Ubiquity: Designing a Multilingual Natural Language Interface

Michael Yoshitaka Erlewine
Mozilla Labs
650 Castro Street
Mountain View, CA 94041
mitcho@mitcho.com

ABSTRACT

This paper describes the design and implementation of Ubiquity, a multilingual textual interface for the Firefox browser developed at Mozilla Labs. The Ubiquity interface facilitates rapid information retrieval and task execution in the browser, leveraging existing open web APIs. The importance of offering equivalent user experiences for speakers of different languages is reflected in the design of Ubiquity's new natural language parser, described here. This paper also aims to advocate the further development of equipotent multilingual interfaces for information access.

Categories and Subject Descriptors

H.1.2 [**Models and Principles**]: User/Machine Systems—*human information processing*; H.5.2 [**Information Interfaces and Presentation**]: User Interfaces—*natural language*; I.2.7 [**Artificial Intelligence**]: Natural Language Processing—*language parsing and understanding*

General Terms

Design, Experimentation, Human factors, Languages

1. INTRODUCTION

Language continues to be one of the greatest barriers to open information access on the internet. The participation of ever more diverse linguistic communities on the web has not only created great linguistic divides in web content, but has also naturally resulted in a multitude of disparate tools created within each community, leaving such projects less able to benefit from each others' innovations. While much effort and increased attention have been devoted to the development of multilingual corpora and resources, less attention has been given to guaranteeing that users with different linguistic backgrounds can use the same quality tools to access that information. As part of Mozilla's goal to make the internet experience better for all users [8], Ubiquity aims to bring a new form of interactivity into the browser which treats user input in different languages equally. Ubiquity offers a platform for rapid information access, with no languages treated as second-class citizens.

The desire of users to access the internet using an interface in the language most natural to them is reflected in Mozilla's latest Firefox browser release which shipped in over 70 languages, each localized by a team of volunteers. The goal of fulfilling this desire is particularly pertinent—and challenging—in the case of a natural language interface.

Ubiquity was born of the Humanized Enso product (http://www.humanized.com/enso/), but is now an open-source community project, with dozens of contributors and active testers. It is available for download at http://ubiquity.mozilla.com and can be installed on the Firefox browser. Similar popular text-based command interfaces which are overlaid on GUI include Quicksilver (http://www.blacktree.com) and GNOME Do (http://do.davesbd.com/), but neither of them attempts a natural language syntax, nor do they support localization of their parser and keywords.

SIGIR Workshop on Information Access in a Multilingual World July 23, 2009 Boston, Massachusetts USA

2. TOWARDS A NATURAL INTERFACE

2.1 Features of a Natural Syntax

The lead of Ubiquity development Aza Raskin argues in his 2008 ACM *interactions* paper that text-based interfaces can be more humane than overextended graphical interfaces [10].[1] Graphical interfaces are easy to learn and apply for concrete tasks but do not scale well with additional functionality and lack the precision required to communicate abstract instruction. While numerous text-based computer interfaces exist, they have been deemed too difficult for lay users. Raskin argues that textual interaction does not entail these difficulties per se; rather, they are products of their oft-times stilted grammars. In reconsidering the text-based interface, ease and familiarity built into the interface are key. A subset of natural language is thus a clear winner.

Many programming and scripting languages—themselves interfaces to instruct the computer—make use of keywords inspired by natural languages (most often English). Many simple expressions neatly mirror a natural language (1a) but more complex instructions will quickly deviate (1b).

(1) a. `print "Hello World"` (Python)
b. `print map(lambda x: x*2, [1,2,3])`

[1]The term "humane" is used in this paper to describe human-computer interfaces which are "responsive to human needs and considerate of human frailties" [12] (see also [11]).

One valiant effort to facilitate near-natural language instruction has been AppleScript, which enables complex English-like syntax (as in 2) and originally was planned to support similar Japanese and French "dialects."

(2) `print pages 1 thru 5 of document 2` (AppleScript)

As a full-featured scripting language, however, more complex expressions push beyond the natural language metaphor and introduce their own idiosyncrasies. Bill Cook, one of the original developers of AppleScript, notes "in hindsight, it is not clear whether it is easier for novice users to work with a scripting language that resembles natural language, with all its special cases and idiosyncrasies" [5]. Raskin notes that this is precisely what must be addressed in designing a humane text-based interface: "if commands were memorable, and their syntax forgiving, perhaps we wouldn't be so scared to reconsider these interface paradigms" [10].

