Build Your Own Speech-Enabled Online CALL Course, No Experience Required

Rayner, Emmanuel; Baur, Claudia; Bouillon, Pierrette; Chua, Cathy; Tsourakis, Nikolaos

BUILD YOUR OWN SPEECH-ENABLED ONLINE CALL COURSE, NO EXPERIECE REQUIRED

Manny Rayner[1], Claudia Baur[1], Pierrette Bouillon[1]
Cathy Chua[2], Nikos Tsourakis[1]

[1]Geneva University (SWITZERLAND)
[2]Swinburne University of Technology (AUSTRALIA)

Abstract

We describe Open CALL-SLT, a framework that allows people who may only have modest computer skills to create interactive speech-enabled CALL courses on the web. User content is uploaded to a server and remotely compiled and deployed. Courses can be created at six levels of increasing sophistication, where the specification of the content ranges from simple prompt/response pairs at one end to multimedia-enabled gamified dialogues with multiple paths at the other. We briefly describe the overall framework and the different levels.

Keywords: CALL, speech recognition, web, open architectures.

1 INTRODUCTION AND MOTIVATION

As the success of RosettaStone, Duolingo and similar enterprises attests, Computer Assisted Language Learning (CALL) has become a big business. Many people want to learn a foreign language and hope that practising with a computer might improve their skills; in particular, it seems plausible that talking to some kind of speech recogniser could be helpful. But most language teaching still takes place in the classroom, and language teachers have on the whole been reluctant to embrace this technology. The content offered seems poorly integrated with school curricula, and speech recognition is viewed with particular suspicion. Not unjustly; for example, a little experimentation with Duolingo reveals that the recogniser is so forgiving that it will consistently accept responses that are quite incorrect. It is possible that the speech recognition functionality still has value in terms of boosting the student's confidence and encouraging them to talk more, but teachers tend to be sceptical about this kind of argument.

CALL-SLT [1,2], a project which has been pursued at Geneva University since 2009, aims to address these issues by creating a framework in which it is easy to develop speech-enabled CALL resources useful in real classroom contexts. One aspect of this has involved developing a speech recognition infrastructure that delivers an appropriate level of strictness. Another, on which we will focus in this paper, is to simplify the process of developing new material.

The basic functionality delivered by CALL-SLT is speech-enabled prompt/response practice on the web. The student logs in to a URL using a normal browser and accesses the course, which consists of a number of lesson modules. Each lesson contains a set of steps. At each step, the system prompts the student, using text, multimedia, or a combination of the two; the student responds, and speech recognition is used to decide whether to accept or reject their answer. In simple versions, the prompts are independent of each other; typically, the student is given a piece of text in their own language (the L1) and have to paraphrase it in the language they are learning (the L2). In more sophisticated versions, the prompts form part of a coherent dialogue for a task like booking a hotel room or ordering in a restaurant.

Previous papers describe how we have developed CALL-SLT content for beginner/low-intermediate courses in French at Bologna University [3,4] and English at several schools in German-speaking Switzerland [5,6]. In both cases, the content was developed by system experts at the University of Geneva, working in consultation with teachers at the schools in question. This process highlighted the importance of rapid content development. It is important not to waste teachers' valuable time, and it is essential to be able to create an adequate amount of content quickly. As a result of these experiences, we performed a complete reimplementation of the system, replacing the original architecture with a much simpler one [7].

In this paper, we describe how we have taken the process a step further and opened up the system infrastructure so that external users can create and deploy their own courses. Since we expect that

potential users will vary widely in their level of expertise, we have divided the functionality into six increasingly sophisticated levels. It is not necessary to master all the levels to be able to write a course; instead, the intention is that users will learn as many of them as they feel comfortable with, and ignore more advanced functionality. Experience shows that users can in fact construct sensible courses that do something useful even if they only use the first level.

The levels are as follows:

1. Basic prompts and responses
2. Multimedia
3. Advanced prompts and responses
4. Basic scripts and multimedia
5. Gamification
6. Advanced scripts

In sections 2 to 7, we outline the functionality added at each level and give examples of how it can be used. Section 8 briefly sketches the remote deployment architecture. Section 9 concludes and gives practical details for people interested in using the system.

