Leanback Learning Handover Notes

Introduction

Leanback Learning (Lbl) is a service which creates audio content from summarised multi-lingual web articles. It allows users to enter topics and select the level of detail required. The service is designed to allow a user to consume the audio content in a 'lean back' manner, i.e. with minimal engagement with the user interface once audio is created. Leanback Learning uses text from www.wikipedia.org and currently uses English, German and French articles.

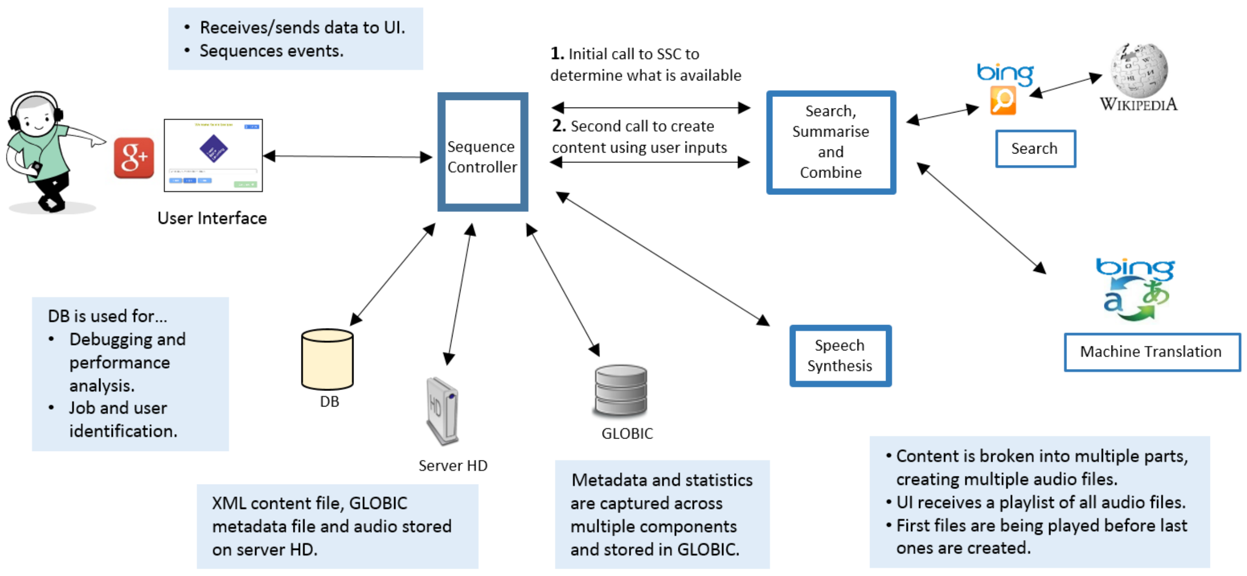
This document outlines the components for Leanback Learning and gives some details of some of the Java classes and other components and files used to implement it.

Overview and Architecture

Leanback Learning is a distributed system and consists of the following components:

Sequence Controller webservice which also serves up the user interface.
Supporting database which records details of each presentation created.
Search, Summarise and Combine component (SSC), this is a remote service which does the retrieval, translation and summarisation of the text used for presentations.
Speech Synthesis service, another remote service. This receives an XML file containing text and returns an audio file in mp3 format.
GLOBIC, a remote service which is stores meta-data about internal operations within Lbl. This data can be used to analyse Lbl for efficiency, usage patterns, etc. GLOBIC was not developed specifically for Lbl and is intended to collect data from other software services in CNGL and ADAPT.

Figure 1 shows the layout of these components.



• Receives/sends data to UI.
• Sequences events.
User Interface
Sequence Controller
1. Initial call to SSC to determine what is available
2. Second call to create content using user inputs
Search, Summarise and Combine
bing
WIKIPEDIA
Search
bing
Machine Translation
Speech Synthesis
DB is used for...
• Debugging and performance analysis.
• Job and user identification.
DB
Server HD
GLOBIC
XML content file, GLOBIC metadata file and audio stored on server HD.
Metadata and statistics are captured across multiple components and stored in GLOBIC.
• Content is broken into multiple parts, creating multiple audio files.
• UI receives a playlist of all audio files.
• First files are being played before last ones are created.

