JAVA AND ITS ROLE IN NATURAL LANGUAGE PROCESSING AND MACHINE TRANSLATION

Tim Read[1], Elena Bárcena[2] & Pamela Faber[2]

[1]Dep. of Experimental Psychology [2]Dep. of Translation, University of Granada
[1]Campus Universitario de Cartuja [2]Puentezuelas 55, 18010 Granada, Spain
tread@platon.ugr.es, ebarcena@goliat.ugr.es & pfaber@platon.ugr.es

Abstract

The Java programming language started as the language Oak when the World Wide Web was still being developed at CERN. It has gained popularity since its launch as a programming language capable of being used to develop applications which can run across the Internet (as well as local stand-alone programs). As with many technologies associated with the World Wide Web, there is a lot of 'hype', confusion, and misinformation. Consequently, while many researchers in the area of Natural Language Processing and Machine Translation will have heard of Java, may be considering using it, or even have got as far as their first 'Hello World' applet, they are probably not fully aware of what the implications of using this language are, and what possible role it could have in the development of computational linguistic applications, either intended to run locally on a wide range of computing platforms, or remotely across the Internet. This paper sets out to address this issue by presenting Java in a clear, concise fashion and considering how it may be used in computational linguistic applications. A requirements analysis for a generic Natural Language Processing and Machine Translation tool is undertaken to consider how Java could be used, and subsequently two example systems developed in Java (which can be accessed on the Internet) are introduced. Finally, pointers to Java resources are presented so that researchers interested in using this language can both install it and learn how to program it.

1. Introduction

Once in a while a new technology becomes available which can offer interesting possibilities for researchers. One such technical advance which is having an increasingly large impact in almost every research and technical community is the Internet (or, to be more specific, the World Wide Web; henceforth the Web). The use of the Web is growing at an amazing rate, and now there are arguably as many individuals connected from home (via local service providers) as there are private companies, research centres, and academic institutions. In the early days of the Web, the addition of the common gateway interface (henceforth CGI; Heslop & Budnick, 1996) and related tools designed by Richard Denny enabled Web pages to do more than simple Hypertext Mark-up Language (henceforth HTML) document retrieval. The CGI enabled Web pages to embed calls to programs on the server which could do such tasks as accessing server resident databases. This technology was subsequently adopted by computational linguists to provide on-line linguistic tools such as mono- and bilingual dictionaries. There are many such examples on the Web at the moment, a small selection of which can be seen below:

ARTFL (French/English, English/ French dictionaries)	http://humanities.uchicago.edu/forms_unrest/FR-ENG.html
Logos (Multiple bilingual dictionaries)	http://www.logos.it/forumget.html
Travlang's translating dictionaries (bilingual between thirty eight languages)	http://www.travlang.com/languages

While these on-line lexical reference systems have been useful linguistic tools, and will continue to be so, the many limitations of the CGI (e.g., basic form based interface, limited graphical capabilities, limited user interaction, non arbitrary data types, high server loads) have contributed significantly to the lack of more sophisticated Natural Language Processing and Machine Translation (henceforth NLP&MT) tools on the Web. Sun Microsystems subsequently released their new programming language called Java, which has had a growing effect on the computing community ever since (judging by the traffic in the comp.lang.java.* network news groups, E-mail discussion lists, large media presence, and a list of over a hundred books on the subject which have been written to date; Howard, 1996a). Java has been presented as a panacea for developing sophisticated portable applications which could be used in a wide range of heterogeneous environments ranging from Web pages accessed across the Internet to local stand-alone applications, running either on a desktop PC or even (in the future) on the new generation of hand-held PCs. Consequently, it is not surprising that interest in Java has grown at an incredible rate.

However, the problem with a new development tool like Java is that it is very hard to separate the fact from the fantasy (which often arises in the many discussion forums) regarding the language and its potential role in the development of new NLP&MT applications. Hence, researchers who are contemplating the development of a computational linguistic system and are tempted to use Java for this purpose would have to invest a lot of time and effort not just to get hold of the basic information, but to disentangle its content. The majority of the books on the subject are little more than 'how to program' guides. The information available on-line at Sun Microsystems (or JavaSoft) looks remarkably like a marketing exercise, which not only does not list the problems with the language, but also does not differentiate between what is currently possible and what may be possible in six months or a year from now. Furthermore, after such an investment of time and effort, researchers might discover that Java is not relevant for their application.

