The First International Conference on The Practical Application of Java

PAJava99 PROCEEDINGS

The Commonwealth Conference
and Events Centre
London UK 21st-23rd April 1999

Proceedings of the International Conference on Practical Applications of JAVA, London, UK, (1999), 193 – 203

Applying Java for the Retrieval of Multimedia Knowledge Distributed on High Performance Clusters on the Internet

Stephen W. Ryan and Arvind K. Bansal
Department of Mathematics and Computer Science
Kent State University, Kent, OH 44242 - 0001, USA
Phone: +1.330.297.6864 and +1.330.672.4004. ext. 214
E-mail: sryan@mcs.kent.edu and arvind@mcs.kent.edu

Abstract

This paper describes a Java application for interfacing with an Internet based distributed multimedia knowledge retrieval system. The aim of this system is to facilitate the access of distributed knowledge residing on remote computer systems. The motivation for this work is the increasing demand for applications that take advantage of the Internet and local intranets for accessing and integrating distributed knowledge bases. Our solution is composed of two parts that integrate multimedia distributed knowledge bases, high performance heterogeneous computing, and internet-based programming. The back end is a high performance distributed knowledge base engine based on our associative logic programming model and the message passing paradigm for heterogeneous distributed computing. The front end consists of a portable graphical user interface, a parser and a compiler, and a query management system for interfacing with the distributed knowledge base system. While the knowledge base engine is implemented in C++ for performance on a variety of high performance computer systems, the front end is based on Java technology for portability, multimedia capability, and Internet functionality.

Keywords: Artificial intelligence, associative computing, distributed computing, distributed knowledge base, heterogeneous computing, intelligent search engine, internet computing, Java, message passing, logic programming, programming language, PVM, MPI, symbolic computing

1. INTRODUCTION

The recent boom in the popularity of the Internet is largely due to its ability to provide sharing of resources such as databases and knowledge bases, applications, and computing power across arbitrary distances and among a large number of users. The growth in the number of applications that take advantage of the Internet and local intranets to distribute information and services has resulted in the need for a high level tool to organize and coordinate these resources. Ideally, one would like to be able to access a desired resource transparently, regardless of where on the network or on what type of computer system it resides. One would also like the ability to pull resources from various sources and assemble them into a complete solution in an *ad hoc* manner, without concern for low-level details such as the host architectures, network protocols, data formats, etc. One would also like to be able to do this from any arbitrary location, independent of any particular workstation or terminal.

In this paper, we describe a system that begins to solve this problem by providing user-transparent information retrieval from distributed multimedia knowledge bases. The model integrates the following three technologies:

1. Associative logic programming [2, 3, 4] for storing and querying knowledge. Associative logic programming uses tabular data structures for efficient pattern matching and goal

reduction. Due to the uniformity of a flat data representation, the associative logic programming system works on any architecture.

2. Message passing to link together heterogeneous high performance computers at a local site into a unified distributed computing platform. Popular message passing libraries such as PVM (Parallel Virtual Machine) [7] and MPI (Message Passing Interface) [13] facilitate data transfer between cooperating processors and translate between the different low-level data formats of heterogeneous architectures.
3. The Java programming language [1] as an architecture independent language that provides a portable graphical user interface, support for multimedia content and access to resources on the Internet via Uniform Resource Locators (URLs). In addition, the use of Java allows the user interface code itself to be mobile. By constraining some of the functionality (file system access for example), we can enable the entire front end to be loaded on demand as an applet from any site with a Java-enabled web browser.

The overall scheme for the system has been illustrated in Figure 1. The Java application (or applet) is used to access remote knowledge bases from the internet. Knowledge is retrieved in the form of logic programs, which are displayed by the GUI, compiled into a low level abstract machine code to be loaded into and executed by an 'Associative Logic Program' server running on the local network. A lexical analyzer, parser, compiler, and interface to the local message passing system are integrated with the GUI.

