intelligence (AI), Automated Speech Recognition (ASR) and Natural Language Processing (NLP) experts, graphic designers, animators, general programmers, sound engineers, actors, etc.) required in order to complete a product which will be pedagogically sound, aesthetically pleasing, technologically robust, interesting, emotionally engaging, and motivating. This necessitates a large and diverse development team (both in expertise areas and skill levels). Therefore, clear communication and knowledge management play a major role in the success and cost of product development.

One of the difficult problems in system development is the gap between two groups of people: the content developers (language experts, script writers) and the technical staff (programmers, animators, ASR, NLP and AI experts). The content developers author the lesson content, but are not able to write the code to execute it. Getting a working system typically requires the technical staff to translate the dialog written by the lesson authors to some form of computer usable code.

Among those “translation” tasks are: extracting recognition grammars from the authored materials; specifying pronunciations for words and utterances; programming the mapping from possible ASR outputs to abstract acts; managing non-player character behavior etc.

Due to the ubiquity of the Web as a teaching medium, and the emergence to content delivery standards such as SCORM [4], most current courseware authoring tools (e.g., Course Genie) [3] currently focus primarily on the definition of Web-based content. Other authoring tools such as REDEEM [2] are concerned with authoring content pages so that they can be selected and linked in different ways depending upon the needs of the learner. Our tools address problems in authoring language learning content and interactive dialog content in particular that are not addressed in these general-purpose authoring systems.

Two categories of authoring tools are most closely related to the work described here: interactive storytelling tools such as Crawford’s Erasmatron and related tools [5], and tutorial dialog development tools such as TuTalk [7] and AutoTutor [6].

Like the Erasmatron, our dialog management system organizes dialog around abstract actions, and allow non-programmers to create interactive dialogs. But whereas the Erasmatron is a proof-of-concept prototype, TLCTS dialog authoring tools are in continuous active use by non-programmers to create dialog content. Other authoring tools for interactive narrative authoring are being developed at the USC Institute for Creative Technologies and elsewhere, however they either assume more programming expertise to use or provide less support for dialog variability. Other dialog toolkits such as Trindikit also exist, but these are designed for use by programmers.

AutoTutor and TuTalk are designed to author tutorial dialog instead of character dialog for interactive scenarios. These tutorial dialogs focus on curriculum topics and elements of the problems being solved by the learner, while TLCTS authoring tools need to support the full range of conversations that TLCTS characters participate in.

At Tactical Language Training, we have undergone a major effort in the last year to build powerful editing tools to reduce this gap and allow content developers (who are not technical experts) to author content that can be produced as automatically as possible. The tools manage access to a library of high level components that the author can easily invoke, and it supports the clear and concise definition of all of the information required to define and execute the content.

2. System overview

In this section we briefly describe the Tactical Language and Culture Training System (TLCTS). For more details see [1]. TLCTS has three major modules: the Mission Skill Builder, the Arcade Game, and the Mission Game.

2.1. Mission Skill Builder

In the Mission Skill Builder (MSB), students learn vocabulary and grammar, by reading, hearing, and saying the words and sentences (with ASR feedback). The lessons are authored as pages to be displayed on the screen. There are several different types of pages and page formats. The page specifications are stored in XML files (including text in standard orthography and/or transcription formats) with links to audio files, animations etc.

Most pages in the MSB allow the user to hear foreign language speech and record his/her own utterances. The recorded speech is passed to a speech recognizer, and the ASR's output is displayed as part of the system's feedback. In this part of the system, the ASR is running with a closed ('flat') grammar of a few tens to a few hundred utterances, and possibly a garbage model to catch "other" words.

The grammars were primarily constructed by collecting all the words from a lesson (or a set of lessons). Since our acoustic models are trained using relatively small training databases (due to the very limited availability of off-the-shelf databases, and in some cases scarcity of available native speakers), we allow some words to be marked as difficult to recognize (e.g. very short words, or word pairs with very close pronunciations), resulting in the exclusion of these words from the recognition grammars.

2.2. Arcade Game

The Arcade Game (AG) allows learners to practice speaking and listening using targeted vocabulary in a video game where they must navigate through a maze by either listening to or speaking voice commands.