2 LEVEL 1: BASIC PROMPTS AND RESPONSES

Level 1 allows the user to construct the most basic type of content: simple prompt/response pairs. When a course of this kind is run, each turn consists of the system displaying a text prompt, and the student replying. The speech recognizer, which knows from the “response” part of the pair which responses should count as correct, either accepts or rejects. Prompt/response pairs are grouped into larger units called “lessons”.

To our surprise, we have found that implementers with a little ingenuity find it quite possible to construct interesting and useful courses even with this very limited framework. [8] describes a pronunciation game designed for French-speaking students of English, where the prompt/response units are grouped together in contrasting pairs linked by pairs of words which illustrate a sound difficult for French students. For example, “hate” is contrasted with “ate” (the French “h” is silent), and “think” is contrasted with “sink” (French has no “th” sound). We illustrate Level 1 using this course.

Nearly all the course definition consists of Prompt units which define the prompt/response pairs. A Prompt contains a piece of L1 text (the prompt) and a piece of L2 text (the response); there is also a line marked Group, which holds a number that allows the course designer to specify the order in which the prompts are displayed. A typical unit, for the “hate” sentence, looks like this:

```
Prompt
Lesson      pronunciation_h
Group       4
Text/french Je déteste les légumes
Response    i hate vegetables
Response    * i ate vegetables
EndPrompt
```

Thus, when the unit is used, the system shows the student the French text *Je déteste les légumes* and they are supposed to respond with the spoken English sentence "I hate vegetables". The second Response line uses the asterisk (*) to explicitly mark an *incorrect* response. This is not obligatory; but in cases like these, where a specific incorrect response is expected, it improves recognition accuracy.

Apart from the Prompt definitions, the pronunciation course contains four Lesson units; the Lessons each collect together a group of Prompts linked by a common theme, the sound that is being practised. Each Prompt is tagged with the Lesson it belongs to; in the example above, the lesson is `pronunciation_h`. The lesson declaration for `pronunciation_h` is

```
Lesson
Name           pronunciation_h
PrintName      Practise H
EndLesson
```

where the line `PrintName` is used to specify the text that identifies the lesson in the system menu. Finally, the Course unit declares the course itself:

```
Course
Name            pronunciation
Languages       french
EndCourse
```

Here, the two lines specify the name of the course (`pronunciation`) and the single student language supported (`french`).

The above gives a complete description of the pronunciation course; the thing worth noting is that the interest of the course is determined much more by the well-chosen content than by the extreme simplicity of the structure.

3 LEVEL 2: MULTIMEDIA

At Level 1, the system prompts the student using a piece of text. In Level 2, this is augmented by allowing the prompt also to include a recorded multimedia (video or audio) file. The course designer records the files, uploads them to the course's multimedia directory, and then adds a line to each prompt to say which file it is associated with. When the course is accessed, the prompt is given by playing the multimedia file in one pane of the browser and showing the text in the other.

A simple example is an English-only game where the student has to name the thing or action shown in the multimedia file, following an English-language text instruction. A typical prompt is:

```
Prompt
Lesson          actions
Multimedia      jump.wmv
Group           3
Text/english    What is the man doing?
Response        jumping
Response        he is jumping
EndPrompt
```

where `jump.wmv` is a video file showing a man jumping and a woman walking. Note that we again include two responses. In general, a prompt can specify any number of possible responses.

When we get to Level 4, we will see that multimedia is particularly useful for lessons which define a dialogue.

4 LEVEL 3: ADVANCED PROMPTS AND RESPONSES

As Level 1 or 2 courses get larger, the designer usually notices after a while that they are typing a great many lines that look more or less the same. Prompts may contain many similar responses to a prompt; the problem is that different variants multiply out, for example different ways of asking for something (“I would like”/”Could I have”/”Do you have”) and politeness (“please”/nothing). It is also normal to find that there are many similar prompt/response pairs; a prompt which tells the student to ask for a coffee is probably going to be very similar to one which tells them to ask for a tea. Level 3 introduces three methods for addressing these issues: a simple version of regular expressions, templates, and a simple version of context-free grammar rules. We’ll look at these in order.

