Multilingual Text Entry using Automatic Language Detection

Yo Ehara and **Kumiko Tanaka-Ishii**
Graduate School of Information Science and Technology, University of Tokyo
13F Akihabara Daibiru, 1-18-13 SotoKanda Chiyoda-ku, Tokyo, Japan
ehara@r.dl.itc.u-tokyo.ac.jp kumiko@i.u-tokyo.ac.jp

Abstract

Computer users increasingly need to produce text written in multiple languages. However, typical computer interfaces require the user to change the text entry software each time a different language is used. This is cumbersome, especially when language changes are frequent.

To solve this problem, we propose TypeAny, a novel front-end interface that detects the language of the user's key entry and automatically dispatches the input to the appropriate text entry system. Unlike previously reported methods, TypeAny can handle more than two languages, and can easily support any new language even if the available corpus is small.

When evaluating this method, we obtained language detection accuracy of 96.7% when an appropriate language had to be chosen from among three languages. The number of control actions needed to switch languages was decreased over 93% when using TypeAny rather than a conventional method.

1 Introduction

Globalization has increased the need to produce multilingual text — i.e., text written in more than one language — for many users. When producing a text in a language other than English, a user has to use text entry software corresponding to the other language which will transform the user's key stroke sequences into text of the desired language. Such software is usually called an *input method engine* (IME) and is available for each widely used language. When producing a multilingual text on a typical computer interface, though, the user has to switch IMEs every time the language changes in a multilingual text. The control actions to choose an appropriate IME are cumbersome, especially when the language changes frequently within the text.

To solve this problem, we propose a front-end interface called TypeAny. This interface detects the language that the user is using to enter text and dynamically switches IMEs. Our system is situated between the user key entry and various IMEs. TypeAny largely frees the user from the need to execute control actions when switching languages.

The production of multilingual text involves three kinds of key entry action:

- actions to enter text
- actions to control an IME[1]
- actions to switch IMEs

Regarding the first and second types, substantial work has been done in the UI and NLP domain, as summarized in (MacKenzie and Tanaka-Ishii, 2007). There has especially been much work regarding Chinese and Japanese because in these languages the number of actions of the second type is closely related to the accuracy of conversion from Romanized transcription to characters in each of these languages, and this directly reflects the capability of the language model used.

[1]When using predictive methods such as completion, or kana-kanji conversion in Japanese, the user has to indicate to the IME when it should predict and which proposed candidate to choose.

In contrast, this paper addresses the question of how to decrease the need for the third type of action. From the text entry viewpoint, this question has received much less attention than the need to reduce the number of actions of the second type. As far as we know, this issue has only been directly addressed by Chen et al. (2000), who proposed integrating English entry into a Chinese input system rather than implementing multilingual input.

Reports on ways to detect a change in the language used are more abundant. (Murthy and Kumar, 2006) studied the language identification problem based on small samples in several Indian languages when machine learning techniques are used. Although they report a high accuracy for the method they developed, their system handles switches between two Indian languages only. In contrast, TypeAny can handle any number of languages mixed within a text.

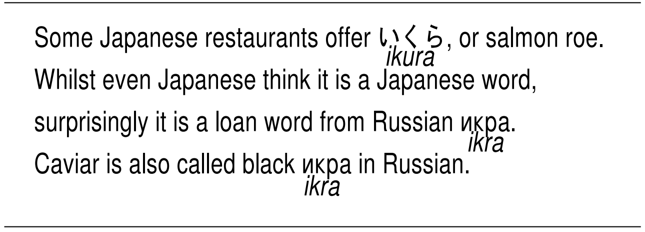
(Alex, 2005) addresses a related task, called *foreign inclusion detection* (FID). The task is to find foreign (i.e., English) inclusions, such as foreign noun compounds, within monolingual (i.e., German) texts. Alex reported that the use of FID to build a polyglot TTS synthesizer was also considered (Pfister and Romsdorfer, 2003), (Marcadet et al., 2005). Recently, Alex used FID to improve parsing accuracy (Alex et al., 2007). While FID relies on large corpora and lexicons, our model requires only small corpora since it incorporates the transition probabilities of language switching. Also, while FID is specific to alphabetic languages, we made our method language-independent by taking into consideration the inclusion problem at the key entry level.

In the following, we introduce the design of TypeAny, explain its underlying model, and report on our evaluation of its effectiveness.

