Helping Non-Expert Users Develop Online Spoken CALL Courses

RAYNER, Emmanuel, et al.

Abstract
We introduce Open CALL-SLT, a Web 2.0 framework which allows non-experts to design, implement and deploy online speech-enabled CALL courses. Course functionality is divided into six increasingly sophisticated levels; the lowest levels assume only basic web-literacy, while the higher ones require some acquaintance with simple software concepts like regular expressions and XML. We describe the different levels of functionality and the deployment process, which permits multiple developers to compile and run courses on a set of shared servers. The framework has recently been opened up for alpha testing, and we briefly summarize early experiences.

Reference

Available at:
http://archive-ouverte.unige.ch/unige:73661

Disclaimer: layout of this document may differ from the published version.
Helping Non-Expert Users Develop Online Spoken CALL Courses

Manny Rayner, Claudia Baur, Pierrette Bouillon, Cathy Chua, Nikos Tsourakis

University of Geneva, FTI/TIM/ISSCO, Geneva, Switzerland

Abstract

We introduce Open CALL-SLT, a Web 2.0 framework which allows non-experts to design, implement and deploy online speech-enabled CALL courses. Course functionality is divided into six increasingly sophisticated levels; the lowest levels assume only basic web-literacy, while the higher ones require some acquaintance with simple software concepts like regular expressions and XML. We describe the different levels of functionality and the deployment process, which permits multiple developers to compile and run courses on a set of shared servers. The framework has recently been opened up for alpha testing, and we briefly summarize early experiences.

1. Introduction and motivation

The basic idea of Web 2.0 is user-generated content: as the slogan has it, we must start thinking that every downloader is a potential uploader. In some areas – social networks and blogs are paradigm examples – this idea has been fully realized and become part of the everyday landscape of the Web. In others, progress has been slow and sporadic.

This paper will examine a case which so far has received comparatively little attention, spoken CALL on the Web. As commercial ventures like RosettaStone testify, hundreds of millions of people are interested in accessing language learning material which includes a speech recognition component. Another recent success story, Duolingo, includes both speech recognition and the availability of methods that in principle allow users to create their own content. In practice, though, only a couple of hundred of Duolingo’s 20 million users have actually contributed towards the creation of any courseware

Why are things this way? It could be that there is no problem; the content being provided by organizations like RosettaStone and Duolingo is all that is required. But we are doubtful about this. Having talked to many language teachers, our impression is that most of them are reluctant to use the currently available tools. The reason they usually give is that the content offered doesn’t integrate well with their courses; hardly surprising, since it wasn’t designed for that purpose. In fact, most of it isn’t designed for use in school situations at all, but for self-study by adults. It is also noteworthy that the use of speech recognition in these systems is very limited, mainly consisting of exercises where the system gives the student a written or spoken sentence and the student imitates it. This has value as a way to practise pronunciation, but does not build up spoken generation skills.

The system we describe here, CALL-SLT [1, 2], is a web-enabled platform that supports rapid development of interactive multimodal speech-oriented language courses. The basic architecture is speech-enabled prompt/response: at each turn the student is given a prompt and produces a spoken response. The system either accepts or rejects the response, possibly giving other feedback as well. This allows the student to practise both pronunciation and productive competence.

Our main focus in this paper will be on rapid development, in particular by non-expert developers. As already noted, our experience is that students enrolled in language courses need material adjusted to their own course: since there are many different language courses, it has to be easy to create material suitable for the course at hand. The goal is thus to simplify the task of creating interactive speech-enabled multimodal internet courseware enough that it can be attempted by a wide range of developers, some of whom will have only very basic technical skills.

When outlining the ideas of this platform, we have received reactions ranging from doubt that teachers have the capability to be internet developers, to claims that this kind of platform already exists, the most commonly quoted example being VoiceXML. While not wanting to suggest that all teachers have the aptitude for, or interest in, creating internet courseware, it is evident that the Web has now become such a pervasive aspect of modern life that many people in all professions are internet-literate. In many countries, including Switzerland, teachers are explicitly encouraged to increase their familiarity with new technology, including the Web [3]. For the platform to be useful, it is not necessary for all teachers to be able to build these courses. If even a few percent of them can do it, that will already be a lot.