Regular expressions allow the course designer to reduce the number of Response lines. For example, the designer can simplify the two lines

```
Response i would like a coffee
Response i would like a coffee please
```

to the single line

```
Response i would like a coffee ?please
```

where the optional part, `please`, is tagged with a question-mark. Any phrase can be marked as optional by enclosing it in parentheses; so for example

```
Response i would like a ?(cup of) coffee ?please
```

is a short way to write the four responses

```
Response i would like a coffee
Response i would like a coffee please
Response i would like a cup of coffee
Response i would like a cup of coffee please
```

Similarly, the three lines

```
Response could i have a single room
Response do you have a single room
Response i need a single room
```

can be written as the single line

```
Response ( could i have | do you have | i need ) a single room
```

where the alternatives are enclosed inside parentheses and separated by vertical bars.

If the course contains several similar prompt/response pairs, e.g.

```
Prompt
Lesson         restaurant
Group          drink
Text/french    Dis : tu voudrais un café
Response       ( i would like | could i have ) a coffee
EndPrompt

Prompt
Lesson         restaurant
Group          drink
Text/french    Dis : tu voudrais un thé
Response       ( i would like | could i have ) a tea
EndPrompt

Prompt
Lesson         restaurant
Group          drink
Text/french    Dis : tu voudrais un coca
Response       ( i would like | could i have ) a coke
EndPrompt
```

a "template" allows the desiger to write the part they share once and then reuse it:

```
PromptTemplate ask_for_drink FRENCH ENGLISH
Lesson         restaurant
Group          drink
Text/french    Dis : tu voudrais FRENCH
Response       ( i would like | could i have ) ENGLISH
EndPromptTemplate

ApplyTemplate ask_for_drink "un café" "a coffee"
ApplyTemplate ask_for_drink "un thé" "a tea"
ApplyTemplate ask_for_drink "un coca" "a coke"
```

Systematic use of regular expressions and templates reduces the size of the course definition a great deal, typically shortening it by a factor of ten or even more for large courses.

“Phrases” (technically, context-free grammar rules) make it possible to write large sets of alternatives in a compact way. For example, prompts in conversational lessons, like the ones immediately above, often involve using request phrases. Rather than constantly repeating a regular expression like

```
( could i have | could you give me | i would like | do you have )
```

it is possible to define a “phrase”, as follows:

```
Phrase
PhraseId $ask-for
Response could i have
Response could you give me
Response i would like
Response do you have
EndPhrase
```

After this, the expression `$ask-for` can be used to mean “any of the alternatives defined in the Phrase `$ask-for`”.

Phrases are particularly useful for defining incorrect responses; as we saw in the pronunciation course from Level 1, it can often be a good idea to let the system recognise incorrect replies and alert the student to the fact that they are doing something wrong. Since there are, in general, many more ways to answer incorrectly than correctly, it can be necessary to define quite large sets of possible responses.

The Coverage unit allows the course designer to specify a set of phrases which the system will be able to recognise; the convention is that all of them will be treated as incorrect responses to every prompt, unless they happen to duplicate responses explicitly marked as correct. So, for example in a lesson about dates, we might have the following Phrase and Coverage units:

```
Phrase
PhraseId $month
Response january
Response february
…
Response december
EndPhrase

Phrase
PhraseId $day-number
Response first
Response second
…
Response thirty-first
EndPhrase

Coverage
Response $month $day-number
Response the $day-number of $month
EndCoverage
```

Here, the Coverage unit defines all phrases like “March fifteenth” or “the twenty-second of April”.

5 LEVEL 4: BASIC SCRIPTS

Up to Level 3, there are only very limited options for controlling the order in which prompts are presented. They can be collected into lessons, so that the user can select a set of similar prompts from a menu, or they can be assigned a number in the Group line, which means that the low-numbered prompts are presented before the high-numbered ones inside each lesson. If no numbers are assigned, the order of presentation is random.

This is enough for simple examples like the pronunciation course, but works less well when the course is supposed to teach the student how to carry out some kind of goal-directed conversation. For example, if the theme of the lesson is “restaurant language”, it is more natural to structure it as a conversation between the student and the waiter, where the student starts by ordering a drink and ends by asking for the bill. The course designer needs a way to break up the lesson into a series of exchanges which take place in a logical sequence that will not always be the same, and which to some extent is determined by what the student does.

