Design of a Novel Web Utility that Provides Multi-lingual Word Definitions for Child E-Book Applications

Deeksha Adiani[1], Daniel Lewis[1], Vanessa Serao[1], Kevin Barrett[1], Amelia Bennett[1], Derick Hambly[1], Martina Shenoda[1], Samuel West[1], Garrett Coulter[1], Sultan Shagal[1], Toheeb Biala[1], Medha Sarkar[1], Joshua Wade[2(✉)], and Nilanjan Sarkar[2]

[1] Computer Science, Middle Tennessee State University, Murfreesboro, TN 37132, USA
[2] Adaptive Technology Consulting, LLC, Murfreesboro, TN 37127, USA
josh@innovateatc.com

Abstract. The use of mobile computing devices to gain access to the Internet and to interact with a range of applications has become ubiquitous, impacting on education, entertainment, healthcare, and many other domains. Engaging applications such as e-books used by children and their parents or educators have also become increasingly common, especially in the context of childhood education. An e-book that presents the reader with challenging words has the potential to improve and expand vocabulary. However, the seamless combination of methods for definition-retrieval, word sense disambiguation, and multi-lingual support are not currently available in a simple tool for the specific application of children's e-book reading. In this work, we present WordWeaver, an open-source tool for use in child reading scenarios where context-sensitive, multi-lingual definitions of unfamiliar words are determined and provided to the user via a simple web API. Our proof-of-concept design includes support for English, Spanish, and French language definitions. Preliminary results support the feasibility of WordWeaver including excellent levels of usability based on the System Usability Scale (i.e., mean SUS = 87.17). Future work will include extending support to include additional languages, definition selection tailored to individual reading skill level, and the ability to address more complicated cases of word and part-of-speech disambiguation.

Keywords: Vocabulary acquisition · Dictionary · E-book · Multi-lingual · Context-sensitive

1 Introduction

For children, an essential component of literacy development is vocabulary acquisition. Evidence shows that the clarification of word meanings during reading training produces greater gains in vocabulary than training that does not provide this information

P. Zaphiris and A. Ioannou (Eds.): HCII 2019, LNCS 11591, pp. 3–12, 2019.
https://doi.org/10.1007/978-3-030-21817-1_1

[1, 2]. E-book use by children and their parents and/or educators has become increasingly common [3, 4] and may offer a unique opportunity to present readers with challenging words that have the capacity to improve and expand vocabulary. A tool that provides context-sensitive definitions for unfamiliar words would therefore be a valuable component of an e-book application. Furthermore, support for a wide variety of languages would facilitate broader adoption and impact.

In this work, we present WordWeaver, an open-source tool for use in child reading scenarios where context-sensitive, multi-lingual definitions of unfamiliar words are provided via a simple web protocol [5]. Our proof-of-concept design currently includes support for English, Spanish, and French language definitions, but support for other languages, including Arabic, is part of ongoing work. The proposed multi-lingual dictionary utility is a web-based resource for an e-book application that provides definitions for words according to both language and context. Unlike existing single-language dictionary APIs, WordWeaver returns definitions tailored to the individual's preferred language, resulting in a more individualized e-book reading experience. While resources like Webster's Dictionary API are robust, they do not offer a one-stop solution for word definition-retrieval, language translation, word sense disambiguation, and other natural language processing functions. By unifying such functionality in a single application, we believe that WordWeaver can serve to fill this gap and offer a useful means by which to perform such functions, and, ultimately, to facilitate learning.

2 System Design

Given a specified word and surrounding text from which contextual information can be inferred, WordWeaver, the novel utility, provides context-sensitive definitions to users in a variety of languages. For proof-of-concept demonstration, the frontend consisted of a simple application supporting word selection via double-click or -tap and was implemented using the Unity game engine (www.unity3d.com). Client queries (i.e., requests for definition) are sent to a server application via HTTP POST requests consisting of JavaScript Object Notation (JSON)-serialized objects, and the server responds with a structured JSON object containing query results. The backend consists of a Python application and utilizes a range of powerful resources including Natural Language Toolkit (NLTK, version 3.3) [6], spaCy version 2.0 [7], Glosbe API [8], and Pywsd [9] for context interpretation and translation functionalities.

