A Framework for Multilingual Searching and Meta-information Extraction

Susumu Shimizu, Takashi Kambayashi, Shin-ya Sato, Paul Francis
{ shimizu, kam, sato, francis}@ingrid.org
Software Laboratory, Nippon Telegraph and Telephone Corporation

---

Abstract

Due in large part to the popularity and global nature of the Web, multi-lingual issues in computers is finally beginning to attract serious attention, from users and developers alike. At the Software Labs in NTT, we are involved in a project that confronts multi-lingual issues in a big way. Namely, we are building software designed to self-configure a global distributed search infrastructure.

This paper describes how we have architected our system in order to handle multi-lingual issues ranging from character encoding translation to language detection to term weighting. It can serve both as a case study in the design of multi-lingual distributed systems, and as a general tutorial on multi-lingual issues. It also presents some surprising conclusions, such as the fact that the popular Unicode international character encoding is not the best one to use in our environment.

Contents

Introduction

Search engines have become one of the most important means of navigating the web. In addition to the well-known and heavily used "global" search engines, such as Alta Vista, many web sites are locally searchable. More and more search software is becoming available, and in some cases, good quality search engines such as Excite's EWS[Excite] and Sony's SSE[Sony] can be downloaded for free.

Along with the proliferation of search services and products, naturally, comes the need to be able to automatically exchange information easily. Currently, search services gather information from web servers with a so-called robot [Koster], which automatically follows HTML links and gathers web documents. It is often difficult, however, to extract useful information out of a document because of a lack formatting information in the document. What is needed is some kind of document *meta-information*---that is, well-formatted information, such as Author, Date, Format, and Language, that describes the document. Meta-information itself is of course nothing new, and pre-dates computers, in the form of for instance library

card catalogs, by many hundreds of years.

What is new is the need to have meta-information available in a standardized form, so that a wide variety of documents can be summarized by different software packages, but still indexed by a single search engine. This need is widely recognized as critical to the evolution of web searching, and there are a number of efforts to define standardized meta-information in the context of the web [Weibel, Guha, Chang].

One of the important distinguishing characteristics of web meta-information (versus more historical uses of meta-information, such as in libraries) is that the meta-information will largely be automatically generated. This lack of human oversight affects every aspect of the meta-information creation process. In some cases, certain meta-information creation is trivial. For instance, structured documents such as email [Crocker] and TeX [Knuth] already tag certain types of information, such as Author, Title, and so on. In other cases, creating exactly the "right" meta-information is for all practical purposes impossible. For instance, in the selection of key words for a document.

Another important aspect of the global web environment is that meta-information creation software can be expected to encounter many different languages. For instance, the robot of a global search service will encounter documents of many different languages, including computer languages.

Not surprisingly, virtually all aspects of meta-information creation are impacted by language---from the character code used, to the mechanics of parsing individual terms out of text, to the weighting of terms. Current efforts at automatic and general meta-information creation, however, do not pay much attention to multi-lingual issues [Hardy].

The authors of this paper are involved in a project, called Ingrid, to develop a distributed, fully automatic, global search infrastructure. In addition to doing the usual tasks of a "stand-alone" search engine---that is, gathering, parsing, and indexing documents---the search engines that participate in the distributed search infrastructure also exchange information with each other. This allows a searcher to find appropriate information on the search engines that have indexed that information without specific knowledge of the individual search engines.

The global scope and distributed nature of this problem brings us face-to-face with multi-lingual issues in a big way. With Ingrid, virtually anybody anywhere can publish a document, of any language, into the single global infrastructure. Because of the automatically configuring search infrastructure, any Ingrid search engine can potentially find itself exchanging information with any other search engine. As a result, we have had to take multi-lingual issues into account in virtually every aspect of our design and implementation.

This paper is about those multi-lingual issues. It describes how we have treated various multi-lingual issues as they have arisen in our design and implementation. As such, this paper does not focus on any single multi-lingual issue, but rather takes a broad view of multi-lingual issues in general. Nor does paper particularly present any new results or solve any specific unsolved multi-lingual problems, even though, taken together, our overall design can be considered as new work.

This paper is, in effect, a case study of how multi-lingual issues were dealt with in a particularly challenging information systems design. This paper is therefore useful to somebody dealing with similar design issues. It can also be read as a general tutorial on multi-lingual issues. In this latter case, the reader who does not have much experience with multi-lingual issues may occasionally come across a concept which is not adequately explained. In general, these can be glossed over without losing the gist of the paper.

