PROJECT MELT

5/8/2013

The Middle English Language Translator

In conjunction with the Virginia Tech English Department, a group of Computer Science students sought to build an online Middle English (12th to 15th century) translator with text to speech pronunciation. This document outlines their research, development, and future plans for the project.

By Zachary Lytle, Tam Ayers, and Michael Goheen
Client: Dr. Karen Swenson, English Department at Virginia Tech

www.project-melt.org
(See Attached Page for Site Map)

projectmelt@vt.edu

Project MELT

THE MIDDLE ENGLISH LANGUAGE TRANSLATOR

Navigation | Table of Contents

Project MELT

THE MIDDLE ENGLISH LANGUAGE TRANSLATOR

Research | Background in Middle English

Middle English describes dialects of English in the history of the English language between the High and Late Middle Ages, or roughly during the three centuries between the late 12th and the late 15th century. The language developed out of Old English after the Norman Conquest of 1066 and was phased out by the arrival of the printing press in Britain (1476). There are two very important linguistic developments that characterize Middle English:

In grammar, English came to rely less on inflectional endings and more on word order to convey grammatical information. (If we put this in more technical terms, it became less 'synthetic' and more 'analytic'.) Change was gradual, and has different outcomes in different regional varieties of Middle English, but the ultimate effects were huge: the grammar of English c.1500 was radically different from that of Old English. Grammatical gender was lost early in Middle English. The range of inflections, particularly in the noun, was reduced drastically (partly as a result of reduction of vowels in unstressed final syllables), as was the number of distinct paradigms: in most early Middle English texts most nouns have distinctive forms only for singular vs. plural, genitive, and occasional traces of the old dative in forms with final –e occurring after a preposition. In some other parts of the system some distinctions were more persistent, but by late Middle English the range of endings and their use among London writers shows relatively few differences from the sixteenth-century language of, for example, Shakespeare: probably the most prominent morphological difference from Shakespeare's language is that verb plurals and infinitives still generally ended in –en (at least in writing). [Oxford]

In vocabulary, English became much more heterogeneous, showing many borrowings from French, Latin, and Scandinavian. Large-scale borrowing of new words often had serious consequences for the meanings and the stylistic register of those words which survived from Old English. Eventually, various new stylistic layers emerged in the lexicon, which could be employed for a variety of different purposes. [Oxford]

Samples of Middle English text can be found at www.project-melt.org/works.

Project MELT

THE MIDDLE ENGLISH LANGUAGE TRANSLATOR

Client Representation | Dr. Karen Swenson

Dr. Karen Swenson of the Virginia Tech English Department designed a course that presents medieval British literature from ca. 700 to 1500 in its representative modes and defining contexts, including the literary influences of pagan antiquity, the native British (Celtic) traditions, Scandinavian and contemporary continental influences, the Crusades, the Byzantine Empire, and the philosophical traditions of neoplatonism and scholasticism. The course is read and taught in Middle English exclusively, with a strong emphasis on verbal pronunciation and grammar.

As a student in the course, Zachary found the Middle English very difficult to pronounce and a conversation regarding a tool for electronic assistance began. Soon after, Project MELT was established under Dr. Swenson's approval and guidance. The tool was established as a mechanical supplement to in-lecture teachings and the website would hold many informative pages on research and supplemental readings.

Requirements | Application Usage

The application was outlined with the following requirements:

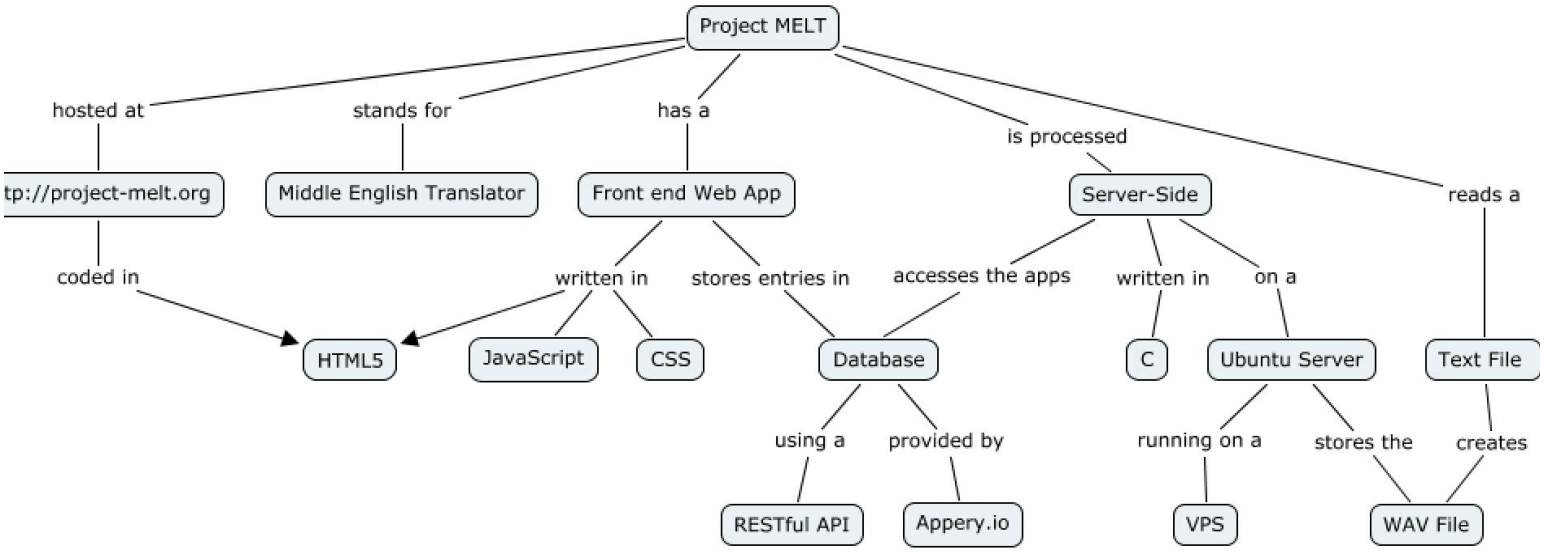
- Publically hosted online for use from any location.
- Compatible with all operating systems, browsers, and devices.
- Provide an input section for any amount of Middle English text.
- Automatically translate any archaic English characters.
- Provide a word-for-word text translation into Modern English.
- Provide a downloadable audio file of the pronunciation.

It was understood early that a word-for-word text translation into Modern English was the lowest priority and that the focus should be on audio pronunciation. The team met with Dr. Swenson often to confirm correct pronunciation and to build the ruleset that would later pronounce any Middle English word, phrase, or sentence. The next page displays a concept map of the application and its functions.

Project MELT

THE MIDDLE ENGLISH LANGUAGE TRANSLATOR

Requirements | Concept Map



Project MELT

THE MIDDLE ENGLISH LANGUAGE TRANSLATOR

TTS | Current Technology

Speech synthesis is the artificial production of human speech. A computer system used for this purpose is called a speech synthesizer, and can be implemented in software or hardware products. A text-to-speech (TTS) system converts normal language text into speech; other systems render symbolic linguistic representations like phonetic transcriptions into speech.

Synthesized speech can be created by concatenating pieces of recorded speech that are stored in a database. Systems differ in the size of the stored speech units; a system that stores phones or diphones provides the largest output range, but may lack clarity. For specific usage domains, the storage of entire words or sentences allows for high-quality output. Alternatively, a synthesizer can incorporate a model of the vocal tract and other human voice characteristics to create a completely "synthetic" voice output.

TTS | Open-Source

Project MELT is modelled after the open-source eSpeak program found in most open-source operating systems today. The software uses a "formant synthesis" method. This allows many languages to be provided in a small size. The speech is clear, and can be used at high speeds, but is not as natural or smooth as larger synthesizers which are based on human speech recordings.

eSpeak is available as:

- A command line program (Linux and Windows) to speak text from a file or from stdin.
- A shared library version for use by other programs. (On Windows this is a DLL).
- A SAPI5 version for Windows, so it can be used with screen-readers and other programs that support the Windows SAPI5 interface.
- eSpeak has been ported to other platforms, including Android, Mac OSX and Solaris.