2 Design of TypeAny

Figure 1 shows an example of text written in English, Japanese and Russian. The strings shown between the lines indicate the Roman transcription of Japanese and Russian words.

With a conventional computer interface, entering the text shown in Figure 1 would require at least six control actions since there are six switches between languages: from English to Japanese and back, and



Some Japanese restaurants offer いくら, or salmon roe.
ikura
Whilst even Japanese think it is a Japanese word,
surprisingly it is a loan word from Russian икра.
ikra
Caviar is also called black икра in Russian.
ikra

Figure 1: Example of Multilingual Text in English, Japanese and Russian

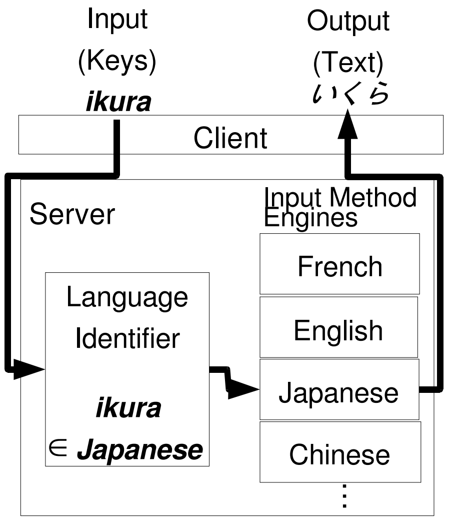


Figure 2: System Structure

twice from English to Russian and back. Note that such IME switches are also required even when the text consists only of European languages. Each European language has its own font and diacritic system, which are realized by using IMEs.

TypeAny solves the problem of changing IME. It is situated between the user's key entry and various IMEs as shown in the system architecture diagram of Figure 2. The user's key entry sequence is input to our client software. The client sends the sequence to the server which has the language identifier module. This module detects the language of the key sequence and then sends the key sequence to the appropriate IME[2]. The selected IME then converts the key entries into text of the detected language.

In our study, IMEs for European languages are built using simple transliteration: e.g., "[" typed in an English keyboard is transliterated into "ü" of German. In contrast, the IMEs for Japanese and Chinese

[2]Precisely speaking, TypeAny detects keyboard layouts (i.e., Qwerty, Dvorak, Azerty, etc.) as well as the languages used.

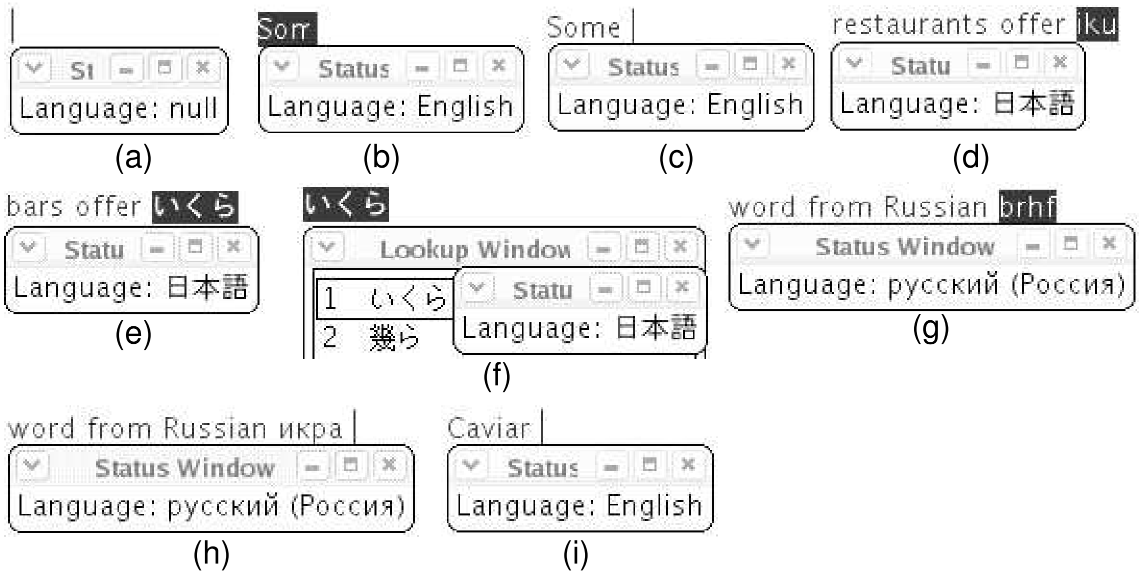


Figure 3: Entry Flow

require a more complicated system because in these languages there are several candidate transcriptions for a key sequence. Fortunately, several existing software resources can be used for this. We use Anthy[3] as the IME for Japanese. As for the IME for Chinese, we used a simple word-based pinyin-hanzi conversion system.