In designing an internationalizable natural language interface, we can conclude that it is not enough to use natural language keywords and mimic its syntax. The grammar must never conflict with a user's natural intuitions about their own language's syntax—a goal I call *natural syntax*. While a user can't expect such an interface to understand every natural language command, a good rule of thumb is that multiple natural alternatives for a given intent are interpreted in the same way. For example, consider the examples (3) in Japanese, a language with scrambling.[2]

(3) a. 太郎に ボールを 投げろ
Taro-ni ball-o nagero
Taro-DAT ball-ACC throw-IMPER

b. ボールを 太郎に 投げろ
ball-o Taro-ni nagero
ball-ACC Taro-DAT throw-IMPER

Both sentences are valid expressions for the command "throw a ball to Taro." An interface with a natural syntax must understand either both of these inputs or, if for example the interface does not understand the verb *nagero*, neither of them. To understand one but not the other goes against the tenet of natural syntax.

2.2 Commands in Ubiquity

Ubiquity actions are requests for actions or information, corresponding functionally to the formal clause type of "imperative" [9], although they may manifest in forms traditionally characterized as "imperative," "infinitive," or "subjunctive," depending on the language [7]. No vocative is entered as the addressee is always the computer, nor do we handle negation,[3] leaving Ubiquity input to simply be composed of a single verb and its arguments (if any). Some example English Ubiquity actions include:

(4) a. `translate hello to Spanish`—previews the text "hola." On execution, inserts the text "hola" in the active text field.

[2]Note that the Japanese examples are given with spaces between words to facilitate the glosses. Japanese does not normally place spaces between words.

[3]When negative imperative meanings are desired, verbs which lexicalize the negative meaning are chosen, e.g. `prevent`, `turn off`, etc.

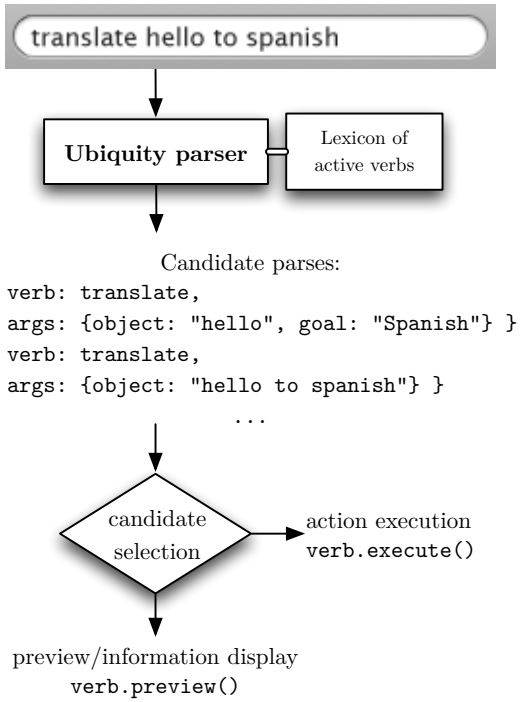

Figure 1: Schematic diagram of user interaction with Ubiquity.

b. `email hello to John`—on execution, composes a new email to contact `John` with message body "hello."

c. `map Tokyo`—previews a map of Tokyo using the Google Maps API. The image can then be inserted into the page.

Verbs are written in JavaScript. Each verb may specify a `preview()` method which displays some information to the user or gives a preview of the action to be executed and an `execute()` method which carries out the intended action.

In order to avoid ambiguity, a list of possible parses is presented to the user for confirmation before execution. Suggestions give a visual indication of the parsing. A scoring mechanism is used to bring more likely candidates to the top, taking user input and browser use habits into consideration.

3. ADDRESSING THE NEEDS OF MULTILINGUAL ACCESS

With the requirements and goals of the project as laid out in section 2, certain architectural choices were made in designing the parser in order to support multiple languages equipotentially. In this section I will review the unique features of our parser and platform which enable equal information access and rapid localization.