2.1 Front End

The user interface was created using built-in User Interface (UI) tools and data structures in the Unity game engine (version 2018.2) as well as TextMesh Pro [10], a text-manipulation package supported by the game engine. All frontend scripting was implemented in C♯. The proof-of-concept tool is composed of a simple story and supports features such as text-selection, text-highlighting, requests for definitions, and definition display (Fig. 1). While reading a story in the e-book, the reader may encounter an unfamiliar word. The text-highlighting feature implemented using Text-Mesh Pro allows the user to select the unfamiliar word by double-clicking on it—or

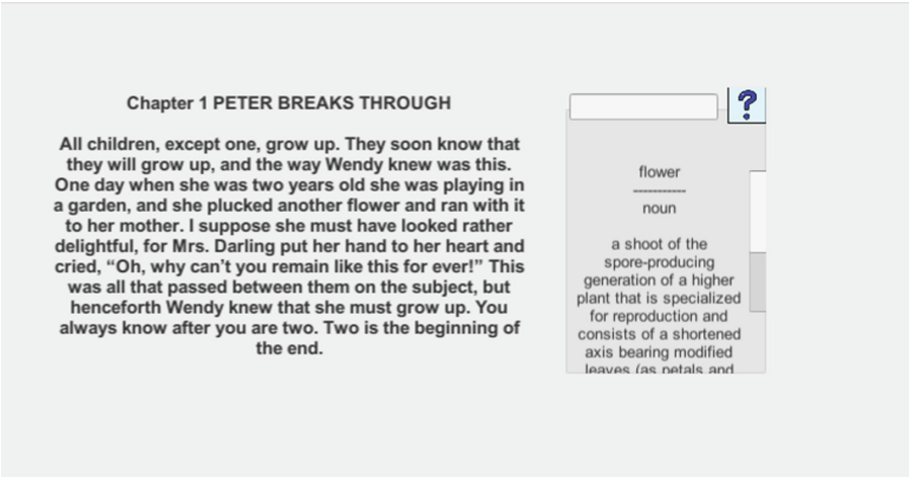

Fig. 1. Simple UI developed in unity to test communication with WordWeaver.

tapping in the case of mobile device use—thus initiating a new definition request. The selected word, the context in which the word is found, and the active story language are transmitted to the server via Transmission Control Protocol (TCP) as a JSON-serialized object, such as that shown in the following example:

```
{
    "word":"quick",
    "context":"The quick brown fox jumped over the lazy
    dog.",
    "language":"English"
}
```

The server, after processing this information, returns the most appropriate definition of the word, which can finally be presented to the user by the client in the manner appropriate to the particular client application.

2.2 Backend

The backend of the tool is written entirely in Python (version 3.6) and consists of a simple Flask server [11]. A word sense disambiguation module returns a context-appropriate definition based on information contained in each JSON object. The JSON object includes a language tag, a word to be defined, and a context in which the word appears. The word is defined with respect to the context. In WordWeaver, this is accomplished by performing word sense disambiguation based on the Lesk algorithm [12]. This is accomplished through utilization of information from Wordnet [13] which is a large lexical database of English words. Before this step occurs, WordWeaver first

checks whether the word is a stop word (high-frequency words like *the*, *to* and *also*; [14]). Because our domain is specific to children's literature, it is likely that stop words will be used in the most common sense, and are, moreover, largely ignored by Wordnet (i.e., they are not defined). We simply return the most common definition found in a separate dictionary if a request is for a stop word.

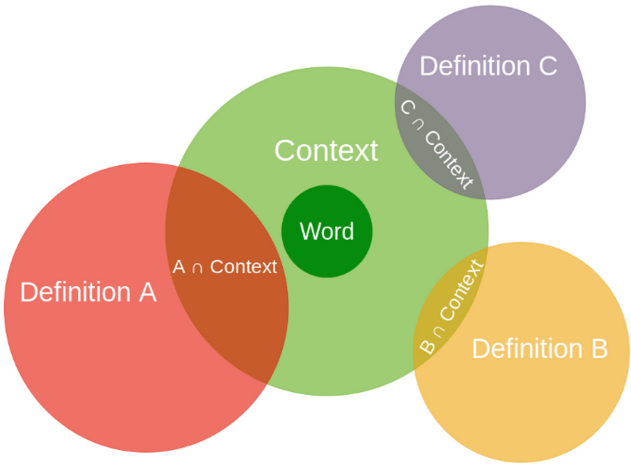

Fig. 2. A visualization of a simple lesk algorithm. In this instance definition A would be chosen.