TypeAny restarts the language detection every time a certain delimiter appears in the user's key sequence. By using delimiters, the system can avoid resorting to a combinatorial search to find the border between languages. Such delimiters naturally occur in natural language texts. For example, in the case of European languages, blank spaces are used to delimit words and it is unlikely that two languages will be mixed within one word. In languages such as Chinese and Japanese, blank spaces are typically used to indicate that the entry software should perform conversion, thus guaranteeing that the sequence between two delimiters will consist of only one language[4]. Therefore, assuming that a text fragment between two delimiters is written in just one language is natural for users. A text fragment between two delimiters is called a *token* in TypeAny.

An example of the TypeAny procedure to enter the text from Figure 1[5] is shown in Figure 3. In each step in Figure 3, the text is entered in the first line, where the token that the user is entering is highlighted. The language estimated from the token is shown in the locale window shown below (called the Status Window). Each step proceeds as follows.

(a) The initial state.

(b) The user first wants to type a token "Some" in English. When "Som" is typed, the system identifies that the entry is in English. The user confirms this language by looking at the locale window.

(c) The user finishes entering the token "Some" and when the user enters a blank space, the token "Some" is confirmed as English text. TypeAny restarts detection for the language of the next token. The tokens up to and including "offer" are entered similarly to "Some".

(d) The user types in a token "ikura" in Japanese. The moment "iku" is typed, "iku" is identified as Japanese, as is confirmed by the user through the locale window.

(e) When the user finishes entering the token "ikura" and types in a blank space, the sequence is sent to a Japanese IME to be converted into "ikura", so that a Japanese text fragment is obtained.

(f) Through conventional kana-kanji conversion,

[3]http://anthy.sourceforge.jp/

[4]Note that a token can consist of a sequence longer than a word, since many types of conversion software allow the conversion of multiple words at one time.

[5]This case assumes use of the Qwerty keyboard.

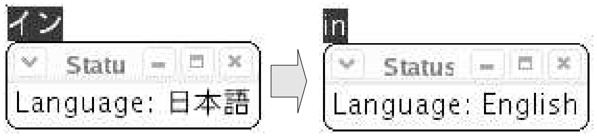


Figure 4: When Detection Fails

the user can select the appropriate conversion of “ikura” among from candidates shown in the Lookup Window and the token is confirmed. TypeAny begins detecting the language of the next token. The tokens between “or” and “Russian” are successfully identified as English in a way similar to procedures (b) and (c).

(g) The key entry “brhf” is the key sequence for the Russian token whose English transliteration is “ikra”.

(h) Since “brhf” is identified as Russian, “brhf” is converted into Russian characters.

(i) The following word “Caviar” is detected as English, as in (b) and (c).

As seen in this example, the user does not need to take any action to switch IMEs to enter tokens of different languages.

Two types of detection failure occur in TypeAny:

Failure A: the language should switch, but the new language is incorrectly selected.

Failure B: the language should not switch, but TypeAny misjudges that it should.

While conventional methods require a control action every time the language switches, TypeAny requires a control action *only* to correct such a failure. Therefore, **Failure A** never increases the number of control actions compared to that of conventional methods. On the other hand, **Failure B** errors are a concern as such failures might increase the number of control actions to beyond the number required by a conventional method. Thus, the effectiveness of introducing TypeAny depends on a trade-off between fewer control actions at language switching points and potentially more control actions due to **Failure B** errors. Our evaluation in §4.2 shows that the increase in the number of actions due to **Failure B** errors is insignificant.

In the event of failures, the user can see that there is a problem by watching the locale window and then easily correct the language by pressing the TAB key. For example, while “in” was correctly judged for our example, suppose it is incorrectly detected as Japanese as shown in Figure 4(a). In this case, the user can manually correct the locale by pressing the TAB key once. The locale is then changed from Figure 4(a) (where “in” is identified as Japanese), to (b) where “in” is identified as English.

Note that the language of some tokens will be ambiguous. For example, the word “sushi” can be both English and Japanese because “sushi” has almost become an English word: many loan words share this ambiguity. Another case is when diacritic marks are considered: for example, the word “fur” is usually an English word, but some German users may wish to use this word as “für” without diacritic marks. Such a habit is widely seen among users of European languages. Some of this sort of ambiguity is disambiguated by considering the context and by online learning, which is incorporated in the detection model as explained next.