2.3. Mission Game

In the Mission Game (MG), the user gets a chance to practice the language and culture knowledge acquired in the Mission Skill Builder. The user is given a mission description (e.g. find the local leader), and then needs to speak with non-player (computer controlled) characters to achieve the mission, using the learned language and cultural skills learned in the MSB and Arcade parts.

3. The Variability Challenge

One of the key challenges in the Mission Game is to both increase and manage variability. On the one hand, we wish to be able to increase variability, meaning that the system should recognize, properly interpret and react to a large number of utterances (language subset). On the other hand, we wish to author the game in order to produce interactions that train specific language and culture skills, and as such we don't really want to accept any possible spoken input. Further, unbound language recognition and interpretation is a technical problem still beyond the current technology state of the art.

We have developed a number of solutions throughout the project to balance these competing needs. A key element of our solutions is to manage variability in two layers. In the speech layer, we constrain the speech to be recognized in such a way that we can limit the possible utterances and improve ASR performance. In the act layer, we manage the possible acts the user can perform, and the mapping from speech to acts. Considerations at this layer are the authoring

of meaningful responses from the non-player characters, the intended level of language and culture mastery of the learner, and the pedagogical goals of the interaction at hand.

For the speech layer, a basic mechanism is to control the speech recognizer through grammars. In the beginning, we defined these grammars based on lists of utterances that were generated based on the specifications of the dialogs (usually a number of scripts). While this way of building lessons is very simplistic and not very efficient (in terms of ASR specification and use, plot logic, and the effort needed for implementation), it was still a good first step, especially for beginner level language learning, where the range of utterances the user can say is very limited. To improve on that, we started to manually add variations of the basic utterances. This improves variability, but is time-intensive and error prone from an authoring perspective.

For the act layer, we originally created scripts and branched scripts, and annotated each utterance with an act name. We would then translate the text scripts into hard-coded programs. This translation included the logic for plot progression and character control, as well as the mapping from user input (the text output received from the ASR) to actions in the game world (in the form of a list of input-action pairs). However, we found out that scripts were too restrictive a medium in that they limited the act variability. We could (and did) improve the situation by manually programming a more complex logic directly in code, but that approach had many problems: it required a programmer to complete the scene (which was both expensive and non-scalable), and it was impossible to verify if the programmed logic was consistent with the intent of the authors.

4. Utterance Templates

For more advanced language learners, or more complex user-system interactions, this simplistic way of specifying the interaction does not scale gracefully. First,

the number of appropriate sentences and sentence variations increases exponentially, making it very hard to list all the alternatives manually. Second, the list of mappings from ASR output text to game actions also grows, making the semantic interpretation of speech much harder.

To solve these problems, we introduced the use of *utterance templates* in the interaction description. An utterance template is a combination of a more flexible grammar definition syntax (which solves the first problem), with additional syntactic constructs used to specify semantic content allowing parameterized utterances (which addresses the second problem).

At authoring time, authors attach an utterance template to each type of act the user can perform, specifying the subset of the language they expect to be used in verbally performing that act, and any parameters for the act.

4.1. Syntax

The grammar definition syntax is similar to the one used in HTK. The following example demonstrates the utterance template grammar.

```
$h_drink = tea | coffee;  
$c_drink = water | coke;  
$drink   = $h_drink | $c_drink;  
$food    = bread | apples;  
$order  = [please] get [...] ($drink|$food);
```

A non terminal (a symbol preceded by '\$') is defined as:

- A terminal (single word): (**tea**)
- An optional terminal / non terminal: [**a**]

- A sequence of terminals/non terminals: (a \$b c)
- A set of alternative terminals/non terminals: (a|b)

The special symbol ‘...’ is a place holder which can match any word (or several words).

4.2.Semantics

The next sample adds semantic objects used in the utterance templates.

```
$h_drink{#drink}{hot} = tea | coffee;
$c_drink{#drink}{cold} = water | coke;
$drink = $h_drink | $c_drink;
$food{#food} = bread | apples;
$order = [please{#polite+=1}] get [...]
($drink|$food);
```

Two types of semantic objects are used:

- Features: (e.g. {hot}) are used to indicate that a specific attribute is present (or absent)
- Variables (e.g. {#type:liquid}) are used to indicate specific values for attributes.