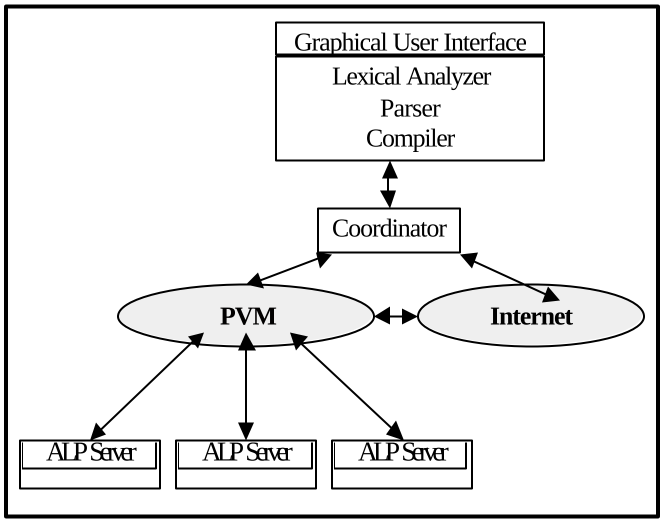


Figure 1. Overview of the distributed multimedia knowledge base system

To the best of our knowledge, this paper describes for the first time how multiple multimedia knowledge bases, distributed over the Internet, can be joined together as one larger virtual multimedia knowledge base on the user's local heterogeneous cluster of high performance workstations, and queried in an efficient manner. The system has been fully implemented, and can be demonstrated using a laptop computer connected to the internet.

The organization of the paper is as follows. Section 2 briefly describes the Distributed Associative Logic Programming System (DALPS) and its implementation. Section 3 describes the application of Java code to create a user interface via which DALPS can interconnect with the internet to facilitate a query from a remote location and to display multimedia results. Section 4 describes applications of the multimedia distributed knowledge base system. Section 5 compares our work with other related works, and the last section concludes the paper.

2. DISTRIBUTED ASSOCIATIVE LOGIC PROGRAMMING

In this section, we describe the concept of distributed knowledge bases using associative logic programming, and describe the architecture of our Distributed Associative Logic Programming System (DALPS).

The rules of formal logic provide a precise language for expressing knowledge. In a logic programming [12] system, knowledge is represented as a collection of facts and rules that describe relationships between objects. For example:

```
part_of(engine, turbofan).
part_of(turbofan, rotor).

contains(X, Z) :- part_of(X, Z).
contains(X, Z) :- part_of(X, Y), contains(Y, Z).
```

Information is retrieved from a logic program by posing a query or 'goal', which asks whether the given relationship between the argument objects holds. For example, the goal:

```
contains(engine, W)?
```

would yield a set {turbofan, rotor} which is represented as a vector in associative logic programs.

The motivation of the Distributed Associative Logic Programming System [11] is to distribute knowledge on a heterogeneous cluster of high performance processors in a user-transparent way. A coordinating process then ties these multiple distributed knowledge bases together into a single, virtual knowledge base. The individual knowledge bases reside in Associative Logic Program server processes (ALPServers), each potentially running on a distinct computer system. Within each server, knowledge is stored in a tabular format that facilitates lookup using associative computing techniques.

Associative computing [10] refers to a pattern matching technique in which individual data items in a collection can be identified by specifying some part of the data that we wish to match. The matching operation is implemented as a primitive operation that operates on the entire data set at one time, rather than relying on the typical sequential search. This primitive operation can be implemented using a variety of data parallel techniques, and can take advantage of high performance parallel computing technology.

The associative data representation of the ALPServer engine is augmented by a simple instruction set which describes the matching operations that are required to solve a goal. For each procedure in the knowledge base, there is a section of instruction code that will find the matching facts as well as a section of code for processing each rule.

When an ALPServer process is initialized, it loads this tabular data and instruction code and reports to its coordinator with a list of the goals that it is able to solve. The coordinator builds a virtual knowledge base by combining this information from all servers. A coordinator may then act as a server itself and participate in even larger virtual knowledge bases (see Figure 2).

Communication in the Distributed Associative Logic Programming System is handled by a message passing library, such as PVM (Parallel Virtual Machine). This library provides for the passing of data between the coordinating and server processes and transparently translates between incompatible data formats when those processes are running on different architectures. It also allows for the spawning and management of tasks running on the hosts in the cluster.

Due to its C++ implementation, the Distributed Associative Logic Programming System runs efficiently on most major computer architectures. The message passing libraries (PVM and MPI) are also available on most architectures. The only requirement for running the system is to have the executables compiled for each architecture in the cluster and to have the message passing system installed and running on all machines in the cluster.

In the prototype DALPS system, the user interface is simply a command line prompt. The configuration of the distributed knowledge base is specified in a schema file that is processed at initialization time. The coordinator process at the user's machine launches the servers specified in the schema file and assembles the virtual knowledge base. The user can then pose queries against the distributed knowledge by entering goals at the prompt.