3 Language Detection

3.1 Language Detection Model

We modeled the language detection as a hidden Markov model (HMM) process whose states correspond to languages and whose outputs correspond to tokens from a language.

Here, the goal is to estimate the languages $\hat{l}_1^m$ by maximizing $\mathrm{P}(l_1^m, t_1^m)$, where $l \in L$ denotes a language in L, a set of languages, and t denotes a token[6]. By applying a hidden Markov model, the maximization of $\mathrm{P}(l_1^m, t_1^m)$ is done as shown in Equation (1).

$$\begin{aligned}\hat{l}_1^m &= \operatorname*{argmax}_{l_1^m \in L} \mathrm{P}(l_1^m, t_1^m) \\ &= \operatorname*{argmax}_{l_1^m \in L} \mathrm{P}(t_1^m | l_1^m)\mathrm{P}(l_1^m) \\ &\approx \operatorname*{argmax}_{l_1^m \in L} \left(\prod_{i=1}^{m} P(t_i|l_i)\right)\left(\prod_{i=1}^{m} P(l_i|l_{i-k}^{i-1})\right) \quad (1)\end{aligned}$$

In the last transformation of Equation (1), it is assumed that $\mathrm{P}(t_1^m|l_1^m) \approx \prod_{i=1}^m P(t_i|l_i)$ and $\mathrm{P}(l_i|l_1^{i-1}) \approx P(l_i|l_{i-k}^{i-1})$ for the first and the second terms, respectively. In Equation (1), the first term

[6]Let $t_u^v = (t_u, t_{u+1}, \ldots, t_v)$ be an ordered list consisting of $v - u + 1$ elements for $v \geq u$.

corresponds to the *output probabilities* and the second term corresponds to the *transition probabilities*.

In a usual HMM process, a system finds the language sequence (i.e., state sequence) l_1^m that maximizes Equation (1) by typically using a Viterbi algorithm. In our case, too, the system can estimate the language sequence for a sequence of tokens. However, as discussed earlier, since it is unlikely that a user enters a token consisting of multiple languages, our system is designed only to estimate the language of the latest token l_m, supposing that the languages of the previous l_1^{m-1} are correct.

In the following two sections, the estimation of each term is explained.

3.2 Output Probabilities

The output probabilities $P(t_i|l_i)$ indicate the probabilities of tokens in a monolingual corpus, and their modeling has been substantially investigated in NLP.

Note that the estimation of $P(t_i|l_i)$ requires monolingual corpora. If the corpora are large, $P(t_i|l_i)$ is estimated from the token frequencies. However, because large corpora are not always available, especially for minor languages, $P(t_i|l_i)$ is estimated using key entry sequence probabilities based on n-grams (with maximum n being n_{max}) as follows:

$$P(t_i|l_i) = P(c_1^{|t_i|}|l_i) = \prod_{r=1}^{|t_i|} P(c_r|c_{r-n_{max}+1}^{r-1}, l_i) \quad (2)$$

In Equation (2), $t_i = c_1^{|t_i|}$ and $|t_i|$ is the length of t_i with respect to the key entry sequence. For example, in the case of t_i="ikura", $|t_i| = 5$ and c_1="i", c_2="k", c_3="u" and $|t_i|$=5. Here, each probability $P(c_r|c_{r-n_{max}+1}^{r-1}, l_i)$ needs to be smoothed.

Values of $P(t_i|l_i)$ are estimated from monolingual corpora. If the corpora are large, $P(t_i|l_i)$ is estimated from the token frequencies. However, because large corpora are not always available, especially for minor languages, $P(t_i|l_i)$ is estimated using smoothed character-based n-grams. *Prediction by Partial Matching*, or PPM is adopted for this task, since it naturally incorporates online learning and it is effective in various NLP tasks as reported in (Teahan et al., 2000) and (Tanaka-Ishii, 2006). PPM uses $c_1^{r-n_{max}}$ as a corpus for training. PPM is designed to predict the next c_r by estimating the n_{max}-gram probability $P(c_r|c_{r-n_{max}+1}^{r-1})$ using backing-off techniques with regard to the current context. Precisely, the probability is estimated as a weighted sum of different $(n+1)$-gram probabilities up to a fixed n_{max}-gram as follows:

$$P(c_r|c_{r-n_{max}+1}^{r-1}) = \sum_{n=-1}^{n_{max}-1} w_n p_n(c_r) \quad (3)$$