Incremental variable assignment (e.g. {#polite+=1}) and several shorthand notations are seen in the example, where instead of re-defining features and variables for each right hand alternative, we can define them just once on the left hand side, with the same meaning. For example, the first line could have equally been declared as:

```
$h_drink = tea{#drink:tea} {hot} |
coffee{#drink:coffee} {hot};
```

4.3.Knowledge Encapsulation

The main advantage of the utterance template is that it avoids having to declare the same information in two separate places: first for specifying and building the ASR grammar, and then again for interpreting the ASR output.

This is an example of knowledge encapsulation, which is a main theme for the improvements to our authoring tools. In this case, knowledge of form (sentence grammar) and meaning (features and variables) are encapsulated into one object, which can be easily passed between system modules without a need to pass additional contextual information: once the author composes the utterance template, it can be passed first to the ASR module (where the ASR grammar can be automatically extracted from it), and then to the input analysis module (where it is used to automatically parse the text output of the ASR).

The parser we use is a finite state transducer (automatically generated from the utterance template), which, for each input text sequence, outputs all of the possible paths through the transducer. Each output path includes the input text as well as the additional features and variables associated with that path's edges.

The meaning of the input (i.e. the effect it has on the state and action of the game world) is derived exclusively from the features and variables associated with the parse path (discarding the original text input). For example, for the input “please get tea” we will get the parse:

“please {#polite:1} get tea {#drink:tea} {hot}”.

This is then stripped down to:

```
{#polite:1} {#drink:tea} {hot}
```

which, by definition, includes all the information needed to process the input at this context (the utterance template author intentionally constructs it this way).

The interpretation of the meaning contained in the semantic objects is designed to be independent of object order, which greatly simplifies the mapping process. The user input is mapped into game acts using the following information:

- Did a specific feature appear?
- Was a specific variable defined?
- What values was assigned to a specific variable?

4.4.Transcriptions

An additional element which is linked to the utterance templates is a set of transcriptions / transliterations which are associated with the terminal symbols in the utterance template grammar (words). Each word is (potentially) associated with:

- Orthography: the way it's written in its native script
- Transliteration: a textual representation intended to help visualize the pronunciation, by using letter combinations the student is familiar with
- Phonetic transcription: phoneme sequence (or sequences) to be used by the ASR for recognition.

The Tactical Language Training System is flexible enough to use different speech recognition engines (e.g. Julius, HTK, Microsoft). For each recognizer, we adapt the conversion program to convert utterance templates to the required grammar definition language.

5.Object Libraries

As mentioned above, the efficient management of information is very important in the development of this kind of language learning system. An important aspect of this is the accumulation of knowledge objects in centralized libraries, which are easier to manage, and allow object re-use. To manage and use these libraries, we are developing dedicated editors, which can create and modify different type of objects, and allow linking them into larger objects (e.g. words into utterance, utterance template into speech act, etc.).

5.1.Utterance library

The utterance library includes objects which describe either full sentences or sentence fragments, and which can be spoken either by the learner to the system, or by the system to the learner.

Each utterance is described by means of its native orthography and/or a transliteration and a phonetic transcription. Since some words can have more than one pronunciation, the authors select the pronunciation appropriate in the context of this utterance. In case there is more than one appropriate pronunciation, they select one as the standard (default), and mark the others as optional. The utterance also points to a set of actual audio recordings.

When authoring a lesson page, the author can search in the utterance library for the utterance object s/he is looking for, reference it in the lesson page. This reference is a new instance of the utterance, which is initialized with the default pronunciation and audio example taken from the utterance library. The author can change the defaults as needed.



Suggestions for pronunciations (in the form of phonetic transcriptions) are obtained from two alternative sources:

First, a pronunciation dictionary (possibly having multiple pronunciations per word) is consulted. Second, if a word is not in the dictionary, a rewrite-rule-based converter generates a set of alternative transcriptions.

The author can select one of the suggested alternatives, or manually edit the phonetic transcription. When a speech synthesizer (with the ability to control the exact phoneme sequence, and a compatible phoneme set) is available for the language being learned, it can be used to aid the authors, who are not always native speakers of the language). Above we show an example of part of the transcription authoring screen.

