INTRODUCTION TO BABELCODE PROJECT

Authoring Machine-Translatable Content

http://www.babelcode.org

Yao Ziyuan (info@babelcode.org)

First-Year Computer Science B.S. Student of Fudan University

December 13, 2003

CONTENTS

OVERVIEW AND QUICK DEMO

Current theories and practices in machine translation and computer-aided translation remain severe problems:

- Analytical and statistical machine translation approaches produce misleading and uncomfortable results mainly due to the impossibility of perfect disambiguation, thus are not eligible or reliable for industry-strength application.
- Computer-aided translation involves human translators and requires additional efforts for multilingual translation versions, adding up to enormous expenses, time and risk of human translation errors.

In BabelCode Project, I propose a set of methods that enable authors to directly create machine-translatable content and guarantee correctness, efficiency and readability of the translation result, plus the benefit that you can automatically obtain multilingual translation versions all from one single source file.

Here you can inspect some translation examples using my system:

1. *Oh, say can you see by the dawn's early light*

   What so proudly we hailed at the twilight's last gleaming?

哦，看，你可以借助拂晓初期的光，看见我们如此骄傲地、向黎明最后的泛动致敬过什么吗?

2. More examples to be added here…

AUTHOR AND READER PREREQUISITE

The author is required to master his own native language (read & write), basic grammar analysis and a SourceScript language. He is NOT required to know any foreign language.

The reader is required to master his own native language (read). He is NOT required to know any foreign language.

TRANSLATION WORKFLOW

Idea in Mind (1. Author & AIE & Dicts) → SourceScript File (2. AIE) → BabelCode File (3. TE & Dicts) → Destination Language Version for Reader

Remarks on the three processes:

(1) The author creates a SourceScript file with an Integrated Authoring Environment (IAE) and one or more dictionaries.

(2) The IAE automatically converts the SourceScript to a standard BabelCode-format file.

(3) A Translation Engine (TE) renders the desired destination language version using the BabelCode file and the dictionaries it is created with.

As you see, human intervention is only involved in the authoring process (step 1).

See also these key components involved: IAE, dictionary, SourceScript, BabelCode, and TE.

INTEGRATED AUTHORING ENVIRONMENTS AND SOURCESCRIPTS

The Integrated Authoring Environment (IAE) and SourceScript are the user interface that the author directly interacts with.

IAE is a software program that provides workspace where the author writes SourceScript files. IAE also automatically converts SourceScript files to BabelCode files.

SourceScript is not a single format, but the general name of any user script language that the author uses to write his idea, while BabelCode is a single, compiled standard format that is recognized by all TEs. SourceScript to BabelCode is like Pascal / C source code to ASM code -- there can be many user programming languages (Pascal, C, C++, BASIC) favored by different users, but there is only one standard compiled file format (ASM). Likewise, there can be many kinds of SourceScripts: ChineseScript, FrenchScript, GermanScript, etc., but BabelCode is the only standard compiled format for translation engines.

AUTHORING OVERVIEW

Sentence is the basic expression unit for normal text. The author has three ways to write a sentence: (1) analytical construction; (2) macros; (3) a mix of analytical construction and macros.

The general principles of choosing which way are:

(1) Always use macros if possible;

(2) Use analytical construction under the guidance of IAE and our recommendations.

ANALYTICAL CONSTRUCTION

Analytical construction resolves grammatical, lexical and pronoun ambiguity. In this mode, the author writes a sentence like an arithmetic expression, with signs and brackets that denote the detailed grammar structure of the sentence, and with index numbers or string keys that specify his intended sense for any multi-sense word.

For example, a type of SourceScript code thus writes:

```
I am a (smart:clever> cat.
```

The “(“ and “>” enclose a modifier “smart”. This modifier modifies “cat” on the right (“(…>” stands for “modifying the right component” while “<…)” stands for “modifying the left component”). Since “smart” is a multi-sense word, an index key “clever” is used to further nail down the intended sense. While editing this sentence, the IAE may provide color-highlighting, assigning light colors to index keys and dark colors to “regular text” such as “smart” and “cat”. The IAE can also provide in-place dropdown menus to prompt the author of available index keys or upcoming grammar structures, like IntelliSense in Visual C++ IDE.

When the IAE converts SourceScript code to BabelCode, all functional tokens (“(…>”, “:”, etc.) and structures are converted to a standard form, which is described in the chapter “BabelCode: Compiled Format for Machine-Translatable Content.”

MACROS

Macros help the author get rid of complex or unclear grammar analysis and avoid intercultural differences of expression. What the author has to do is just choose a macro and “fill in the blank” the macro’s arguments. For example, a macro’s prototype may look like this pseudo code:

```
MODIFIER better_than(NOUN ThanWhom, NOUN InWhat);
```

So if the author wants to say “I’m better than Tom in running”, he can do so by invoking the macro in his code:

```
I am better_than (Tom, running).
```

As you see, the macro will handle all details of internal grammar structures and even the choice of additional words (a preposition here). This is specifically important when word-for-word translation can’t help: English expression uses “better than sb. in sth.” but Chinese actually uses “better than sb. on sth.”. Without macros, an English author will cause the prep “in” to get into the final Chinese translation. But if macros are used, they will handle everything for the author because they have different implementations for each target language, which look like this pseudo code:

```
MODIFIER better_than(NOUN ThanWhom, NOUN InWhat); English;

{

 _superior(good:fine) <than:prep ThanWhom) & <in:about InWhat);

}
```

```
MODIFIER better_than(NOUN ThanWhom, NOUN InWhat); Chinese;

{

 _superior(good:fine) <than:prep ThanWhom) & <on:about InWhat);

}
```

The code above in brackets { } is analytical construction for internal details of the macros. As you would imagine, an author can create his own macros since he can do analytical construction.

Macros in practical use exert much more power than the above example, because they can virtually encapsulate a more natural target language expression, rather than a stiff word-for-word interpretation.

BABELCODE: COMPILED FORMAT FOR MACHINE-TRANSLATABLE CONTENT

The BabelCode Specification defines the standard, machine-oriented format to describe analytical construction and macro calls, plus some decorative and commentary elements. All translation engines (TEs) must recognize BabelCode files and render target language versions based on this information. But it is not intended for authors to directly view or edit these files.

You can view the BabelCode Specification in the document “babelspec”.

DICTIONARIES

Dictionaries have two uses:

(1) Provide a list of all senses for each multi-sense word or macro so that the author can choose his intended one from it.

(2) Provide translation candidates and related information for each sense and macro so that translation engines can render a target language version based on the dictionary’s self-explanatory information.

A SourceScript / BabelCode document can designate one or more dictionaries for interpretation purposes. It is like you include several header files in a C source file:

```
#include <stdio.h>

#include <stdlib.h>

// main code goes here.
```

Similarly we can have:

```
#include <general.dict>

#include <computer_science.dict>

// main babelcode goes here.
```

As you see, the author can designate a general-purpose vocabulary and a specialized vocabulary for the computer science domain.

You can view the Dictionary Format Specification in the document “dictspec”.

TRANSLATION ENGINES

Translation Engines (TEs) are software programs that accept BabelCode documents and convert them to certain target language versions. TEs can also be integrated into an IAE, or Web browsers, instant messaging clients, etc.

Besides outputting a translation result, a TE may also attach a “Reader’s Notes” to the result when there are inevitable intercultural differences which the reader should be aware of.

Thanks to the designs of BabelCode and dictionaries, implementing a translation engine is just a matter of programming. No significant technical considerations to be discussed here.