Currently, if a request is not in English, then a translation to English must be performed before the disambiguation step can be executed. This translation is performed through a call to the Glosbe API [8]. Next, word sense disambiguation happens in two main steps. First, two versions of the lesk algorithm from the Pywsd module are used [9]; Pywsd shares an implementation and author with the built in Wordnet version of the Lesk algorithms. A simple Lesk (shown in Fig. 2) and a cosine Lesk are performed with the given context (i.e., the surrounding text), the word, and the part of speech. The results from these two algorithms are compared by checking which definition has the highest percentage of overlap with the context. Overlap is defined as the set of words in common with the context. After this step is done, some checks are made to ensure that the result is reasonable before returning result to the client. For instance, given the domain of children's literature, we can disregard rare and archaic uses of a word as uses of these words imply a higher reading level. A simple check is made to ensure that the definition returned is in the top four most common definitions for the given word. If an error occurs at any point during this process, an error message is returned and displayed to the user. WordWeaver's step-by-step process is detailed in Appendix A.

3 Results and Discussion

The preliminary usability of WordWeaver was gauged using the System Usability Scale (SUS), which is a widely used measure of the perceived ease of use of digital systems [15]. N = 15 volunteers in an undergraduate program in the lead author's university interacted with the frontend application. Subjects were asked to progress through an example story and to select words of their choosing for definition by the novel tool. The mean cumulative SUS reported by participants was 87.17 (SD = 13.46), which is interpreted as "excellent" usability based on benchmarks reported in the literature [16]. Detailed participant responses on the SUS are given in Table 1. In addition, Fig. 3 shows the results of WordWeaver for an example input word ("quick") and context ("the quick brown fox jumped over the lazy dog") in English, Spanish, and French. Cumulatively, our preliminary results indicate that WordWeaver is capable of reliably reporting user-requested definitions and that client-server interactions perform as expected.

Table 1. Interpretation of SUS cumulative scores.

Adjective	SUS cutoff	% Respondents above cutoff
Worst Imaginable	12.5	0%
Awful	20.3	0%
Poor	35.7	0%
OK	50.9	13%
Good	71.4	20%
Excellent	85.5	20%
Best Imaginable	90	47%

SUS = System Usability Scale [15]. See [16] for interpretation of scores.

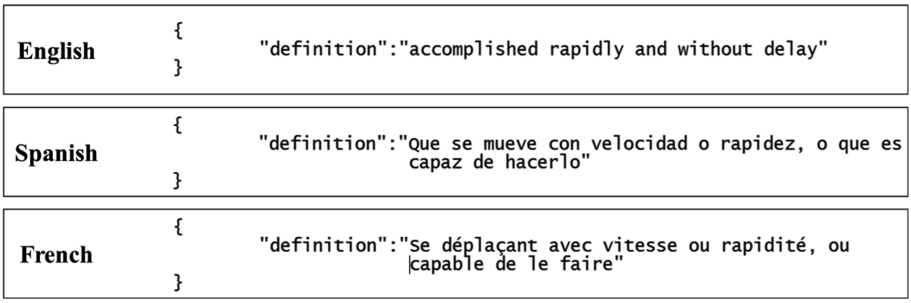

Fig. 3. WordWeaver output for the word "quick" and context ("the quick brown fox jumped over the lazy dog") in English, Spanish, and French.