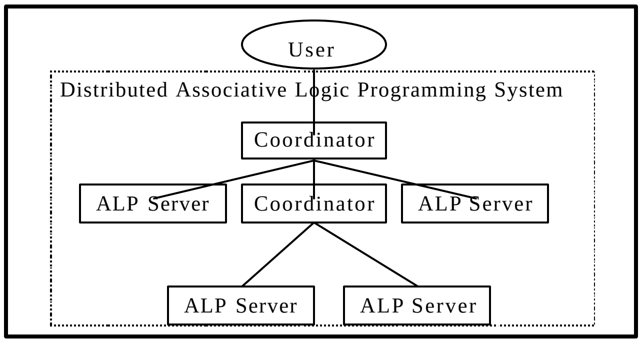


Figure 2. Overall scheme of the distributed logic programming system

3. APPLYING JAVA FOR REMOTE KNOWLEDGE RETRIEVAL

In this paper, we describe how we have applied Java technology to augment the Distributed Associative Logic Programming System. In short, we have:

- Developed a portable graphical user interface for loading, saving, and compiling programs, launching and destroying server processes, and entering goals and assertions,
- Implemented the lexical analyzer, parser and compiler in Java so that it is tightly integrated with the GUI,
- Added the ability to retrieve knowledge bases and data easily from the Internet via URLs,
- Added the ability to display image and sound data using Java's built-in multimedia functionality.

3.1 An Example Multimedia Knowledge Base

To demonstrate the use of the system, we will use as an example a small knowledge base about famous artists and their paintings. The knowledge base contains two types of facts and one rule. Facts of the form

painted(Artist,Painting).

associate the various paintings in the knowledge base with the artists that painted them. Facts of the form

picture(Painting,URL).

indicate a location on the Internet where an image of a painting may be found. The rule pictureby(Artist,Picture) :- painted(Artist,Painting), picture(Painting,Picture).

matches the artists in the multimedia knowledge base with images of their work on the Internet.

There may be several sites on the Internet which keep track of great artists and store knowledge about them in the form of facts like the above. With this system, it would be possible to access this distributed multimedia knowledge, create a server for each individual knowledge base on a local high performance computer cluster, and query the knowledge transparently as if it were a single local knowledge base.

3.2 The Java User Interface

In this section, we describe the implementation of the Java graphical user interface and front end to our Distributed Associative Logic Programming System. This front end includes a lexical analyzer and parser for the input of logical goals and assertions, a compiler to generate instruction code and data for the Associative Logic Program engine, the communication routines for configuring and communicating with the distributed logic engine, and the graphical user interface.

Figure 3 illustrates the primary window for the Java application. It consists of a text window for viewing the text of a logic program and a text entry field for entering new facts, rules and goals. From the File menu, existing programs can be opened either from a local file or from the Internet by specifying a URL. The programs can be saved to the local disk or published to a web address. A compiled version of the program can also be saved to the local disk by selecting the Compile menu.

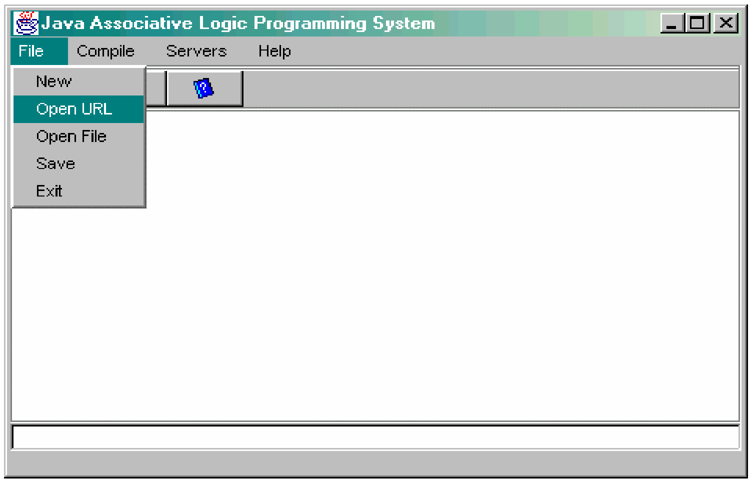
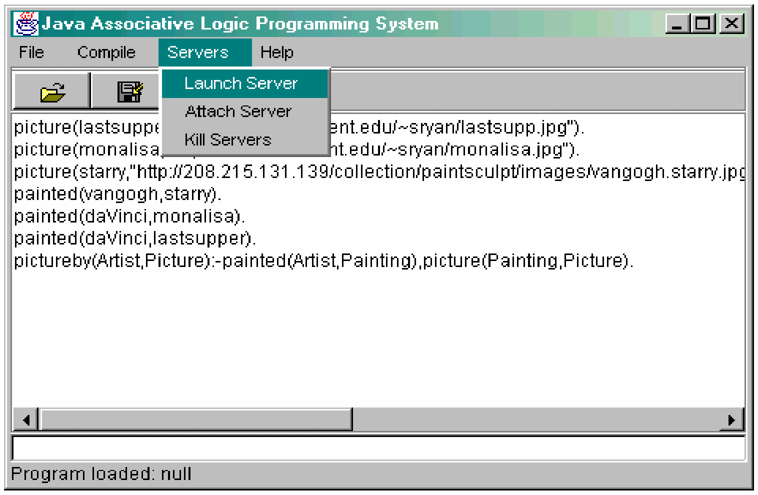