Level 4 adds a mechanism for doing this; a lesson can include a "script", which defines the lesson as a set of "steps". As before, each step picks a Prompt from a specified group, presents it, and processes the student’s spoken response. What happens next now depends on whether the student manages to answer correctly or not, and possibly on other things (this is described in Level 6). For the moment,

there are three choices. If the speech recognizer accepted the student's response, the conversation proceeds to the next step, as defined by the script. If it rejected the student's response, it repeats the step, possibly with a different multimedia file. If the student is rejected too many times, the system gives up and offers them something easier.

Scripts are written in XML, which is a little more complicated than the plain text files used to specify Prompts, but still not very complicated: XML is more or less like HTML, with which many nontechnical people now have some familiarity. A typical step, from a hotel booking lesson, looks like this:

```
<step>
   <id>ask_for_number_of_nights</id>
   <group>room_for_number_of_nights</group>
   <next_success>ask_how_to_pay</next_success>
   <next_limit>is_one_night_okay</next_limit>
</step>
```

The meanings of the individual lines are as follows:

1. The name of the step is `ask_for_number_of_nights`.
2. The prompt will be from a group called `room_for_number_of_nights`. These prompts will tell the student to say things like "I want a room for three nights".
3. If the student's response is accepted, the next step will be `ask_how_to_pay`. In this step, the student will be given a prompt to get a response like "Can I pay by Visa?"
4. If the student is rejected too many times, the system will move to the step `is_one_night_okay`. Here, the system will say something like "I don't understand, but is one night okay?", and the student will be told to say "Yes".

The system's side of the conversation is supplied by multimedia prompts. So here, for example, we might have the Prompt

```
Prompt
Lesson          hotel
Multimedia      ask_how_many_nights
Group           room_for_number_of_nights
Text/french     Dis : tu voudrais une chambre pour 3 nuits
Response        ( i would like | could i have ) a room for three nights
EndPrompt
```

The Multimedia line is slightly different from the ones shown earlier; it does not contain the name of an actual multimedia file, but rather a "multimedia unit" which contains several files. Since the step may be repeated more than once if the student is rejected, the different files in the multimedia unit are played in order. Here, the multimedia unit is

```
Multimedia
Id ask_how_many_nights
File ask_how_many_nights1.flv
File ask_how_many_nights2.flv
EndMultimedia
```

`ask_how_many_nights1.flv` is a video file showing an animated desk clerk saying "How many nights do you wish to stay?" while `ask_how_many_nights2.flv` is a file with the same clerk saying "I'm sorry, how many nights was that"?

A lesson which uses a script defines it in the Lesson unit, e.g.

```
Lesson
Name            hotel
PrintName       Hotel booking
Description     Simple hotel booking dialogue
Script          hotel_booking_script
EndLesson
```

6 LEVEL 5: GAMIFICATION

Gamification [9] is a technique that has become very popular in CALL. Making the course more like a video game motivates many students; even if the methods may seem unsophisticated, they often turn out to work surprisingly well.

CALL-SLT provides tools to enable simple gamification of courses using scores and badges. The student starts each lesson with a specified score. They lose points every time they get a response rejected or have to skip a step because they have run out of tries, but gain points if they use phrases the course designer wishes to encourage. If they have enough points left at the end of the lesson, they get a credit towards their next badge; there are four badge levels, starting at "plain" and then moving through "bronze", "silver" and "gold". A tab in the browser's display shows the badges the student has collected during the current course.

To include gamification, the course designer needs to add the following declarations:

1. A line in each Lesson which has badges saying what badge pictures to use, for example:

   ```
   Lesson
   Name           hotel
   PrintName      Hotel booking
   Description    Simple hotel booking dialogue
   Badges         default
   Script         hotel_booking_script
   EndLesson
   ```

   The `default` value in the Badges line means that the badges displayed will be the standard ones. Alternatively, course designer can draw their own badges or download clip-art, and then declare the customized badges. This may seem like wasted effort; but gamification, as noted, is a powerful motivator, and students seem to be more motivated by pretty badges.

2. A declaration in the Course unit to specify how many times the student is allowed to attempt each step, what the starting value is for the score (by default, 100), what the penalties are for being rejected or for missing a step completely, and how many points are needed for each of the different badges. The following illustrates typical values:

   ```
   Course
   Name           english_course
   Languages      french german
   MaxTries       2
   StartScore     100
   RejectPenalty  2
   SkipPenalty    5
   PlainBadge     0
   BronzeBadge    0
   SilverBadge    90
   GoldBadge      100
   EndCourse
   ```

3. Definitions for bonus phrases. A typical `PhraseScore` declaration looks like this:

   ```
   PhraseScore
   Phrase would like to
   Score 2
   EndPhraseScore
   ```

   and says that the student will acquire 2 bonus points for using any response including the phrase "would like to".