4 Conclusion

There are several opportunities for future development and improvement of WordWeaver. First, the use of the Glosbe API to translate non-English requests into English has some key limitations. The most important is that many words simply do not translate directly into another single word, and machine translation is often incorrect—the reader has likely experienced this when interacting with commercial tools such as Siri, Amazon Alexa, or Google Assistant. The versions of the Lesk algorithm we used rely on Wordnet, and not every language's version of Wordnet currently provides this functionality. In the future, we hope to rectify this and fully realize the design equally across languages rather than relying on an English translation, or perhaps by using other methods such as cross referencing among the translated languages to disambiguate word meaning. In the future, we hope to additionally provide definitions at the optimal reading level of the child, but because the Lesk algorithm requires a definition and example sentences in which the word is used, a simple children's dictionary alone may not provide the algorithm with enough information to permit such fine-tuning at this time. Also, the availability of children's dictionaries and example sentences is quite limited. Another area of future work involves distinguishing proper nouns such as character names from their literal meanings. For instance, if a character is named "Mrs. Red" and the user taps on the word "Red", then a definition should not necessarily be returned by WordWeaver. One way to address cases such as these may be to have the client application maintain a list of keywords that are unique to a particular story. Then, when the user attempts to request a definition for an identified keyword from WordWeaver, the client-side application could preempt the request and returns the appropriate definition locally. Such changes as those described are in fact part of ongoing development with WordWeaver.

Appendix

Appendix A. Detailed Procedures for Returning a Definition in WordWeaver

See Fig. 4.

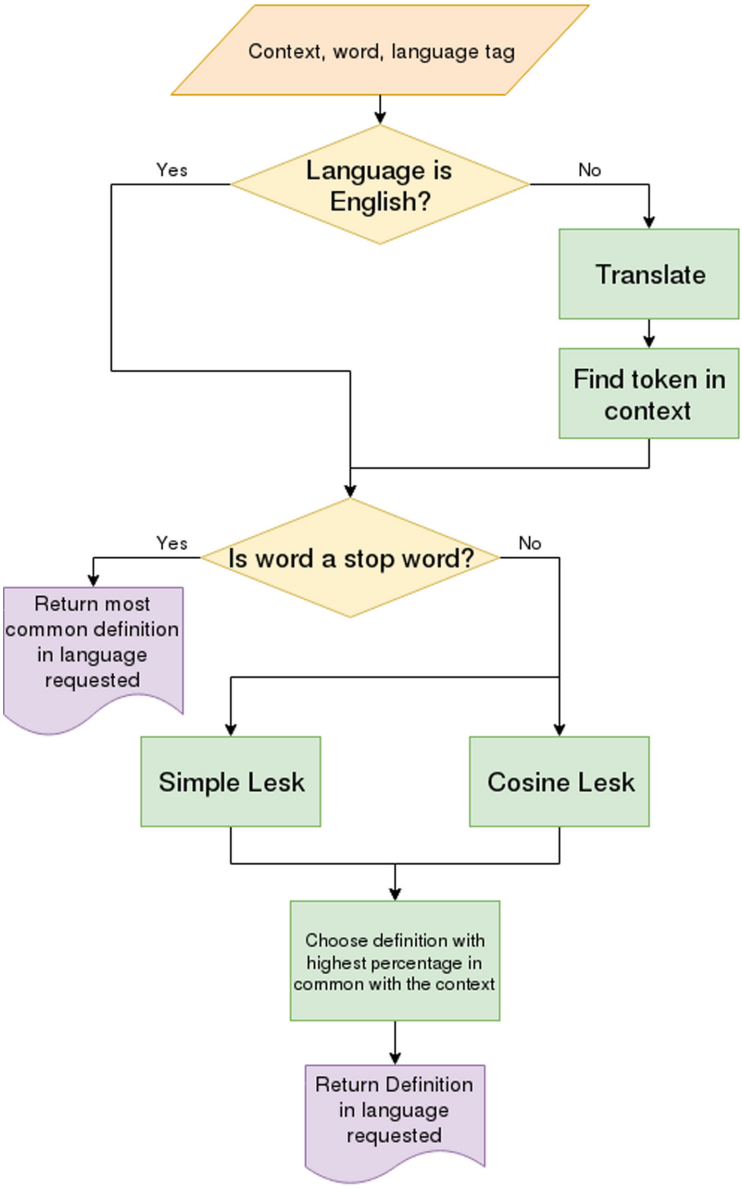

Fig. 4. WordWeaver's procedures for returning a context-sensitive and multi-lingual definition.