Though eSpeak speech synthesizer was not used, the MELT team chose to model their markup language based on that written by the eSpeak developers. This text to phoneme translation is more robust than SSML (Speech Synthesis Markup Language) and allows for altering syllables through the ruleset.

Project MELT

THE MIDDLE ENGLISH LANGUAGE TRANSLATOR

TTS | Text Breakdown

A text-to-speech system (or "engine") is composed of two parts: a front-end and a back-end. The front-end has two major tasks. First, it converts raw text containing symbols like numbers and abbreviations into the equivalent of written-out words. This process is often called text normalization, pre-processing, or tokenization. The front-end then assigns phonetic transcriptions to each word, and divides and marks the text into prosodic units, like phrases, clauses, and sentences. The process of assigning phonetic transcriptions to words is called text-to-phoneme or grapheme-to-phoneme conversion. Phonetic transcriptions and prosody information together make up the symbolic linguistic representation that is output by the front-end. The back-end—often referred to as the synthesizer—then converts the symbolic linguistic representation into sound. In certain systems, this part includes the computation of the target prosody (pitch contour, phoneme durations), which is then imposed on the output speech.

TTS | Synthesizer Methods

The most important qualities of a speech synthesis system are naturalness and intelligibility. Naturalness describes how closely the output sounds like human speech, while intelligibility is the ease with which the output is understood. The ideal speech synthesizer is both natural and intelligible. Speech synthesis systems usually try to maximize both characteristics.

The two primary technologies for generating synthetic speech waveforms are concatenative synthesis and formant synthesis. Each technology has strengths and weaknesses, and the intended uses of a synthesis system will typically determine which approach is used.

Formant synthesis does not use human speech samples at runtime. Instead, the synthesized speech output is created using additive synthesis and an acoustic model (physical modelling synthesis). Parameters such as fundamental frequency, voicing, and noise levels are varied over time to create a waveform of artificial speech. This method is sometimes called rules-based synthesis; however, many concatenative systems also have rules-based components. Many systems based on formant synthesis technology generate artificial, robotic-sounding speech that would never be mistaken for human speech. However, maximum naturalness is not always the goal of a speech synthesis system, and formant synthesis systems have advantages over concatenative systems.

Project MELT

THE MIDDLE ENGLISH LANGUAGE TRANSLATOR

TTS | Synthesizer Methods (cont'd)

Formant-synthesized speech can be reliably intelligible, even at very high speeds, avoiding the acoustic glitches that commonly plague concatenative systems. High-speed synthesized speech is used by the visually impaired to quickly navigate computers using a screen reader. Formant synthesizers are usually smaller programs than concatenative systems because they do not have a database of speech samples. They can therefore be used in embedded systems, where memory and microprocessor power are especially limited. Because formant-based systems have complete control of all aspects of the output speech, a wide variety of prosodies and intonations can be output, conveying not just questions and statements, but a variety of emotions and tones of voice.

Examples of non-real-time but highly accurate intonation control in formant synthesis include the work done in the late 1970s for the Texas Instruments toy Speak & Spell, and in the early 1980s Sega arcade machines and in many Atari, Inc. arcade games using the TMS5220 LPC Chips. Creating proper intonation for these projects was painstaking, and the results have yet to be matched by real-time text-to-speech interfaces.

TTS | Diphone Concatenation

Concatenative synthesis is based on the concatenation (or stringing together) of segments of recorded speech. Generally, concatenative synthesis produces the most natural-sounding synthesized speech. However, differences between natural variations in speech and the nature of the automated techniques for segmenting the waveforms sometimes result in audible glitches in the output. There are three main sub-types of concatenative synthesis.

Diphone synthesis uses a minimal speech database containing all the diphones (sound-to-sound transitions) occurring in a language. The number of diphones depends on the phonotactics of the language: for example, Spanish has about 800 diphones, and German about 2500. In diphone synthesis, only one example of each diphone is contained in the speech database. At runtime, the target prosody of a sentence is superimposed on these minimal units by means of digital signal processing techniques such as linear predictive coding, PSOLA or MBROLA.