5.2. Utterance Template Library

Utterance templates are created and collected into an utterance template library, in a way very similar to the simple utterances, except that their format is a little different. Utterance templates can be created by composing previously authored utterance templates from the library. They are entered and managed in the context of the valid user game acts in the system.

6. Interactive Dialog Management

In the current system, dialogs (and plot progression) are managed by finite state machines, and are defined using flow graphs, which describe the flow of the plot, as well as the associated logic, and all the possible inputs and outputs.

For each flow graph, there is a start node, and one or more end nodes, which correspond to different dialog outcomes (e.g. successful/failed mission), and are typically associated with mission debriefing/feedback/evaluation.

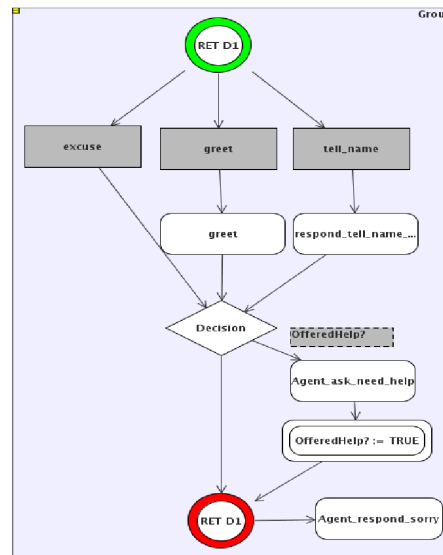
The user's speech (which is mapped into a specific speech act, using the input mapping described above), as well as non-verbal input (such as body gestures, which the user can select), determine the branch which will be taken from the current node to the next. Progressing to a subsequent node can also result in setting or modifying of a state variable, which can also be used in the logic of the state machine. For example: if the learner introduces themselves politely and uses polite body language, a variable for politeness gets a high value, which can influence the willingness of some characters to cooperate, at some later point in the dialog.

6.1. The TIDE Dialog Editor

We have built a graphical dialog editor (called TIDE: *Tactical Interactive Dialog Editor*), which simplifies the authoring of the dialogs. Authors create graphical descriptions of the dialog – available types of acts, user inputs (verbal and non-verbal), game decisions, non-player outputs (verbal and non-verbal), etc. User inputs are specified in terms of valid acts the user can perform, which in turn are specified using utterance template. Authors are also able to annotate the dialog elements with links to objects related to production, such as the specific audio file and/or animation to play for an output. This simplifies the scene production work.

The figure below shows a screen shot of a sample from the dialog editor. This sample shows the definition of one modular dialog fragment (a whole dialog is too large to show here). The figure shows examples for user input, character response, branching according to variable values etc. When lesson authoring is complete, the editor automatically generates the script code that controls the scene in the TLCTS

system, as well as a reusable XML file for other types of manipulation.



7. Conclusion and further work

We are continuing to improve our lesson authoring tools to make TLCTS easier, cheaper and faster to author, and to produce an engaging, pedagogically sound game with as much input variability as possible.

We have developed TLCTS systems for Levantine Arabic, Iraqi Arabic, Pashto, and sub-Saharan French, both for military and civilian users. The authoring tools have proved themselves to be commercial grade tools, able to handle realistically complex lesson material.

Further work is under way on improving the editors' features and user interface, language reference tools, and ASR, and expanding the scope of the system beyond language training applications.

8. References

- [1] W.L. Johnson et al., "Tactical Language Training System: Supporting the Rapid acquisition of Foreign Language and Culture", InSTIL/ICALL 2004
- [2] Evaluating the REDEEM Authoring Tool: Can Teachers Create Effective Learning Environments? International Journal of Artificial Intelligence in Education (2004), 14, 279-312.
- [3] Course Genie: Author courses with Word. http://www.wimba.com/docs/course_genie_brochure.pdf
- [4] SCORM 2004 Documentation suite. <http://www.adlnet.org>
- [5] Chris Crawford on Interactive Storytelling. New Riders Games, 2004.
- [6] Susarla, S. et al. Development of a lesson authoring tool for AutoTutor. In AIED2003 Supplemental Proceedings (pp. 378-387). Sydney, Australia.
- [7] Jordan, P.W. et al. (2007). Tools for authoring a dialogue agent that participates in learning studies. In Artificial Intelligence in Education, 43-50. _