7 LEVEL 6: ADVANCED SCRIPTS

7.1 Tags

The way a script works is that it makes a random choice of prompt at each step out of the ones available in the step's group. This is certainly a good idea; it is not helpful for the student to practise the same dialogue every time, since they will only get the same small number of vocabulary items over and over again. But if the script is at all complicated, it can often happen that two steps are related. Perhaps the theme is shopping, and the student is told to say they want to buy a particular article in one step, and then say in a later step that the article is the wrong size. If the student has been told to say they want to buy a pair of jeans in the first step, it makes no sense for them to be told in the second step that they should complain about the shirt.

This kind of problem can be solved by attaching "tags" to the Prompt units. The point of a tag is that it gives the system some basic knowledge of what the Prompt unit is about, so that it can keep the dialogue consistent. For the example above, tags can be used as follows. In the first step, where the student is told to ask for the article, the prompts for "jeans" and "shirt" are as follows:

```
Prompt
Lesson          shopping
Group           ask_for_article
Text/french     Dis : tu voudrais un jeans
Tags            garment=jeans
Response        i would like to buy a pair of jeans
EndPrompt

Prompt
Lesson          shopping
Group           ask_for_article
Text/french     Dis : tu voudrais une chemise
Tags            garment=shirt
Response        i would like to buy a shirt
EndPrompt
```

Here, the value of the tag `garment` is `jeans` in the first prompt, and `shirt` in the second. (In practice, one would probably write the prompts using a template, but we present them explicitly to make it clearer what's going on). Later, in the second step, the prompts are:

```
Prompt
Lesson          shopping
Group           too_expensive
Text/french     Dis : le jeans est trop cher
Tags            garment=jeans
Response        the jeans are too expensive
EndPrompt

Prompt
Lesson          shopping
Group           too_expensive
Text/french     Dis : la chemise est trop chère
Tags            garment=shirt
Response        the shirt is too expensive
EndPrompt
```

At the first step, the system picks a prompt at random. But when it reaches the second step, it has to keep the tags consistent, so it will choose the prompt with the same value for the tag `garment`.

The tags also make it possible to choose different multimedia files depending on the prompt. So the prompts for the second step could be

```
Prompt
Lesson          shopping
Group           too_expensive
Multimedia      jeans_question
Text/french     Dis : le jeans est trop cher
Tags            garment=jeans
Response        the jeans are too expensive
EndPrompt

Prompt
Lesson          shopping
Group           too_expensive
Multimedia      shirt_question
Text/french     Dis : la chemise est trop chère
Tags            garment=shirt
Response        the shirt is too expensive
EndPrompt
```

Here, the multimedia resource `jeans_question` contains files with words like "Are the jeans okay?" while the multimedia resource `shirt_question` has files with words like "Are you happy with the shirt?"

7.2 Conditional steps

Another technique for making dialogues more interesting is to include extra paths. This can be done by adding “conditional steps”, where the <next_success> tag in the step includes a condition that has to be satisfied. The step will then have several different <next_success> tags, and the first one whose condition matches will be chosen.

The simplest type of condition is a random choice. For example, in this step from a hotel lesson,

```
<step>
   <id>ask_type_of_room</id>
   <multimedia>what_kind_of_room</multimedia>
   <group>type_of_room</group>
   <next_limit>is_single_okay</next_limit>
   <next_success probability="50">here_is_the_price</next_success>
   <next_success>not_available</next_success>
</step>
```

the student is asked what kind of room they want, and are told to say something like "I want a double room" or "I want a suite". If they are accepted, there will be a 50% chance of moving to the step `here_is_the_price`, where they are told how much it costs, and a 50% chance of moving to the step `not_available`, where they are told that a room of the kind they want is unavailable and offered an alternative.