Project MELT

THE MIDDLE ENGLISH LANGUAGE TRANSLATOR

TTS | Existing Challenges

Speech synthesis systems use two basic approaches to determine the pronunciation of a word based on its spelling, a process which is often called text-to-phoneme or grapheme-to-phoneme conversion (phoneme is the term used by linguists to describe distinctive sounds in a language). The simplest approach to text-to-phoneme conversion is the dictionary-based approach, where a large dictionary containing all the words of a language and their correct pronunciations is stored by the program. Determining the correct pronunciation of each word is a matter of looking up each word in the dictionary and replacing the spelling with the pronunciation specified in the dictionary. The other approach is rule-based, in which pronunciation rules are applied to words to determine their pronunciations based on their spellings. This is similar to the "sounding out", or synthetic phonics, approach to learning reading.

Each approach has advantages and drawbacks. The dictionary-based approach is quick and accurate, but completely fails if it is given a word which is not in its dictionary.[Helsinki] As dictionary size grows, so too does the memory space requirements of the synthesis system. On the other hand, the rule-based approach works on any input, but the complexity of the rules grows substantially as the system takes into account irregular spellings or pronunciations. (Consider that the word "of" is very common in

English, yet is the only word in which the letter "f" is pronounced [v].) As a result, nearly all speech synthesis systems use a combination of these approaches.

Languages with a phonemic orthography have a very regular writing system, and the prediction of the pronunciation of words based on their spellings is quite successful. Speech synthesis systems for such languages often use the rule-based method extensively, resorting to dictionaries only for those few words, like foreign names and borrowings, whose pronunciations are not obvious from their spellings. On the other hand, speech synthesis systems for languages like English, which have extremely irregular spelling systems, are more likely to rely on dictionaries, and to use rule-based methods only for unusual words, or words that aren't in their dictionaries.

Project MELT

THE MIDDLE ENGLISH LANGUAGE TRANSLATOR

Language Parsing | Phoneme Translation

Letter-to-sound rules, also known as grapheme-to-phoneme rules, are important computational tools and have been used for a variety of purposes including word or name lookups for database searches and speech synthesis.

These rules are especially useful when integrated into database searches on names and addresses, since they can complement orthographic search algorithms that make use of permutation, deletion, and insertion by allowing for a comparison with the phonetic equivalent. In databases, phonetics can help retrieve a word or a proper name without the user needing to know the correct spelling. A phonetic index is built with the vocabulary of the application. This could be an entire dictionary, or a list of proper names. The searched word is then converted into phonetics and retrieved with its information, if the word is in the phonetic index. This phonetic lookup can be used to retrieve a misspelled word in a dictionary or a database, or in a text editor to suggest corrections.

Such rules are also necessary to formalize grapheme-phoneme correspondences in speech synthesis architecture. In text-to-speech systems, these rules are typically used to create phonemes from computer text. These phonemic symbols, in turn, are used to feed lower-level phonetic modules (such as timing, intonation, vowel formant trajectories, etc.) which, in turn, feed a vocal tract model and finally output a waveform and, via a digital-analogue converter, synthesized speech. Such rules are a necessary and integral part of a text-to-speech system since a database lookup (dictionary search) is not sufficient

to handle derived forms, new words, nonce forms, proper nouns, low-frequency technical jargon, and the like; such forms typically are not included in the database. And while the use of a dictionary is more important now that denser and faster memory is available to smaller systems, letter-to-sound still plays a crucial and central role in speech synthesis technology.

Project MELT

THE MIDDLE ENGLISH LANGUAGE TRANSLATOR

Language Parsing | Pronunciation Ruleset

Project MELT uses a system of phonetic markups based off the Arpabet. Arpabet is a phonetic transcription code developed by Advanced Research Projects Agency (ARPA) as a part of their Speech Understanding Project (1971–1976). It represents each phoneme of General American English with a distinct sequence of ASCII characters. Arpabet has been used in several speech synthesizers, including Computalker for the S-100 (Altair) system, SAM for the Commodore 64, SAY for the Amiga and TextAssist for the PC and Speakeasy from Intelligent Artefacts (see ST_Robotics) which used the Votrax SC01 speech synthesiser IC. It is also used in the CMU Pronouncing Dictionary.