The weights w_n are determined through *escape probabilities*. Depending on how the escape probabilities are calculated, there are several PPM variants, which are named PPMA, PPMB, PPMC, and so on. PPMC, the one that we have used, is also known as Witten-Bell smoothing in the NLP field (Manning and Schuetze, 1999). The escape probabilities are defined as follows.

$$w_n = (1 - e_n) \prod_{n'=n+1}^{n_{cont}} e_{n'} \quad (-1 \le n < n_{cont}) (4)$$

$$w_{n_{cont}} = (1 - e_n)$$

Here, n_{cont} is defined as the maximum n that satisfies $X_n \neq 0$. Let X_n be the number of c_{r-n}^{r-1}, x_n be the number of c_{r-n}^{r} and q_n be the number of different keycodes followed by c_{r-n}^{r-1} found in c_1^{r-n-1}.

Using these notations, $p_n(c_r)$ is defined as

$$p_n(c_r) = \frac{x_n}{X_n} \quad (5)$$

In PPMC, the escape probabilities are calculated as

$$e_n = \frac{q_n}{X_n + q_n} \quad (6)$$

For further details, see (Bell et al., 1990).

3.3 Language Transition Probabilities

Only a small corpus is typically available to estimate $P(l_m|l_{m-k_{max}+1}^{m-1})$, where k_{max} is the longest k-gram in the language sequence to be considered. Thus, the transition probability is estimated on-line, making use of language that will be corrected interactively by the user. For this on-line learning, we adopted PPM as well as the output probabilities.

Note that a large k_{max} may reduce accuracy, which is intuitively explained as follows. While there is typically a high probability that the subsequent language will be the same as the current language, it is unlikely that any language sequence will have long regular patterns. Therefore, k_{max} should be fixed according to this consideration.

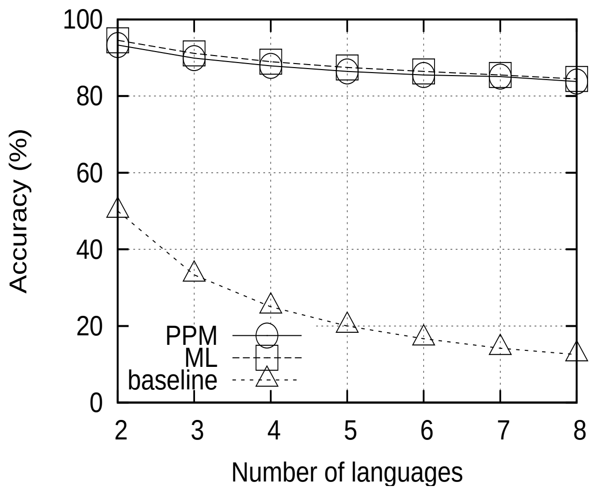


Figure 5: Detection Accuracy Test1

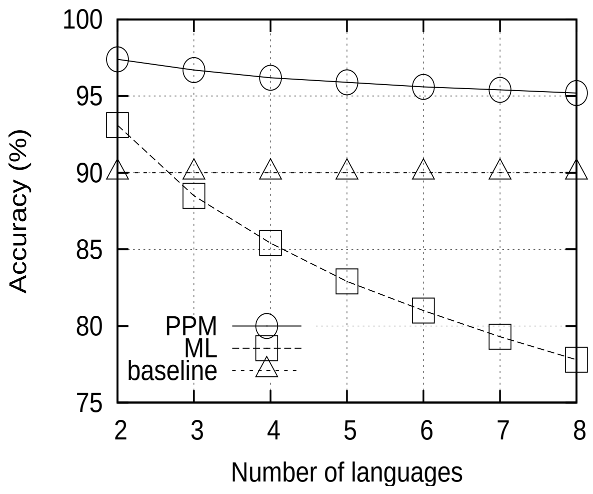


Figure 6: Detection Accuracy Test2

4 Evaluation

We evaluated TypeAny with respect to two measures: language detection capability when using artificially generated multilingual corpora, and the number of required control actions when using actual multilingual corpora.

4.1 Language Detection Accuracy

The ideal experiment would be to use actual multilingual corpora for many language sets. However, it is still difficult to collect a large amount of multilingual corpora with adequate quality for the test data of languages.