Figure 3. A Java user interface

The program is automatically compiled when launching a server (see Figure 4). The first option on the 'Servers menu will spawn a server process on any host in the currently configured cluster, compile the current program, and load it into the new server.

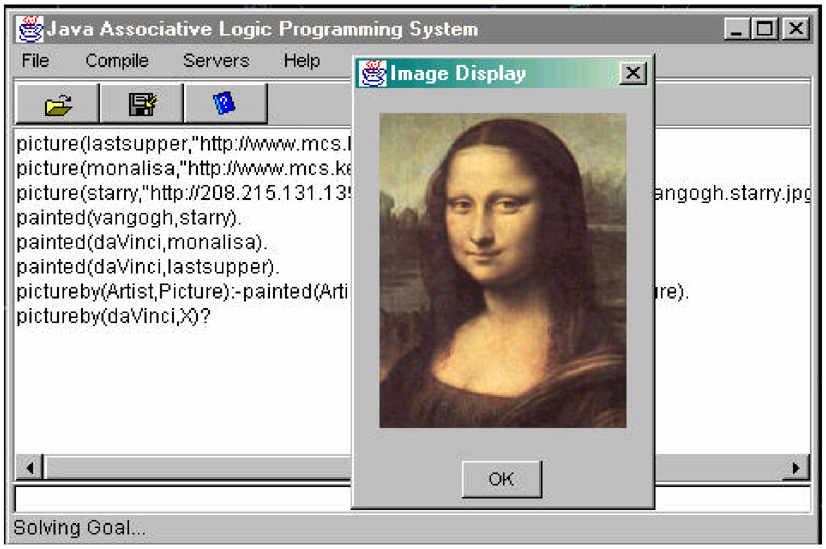
Other functions on the Servers menu include attaching to an already running server or terminating any previously spawned server. After launching one or more servers, a user may submit a query. The goal will be submitted to all servers, which will return their solutions. In the event of a multimedia result, such as the image in the example, the result will be displayed appropriately as shown in Figure 5.



```
picture(lastsuppe                          ent.edu/~sryan/lastsupp.jpg").
picture(monalisa,                          nt.edu/~sryan/monalisa.jpg").
picture(starry,"http://208.215.131.139/collection/paintsculpt/images/vangogh.starry.jpg
painted(vangogh,starry).
painted(daVinci,monalisa).
painted(daVinci,lastsupper).
pictureby(Artist,Picture):-painted(Artist,Painting),picture(Painting,Picture).
```

Program loaded: null

Figure 4. Launching an ALPServer



```
picture(lastsupper,"http://www.mcs.l
picture(monalisa,"http://www.mcs.ke
picture(starry,"http://208.215.131.13!                angogh.starry.jpg
painted(vangogh,starry).
painted(daVinci,monalisa).
painted(daVinci,lastsupper).
pictureby(Artist,Picture):-painted(Arti               re).
pictureby(daVinci,X)?
```

Solving Goal...

Figure 5. An example of multimedia results of a query

3.3 Object-Oriented Java Implementation

The structure of the Java application is shown in Figures 6 and 7. The user interface is integrated with a parser, defined by the ALParser class, for handling the input from the user plus an array of RemoteAbsMachine objects each of which provides the interface for communicating with a server process, indicated here as an ALPServer.

The ALParser class receives logic program statements from the user or from an input file and builds an abstract representation of the logic program. The parser incorporates a lexical analyzer (the LexAn class), a symbol table (the SymbolTable class), and a representation of the logic program (the ALProgram class).

The ALProgram class actually contains two representations of the program. The first is an array of procedures, which in turn are each an array of clauses. This is the representation that is built via the parser and corresponds directly to the text of the program. The second representation of the program is the tabular data and instruction code that is used by the associative logic programming engine.