At the other end, it is undoubtedly the case that the kind of interactive course we describe here can be built on existing platforms by a sufficiently competent developer. The question, however, is the level of competence required. Building and deploying an app on iSpeech is something that requires nontrivial software development expertise; for every person capable of doing this, there are at least fifty who can write a basic HTML web page. We are interested in catering for the people who can write some HTML, have an idea of what a piece of simple program might look like and can follow online documentation and tutorials.

In the rest of this paper, we will describe the latest version of our platform, Open CALL-SLT, which allows non-expert users to build web-enabled spoken CALL applications good enough for real use. Section 2 gives a user’s-eye view of the functionality that can be created and summarises results from a substantial evaluation exercise carried out in late 2014. Section 3 and 4 respectively describe the course implementation and deployment frameworks. Section 5 sketches a few initial example courses that have been built using the platform. The last section summarises and suggests further directions.

1http://incubator.duolingo.com/
2http://www.ispeech.org/
2. CALL-SLT functionality

CALL-SLT courses are deployed over the web and can be run either through a normal browser or on an Android device. From the student’s perspective, interaction is as follows. The student logs in and selects a course; most courses are divided into smaller units called “lessons”, and if so the student also selects a lesson. At each turn during the lesson, the system prompts using a piece of text, a piece of multimedia, or a combination of the two. Related systems have for example been built by Seneff and her colleagues at MIT [4, 5] and by the GOBL project [6].

Going back to CALL-SLT, the student usually has the option of requesting help, in which case they are shown a correct answer in written and spoken form. (Help can optionally be turned off by the course designer; spoken help examples are automatically logged from previous successful interactions with users registered as native speakers of the L2). After possibly listening to a help example, the student presses the “Record” button and holds it down while speaking. The system plays back what they have said, both for pedagogical reasons and to give feedback on microphone or background noise issues. The system then signals either an “Accept” or a “Reject”, optionally accompanying this with other behavior. Recognition is grammar-based, with the grammar constructed from specifications of anticipated correct and incorrect responses given in the course descriptions; these are compiled into packages run on the commercial Nuance Recognizer engine.

Three different kinds of courses are currently supported: plain text prompt-response, plain multimedia prompt-response, and scripted multimedia dialogues. We describe these in turn.

2.1. Plain text prompt-response courses

The simplest kind of course uses plain text prompts: the system shows the student a piece of text, and the student responds.

A minimal example of this type of course is the one described in [7], which is designed to help French-speaking students improve their English pronunciation. The course is divided into four lessons, each one concentrating on an English sound which French native speakers find difficult. Inside the lesson, the content consists of sentences built around minimal pairs of words which differ with respect to the sound in question. The screenshot below shows a typical interaction. The student has been prompted, in French, to say “I hate vegetables”. (The French “h” is silent). The system has responded by rejection (the red border) and also showing the response with the incorrect word highlighted.

2.2. Plain multimedia prompt-response courses

A slightly more complicated type of course can be built by including multimedia in the prompts: this includes static pictures (JPEGs/PNGs), video clips (WMV/MP4) and audio files (WMA/WAV). The screenshot below shows a typical interaction from a picture game where the task is to identify animals. This time the student has answered correctly, as shown by the green border.