Therefore, we measured the language detection accuracies using artificially generated multilingual corpora by mixing monolingual corpora for every combination varying from two to eight languages.

First, the following monolingual corpora were collected: editions of the Mainichi newspaper in 2004 for Japanese, the Peking University corpus for Chinese, and the Leipzig corpora (Biemann et al., 2007) for English, French, German, Estonian, Finnish and Turkish. The text of each of these corpora was transformed into a sequence of key entries.

Two test sets, Test1 and Test2, were generated by using different mixture rates. In Test1, languages were mixed uniformly and randomly, whereas in Test2 a major language accounted for 90% of the text and the remaining 10% included different languages chosen uniformly and randomly. Test2 is more realistic since a document is usually composed in one major language.

The output and language transition probabilities were estimated and smoothed using PPMC as described in §3. Since part of the target of the experiment was to clarify the relation between learning size and accuracy, the output probabilities and transition probabilities were *not* trained on-line while the text was entered using PPMC, thus accuracy was measured by fixing the language model at this initial state. We used $n_{max} = 5$ for the output probability and $k_{max} = 1$ for the transition probability since the distribution of languages in the corpus was uniform here as we generated it uniformly. (See formula (4) in §3.2).

A 10-fold cross validation was applied to the generated corpora. Each generated corpus was 111 Kbytes in size, consisting of a disjoint 100-Kbyte training part and an 11-Kbyte testing part. The output probabilities were trained using the 100-Kbyte training part. The language transition probabilities were trained using about 2000 tokens.

The results for Test1 and Test2 are shown in Figure 5 and Figure 6, respectively. The horizontal axis shows the number of languages and the vertical axis shows the detection accuracy. There are three lines: PPM indicates that the transition probabilities were trained by PPM; ML indicates that no transition probability was used and the language was detected using only output probabilities (maximum likelihood only); Baseline is the accuracy when the most frequent language is always selected.

As shown in Figure 5 (Test1), when the mixture was uniform, the PPM performance was slightly lower but very close to that of ML. This was because PPM would be theoretically equivalent to ML with infinite learning of language transition probabilities, since languages were uniformly distributed in Test1. These results show that our PPM for transition probabilities learns this uniformity in Test1.

As shown in Figure 6, PPM clearly outperformed ML in Test2. This was because ML has no way to learn the transition probability, which was biased with the major language being used 90% of the time. This shows that the introduction of language transition probabilities accounts for higher performance. Interestingly, ML falls below the baseline case when more than three languages were used in Test2, a situation that has rarely been considered in previous studies. This suggests that language detection using only ML requires large corpora for learning to select one appropriate language, and that this requirement can be alleviated by using PPM.

Another finding is that the detection accuracy depends on the language set. For example, the accuracy for language sets consisting of both French and English tended to be lower than for other language sets due to the spelling closeness between these two languages. For example, the accuracy for test data consisting of 90% English, 5% French and 5% German was 94.4%. This is not surprising since the detection was made only *within* a token (which corresponds to a word in European languages): naturally there were many words whose language was ambiguous within the test set. In contrast, high accuracies were obtained for test sets consisting of languages more different in their nature. We obtained 97.5% accuracy for test data consisting of 90% English, 5% Finnish and 5% Turkish; this accuracy was higher than the average for all test sets.

4.2 Number of Control Actions

The second evaluation was done to compare the number of control actions needed to switch languages with TypeAny and with a conventional method. As mentioned in §1, three types of *keyboard actions* are used when entering text. Our work only concerns the control action to switch language, though, and the comparison in this section focuses on this type of action.

[7]E.: English, J.: Japanese and C.: Chinese.

[8]http://en.wikitravel.org/

[9]http://en.wikipedia.org/

Table 1: Articles Used in the Decrease Test

article	Article 1	Article 2
Foreign tokens	286	55
Total tokens	1725	5100
Inclusion ratio	16.6%	1.1%
languages	E., J.	E., J., C.[7]
content	Introduction of Japanese phrases for traveling	About tofu (bean curd)
Source	Wikitravel [8]	Wikipedia [9]

Table 2: Required Number of Control Actions

		Article 1	Article 2
Conventional Number of switches		572 (100%)	110 (100%)
Ours	**Failure A**	2.8%	3.6%
	Failure B	1.6%	2.7%
Total Failures		4.4%	6.3%
Decrease		95.6%	93.6%

This evaluation was done using two samples of actual multilingual text collected from the Web. The features of these samples are shown in the top block of Table 1. In both cases, the major language was English.