Overview of the Ingrid Architecture

As mentioned briefly above, Ingrid is a distributed search infrastructure. The basic idea behind Ingrid is as follows.

Individual search engines exchange information about their local indexes. Thus, each search engine can not only answer queries about its own index, but can also refer queries to other search engines that may contain information relevant to the query. The result is that the totality of information across all search engines can be efficiently searched.

The (human) searcher does not need to have any knowledge about individual search engines. The search software accessed by the searcher makes a series of queries to multiple search engines on behalf of, and transparently to, the searcher. In addition, the administrator of an Ingrid search engine does not need to have any knowledge about other search engines. The software that exchanges information with other search engines does so automatically and transparently to the administrator.

Note, however, that both the searcher and the administrator can, if they desire, limit the interactions of their Ingrid systems to specific other Ingrid systems.

There are three major components to the Ingrid architecture, the *Publisher*, the *search engine*, and the *Navigator* (see Fig.1).

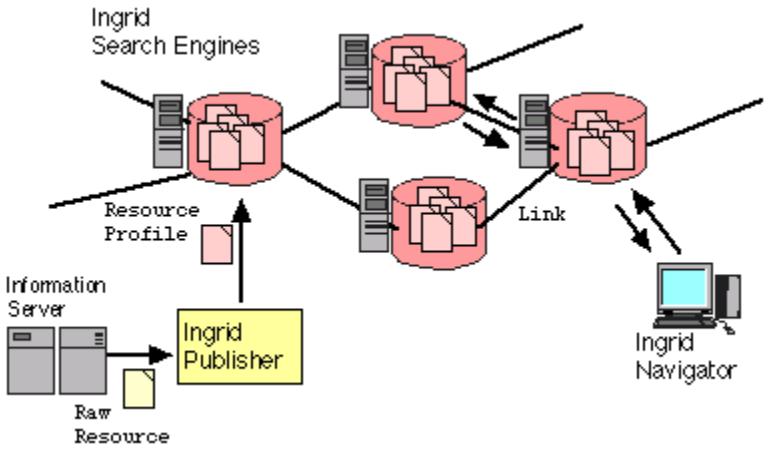


Fig.1) Ingrid architecture

There are also two basic units of information, the *resource profile* and the *link*. The Publisher gathers resources (documents on web servers, for instance), generates meta-information for them in the form of a resource profile, and gives them to the search engine. (Note that we use the terms resource and document interchangeably.)

Upon receiving a resource profile, the search engine indexes the resource profile, and exchanges information in the form of links with other search engines with similar resource profiles. Finally, when a searcher wants to find resources, he submits a query to a Navigator, which then makes a series of queries to search engines. Each search engine will tell the Navigator of 1) relevant resources that it has, and 2) other search engines that may have relevant resources. The Navigator dislays the final results to the searcher.

Ingrid's resource profiles consist of a list of (type: value) pairs, such as:

```
URL: http://www.ingrid.org/
Title: Ingrid Home Page
KeyWords: term1, term2, term3, ..., termN
```

This format is similar to that of other meta-information definitions [Weibel, Guha, Chang]. The exact syntax

is not particularly important to us, and we will follow whatever meta-information standard evolves. The main difference between the Ingrid resource profile and other meta-information is the fact that a short list of key words are mandatory for an Ingrid resource profile. They are needed by the search engines for the purpose of creating the appropriate links. This need to automatically pull key words out of text is in large part what forces Ingrid to deal with language.

An Ingrid link consists of a search engine identifier, a resource profile identifier, and a set of relevant terms taken from the list of key words in the resource profile and shared by the resource profiles on either end of the link:

```
(search engine ID: RP ID: term1, term2, ..., termN)
```

Selecting a Character Encoding

Though it basically goes without saying, we should note that all terms in the resource profiles, the links, and in the queries sent by Navigators, for a given language, must share a common character encoding (where examples of character encodings are ASCII [ANSI], ISO2022 [ISO2022], and Unicode [UNICODE]). If they contained different character encodings, then a query for a given term in one character encoding would not match the same term in a resource profile with a different encoding.

While it is possible to select an encoding on a per language basis, it makes a lot more sense to select a single international character encoding scheme and apply it across all languages. Given the growing acceptance of Unicode, it may surprise the reader that we did not choose Unicode as our common encoding scheme. Rather, we chose the Mule [Nishikimi] character encoding scheme.