2.3. Scripted multimedia dialogue courses

More elaborate courses can be built by integrating multimedia elements in the prompts, as well as by creating scripted dialogues, in order to simulate a virtual conversation partner for the language learner. This approach has been chosen to create an eight-lesson course for beginner German-speaking learners of English, based on a commonly used English textbook in Switzerland. Depending on language and level of difficulty, the dialogue-design can be either linear or branched. A branched dialogue-design allows for different turns, depending on the student’s response, as well as for uncooperative dialogues which increase lesson difficulty [8]. The example displayed below shows one of the first steps of the hotel lesson, where the students are asked how many nights they want to stay. As in all course designs, the students need to answer using the variable given in the L1 (here Ask for: room for 1 week). The only constraints for the answer to be accepted by the system are its grammatical correctness, understandable pronunciation and correct use of the given variable. If need be, the help function can be used, giving an example of a correct answer in both written and spoken form (recorded by native L2 speakers).

Another element that can be added to more elaborate courses, are gamification elements. In the course described above we mainly focused on scores and badges, in order to increase learner’s motivation to engage in the game [9].

2.4. Evaluations of CALL-SLT courses

Various versions of CALL-SLT courses have been evaluated, with different language combinations and course designs. A text prompt-response course for Italian-speaking university students learning French as an L2 that was integrated in an already existing e-learning platform was evaluated in [10]. Another text-based course covering the restaurant domain was tested for Arabic/Chinese L1 and French L2 [11, 12]. These exercises suggested that the user’s subjective appreciation was high
across different languages and that small and domain-focused courses can help acquiring pronunciation, lexical and grammatical skills in the L2 with systematic exercises.

The course described in section 2.3 was evaluated in an extensive experiment, where the course was tested in five secondary schools in German-speaking Switzerland, comprising fifteen school classes and a total of 215 users, of which 185 were active users with a minimum of twenty interactions. The students were asked to regularly use CALL-SLT during a four week experiment phase, as well as to take a placement test before and after the experiment and to fill in a pre- and post-questionnaire, providing qualitative feedback on CALL-SLT. During this comprehensive evaluation we were able to record more than 43,000 L2 utterances (= interactions with the system). The recorded interactions were used both for evaluation purposes and to create a large non-native speech corpus [13]. In this large-scale evaluation we got positive feedback both from users, as well as from teachers who integrated CALL-SLT in their traditional classroom teaching. Quantitative evaluations (focussing on increased language skills) show positive results.

The evaluations described above confirm that CALL-SLT seems to be a very useful tool to support traditional language acquisition strategies. However, for all participating teachers one of the main requirements before engaging in CALL-SLT projects was that the content had to be tailored to their use case.

3. Writing courses

Since the intention is to produce a framework that can be used by developers with a wide variety of different levels of sophistication, functionality is divided into six ascending levels. The lowest levels are intended to be accessible to people with extremely basic skills, comparable to those required to write and upload a simple page of HTML. The higher ones require some acquaintance with software concepts.

We briefly describe the levels in turn; we concentrate on the earlier ones, which have so far been used most. A complete tutorial introduction and reference can be found in the online documentation [14].

3.1. Level 1: Basic prompt/response

The simplest type of course consists of plain prompt/response pairs: each prompt is a piece of text, associated with one or more responses. It is possible to include explicitly incorrect responses, in order to improve recognition of expected errors. Each prompt/response pair is defined by a Prompt unit. For example, the Prompt unit used for the French pronunciation game example from section 2.1 is:

```
Prompt Lesson pronunciation_h
Group 4
Text/french dis_que: tu detestes les legumes
Response i *ate vegetables
Response i *i hate vegetables
EndPrompt
```

Here, the asterisk in the second Response line marks it as incorrect. The Group line is to specify the order in which the prompts are presented (the contrasting example for “ate” is also in group 4, so gets presented immediately before or after this one). The Lesson line marks the Prompt as belonging to the lesson pronunciation_h, which is defined as follows:

```
Lesson Name pronunciation_h
PrintName Her hair floats in the air
HelpFile pronunciation_h_help.html
EndLesson
```

There are three more Lesson units for the other lessons. The course itself is defined with the Course unit:

```
Course Name pronunciation
L2 english
Languages french
Feedback colour_highlighting_on_response
AcceptBonus 0
EndCourse
```

The Prompt, Lesson and Course units constitute the whole course.