For each of these articles, the number of control actions required with the conventional method and with TypeAny was measured. The conventional method requires a control action every time the language switches. For TypeAny, control actions are required only to correct language detection failures. In both cases, the action required to switch languages or correct the language was counted as one action.

For the language model, the output probabilities were first trained using the 100-Kbyte monolingual corpora collected for the previous evaluation. The transition probabilities were not trained beforehand; i.e., the system initially regarded the languages to be uniformly distributed. Since this experiment was intended to simulate a realistic case, both output and transition probabilities were trained on-line using PPMC while the text was entered. Here, both n_{max} and k_{max} were set at 5.

The results are shown in Table 2. First, some detection errors occurred for Article 2 because “tofu” was detected as Japanese at the beginning of entry, even though it was used as an English word in the original text. As noted at the end of §2, such loan words can cause errors. However, since our system uses PPM and learns on-line, our system learned that “tofu” had to be English, and such detection errors occurred *only* at the beginning of the text.

Consequently, there was a substantial decrease in the number of necessary control actions with TypeAny, over 93%, for both articles. An especially large decrease was observed for Article 2, even though the text was almost all in English (98.9%). There was only a small increase in the incidence rate of **Failure B** for Article 2, so the total decrease in the number of required actions was still large, putting to rest the concern discussed in §2. These results demonstrate the effectiveness of our approach.

5 Conclusion

TypeAny is a novel multilingual text input interface in which the languages used for entries are detected automatically. We modeled the language detection as an HMM process whose transition probabilities are estimated by on-line learning through the PPM method.

This system achieved language detection accuracy of 96.7% in an evaluation where it had to choose the appropriate language from among three languages with the major language accounting for 90% of the sample. In addition, the number of control actions required to switch IMEs was decreased by over 93%. These results show the promise of our system and suggest that it will work well under realistic circumstances.

An interesting objection might be raised to the conclusions of this study: some users might find it difficult to watch the locale window all the time and prefer the conventional method despite having to work with a large number of key types. We plan to examine and clarify the cognitive load of such users in our future work.

References

B. Alex, A. Dubey, and F. Keller. 2007. Using foreign inclusion detection to improve parsing performance. In *Proceedings of EMNLP-CoNLL*, Prague, Czech, June.

B. Alex. 2005. An unsupervised system for identifying English inclusions in German text. In *Proceedings of the ACL Student Research Workshop*, pages 133–138, Ann Arbor, Michigan, June. Association for Computational Linguistics.

T. C. Bell, J. G. Clear, and I. H. Witten. 1990. *Text Compression*. Prentice-Hall, New Jersey.

C. Biemann, G. Heyer, U. Quasthoff, and M. Richter. 2007. The Leipzig corpora collection - monolingual corpora of standard size. In *Proceedings of Corpus Linguistics*, Birmingham, United Kingdom, July.

Z. Chen and K. Lee. 2000. A new statistical approach to Chinese input. In *The 38th Annual Meeting of the Association for Computer Linguistics*, pages 241–247, Hong Kong, October.

I. S. MacKenzie and K. Tanaka-Ishii. 2007. *Text Entry Systems —Mobility, Accessibility, Universality–*. Morgan Kaufmann.

C. D. Manning and H. Schuetze. 1999. *Foundations of Statistical Natural Language Processing*. MIT Press.

J.-C. Marcadet, V. Fischer, and C. Waast-Richard. 2005. A transformation-based learning approach to language identification for mixed-lingual text-to-speech synthesis. In *Interspeech 2005 - ICSLP*, pages 2249–2252, Lisbon, Portugal.

K. N. Murthy and G. B. Kumar. 2006. Language identification from small text samples. *Journal of Quantitative Linguistics*, 13:57–80.

B. Pfister and H. Romsdorfer. 2003. Mixed-lingual analysis for polyglot TTS synthesis. In *Eurospeech*, pages 2037–2040, Geneva, Switzerland.

K. Tanaka-Ishii. 2006. Word-based text entry techniques using adaptive language models. *Journal of Natural Language Engineering*, 13(1):51–74.

W. J. Teahan, Y. Wen, R. MacNab, and I. H. Witten. 2000. A compression-based algorithm for Chinese word segmentation. In *Computational Linguistics*, volume 26, pages 375–393.