English Consonants:

[p]		[b]	
[t]		[d]	
[tS]	church	[dZ]	judge
[k]		[g]	
[f]		[v]	
[T]	thin	[D]	this
[s]		[z]	
[S]	shop	[Z]	pleasure
[h]			
[m]		[n]	
[N]	sing		

[l]		[r]	red
[j]	yes	[w]	

Project MELT

THE MIDDLE ENGLISH LANGUAGE TRANSLATOR

Language Parsing | Pronunciation Ruleset (cont'd)

English Vowels:

[@]	alpha schwa	[3:]	nurse
[@L]	simple	[@2]	the
[@5]	to	[a]	trap
[aa]	bath	[a#]	about
[A:]	palm	[A@]	start
[E]	dress	[e@]	square
[I]	kit	[I2]	intend
[i]	happy	[i:]	fleece
[i@]	near	[0]	lot
[V]	strut	[u:]	goose
[U]	foot	[U@]	cure
[O:]	thought	[O@]	north
[o@]	force	[aI]	price
[eI]	face	[OI]	choice
[aU]	mouth	[oU]	goat
[aI@]	science	[aU@]	hour

Project MELT

THE MIDDLE ENGLISH LANGUAGE TRANSLATOR

Language Parsing | Markup Language

The utility 'phonemes' are:

'	primary stress
,	secondary stress
%	unstressed syllable
=	put the primary stress on the preceding syllable
_:	short pause
_	a shorter pause
||	indicates a word boundary within a phoneme string
|	can be used to separate two adjacent characters.

It is not necessary to specify the stress of every syllable. Stress markers are only needed in order to change the effect of the language's default stress rule.

Beyond a description of the phonemes and how to utilize them, Project MELT cannot go into the details of the markup language used for language parsing. The intellectual property is still undergoing commercial license.

Project MELT

THE MIDDLE ENGLISH LANGUAGE TRANSLATOR

Application | Overview

Project MELT using a responsive JavaScript, HTML5, and CSS front end interface. This option allows the user to access the project using almost any web enabled device with a web browser. The Project MELT web app provides a consistent user experience on both mobile and desktop modes. The web app platform means that we are not locked to one set of users on mobile. We have tested the front end experience on almost all mobile and desktop environments.

Application | Server-Side Processing

The MELT project uses extensive server computation to handle all of its processes. We decided that we wanted all of the server code to be 100% portable. The main reason for this choice was due to the fact that we had no idea what the long term location storage of the project was going to be. Right now, the server that we are using is a small VPS slice based in Chicago, IL. The VPS runs a 32-bit version of Ubuntu Linux. We are using the 12.04 build of the operating system. Right now, the VPS only has 128mb of RAM and 5GB of storage space. We have found that this is not really an issue when it comes to the small amount of RAM as our program is very efficient which text-to-speech even when handling large amounts of text. We have tested our server with an entire book and found the server able to handle the load without any issue. The main issue that we face with our current server situation is the amount of storage. The above referenced book took up 490MB of storage. We have cut the amount of installation data down to a minimum to try to free up as much storage as possible. But with only 5GB to start with, there is not much room for long term storage.

To help try to fix this problem, we have created a check in our code that removes large files every time another large file is created. This is not the perfect way to do this but its a start in the correct direction. Right now, the server side code is written in C and make many Unix system calls. The reason for this design choice was to increase portability of the code. We started to use Java for the code but decided against it so that our code could run natively on any UNIX-based system without the need of third party software.