If the course includes gamification, it is also possible to make the choice of next step conditional on the student's current badge level, so that they can be offered new possibilities when they reach a higher level. For example, near the end of the hotel script one could have a step like

```
<step>
   <id>enjoy_your_stay</id>
   <multimedia>enjoy_your_stay</multimedia>
   <group>thanks</group>
   <next_limit>exit</next_limit>
   <next_success cond="level >= silver">is_everything_okay</next_success>
   <next_success>exit</next_success>
</step>
```

Here, the multimedia file says something like "I hope you will enjoy your stay with us", and the student is prompted to say something like "Thank you". The first `<next_success>` line says that, if the badge level is at least silver, the dialogue will transition to a step where the desk clerk asks if everything is okay and the student is prompted to complain about something. The second `<next_success>` line says that the dialogue will otherwise exit the lesson.

8 UPLOADING, COMPILING AND DEPLOYING

The infrastructure for remote compilation and deployment of courses is at an early stage of development, and so far only offers the following basic functionality. Every user is assigned a directory on the server, which they can access by password-protected FTP. The user needs to create a new subdirectory for each course they write. This subdirectory must contain three second-level subdirectories, called `grammars`, `scripts` and `multimedia` respectively. The `grammars` subdirectory will usually contain only one file, the grammar for the course. The `scripts` subdirectory will contain one file for each script, if the course uses scripts. The `multimedia` subdirectory will contain any multimedia files that may be used, including badge pictures.

The user starts by writing a first version of the grammar file for the course and uploading it to the `grammars` subdirectory, after which they can register the course on the server. After uploading and registering the course and uploading any other files that may be required, the user can compile it. Compilation typically takes about half a minute to run, and returns an indication of whether compilation was successful together with a list of trace output. When the course has been successfully compiled, the user can deploy it on the server. This normally takes about a minute, during which time the server is unavailable.

9 IF YOU WANT TO GET INVOLVED…

This paper has presented a description of the new version of CALL-SLT, which has been purposely designed so that it can be used by people who would like to build interactive speech-enabled courseware that can be used on the web, but have only modest computer skills. The reason why we have been unusually explicit about the details is to show that there isn't, in fact, a great deal to learn. What we've described is pretty much it.

If you think this is interesting and would like to try to create and deploy your own courses, the Open CALL-SLT framework is currently in alpha testing. Please write to Emmanuel.Rayner@unige.ch, Claudia.Baur@unige.ch or Nikolaos.Tsourakis@unige.ch with a brief description of the kind of course you are thinking of developing, the languages involved, and a little background on yourself.

REFERENCES

[1] Rayner, M., P. Bouillon, N. Tsourakis, J. Gerlach, M. Georgescul, Y. Nakao, and C. Baur. (2010). A multilingual CALL game based on speech translation. In Proceedings of LREC 2010, Valletta, Malta.

[2] Rayner, M., Tsourakis, N., Baur, C., Bouillon, P., & Gerlach, J. (2014). CALL-SLT: A spoken CALL system based on grammar and speech recognition. Linguistic Issues in Language Technology, 10(2)

[3] Bouillon, P., Cervini, C., Mandich, A., Rayner, E., & Tsourakis, N. (2011). Speech recognition for online language learning: Connecting CALL-SLT and DALIA. Proceedings of the international conference on ICT for language learning

[4] Cervini, C., Bouillon, P., & Gasser, R. (2013). Jeu de traduction orale en ligne et apprentissage des langues. Les Langues Modernes, (4), 83-94.

[5] Baur, C., Rayner, E., & Tsourakis, N. (2013). A textbook-based serious game for practising spoken language. Proceedings of the 6th international conference of education, research and innovation (ICERI)

[6] Baur, C., Rayner, E., & Tsourakis, N. (2014). Using a serious game to collect a child learner speech corpus. Ninth international conference on language resources and evaluation (LREC)

[7] Rayner, M., Baur, C., & Tsourakis, N. (2014). CALL-SLT lite: A minimal framework for building interactive speech-enabled CALL applications. Workshop on child computer interaction (WOCCI)

[8] Jolidon, A. (2013). Reconnaissance vocale et amélioration de la prononciation : élaboration et évaluation de leçons avec le logiciel CALL-SLT. Masters Thesis, University of Geneva.

[9] Werbach, K. and D. Hunter. (2012). For the Win: How Game Thinking Can Revolutionize Your Business. Wharton Digital Press