This paper has been written in an attempt to solve this problem by presenting the Java programming language at a high level of description, avoiding actual reference to the nuts and bolts of how it works and should be programmed, for researchers who might want to consider using it, but so far have only heard about it in a very superficial way. Together with this information, an examination of the claims which have been made about Java and a discussion of its problems have been presented. To give an idea of what this language has to offer to NLP&MT researchers, a requirements analysis of a generic computational linguistic system is introduced against which Java is evaluated. Subsequently, two systems are outlined which have been built with Java and are both available on the Web. These systems can be accessed on-line to provide researchers with experience of state-of-the-art systems built with this language. Finally, a few pointers are included to enable researchers who are interested in developing their own Java based NLP&MT tools to access sources of information that will prove to be useful (e.g., where to get the Java development kit, an on-line tutorial, documentation, examples).

While some computational linguistic researchers may already be working on Java based tools, general widespread knowledge about the language and what it has to offer remains quite limited. Therefore, the goal of this paper is to remove the confusion and hype surrounding Java, and examine how it may be of use to researchers. It is important to note that the objective of this paper is neither to push Java as a replacement for traditional languages used in NLP&MT applications (like Lisp, Prolog, or C) nor to dismiss its usefulness in a superficial way. The eventual choice of programming language remains (as ever) with the researcher.

2. A requirements analysis for generic NLP&MT

The design and development of a computational linguistic system usually starts with an analysis which specifies what requirements need to be fulfilled. While these goals and conditions vary from

research project to project (e.g., the goal could be testing and implementing an existing linguistic theory), it is possible to make a list of generic requirements which might be common to future systems, and can be used to illustrate how to apply Java to such requirements. A full analysis and comparison of Java with other programming languages such as Lisp, Prolog, or C are not undertaken here for a number of reasons, including the fact that, due to the relative newness of Java, very few real Java applications exist to date, from which comparisons can be drawn.

Before presenting the requirements analysis, it should be noted that the ultimate goal of most computational linguistic systems is the quality of their output, e.g., the fidelity, intelligibility or clarity, and style in MT systems (Lehrberger & Bourbeau, 1988) and accurate cross-linguistic links between entries and lack of circularity in computational lexica. However, this fundamental objective is not included in the list of requirements below because it depends more on the underlying linguistic theory implemented in the system, than on the programming language (since, with varying degrees of difficulty, a computer scientist can implement most linguistic algorithms in programming languages ranging from 6802 assembler to Common Lisp). The requirements presented below refer to the more computationally based goals of a generic system (it should be noted that this list is not intended to be complete, but merely a selection of standard requirements for NLP&MT systems):

1. Simple updating of linguistic knowledge in the system, i.e., this knowledge must be stored in a way that enables linguists and non computer experts to access and modify it easily. While some of the knowledge will actually be built into the program itself (e.g., how to parse sentences), and therefore can not be changed, there are usually data files which contain other types of information (e.g., dictionary files). The knowledge should be structured in such a way as to both make it portable to future applications and facilitate human understanding (rather than coded in a way that makes it easy for the program to access). An example of how this type of information is usually coded can be seen in the following extract taken from a data file from the Anthem project (Barcena et al., 1995):

   OPHIASIS SUBS snomed($DO-53112$,$$,01,$Ophiasis$,$(T-01400)$,704.01,$$)
   OPHTHALMOMYIASIS SUBS snomed($DE-71402$,$$,02,$Opthalmomyiasis$,$
   (T-AA000)(L-0020)$,$134.0$,$$)
   OPIOD ADJ snomed($C-60600$,$$,02,$Opioid$,$$,$$,$$)

   Furthermore, the linguistic knowledge should be managed in such a way that, when extended, it is easy to supply users with new updated versions, perhaps in terms of only the differences with the older information, rather than a complete new copy of the knowledge source.

2. An interface which requires no expert (computer) knowledge to be used, and that offers fast, linguistically sophisticated, and flexible (easily tailorable) access to the information.

3. Complete system independence, i.e., a system which runs on any combination of hardware and software without the need to make a port for each combination.

4. A stand-alone application which does not require a supporting runtime harness which is difficult to install or manage for non computer experts.

5. Application accessibility, i.e., a system which is accessible both locally (on a computer which does not have a network connection) and remotely across the Internet.

6. Reusability of linguistic knowledge, i.e., the portability of the linguistic knowledge to a new system without any additional effort.

3. Java: fact and fantasy

Now that a requirements analysis has been presented for a generic NLP&MT system, a key point description of Java is necessary prior to the consideration of how this programming language may be applied to this analysis. The review and discussion about Java presented here is built upon information drawn from many different sources including Blundon, 1996; Brugués, 1996; Campione and Walrath, 1995; Cuenca Jiménez, 1996; Harold, 1996a, 1996b, 1996c; Jamsa, 1996; Lea, 1996; Naughton, 1996; Van der Linden, 1996; http://java.sun.com; http://www.javasoft.com; http://www.stars.com/Software/Java; and http://www.gamelan.com/index.shtml.