3.2. Level 2: Multimedia

Level 2 differs from level 1 in the addition of multimedia to prompts. A Prompt unit will now contain a line starting with the word Multimedia and referencing a multimedia file. For example, the Prompt unit which produces the tiger picture shown in section 2.2 is the following:

```
Prompt Lesson animals
Group 1
Multimedia tiger.png
Text/english What is it?
Response a tiger
Response it’s a tiger
EndPrompt
```

Using multimedia hardly makes the structure more complex, but opens up many new possibilities for constructing interesting courses. Up to Level 2, as can be seen, the comparison with HTML is not unreasonable. If anything, a CALL-SLT course of this kind is rather easier to construct than an HTML page.

3.3. Level 3: Regexps, templates, grammar

Experience with writing Level 1 and 2 courses exposes some predictable problems. The course developer discovers that it is often necessary to list many similar responses to a prompt. (“I would like a coke”, “I want a coke” “I would like a coke please”, etc), and the different possibilities multiply out. Similarly, the developer will also find themselves writing many similar Prompt units; the unit for ordering a Coke will probably be almost the same as the one for ordering a Pepsi. A third problem is negative examples. In the animal game, it is easy to list a hundred or so extra animals which are not in the prompts, but which can be recognized in case the student gives an incorrect answer. If the lesson involves telling the time or giving a date, it is evidently not feasible to list all possible times and dates.

Users who are willing to acquire some basic software concepts can move to Level 3, which offers simple tools to deal with these problems. A minimal form of regular expression syntax allows compact formulation of sets of responses: for example, the body of the Response line in the Coke example can be written as

```
i (want | would like ) a coke ?please
```
where the vertical bar expresses alternation and the question-mark optionality.

Similarly, a template mechanism supports abstraction over common structure in Prompt units: instead of writing two prompts for Coke and Pepsi, a game which teaches the student to order in a restaurant could have one PromptTemplate and two ApplyTemplate lines:

```xml
PromptTemplate order WORD PICTURE
Lesson ordering
Multimedia PICTURE
Text/english Order politely
Response could i have a WORD
EndPromptTemplate

ApplyTemplate order "coke" "coke.jpg"
ApplyTemplate order "pepsi" "pepsi.jpg"
```

Finally, a simple version of context-free grammar rules permits compact definition of constructs like dates and times.

### 3.4. Level 4: Basic scripting

For the first three levels, the only tools available for controlling the order in which prompts are presented are the Lesson and Group constructs. The first allows prompts to be collected into thematic units, and the second forces them to be presented in a specified order within the lesson. By default, prompts are presented in a random order.

Level 4 introduces primitives for creating scripted dialogues. A dialogue lesson is associated with an XML file which contains the lesson’s script; the contents of the file are a list of `<step>` units. In the version presented at this level, a `<step>` specifies a group of prompts, a `<step>` to move if the student succeeds, and a `<step>` to move if they fail. A typical example, from a hotel booking lesson, is the following:

```xml
<step>
  <id>ask_for_number_nights</id>
  <group>room_for_n_nights</group>
  <limit>is_one_night_okay</limit>
  <success>ask_type_of_room</success>
</step>
```

The `<step>` is called ask_for_number_nights, and defines the exchange in the dialogue where the desk clerk asks how many nights the student wishes to stay. The `<group>` line says that a Prompt will be randomly chosen whose Group is room_for_n_nights. These are multimedia video prompts featuring a cartoon desk clerk asking a question like “How many nights will you be staying?”, together with a Text line with text indicating a number of nights.

The student is allowed a maximum number of attempts at a response, specified in the Course unit. If they succeed, they transition to the `<step>` indicated in the `<success>` line; if they fail, they transition to the `<step>` in the `<limit>` line.

Writing a Level 4 course is not particularly difficult, but it requires some ability to understand the notion of flow of control and a little familiarity with XML syntax; there is no doubt that it is qualitatively more demanding than the lower levels.