Figure 1: Leanback Learning Architecture and Layout

Workflow Summary

Leanback Learning is available at `www.lbl.ie`. The user signs using their Google sign-in credentials and then enters their chosen output language and topics (i.e. the terms which will be searched for). On clicking 'continue', the service uses the remote SSC service to determine the amount of content available for that topic. This is used to present the user with option which allow them to adjust the length of their presentation. The user can make two selection, one which determines the level of detail presented to the user ('Overview', 'Normal' or 'Detailed'), and one which determines the level of modularisation which is applied to the available text. The modularisation selection is made by clicking on one of a range of time option buttons. The user then clicks 'continue'; the process of retrieving, translating and summarising the text then starts in the SSC service. When this output is completed and returned to the Sequence Controller in XML format, this file is broken into multiple smaller XML files. These are sent in sequence to the Speech Synthesiser. When the first 'small' file is converted to audio, it is returned to the UI and listening can commence. The user is presented with a progress bar and 'Play', 'Pause' and 'Stop' buttons. While listening is under way, the remainder of the XML text files are synthesised and are available to the listener when required. For a detailed description see the following publication (NLDB 2015, Passau, Germany), sections 1 and 3 are most relevant to this document.

`http://link.springer.com/chapter/10.1007/978-3-319-19581-0_28`

Classes, Services and ancillary properties files.

Lbl is implemented as a Dynamic Web Project with the HttpServlet class `SequenceServlet` as the main service within that service. Several other 'utility' classes support the process also. The service also uses a database which generates an incremental unique ID number for each presentation (or job) and also stores details and metrics from each job. Constants and strings are stored in a `properties` file in the `config` package.

Two different methods, doPost and doGet are invoked in the `SequenceServlet` class during the preparation of an audio presentation. The `doPost()` method is invoked when the user first clicks 'continue' at the UI. User details (unique ID number and user name from Google sign-in) and chosen topics and output language are recorded in a database table. On successfully doing this, the topic strings are passed to the SSC service. This returns an XML file containing three word-count values, one for each level of detail. These details are returned to the UI.

At the UI, the three word-count values are converted to presentation duration values (in seconds). The user is presented with three buttons, one for each level of detail. As any of these buttons is selected, a range of buttons are generated which reflect 30 to 80 percent modularisation of the available text for that given level. The available times for each level of detail all have a default value pre-selected, allowing the user to simply click 'continue' if no changes are required.

On clicking continue for the second time, the `doGet()` method is invoked in the `SequenceServlet` class. Details of the users final choices on detail and modularisation levels are passed to this method. Within this method, some validation is carried out (this is partially redundant as validation is also carried out in the UI). The topic words chosen by the user are combined with the ID number generated by the database to form the basis for the final names for all XML and MP3 files generated for the presentation.

(1) The `SequenceServlet` class then does a number of steps. After each one, a Boolean variable is updated. If after any step, this variable becomes false, the job is aborted and an appropriate error (from the properties file) is returned to the UI. The steps are as follows:

(2) Call the SSC service and receive an XML file by return. The validity of the XML in this file and the absence of some other failure modes is confirmed before continuing. If the file passes the validation checks, it is stored to disk on the VM which hosts service.
(3) Update the database on the status of the job so far.
(4) The XML file is then prepared for sending to the speech synthesis. The XML file is broken into multiple parts. In preparation for this, the lengths of all sentences are balanced to be between 30 and 50 words long. This is done to ensure that all audio files at the start of the presentation are roughly the same length.
(5) In the `BalanceSentencesUtils` class, all sentences under 30 words are concatenated. Following that, all sentences over 50 words are divided roughly in half by locating the comma nearest the median word in that string. The balanced sentences are returned in an ArrayList.
(6) Next, using the number of items in this ArrayList, the number of parts in the audio presentation is calculated. This, in conjunction with the base name for the presentation is used to generate a JSON playlist. The playlist contains a URL from which each part for the overall audio presentation can be accessed over the internet.
(7) The first sentence from the ArrayList is then extracted and sent to the Speech Synthesis (SS) service.
(8) SS returns the first audio MP3 file. After some validation, this is stored to the appropriate location on the disk of the host VM. Some details are also sent to the database.
(9) Some details from the job are prepared and sent to the GLOBIC service.
(10) The JSON playlist is then returned to the UI.
(11) Within the UI, the playlist is passed to a JPlayer media player and the user can start listening to the first part of the presentation.
(12) Meanwhile, in the `SequenceServlet` class, the information required to create the remaining audio file (naming details, constants, etc.) are passed to an method as a new thread. The remaining unprocessed sentences are synthesises into audio parts in turn. The size of each audio part (i.e. how many sentences it contains) is set using constants from the properties file. Files increase in size as the processing goes on. Short files are used for early parts so that a given audio file will be available when its predecessor is completed.

Hosting of LeanbackLearning

Leanback learning is built as a war file and runs in a Tomcat 7 container on a SCSS Cloud VM (IP address 10.63.0.55). The service is available to the public a the lbl.ie URL be means of a reverse proxy arrangement put in place by the SCSS technicians.