Project MELT

THE MIDDLE ENGLISH LANGUAGE TRANSLATOR

Application | Database Listener

Our C code program accesses the web applications database using RESTful API calls. In this current Beta version of our code, the database is checked for new entries every 3 seconds. The reason for this is because we are only allowed 1,000,000 database calls per month. We were limited to this number of calls since we are not currently paying for the Appery.io system. The use of any more calls would require a paid plan starting at an additional $15 per month. We tried to find the correct time interval to match the demand of the server without running out of database calls or delaying output to the user. At the current rate, we should be able to handle about 200,000 translation requests per month. The 1,000,000 calls is a part of the free service that was set up for us by the database provider. We could purchase 5,000,000 calls if needed. However, the better option would be to explore other options that more efficiently utilized the number of calls made to the server. This is a feature that we are continuing to explore.

Application | Building the Audio File

Our program reads a .txt file and converts it to a .wav file. Once this process is completed, a link is sent to the user for their browser to handle the file. Depending on what system the user is accessing Project MELT from, the user experience will vary. We decided that almost all modern browsers have built in support for .WAV files so there was no need to reinvent the wheel. .WAV files are also uncompressed and lossless so we are able to provide the user with a high quality level of audio.

Application | Format Choice

Another reason that we decided to use .WAV for this project is that the Ubuntu operating system has built in support for creating .WAV files using text. We used and modified the espeak program that is built into the core of the Ubuntu operating system. For more information about this, see the Text-to-Speech section.

Project MELT

THE MIDDLE ENGLISH LANGUAGE TRANSLATOR

Application | Testing & Improvements

As always, there are many areas that we can improve this project's application. We are looking into more efficient ways to access the application's database that requires less system calls and API calls. We also plan to give the user more options when it comes to speed and pitch of the Middle English voice. Before we give the application more features, we plan to fix performance and storage issues first.

Right now, we feel that our server might also not be very stable. Certain test cases can break the system. We are looking to work out those bugs in the code as soon as possible. Since the program is web-based, we have created a contact page on the website that allows users to alert us of any bugs. Over the course of the last week of the project, we committed major changes to the server structure and code design that dramatically cut back on the number of bugs. That being said, there are still many that need to be sorted out. We argreed that within the next month, all known bugs from our test will be removed.

Project MELT

THE MIDDLE ENGLISH LANGUAGE TRANSLATOR

Conclusion | The Future of MELT

Currently, the project does not require much resources to run. The only cost that we have to pay for it the domain name and the VPS. This cost the team of a total of about $25 per year. We have promised to run this service for another 5 years at minimum. Based on use and future direction of the project, we will decide after 5 years what we are going to do.

At the start of this project, we never knew that there would be a longtime future for this project. While making Project MELT, we realized that we created a product that can be packaged and licensed for commercial use with other languages. The start of this kind of program could help digitally preserve dying languages forever. This would create a valuable tools for linguists around the world. We are working to creating a completely modular package that can be licensed for commercial and academic use.

Conclusion | Acknowledgements

This project would not be possible without a few crucial influences. Firstly, we would like to thank Dr. Fox for his input and support during the Spring 2013 semester. His approval and suggestions helped define the course of this project. We would also like to thank Dr. Swenson for helping put together data from recreating the Middle English language in a digital format. Thanks to Jon Doddington for his help with computer based pronunciation with the open source platform. Finally, thank you to Appery.io for giving Project MELT the RESTful database and API for the front end app development.

Conclusion | Sources

Divay, Michel. *Algorithms for Grapheme-Phoneme Translation for English and French: Applications for Database Searches and Speech Synthesis.* http://acl.ldc.upenn.edu/J/J97/J97-4001.pdf

Durkin, Philip. *An Overview of Middle English.* http://public.oed.com/aspects-of-english/english-in-time/middle-english-an-overview/

Helsinki University of Technology. *History and Development of Speech Synthesis.* http://www.acoustics.hut.fi/publications/files/theses/lemmetty_mst/chap2.html

Below is a Project MELT Sitemap Page.

Official Sitemap can be found http://project-melt.org/sitemap

Edit this page

Pages

- About
- Acknowledgements
- Contact
- History of Middle English
- Pronunciation Guide
- Sample Works
- Sitemap
- The App

Categories

- News & Updates
- Uncategorized

Tags

Authors

- melt (11)