### 3.5. Level 5: Gamification

The concept of gamification has become pervasive in CALL during recent years, and there is a general belief that it improves student motivation. Level 5 introduces simple gamification methods, using a score/badge framework. In each lesson, the basic idea is that the student loses points for rejections, gains them for bonus phrases, and acquires a credit towards their next badge if they end on a high enough score. The designer adds lines to the Course unit to specify global parameters (penalties, score thresholds etc.), and to the Lesson units to define associated badge icons.

#### 3.6. Level 6: Advanced scripting

The final level introduces two more primitives, which allow construction of more elaborate scripts. The first of these permits the designer to link together two steps so that the choice of prompts is consistent; for example, if the student has been prompted in an early step from a restaurant lesson to order a certain dish, and in a later step to complain about it, the dish in question has to be the same in both cases. Consistency of steps is enforced by attaching “tags” to Prompt units, with the system making sure that the same tag is selected in all steps where it is referenced. Thus, in the restaurant example, if the prompt chosen in the “ordering” step is tagged food=hamburger, then the one chosen at the “complaining” step must be similarly tagged.

The second primitive lets the designer introduce branching dialogues with conditional steps; the conditions can depend on previously assigned tag values, badge level, or just be random choices. It is possible to design moderately non-trivial dialogues using just these two extra primitives; examples are given in [8].

### 4. Deploying courses

One of the main obstacles to creating speech-enabled multimedia content on the web is that the deployment process is typically very complex. The details vary widely between frameworks; we summarise the operations that need to be performed in ours, then describe how we have organized the developer interface so as to provide a simple and intuitive view of the underlying functionality which abstracts away the low-level details.

We begin by outlining the technical issues that need to be addressed. The most serious of these is that many users share the same server, and in particular the same grammar-based recognition resources. When one user uploads new content, this will in general change the grammar, which then needs to be reloaded. It is thus necessary to organize the uploading process in such a way that syntactically ill-formed content uploaded by one user cannot easily break the system for other users. Additional problems come from the fact that the same recognition resources are typically shared between two servers. The user of multimedia content creates further complexity. Some multimedia files, e.g. JPEG images, must be copied to the webserver; others, like MP4s, are copied to another directory associated with the Flash Media Server. (The clients are currently written in Flash). Some types of multimedia files need to be converted to a different format before they can be used (WMV, WMA, MP3 and WAV files must be converted to FLV). It goes without saying that all these details should be hidden from the users, not least because they often change.

The solution we have implemented uses three levels of deployment, which we call Compilation, Staging and Production; content can only reach a higher level by first going through the lower ones. Compilation checks well-formedness, only involves the user’s own content, and does not allow interactive use. At the Staging level, the user’s content is compiled together with the common pool and made available for testing
in text and speech mode. After sufficient testing has been performed, the user finally has the option of moving their content to the Production server and making it generally accessible.

From the user’s point of view, they follow a sequence of six steps which we call Constructing, Uploading, Selecting, Compiling, Testing and Releasing. The first two of these take place outside the platform itself, the other four inside it:

4.1. Constructing

In order to avoid requiring the user to write a metadata file or similar, we enforce a uniform directory structure. Each user is assigned one or more named directories, their namespace directories, which contain their CALL-SLT projects. Immediately under each namespace directory, there are one or more course directories. A course directory may currently have up to four subdirectories, called grammars, multimedia, scripts and doc; these respectively contain course descriptions, multimedia files, lesson scripts and lesson help files. Only the first is obligatory.

4.2. Uploading

Each user is assigned a password-protected upload directory on the server machine, which can be accessed over secure FTP. To upload course files, the user connects to the server using FileZilla or a similar tool, then drags and drops one or more namespace directories containing the new material. This must include the grammars directory and the course description.