Database Details

The database is implemented using MySql. Details of its location (VM, IP address) are specified in the file `WebContent/META-INF/context.xml` file. The database is hosted on the same VM which hosts LeanbackLearning. The name of the database is `db_leanbacklearning.` The database has two tables. `tbl_jobs` contains details of each individual presentation (or job), using the unique job ID number as the primary key. `tbl_parts` has details for individual parts of every job and uses a combination of the job ID number and the part number assigned to an individual part as the primary key. The file `SqlNotesOnDbaseCreation.txt` in the `databaseutils` package contains the schema for both these tables as well as the SQL statements needed to recreate them.

External Components

Search, Summarise & Combine (SSC)

The summariser component is a separate service and is written in Javascript and node.js. It is hosted on a SCSS VM (IP address 10.63.0.7). SSC is not available outside the SCSS network. Currently, two versions of this service run on this VM, one for an earlier version of Lbl and one for the latest version. These run on ports 9999 and 9998 respectively. The newest version (on port 9998) will be described here.

The service is called twice, once to get an estimate of how much content is available, and subsequently to retrieve, translate if necessary and summarise that content.

Differences in the old and new versions:
This version used by the current Lbl service is modified to collect some metrics for GLOBIC and wordcount details for each section for use in the Treemap visualisation at the Lbl UI. This data is added to a new element in the XML file which contains the textual content for the presentation. The presence of this element does not impact on the old version of Lbl and therefore can be used with both.

Protection Script for SSC:
Due to variations in layout and characters types used in some Wikipedia articles, SSC can crash occasionally. To ameliorate this, an cron controlled script checks for the presence of the SSC process. On detecting that SSC has crashed, it immediately restarts it. This check takes place every 10 seconds. The cron job is owned by the `lavinpe` user on the VM and runs a bash script called `monitorssc2.sh` (located in the home directory for `lavinpe`). This script outputs to two different files. The file `ssclog_9998` contains the date and time of each check on the SSC service, the status at that time and the PID number for the process running. The file `nohanguplog.txt` contains the output of the SSC process. This maybe useful for debugging and was necessary as the output is otherwise unavailable (this is a consequence of running SSC via a script). NB: to stop SSC, comment out all cron jobs and kill the SSC process.

Speech Synthesis (SS)

The Speech Synthesis is a Java WAR webservice and runs as separately and remote to Lbl. It receives an XML file and returns one MP3 file. Currently, the SS service is hosted on an SCSS VM, IP address 10.63.0.5. This VM is owned and controlled by Joao Paulo Cabral (cabralj@scss.tcd.ie) and Ketong Su (kesu@scss.tcd.ie).

Code security, github access and location.

The current version of Leanback Learning is stored in the CNGL github repository at...
`https://github.com/CNGL-repo/LeanbackLearning`
Access to this repository can be granted by Dave Lewis (Dave.Lewis@scss.tcd.ie). Github also contains the source code for two previous iterations of Lbl, these are...

1. LeanbackPrototypes.
2. LeanbackLearningDemo.

Development Notes

The Lbl front webpage is named `index.jsp`. This page includes all the necessary html divisions for implementing google sign-in. Most of the development work was done on a PC which was in the SCSS network but which did not have a public IP address. Therefore, it was not possible to fully test the JSP page. The page `input_dev.jsp` contains the same layout as `index.jsp` and can be run viewed on localhost and therefore used for testing, etc. Both files import javascript called `lbl_javascript/lbl_specific.js`. This file contains all the logic for user inputs and the various steps of the process. Each other aspect of the UI implementation, e.g. the JPlayer media player, the Treemap and Google sign-in have separate javascript files. The imports for these are commented in the header section of `index.jsp`. A similar import strategy is used for css files.

There are also a number of JSP pages which are used for development and prototypes only. The files `index.jsp` is the only one used in the UI of Lbl.

Potential Future Work

G+ login and personalisation

The google sign-in facility can be used for associating users with their past usage patterns and topic choices. The JSP page which implements the Lbl home web UI has javascript code which is able to query the Google APIs and retrieve the name and unique ID for the given user which is signed in. These are passed to the HttpServlet and stored in the database table `tbl_jobs` for each individual job or presentation.

Treemap Visualisation

A prototype of a Treemap visualisation is already implemented. This aims facilitate the creation of personal presentations. It is intended that the user can click/touch various sections of an article which they would like to add to a separate presentation which is to be created later. The data which is used to create the Treemap is extracted from a process in the SSC service. The data consists of a list of value pairs, the section name and the word count in that section. This enables the Treemap to show the size of each section in the overall article.

On clicking on a section name, the article name (already known) and the section name can be added to the Http Session. This allows it to be accessed after a number of presentations have been viewed.

Information and examples of Treemaps can be found at... `http://www.highcharts.com/demo/treemap-with-levels`. Implementation of this feature will require significant modifications to the summariser.

If you need any further information on Leanback Learning, please contact Peter Lavin, ADAPT Centre, (peter.lavin@scss.tcd.ie).