Appendix B. Instructions for Setting up and Utilizing the Open-Source Project WordWeaver

In order to communicate with WordWeaver's server-side application, clients must submit queries via HTTP POST requests. A complete example is given in the code listing below (C# console application), which utilizes the Json.NET implementation of JSON (i.e., Newtonsoft.Json.dll) (Fig. 4).

```
using System;
using System.Net;
using System.Collections.Generic;
using System.Text;
using Newtonsoft.Json;

public static class TCP
{
  public static void Main() {
    string serverName = "https://0.0.0.0/"; //server name
    WebClient webClient = new WebClient();
    byte[] resByte;
    byte[] reqByte;

    var data = new Dictionary<string, string> {
      {"word", "quick"},
           {"context", "The quick brown fox jumped over
             the lazy dog."},
      {"language", "English"}
    };

    webClient.Headers["content-type"]="application/json";
        reqByte = Encoding.Default.GetBytes(
          JsonConvert.SerializeObject(
          data,Formatting.Indented);
        resByte = webClient.UploadData(serverName +
          "get_def","post",reqByte);

        Console.WriteLine(Encoding.Default.GetString(
          resByte));
    webClient.Dispose();
  }
}
```

WordWeaver was designed to be easily deployed as a lightweight server application. WordWeaver can be setup on either a dedicated hosting service or locally using a

service such as ngrok (www.ngrok.com). In order to launch the server, users must execute the file Server.py (see code listing below; source code available at [5]):

```
from flask import Flask, request, jsonify
import json
from wsd import wsddef
app = Flask(__name__)


@app.route('/get_def', methods=['POST'])
def get_definition():
    definition = wsddef.get_def(request.json)
    return jsonify(definition)


if __name__ == '__main__':
    app.run(host='0.0.0.0', port=4321, debug=True)
```

References

1. Elley, W.B.: Vocabulary acquisition from listening to stories. Read. Res. Q. **24**, 174–187 (1989). https://doi.org/10.2307/747863
2. Biemiller, A., Boote, C.: An effective method for building meaning vocabulary in primary grades. J. Educ. Psychol. **98**(1), 44 (2006). https://doi.org/10.1037/0022-0663.98.1.44
3. Yuill, N., Martin, A.F.: Curling up with a good e-book: mother-child shared story reading on screen or paper affects embodied interaction and warmth. Front. Psychol. **7**, 1951 (2016). https://doi.org/10.3389/fpsyg.2016.01951
4. Baron, N.S.: Words Onscreen: The Fate of Reading in a Digital World. Oxford University Press, Oxford (2015)
5. WordWeaver (2018). https://github.com/danielyoureelewis/dictionary_group. Accessed 30 Jan 2019
6. Bird, S., Loper, E.: NLTK: the natural language toolkit. In: Proceedings of the ACL 2004 on Interactive Poster and Demonstration Sessions, p. 31 (2004). https://doi.org/10.3115/1219044.1219075
7. spaCy. https://spacy.io/usage/v2. Accessed 21 Jan 2019
8. Glosbe - the multilingual online dictionary. https://glosbe.com/. Accessed 30 Jan 2019
9. Tan, L.: Pywsd: python implementations of word sense disambiguation (WSD) technologies (2014)
10. TextMesh Pro 1.2.2 (2018). https://assetstore.unity.com/packages/essentials/beta-projects/textmesh-pro-84126. Accessed 30 Jan 2019
11. Ronacher, A.: Welcome—flask (a python microframework) (2010). https://flask.pocoo.org/. Accessed 30 Jan 2019
12. Banerjee, S., Pedersen, T.: An adapted lesk algorithm for word sense disambiguation using wordnet. In: Gelbukh, A. (ed.) CICLing 2002. LNCS, vol. 2276, pp. 136–145. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45715-1_11

13. Miller, G.: WordNet: An Electronic Lexical Database. MIT Press, Cambridge (1998). https://doi.org/10.1002/9781405198431.wbeal1285
14. Bird, S., Klein, E., Loper, E.: Natural language processing with python (2014). https://www.nltk.org/book/ch02.html#stopwords_index_term. Accessed 30 Jan 2019
15. Brooke, J.: SUS-a quick and dirty usability scale. Usability Eval. Ind. **189**(194), 4–7 (1996). https://doi.org/10.1.1.232.5526
16. Bangor, A., Kortum, P.T., Miller, J.T.: An empirical evaluation of the system usability scale. Intl. J. Hum.-Comput. Interact. **24**(6), 574–594 (2008). https://doi.org/10.1080/10447310802205776