4.3. Selecting

Having uploaded their material over FTP, the user accesses the CALL-SLT site through a normal browser, logs in, and opens the Compile & Redeploy tab. They will find themselves in the initial Select subtab, where they can press the Refresh button. If the uploaded material has an appropriate structure, the user will see a tree of available namespaces and courses. (If not, the system produces an error message, for example that a namespace contains an unknown directory). They will then be able to select one of the courses, after which the tab will look something like the following:

The system copies the selected namespace directory and its contents to the Compilation server.

4.4. Compile

The user can now move to the Compile tab and press the Compile button. This compiles the material in the selected namespace and returns feedback in the trace pane. A typical response for successful compilation looks like this:

4.5. Test

If compilation of the selected namespace was successful, the user can move to the Test tab and press the Test button. This will again produce feedback in the trace window.

Unless an internal error has occurred, the operation should succeed. The processing carried out is as follows. The system copies the selected material from the Compilation server to the Staging server and recompiles the whole set of namespaces there; it also copies any multimedia files to the places where they will be used, converting their formats if necessary. It then updates the recogniser module with the new recognition grammar for the changed namespace. This series of steps proceeds invisibly to the user and typically takes about two minutes, during which time the Staging server is offline.

When the redeployment operation has finished, the user is able to run their course interactively on the Staging server. This enables them to identify problems; they can then address them, upload the course again, and repeat the edit-debug-test cycle as many times as necessary.

4.6. Releasing

When the user is satisfied with their course, they move to the Release tab. Pressing the Release button carries out the same processing as that performed at the Test stage, but now copying material from the Staging server to the Production server and redeploying it there. Again, the user needs to wait about two minutes until they receive feedback in the trace window.
5. Initial experiences

We have recently begun alpha testing of the Open CALL-SLT platform. Most of the first group of courses are variants on the spoken multimedia game. These courses are structurally simple, hence easy to construct, but there are surprisingly many interesting things that can be done by combining a multimedia prompt and a spoken response. We present some initial anecdotal examples, each of which required less than a person-day of work to build, test and deploy. They are all freely accessible at http://callslt.unige.ch/demos-and-resources/.

Arithmetic is a course that allows students to practise using French numbers in the context of performing simple mental arithmetic. There are four lessons, for addition, subtraction, multiplication and division. A prompt is a recorded audio file with a mental arithmetic task, e.g. “quarante-trois moins dix-sept” (43 – 17). The student replies with the result, here “vingt-six” (26).

Shopping lets students practise English language skills useful for shopping at a food market. There are five lessons. In the first, a prompt is a picture of a fruit or vegetable, and the student responds by naming it (“Apple”). In the second, the prompt is a picture and either a number or a quantity, and the student responds by combining them (“Six apples”/“two kilograms of apples”). In the third, the prompt is again a simple picture, but now the student must request it politely (“Could I have an apple?”). The fourth lesson asks the student to request a number politely, without naming the article (“Six please”). The final lesson combines the second and third: the student must request a number or quantity politely (“Could I have six apples/two kilograms of apples?”).

Pokémon is a pair of courses built for young French-speaking children, where the task is to identify Pokémon characters. The course exists in both a French-language and an English-language form. In both cases, a turn consists of the system displaying a picture of a Pokémon character, which the child has to name. The games currently contain pictures of 25 characters, with 125 more in the recognition vocabulary.

Recognition with children around third grade (the core demographic for Pokémon) is good enough that they find the game very enjoyable. The French-language version only has entertainment value; some children, however, are also curious to try the English-language version, which allows them to practice English pronunciation in a play context.

6. Conclusion

We have presented Open CALL-SLT, a platform which allows non-expert users to design and deploy spoken CALL courses on the Web. Though initial results are extremely encouraging, testing is still at an early stage, with only a handful of alpha testers using it. The short-term focus is on fully stabilizing the platform and documentation (in particular, error feedback needs to be improved). We expect to enter the next phase of the project in late Q3 2015, when the system will be deployed on a larger numbers of servers and more users will be allowed in.

7. Acknowledgements

The work described in this paper was performed under funding from the Swiss National Science Foundation. We would like to thank Nuance for generously allowing us to use their software for research purposes.

8. References