According to Sun Microsystems, Java is a simple, robust, dynamic, multi-threaded, general-purpose, object oriented, platform independent programming language! It has some similarities to languages like C and C++, but was designed from scratch to avoid some of their key problems (e.g., pointer arithmetic, memory management). The claimed strengths of Java can be split into four key issues, namely, portability, security, robustness and ease of usage, and distributed operation across the Web. Each of these issues is considered here in more detail.

Firstly, the claim that Java applications are portable across different hardware platforms, not just in terms of its source code like C, but also its binary code. Java achieves its platform independence by the combination of a compiler which produces machine independent (Virtual Machine, henceforth VM) byte code and an interpreter which runs on the execution machine to convert the VM byte code into native machine code calls. Java is available on the majority of platforms including Sun SPARC and X86 Solaris; Microsoft Windows 3.1 (with more than 8 megs of RAM), NT and Windows 95; MacOS 7.5 on PowerMacs, 68030 (25Mhz and faster) and 68040 Macs; IRIX, Linux, OS/2, and AIX; UnixWare, Bull Estrella or PowerPCs running AIX4.1; X86 running DASCOM OSF/1; the Digital Alpha running Digital UNIX 3.2; the Hewlett Packard HP7000 series running HPUX 10.x; the NCR Globalyst (Pentium) running UNIX Sys V; Sony NEWS (MIPS) running Sony NEWS 6.1.1; Nexstep; SunOS 4.1; and the Amiga.

The problem with this technique boils down to the issue of speed. Interpreted languages run slower than compiled ones. Hence, the speed of a program written in Java depends on both the size of the application running (especially if it is being run across the Internet) and the power of the client computer. A bad combination can lead to very slow programs (between two and ten times slower than native C code). A partial solution has been provided by modifying the design of the Java VM interpreter to include a technique known as compiling 'just in time' (or JIT), which means that once a byte code is interpreted it is held in memory, so that, should it be used again, the native machine code equivalent can be used directly from memory without the need to reinterpret it. This can speed up code execution up to fifty times. A second solution to this problem (which unfortunately removes the portability claim!) was the development of the Java VM in silicon, that is to say, the development of microprocessors which perform the interpretation of the Java VM code. Independently of these dedicated processors, the ongoing development of new microprocessor technology in general (resulting in power and performance improvements) combined with redesigned interpreters will undoubtedly remove the speed of interpretation problem in the future. The reality of the claim of real binary portability across platforms appears to be true. Small Java applications embedded within Web pages (called applets) work on as many platforms as there are Java enabled Web browsers to support them (that is to say, browsers which implement the Java VM interpreter). However, regarding large stand-alone Java applications, this claim remains something to be seen. Certainly, the first version of Sun's own Java programming environment (written in this language), Java Workshop, did not run reliably on non SPARC/Solaris combinations.

Secondly, the claim that Java provides secure operation of code across a network. Secure code operation is claimed to be achieved in two ways. Firstly, the compilation of the VM byte code does not permit any direct memory access or other insecure operations. Secondly, in order to prevent the generation of specially modified code from malicious compilers, the VM interpreter checks the integrity and security of the byte code before executing it. Thirdly and finally, the two main Web browsers (namely Netscape Navigator and Microsoft Internet Explorer) also add an extra level of security by preventing Java applets from accessing the local file system (but, interestingly enough, not Sun's own browser HotJava). So far, Java has proven to be 'relatively secure'. The security holes which have been found in the Java VM to date have been repaired quickly, and future planned extensions using encryption keys and signed code will continue to improve security.

Thirdly, the claim of application robustness and ease of use. In order to assist with the development of robust programs, Java does not include pointers, thereby avoiding many of the memory related pointer errors so common in C variants (it has been estimated that native C code has on average one bug every fifty five lines, often related to memory allocation and deallocation). Memory allocation is handled by a garbage control mechanism, which removes a source of many memory related problems in C and C++. Java minimises program bugs by including strong typing, no unsafe constructs, consistent error handling, and no undefined or architecture dependent constructs. It is a small language and therefore easy to learn. Its object oriented structure leads to code which is easy to read and write, and facilitates reusability. The only minor problem for the adoption of Java is its enforced object oriented view of the world. Non experienced object oriented programmers will need to put up with the learning curve associated with adopting this new view before they become fluent with the task of code design and implementation. A powerful feature of Java is its threading mechanism, which allows task concurrency. While multi-threaded applications can enhance performance of a program, they can also cause a few headaches until the basic principles are sorted out.