A little background on Unicode and Mule. Mule is a variable-length encoding scheme where, roughly speaking, one byte is used for ASCII encodings, 2 bytes for ISO-8859 encodings, 3 bytes for ideograph based languages, and, less commonly, 4 or 5 bytes can be used with bi-directional languages. For the non-ASCII codes, the first byte of the code essentially identifies the character set used by the remaining bytes. Thus, Mule is a tag/value scheme. Unicode, on the other hand, is a fixed-length, two-byte encoding scheme where all characters come out of the same space. In other words, Unicode characters are not tagged.

The main problem with Unicode, when used in a global search-engine context, is that Chinese, Japanese, and Korean (CJK) ideographs share the same code space (Han Unification). Thus, if a Japanese searcher inputs a string for searching, it can equally match against its Chinese and Korean counterparts, which generally is not what the Japanese searcher would want. This is a serious problem for CJK users.

Another problem with Unicode, again mainly for CJK users, is that it doesn't contain enough code space to capture all ideographs, though it does capture most of them. To ease this problem Unicode makes available a part of the space for purely local use, but this is of no value for a global search scheme. Note that the above comments are with respect to Unicode. 32-bit-wide character set, such as ISO10646/UCS4 [ISO10646] won't have this problem. Given that 16-bit-wide character set solves most people's encoding problems, however, we cannot rely on 32-bit-wide character set becoming widely available.

The Navigator Architecture

The Navigator consists of three basic components (Figure 2). The I/O component takes input from the searcher, including terms specifying the search, and forwards the information to the search manager. The search manager, through a series of queries to search engines, obtains relevant resource profiles. These are sorted and filtered by the presentation component, which finally hands the results back to the I/O component. The I/O component then presents the results to the searcher.

Both the search manager and presentation components deal entirely in the common encoding Mule. Character encoding translation takes place in the I/O module. We have implemented I/O modules for Motif (X windows), web browsers, and Java. The Motif I/O modules is shown in more detail in Figure 2.

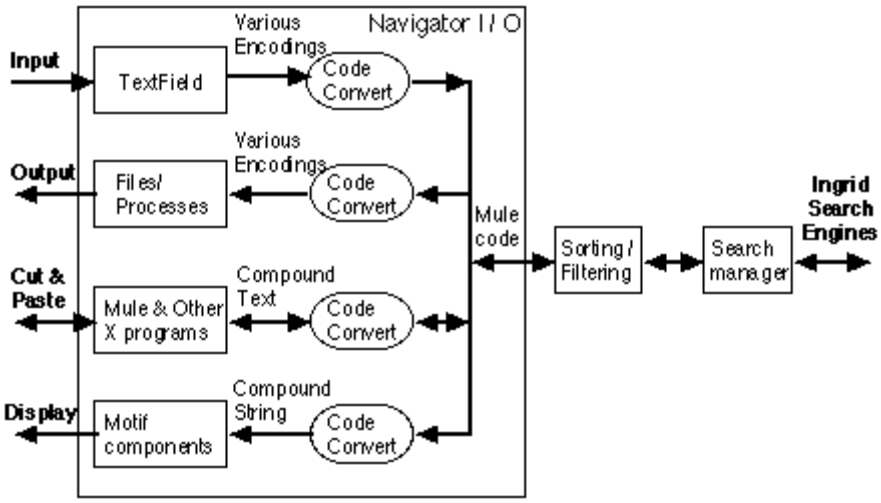


Fig. 2) Ingrid Navigator, Motif Version

The various translations required include Compound Text of the X window system [X11], and Compound Strings used by Motif [OSF]. This I/O module makes heavy use of the Mule library for encoding translation. Since Java uses Unicode as its character set [Java], for the Java I/O module we find ourselves in the rather odd position of translating between two international character sets (Mule and Unicode).

Multi-lingual I/O is a non-trivial problem. Each of the user interfaces we worked with requires different techniques. Further, we are subject to the limitations of the input modes of our interfaces. In most cases, such interfaces are limited to one character set at a time (with ASCII as an always-available alternative). Thus, we today don't have true multi-lingual typed input in our Navigator software. This is one area where Unicode should greatly improve things.

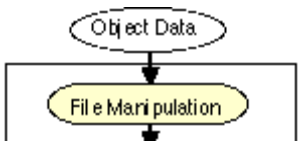
Although multi-lingual I/O is a difficult and important problem, it is also a common one, and there is little about our architecture that makes our experience with it significantly different from others'. Therefore, we don't discuss it in any more detail in this paper.