3.1 Identifying Arguments by Semantic Role

Ubiquity commands' ease of creation is a great strength for the platform, with many contributors around the world creating and sharing their own verbs as well as writing new verbs for personal use. In order to let users of different languages benefit equally from the platform, however, there is a

Figure 2: Equivalent Ubiquity queries in three languages: English, French, and Japanese. Note that the two suggestions returned in each case are semantically different, reflecting the ambiguity between translating "hello to span" into an as yet unspecified language and translating "hello" into the Spanish language.

need to internationalize the verbs themselves. Verbs include some strings which must be translated, such as the verb's name, but they also include a specification of the type of arguments it accepts, known as the *syntactic frame* of the verb. For example, in English an `email` verb may take a direct object and a recipient introduced by the preposition "to," while a `translate` verb may take an arbitrary direct object, a goal language marked by "to," and a source language marked by "from."

In order to facilitate this localization, we chose to let verbs specify their syntactic frames using abstract semantic roles such as `object`, `goal`, `instrument`, `position`, etc. which are morphosyntactically coded in most languages.[4] For example, suppose an English-speaking contributor wrote a verb called `move`, whose action was to move an object from one location to another. Its syntactic frame could be specified as follows, where `physical_object` and `location` are noun types which specify a class of arguments and their associated semantic forms.

```
{ object: physical_object,
  source: location,
  goal:   location }
```

The command author could then use this command in English, entering input such as (5). The parser recognizes the English prepositions "to" and "from" as corresponding to the `goal` and `source` roles (underlined below), and recognizes the unmarked argument as an `object`.

(5) `move truck from Paris to Beijing`

(6) トラックをパリから北京へmove

However, given a set of localized noun types, the exact same command code could be used with the Japanese parser by entering the input (6). Here, the parser recognizes that the postpositions "を," "へ," and "から" mark `object`, `goal`, and `source` arguments, respectively. The only manual localization required for the `move` command, then, is the translation of the verb name "move" itself. As shown by this example, the specification of arguments using abstract semantic roles supports the rapid and, indeed, semi-automatic localization of commands, ensuring that users of all languages benefit from individual improvements to the Ubiquity platform's functionality.

[4]In our use, "semantic role" is equivalent to the related notions of "grammatical function" and "thematic relation." An inventory [2] was chosen based on [6] and subsequent cross-linguistic work.

Table 1: Argument-first Suggestions

Sample argument parses	suggested verbs
`{ object:…, goal:… }`	email, send
`{ object:…, instrument:… }`	search, look up
`{ object:…, source:…, goal:… }`	move, translate

3.2 Argument-first Suggestions

In parsing Ubiquity input, a key task is the identification of the verb, if any. In many languages the verb naturally comes at the beginning of the sentence (see English examples in 4). In this case, as the verb can be identified early in the user input, we can then annotate the candidate parses with information on the missing arguments to guide the user in entering the rest of their input (see figure 2). However, not all languages enter the verb first in commands. Some languages are strictly verb-final (e.g. Japanese), while in some other languages (e.g. German, Dutch, Modern Greek) it is equally valid to express commands using the imperative or subjunctive verb form at the beginning of the sentence, or using the infinitive at the end of the sentence.

Rather than being discouraged by this conundrum, thought was given to how we can leverage the unique qualities of verb-final (or argument-first) input to make a more humane and supportive interface. As different verbs in our lexicon specify different syntactic frames, by parsing arguments and identifying semantic roles in the input before the verb is known, we can then suggest verbs to the user which match that particular argument structure (see examples in table 1 of some such suggestions). This smart argument-first suggestion aids in command discoverability by suggesting verbs for a given target which the user may not have known

existed. This approach crucially takes advantage of the argument-first input and offers unique value and increased usability to users with verb-final languages.

Note also that the suggestion of verbs based on argument-only input may also be useful for regularly verb-initial langauges such as English. Studies of general interactive systems concur that *noun-verb* (or *object-action*) paradigms result in error reduction, increased speed, and better reversibility during input [11]. For these reasons, argument-first suggestions are supported in Ubiquity for all languages equally.

3.3 Minimal Language Descriptions

The Ubiquity parser attempts to make as much of its parser algorithm universal as is practical, taking a page from the Principles and Parameters framework in generative linguistics.[5] A single universal parser was designed, with settings for different languages built on top of that base [1]. The settings for each language are written in JavaScript and range from ten to thirty lines of code. Various hooks exist in the code for language-specific processing when necessary, but the majority of the language settings are simply lists of special lexical items such as the prepositions or postpositions in a language. In this way, for the limited range of data which constitute Ubiquity input, the very difficult problem of writing a language-specific parser is reduced to little more than some native speaker consultation and string translation.