Fourthly and finally, Java is claimed to allow distributed application operation on the Web. Applets can be developed to run on Web pages, which enables programs to run across the Internet. The main issue with such programs is that they run on the client machine, and not on the server, which means that the VM code has to be transferred across the Internet before it can run. This, depending on the speed of the Internet connection, can result in a long wait! However, once the applets are loaded into memory, they run as quickly as local programs. This means that either all the data needed by an applet must be loaded when the applet is first transferred across the Internet (which could be very slow), or else, the applet will need to be able to contact the server to access information held at that end.

At this point, it should be apparent that Java has many features which make it an interesting alternative language for computational research applications. Before the application of Java to the requirements analysis, a comment must be made about a traditional problem found in the use of languages like C for NLP&MT, namely, the lack of representational and manipulative processing facilities for complex abstract data structures like words and sentences. Traditionally, languages like Lisp and Prolog have been selected for such applications since they contain computational mechanisms (e.g., list manipulation, pattern matching) which would need to be incorporated (to some degree) in languages like C. However, in these days in which more systems are being designed to run as stand-alone programs on home computers, away from the computational linguistic laboratories and their associated computing tools, there has been a move toward C and C++ based applications. While there will always be a place for Lisp and Prolog in the development of computational linguistic systems, it should be noted that a key feature of object oriented languages like C++ and Java is reusability of code. Hence, the initial overhead of designing libraries for complex abstract data type representation and manipulation which would be needed in Java (beyond the many libraries which come with the basic language) will only need to be designed once, at which point, hopefully, the libraries will start to be distributed in the NLP&MT research community.

4. Java for NLP&MT: the requirements analysis revisited

It should be emphasised that the examination below is intended to be illustrative, and not exhaustive, of some of the ways in which Java can be used to meet the requirements of generic NLP&MT applications:

1. Simple updating of linguistic knowledge... One of the most promising aspects of Java as a tool (especially for server side application development in a network context), is its database access routines, namely, the Java Database Connectivity (JDBC) Application Program Interface (API). This enables both applications and applets to access databases. Consequently, an obvious choice to ensure ease of access to linguistic knowledge stored in a user-friendly way is to store the information in a database. It can then be loaded at run time or when the applet is transferred across the network.

2. An interface which requires no expert (computer) knowledge... The interface can be developed in a number of ways since Java contains libraries to aid the construction of interfaces. The program interface could vary from one or more applets which reside on a Web page through to a complete stand-alone application which has its own 'look and feel'. Both types of interface could run locally, across an intranet, or across the Internet. As was pointed out above, Java is capable of multitasking via its threading mechanism, so tasks within an application can be split into threads (e.g., one controlling source language access and one controlling target language access) to speed up processing.

3. Complete system independence... In principle, Java applications are completely system independent. Consequently, if implemented as an applet, the tool can be run locally, on an intranet, or across the Internet, without needing to change a single line of code. The only difference in the three cases is the location of the database (on the local machine or on a server somewhere on the network). Such information can be set by a single environmental variable. The use of method overloading (a feature of object oriented languages) means that different versions of the same database access method can be written to cater seamlessly with the different locations of the database. An advantage of a single version of the database located on a network server is that it can be modified and updated without the need to send new data files to all users.

4. A stand-alone application ... A benefit of embedding of the tool as an applet in a Web page is that the majority of Web browsers come with Java VM interpreters built in, and hence, no extra Java run time systems are needed.

5. Application accessibility... As has been explained above, the computational linguistic tool can be used both across a network and stand-alone. For users who have a local copy and occasional access to the Internet, new versions of the database files could be made available for on-line updating without the need to ship new copies on disk or CD.

6. Reusability of linguistic knowledge... The separation of the linguistic knowledge (in the database files) from the corresponding Java program means that the knowledge can be easily reused for new applications. Furthermore, since it is easy to change the structure of tables within a database (or create new tables by partially merging existing ones), it is easy to modify the linguistic properties of the knowledge without having to re-enter the information from scratch. Finally, providing the original forms and entries are left intact with the tables, it is irrelevant whether new entries are added since the nature of Structured Query Language access will not generate error messages (unlike a program parsing a file which contains the linguistic knowledge).