The Publisher Architecture

The bulk of multi-lingual issues confronted in the Ingrid project come from the Publisher. The Publisher must be able to parse and select key words for documents with arbitrary contents, in various file formats, with multiple encodings of different languages, including software languages.

The Publisher is largely patterned after the Essence package [Hardy] in the Harvest project [Harvest], particularly in its attempts towards modularity. The Harvest project, however, did not make any particular effort to accommodate multiple languages. To handle multiple languages, we have had to enhance the basic Essence architecture in a number of ways.

The basic operation of Essence is as follows (Figure 3).



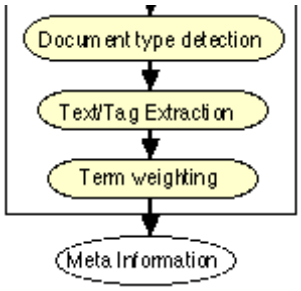


Fig.3) Current framework

First, it uncompresses/unbundles the input file into separate files (documents) if necessary. Then, it detects, where possible, the document types (such as postscript, HTML, README, and so on). It throws out document types that are of no indexing value (such as computer object files). Based on file type, it extracts the text (individual terms) and whatever tags (author, section headings, etc.) may exist. From the text and tags, key words are selected. The Essence code is designed to allow for easy insertion of custom file-type recognizers and text/tag extractors.

The Ingrid Publisher also does each of these things. However, in order to do so in the multi-lingual environment, it must deal with multi-lingual issues at almost every step.

Figure 4a shows the basic architecture of the Ingrid Publisher at the same level of detail as Figure 3.