4. EVALUATION METRICS

As an open-source community project, the success of Ubiquity must be evaluated in terms of developer involvement as well as user adoption. The design choices laid out here are intended to lower the barrier of contributing to Ubiquity's localization, and the level of localizer engagement is a direct reflection of the facility or difficulty of contributing to individual parser language setting files and localizations. An initial survey of this metric can be interpreted as optimistic, with language settings having been written for ten languages and Ubiquity's built-in commands having been completely localized into three of those languages as of this writing, even before widespread public release of the new parser.

Approximate user adoption rates can be calculated based on download and update counts, though this does not currently directly reflect active usage of Ubiquity nor give us much insight into its different use cases. Current user adoption is expected to be limited by the fact that previous versions of Ubiquity were almost exclusively suited for English use, and we expect a slow uptick in usage and interest by users in other languages which we are beginning to support. Future collaboration with the Mozilla Labs' Test Pilot project [4] to collect anonymous user behavior data in Ubiquity is also being planned [3]. This data will help yield more accurate usage statistics, including usage breakdowns by language, as well as yield valuable information on parser accuracy and user interaction patterns.

[5]It is worth noting that this architectural choice also complemented the object-oriented architecture and Don't Repeat Yourself design goals of the project.

5. CONCLUSIONS

Further globalization of the web without serious consideration of multilingual information access could spur the further fragmentation of information and ideas. Equal access to information will require more than just cross-language search and retrieval systems, but also universal interfaces which are designed for rapid localization and treat all languages equally.

In this paper I outlined some of the design features of Ubiquity's interface and natural language parser which bring the system closer to this goal. Formal approaches to the study of language were applied in order to design a system which can be extended to a wide range of languages. As of this writing, settings for ten languages have been written for Ubiquity, while the community process of setting technical standards for verb and noun type localization is in progress.

Ubiquity is quickly becoming a compelling text-based interface for both advanced and casual users. The forgiving "natural syntax" philosophy and the smart suggestion of verbs and arguments to the user help make Ubiquity a humane interface which cooperates with users rather than confounds them. These qualities make Ubiquity a natural choice of interface platform for multilingual and cross-languge information access applications.

6. ACKNOWLEDGMENTS

Thank you to comments from Aza Raskin and Jonathan DiCarlo at Mozilla and audiences at BarCamp Tokyo; Chuo, Waseda, and Keio Universities; Tokyo 2.0; Tokyo Institute of Technology; as well as comments on related material on my blog (http://mitcho.com/blog/).

7. REFERENCES

[1] Parser 2 - MozillaWiki. https://wiki.mozilla.org/Labs/Ubiquity/Parser_2 .

[2] Semantic Roles in Parser 2 - MozillaWiki. https://wiki.mozilla.org/Labs/Ubiquity/Parser_2/Semantic_Roles.

[3] Ubiquity Roadmap - MozillaWiki. https://wiki.mozilla.org/Labs/Ubiquity/Roadmap.

[4] C. Beard. Introducing test pilot. https://labs.mozilla.com/2008/03/introducing-test-pilot/.

[5] W. R. Cook. Applescript. In *The Third Conference on the History of Programming Languages*, 2007.

[6] C. J. Fillmore. *Types of lexical information*. Reidel, Dordrecht, 1969.

[7] S. Iatridou. De modo imperativo. Lecture notes, ENS, Paris, September 2008.

[8] Mozilla Foundation. The Mozilla manifesto, v0.9. http://www.mozilla.org/about/manifesto.en.html .

[9] P. Portner. The semantics of imperatives within a theory of clause types. In *Proceedings of Semantics and Linguistic Theory*, volume 14, 2005.

[10] A. Raskin. The linguistic command line. *interactions*, 15(1):19–22, 2008.

[11] J. Raskin. *The Humane Interface: New Directions for Designing Interactive Systems*. Addison-Wesley, 2004.

[12] A. Varma. Humanized > Why 'humane' is a better word than 'usable'. http://humanized.com/weblog/2006/06/01/why_humane_is_a_better_word_than_usable/.