Now that an illustration has been provided for ways in which Java fits with the generic requirements of an NLP&MT system, the existence of two systems which have been developed in Java will be outlined. These systems are the only examples (at the time of writing this paper) available on the Web written in Java. The first of these systems is called MoLeX (developed by a research team including the authors; Barcena et al., 1997). It is an on-line lexical reference system which uses a semantic feature based representation to facilitate multilinguality. Its structure is similar to the generic tool presented above for the requirements analysis, with a database containing monolingual lexica and a Java based front end which consists of two separate applets, one for the source language and one for the target language. It can be accessed on-line at http://www.ugr.es/~tread/molex.html (with a Java enabled Web browser such as Netscape Navigator 3.0 Gold - or later, accessible from http://www.netscape.com). The second tool is a Japanese-English dictionary which includes four methods of Kanji 'lookup'. The dictionary can be accessed in four ways: skip-code, selecting a line in the dictionary, handwriting recognition (interpretation of lines generated by mouse movements), and radical selection. This system has been written by Todd Rudick, and can be readily accessed on-line at http://www.cs.arizona.edu/people/rudick/DrawApplet/index.html.

5. Conclusion

In this paper the role of Java in NLP&MT has been discussed by considering the requirements of a generic computational linguistic system, and then, after examining Java, an illustration was presented about how this language might be used to meet these requirements. The strengths of Java, both for computational research applications in general, and specifically for NLP&MT systems, have been presented, and were split into four categories: portability, security, robustness and ease of usage, and distributed operation on the Web. However, it is important that the developer has had some experience with previous object oriented programming languages, a fast computer, and does not mind waiting a while for some of the newer development tools and libraries to become available.

For researchers who have read this paper and are interested in using Java for their own NLP&MT applications, they are encouraged to contact the authors for more information, help, and possible collaboration (perhaps if sufficient interest is shown, a discussion E-mail-list could be started on topics related to NLP&MT using Java). Furthermore, a list of resources is included here:

Firstly, to start developing Java programs all that is needed is the Java development kit (JDK) which can be obtained (together with a wide range of related information) freely from Sun at the following address: http://java.sun.com/products.

Secondly, a rather complete on-line tutorial is available at Sun Microsystems Java site at the following address: http://www.javasoft.com/nav/read/Tutorial.html.

Thirdly and finally, a small list of the many sources of information available of Java is included below:

General information about JavaSoft	http://www.javasoft.com
Documentation on Java	http://www.javasoft.com/doc
General Java information list	http://www.stars.com/Software/Java
Java resources list	http://www.sun.com/java/list.html
Java FAQ	http://sunsite.unc.edu/javafaq/javafaq.html
Cafe Au Lait links	http://sunsite.unc.edu/javafaq/links.html
Java news groups	comp.lang.java.*

References

Bárcena E., Vanallemeersch T. & Gerardy C. 1995. Distributional Underlying Predications for the semantic description of medical diagnoses'. In *Actes de la Première Rencontre de Jeunes Linguistes.* Université du Littoral (Dunkerque). 17-18 March 1995.

Bárcena E., Faber P. & Read T. 1997. 'El diccionario del traductor del mañana'. In submission to I Congreso Internacional de Estudios de Traducción. Universidad de la Coruña.

Blundon W. 1996. The truth about Java'. *Internet World,* 7(12).

Brugués A. 1996, 'Ultimas tendencias en el Web'. In *Net Conexión,* **13.**

Campione M. & Walrath K., 1995. 'The Java Tutorial' (accessible at http://www.javasoft.com/nav/read/Tutorial.html).

Cuenca Jiménez P. M. 1996. *Programación en Java para Internet.* Anaya.

Heslop B. & Budnick L. 1996. *Publicar con HTML en Internet.* Paraninfo.

Howard E.R. 1996a. 'The Java Book List' (accessible at http://sunsite/unc.edu/javafaq/books.html).

Howard E.R. 1996b. comp.lang.java FAQ (accessible at http://sunsite/unc.edu/javafaq/javafaq.html).

Howard E.R. 1996c. *The Java Developer's Resource.* Prentice Hall.

Jamsa K. 1996. *Java Now!* Jamsa Press.

Lea D., 1996. *Concurrent Programming in Java: Design Principles and Patterns.* Addison-Wesley.

Lehrberger J. & Bourbeau L.1988. *Machine translation: linguistic characteristics of MT systems and general methodology of evaluation* (Linguisticae Investigationes: Supplementa, 15). John Benjamins.

Naughton P. 1996. *The Java Handbook.* Macgraw Hill.

Van der Linden P. 1996. Just Java. 2nd edition. Sunsoft Press/Prentice Hall.