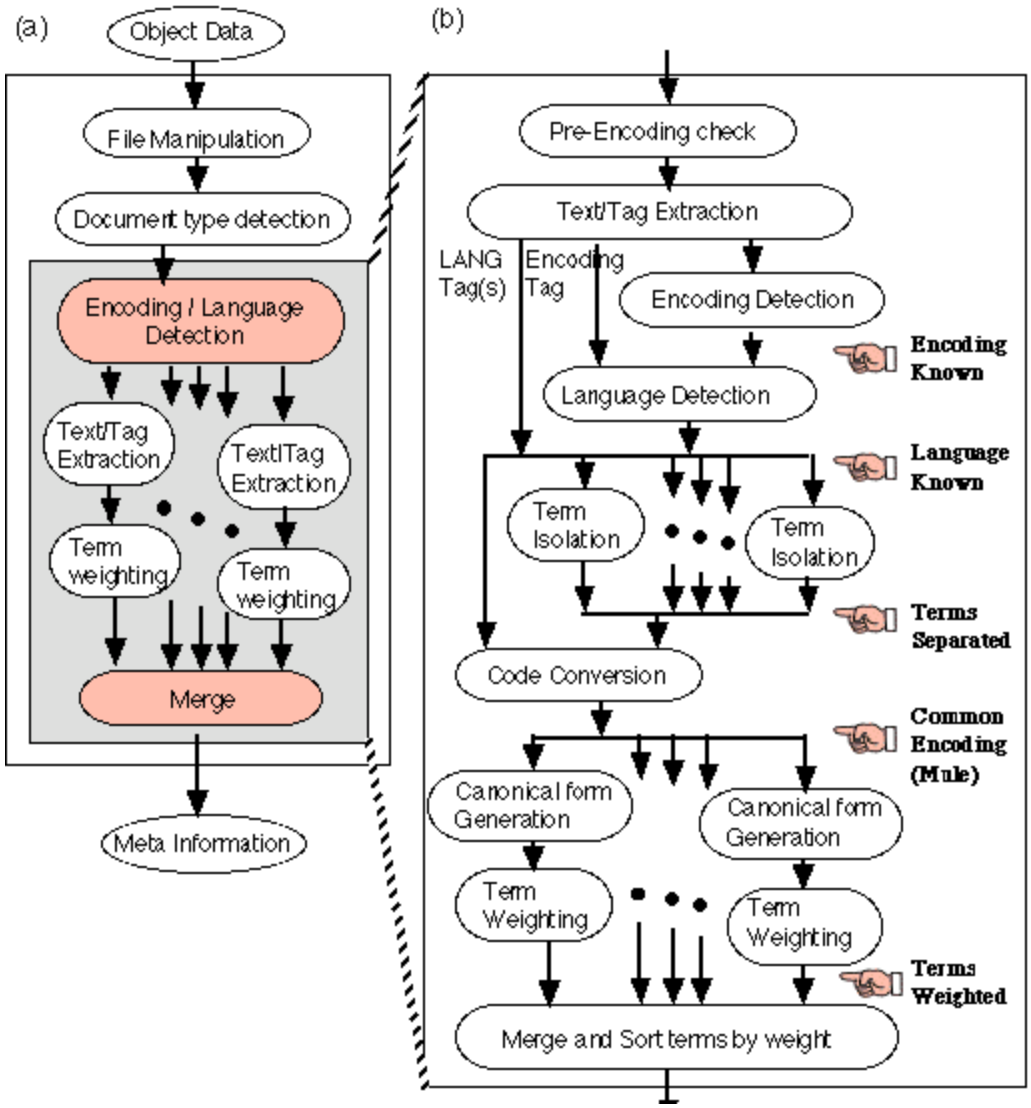


Fig.4) Multilingual framework

Here we see that, after document type detection, the Ingrid Publisher must identify the character encoding and languages of the text. Since a given document may be in multiple languages, the Ingrid Publisher detects language on roughly a per-paragraph basis. Different text/tag extraction and term weighting modules will be called based on the language detected. In the case of multiple languages, the per-language document meta-information will be merged back together to create a single resource profile.

Figure 4b shows in more detail how language and character encoding is handled. After the document type is determined, an initial check for encoding type is made in order to select the appropriate text/tag extraction software. In particular, JIS (Japanese Internation Standard) characters are each two bytes, but each byte comes out of the ASCII code space. Without special handling, the JIS characters can be confused as, for instance, document formatting commands, by software that is not sensitive to JIS encodings.

Text/Tag Extraction

During text/tag extraction, the language or encoding types may be detected from the tags themselves. For instance, in Japanese postscript, a font name of 'Ryumin-Light-H' implies that the text is encoded in JIS. Or, Japanese Mail messages with MIME [Borenstein] headers have a MIME sub-type of:

```
Mime-Version: 1.0
Content-Type: text/plain; charset=iso-2022-jp
```

This implies the messages are encoded with ISO2022-JP escape sequences. Where language or encoding tags are not available, the text is fed through software that is able to detect encoding or language (more on this later).

Term Isolation

Once language and encoding are known, it may be necessary to feed the text through software that can extract the individual terms from the text. This is necessary for languages, such as Chinese and Japanese, that do not contain white space between individual terms. Term isolation is not a trivial task, requiring that the software understand the language's grammar and has a complete dictionary. Clearly term isolation is a language specific task (different software modules for different languages).

Generate Canonical Forms

After terms have been isolated, the individual terms are put into what we call their canonical form. Since the exact canonical form varies from language to language, generating the canonical form is a language specific task. The canonical form of a term can be loosely defined as the simplest form the term can take. For English, the canonical form for a given term is the shortest form the term takes in the document. In addition, the canonical form is completely lower-case.

For instance, say an English document contains, at various points in its text, the following terms: "Route", "routing", and "routable". The three terms are recognized by our software as representing the same word because the share a common stem ("rout"). The canonical form is "route", because "Route" is the shortest of the three, and further it is made lower case. Each of these terms, then, is represented as a pair of terms, {route|Route}, {route|routing}, and {route|routable}. These composite terms are what is transmitted from the Publisher to the search engine.

The basic reason that the Ingrid Publisher must create the canonical form is that the Ingrid search engine is designed to have zero language dependencies. The search engine is the common "infrastructure" component of Ingrid, so it is basically impossible to put language smarts there. Thus, unlike most search engines, the Ingrid search engine per se does not understand the notion of, say, matching on both upper and lower case English characters. Therefore, for the above example, the publisher sends the lower case version to the search engine. The search engine indexes both the canonical and actual form of each term.

Term Weighting

After term canonicalization, the terms are weighted so that key-words can be selected. Term weighting, like term isolation and term canonicalization, is a language specific task. We use the term frequency/inverse document frequency method of term weighting [Salton]. For English and Japanese (the only two language modules we have yet implemented) we calculate term weights using stemmed terms, and we filter for a short list of stop-words in advance. (Stop words are those words, such as "the", "a", and "and" in English, that are of virtually no indexing value, and so are filtered out of the text.) In addition, we take into consideration how often each term appears in document headings and enhance weights accordingly.

Various term weighting schemes have been developed (most search engines have one of their own), and in principle can be used in place of ours. However, it is necessary that the final merging and sorting be able to directly compare the weights assigned to terms from different languages. Therefore, at a minimum, weights calculated by different schemes should be normalized to fall within the same range. Preferably, the same scheme is used for each language.

Code Translation

The language specific modules can operate under any encoding scheme that is appropriate. For instance, the Japanese term isolation package we use, called JUMAN [JUMAN], requires EUC encoding for input. Our term weighting modules work in Mule. Other term weighting modules, if used, are more likely to operate in the most common encoding for that language (for instance, ASCII for English). Thus, for the language specific modules, it may be necessary to translate the encoding before feeding it to the module. This is not shown in Figure 4b.

In Figure 4b, we show the final code conversion into Mule taking place before the term weighting modules because that is how our software currently operates. However, if a term weighting scheme requiring a different encoding is used, we can just as easily convert into Mule after term weighting.

Note also that canonicalization could also take place after term weighting. However, in our term weighting we recognize terms with common stems where possible. Thus, it is convenient for us to do the canonicalization at the same time we parse for common stems.

Various Issues, Our Solutions, and Open Problems

Encoding Detection

To detect encoding in documents at the Publisher, we use the COCO software from the Mule library. This software has a few limitations. First, it currently does not detect Unicode, although it is under development.

Second, it does not distinguish three different EUC text (EUC-china [Lunde96], EUC-japan [Lunde93] and EUC-korea [Lunde96]). This is because these codes come out of the same code space, and so from a coding perspective look identical. Thus, the user must select as a compile option which of the three should be

selected. It also sometimes mistakes EUC-japan for shift-JIS [Lunde93] and vice versa. The reason is similar---both come from the similar code space---though COCO does make an effort to distinguish by looking for certain codes that appear frequently in Japanese.

Both of these problems can be solved through the language recognition techniques mentioned below. We simply have to make some rather odd stop word lists. For instance, a stop word list for Japanese using EUC-china would be required to recognize where EUC-japan is mistaken for EUC-china.

Language Detection

Detecting language basically involves scanning the text for certain characteristics that are frequent in that language. In the Publisher, we look for common term such as, in English, "the", "a", "and", "to", and so on. These are the same terms that would normally be found in a stop-word list.

This technique works less easily for non-white space languages since the terms are not distinct (and cannot be made distinct through the term isolation software until after the language is recognized). Other researchers have proposed an n-gram technique [Cavnar], where rather than scanning for frequent individual terms, a profile of the frequency of every string of n characters, regardless of whether it falls on a word boundary, is made for each language as a whole. Then, a similar profile is made for the document. The document's profile is compared against the languages' profiles, and the best matching one is picked. This technique is more general than the stop-word technique, but also more costly in terms of memory, processing, and profile preparation.

Since, with a few exceptions (see the previous section), we know the encoding at the time we do language detection, we can take advantage of that fact to simplify language detection. For instance, if the encoding is ISO-8859 part 5 (Latin/Cyrillic), and the "non-ASCII" subset of the code is in heavy use, the language is limited to one or more of Russian, Ukrainian, Bulgarian, etc. By taking advantage of this, we feel we will not need to use n-grams.

However, we also must be careful. Sometimes pure ASCII can be used for non-English languages with a Latin character set, such as French, because for instance the writer did not have a French keyboard or input system at the time. Thus we must be fairly broad in our initial assumptions about what languages to look for based on encoding.

Term Isolation for Non-White Space Languages

As mentioned above, term isolation in non-white space languages is difficult. The package we are using [JUMAN] does a relatively poor job, mainly because of its incomplete dictionary. In particular, it often incorrectly chops up longer terms into its shorter components.

To make up for this, our software does a certain amount of post-processing in the form of phrase detection. That is, we scan the document for term pairs that appear multiple times in the document, and assume that they are a single term. Preliminary test show that this technique works surprisingly well, though naturally it sometimes glues together two terms that should remain separate.

Another technique we plan to experiment with to ease this problem is to intentionally glue together every pair of terms in the search engine to insure that the full terms are properly indexed, and then use partial string matching to catch the case where two terms have been incorrectly glued together. For instance, if the output of term isolation is T1, T2, T3, T4, we will glue together each term pair to produce T1-T2, T2-T3, and T3-T4. If the user searches for, say, just T3, we will treat it in the search engine as searching for any term starting with T3, and match on the T3-T4 composite term.

Future Work

In general, our multi-lingual architecture remains inadequately tested. So far, we have incorporated only two languages into our system, Japanese and English. These two languages are different enough that we feel we have encountered many if not most of the multi-lingual problems that will effect our architecture. Still, we need to add languages to flush out the architecture further. In particular, we want to work with languages that build compound characters or have right-to-left ordering, such as Arabic.

In general, we are looking for people who are interested in using the Ingrid system to index their language, and can contribute, where necessary, code modules to our Publisher.

Acknowledgements

The authors would like to acknowledge the efforts and help of Kazuhiro Kazama, who is the implementor of the Navigator I/O parts and foremost multi-lingual expert in the Ingrid group.

References

[ANSI]
ANSI/NISO X3.4-1986, *Coded Character Set - 7-bit American Standard Code for Information Interchange*

[Borenstein]
Borenstein, N., Freed, N., *MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies*, RFC1521 ftp://ds.internic.net/rfc/rfc1521.txt

[Cavnar]
Cavnar, W., Trankle, R. , *N-gram Based Text Categorization* Proceedings of the Third Annual Symposium on Document Analysis and Information Retrieval," April 1994, pp 161-169

[Chang]
Chang, K., Garcia-Molina, H., et al., *STAIRS ver1.0*, http://www-diglib.stanford.edu/diglib/WP/PUBLIC/DOC82.html

[Crocker]
Crocker, D. H. *Standard for the format of arpa internet text messages*, RFC822 available from ftp://ds.internic.net/rfc/rfc822.txt

[Excite]
Excite, *Excite for Web servers*. download page is http://www.excite.com/navigate/home.html.

[Guha]
Guha, R.V. *Meta-Content Format*, Apple Computer Technical Draft, http://mcf.research.apple.com/hs/mcf.html

[Hardy]
Hardy, D. R., Schwartz M. F. *Customized Information Extraction as a Basis for Resource Discovery*., ACM Trans. on Computer Systems(TOCS) Vol.14 Number.2

[Harvest]
Bowman, C. M. et.al., *The Harvest Information Discovery and Access System*, in Proc. of the 2nd Intl. WWW Conf. pp763-771, Chicago 1994

[ISO10646]
ISO/IEC 10646-1:1993, *International Standard, Information technology - Universal Multiple-Octet Coded Character Set(UCS)*

[ISO2022]
ISO/IEC 2022:1994, *International Standard, Information technology - Character code structure and*

extension techniques

[Java]
Gosling, J., Joy, B., Steele, G. *The Java Language Specification*. http://java.sun.com/doc/language_specification/index.html

[JUMAN]
JUMAN Information, http://pine.kuee.kyoto-u.ac.jp/nltools/juman-eng.html

[Knuth]
Knuth, D. E. *The TeXbook*, Addison-Wesley, 1984.

[Koster]
Koster, M. *The Web Robots Pages.* http://info.webcrawler.com/mak/projects/robots/robots.html

[Lunde93]
Lunde, K., Understanding Japanese Information Processing, O'Reilly & Associates, Inc. 1993

[Lunde96]
Lunde, K., CJK.INF, V2.1(July 12,1996) available at ftp://ftp.ora.com/pub/examples/nutshell/ujip/doc/cjk.inf

[Nishikimi]
Nishikimi, M., Handa, K., Tomura, S. *Mule: MULtilingual Enhancement to GNU Emacs*. in Proc of INET'93, available at ftp://etlport.etl.go.jp/pub/mule/papers/INET93.ps.gz

[OSF]
Open Software Foundation, *OSF/Motif Programmer's Guide Release 1.2*, Prentice-Hall

[Salton]
Salton, G. and McGill, J. Introduction to Modern Information Retrieval, McGraw-Hill,1986

[Sony]
Sony, *SonyDrive Search Engine*. http://www.sony.co.jp/Search/.

[UNICODE]
The Unicode Consortium, *The Unicode Standard Version 2.0* Addison-Wesley Publishing, 1996

[Weibel]
Weibel, S., et al., *OCLC/NCSA Metadata Workshop Report*, The March 1995 Metadata Workshop, http://www.oclc.org:5046/conferences/metadata/dublin_core_report.html

[X11]
X Consortium Standard, *Compound Text Encoding*. comes with X11R6 distribution package