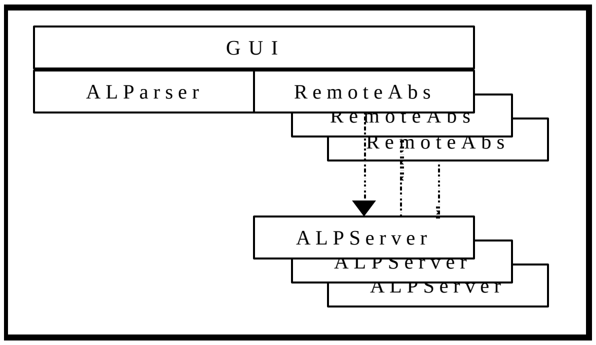


Figure 6. The structure of the Java application

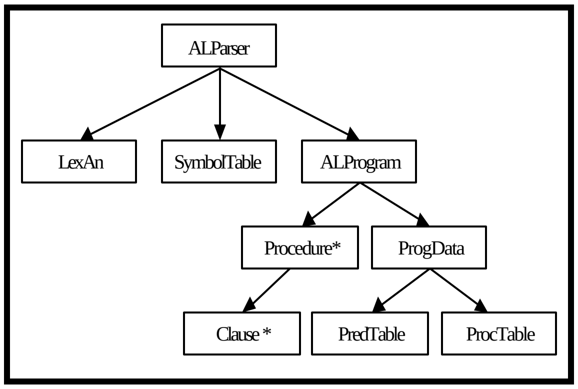


Figure 7. The ALParser Class Structure

When the user enters a clause at the keyboard or reads a logic program from a file, it is passed through the lexical analyzer and parsed. This populates the symbol table and the initial representation of the program in the Program class. At compile time (when the user selects 'Compile' from the Compile menu or launches a server for the program), the associative representation of the program is created within the ALProgram object. This is then written to disk or passed to the server via the RemoteAbsMachine class.

The RemoteAbsMachine class masks the remote nature of the ALPServer. Requests made against the RemoteAbsMachine are actually packed into messages and sent to the server process via the message passing system. This functionality is currently provided by jPVM [15], a Java native methods interface to the PVM library developed at the Georgia Institute of Technology. Each RemoteAbsMachine object can represent a single server or, by specifying a schema file instead of a simple program, a coordinator that represents a system of distributed servers.

After all servers have been launched, goals may be submitted via the text entry line of the user interface. If multiple servers have been launched, then the goal will be submitted to all of those servers. Textual results of the query will be displayed in the text window. Multimedia results, such as images or sounds, will be displayed accordingly.

4. APPLICATIONS

One interesting application that we see for this work is an advanced, intelligent Internet search. Imagine a simple program that would collect the results of a search from each of the major Internet search engines and represent those results as logic programs. With this system, complex queries could be performed against the combined knowledge base to select precisely the items of interest, or weed out the non-interesting results.

An area for further research is the possibility of constructing distributed simulations, using the strength of the logic programming paradigm in abstract modeling. We envision constructing a logical model of the system being modeled, where each component in the system is represented by a logical term. Possible solutions for each term are drawn from a distributed knowledge base that can be linked together in real time to solve the overall system. With an intermediary similar to that mentioned above for the search engine interface, results from numerical calculations or other computations could be combined and included in our logical model.

At NASA's Lewis Research Center, for example, there is an interest in creating computational simulations of gas turbine engines for use in aircraft research. Consider the following simplified logical model of such an engine:

```
engine(Env) :- compressor(Env,V1), combustor(V1,V2), turbine(V2,V3), nozzle(V3,Env).
```

The compressor term, for example, could be mapped either to static tables representing empirical information about a number of different compressors, or possibly to a computational code that could be executed in real time to model the compressor numerically. With the proper interconnection technology, we could construct models that would yield a number of different solutions.

5. RELATED WORKS

Jinni [14] is a lightweight, multi-threaded logic programming engine that is implemented in Java. The Jinni system has the ability to suspend and move a locally executing logic program to a remote Prolog engine for solution. The remote server may be another Jinni engine or a server based on the BinProlog engine. BinProlog is a native language, compiled engine, and thus offers higher performance than the Java engine. Jinni is able to call Java methods from within a logic program and the Jinni engine can be accessed from a Java program, facilitating the creation of mixed Java/logic programs. The target applications for the Jinni system are mobile intelligent agents and knowledge-based assistants for Java applications.

In contrast, our system focuses on a higher level coordination language for integrating distributed knowledge and Internet resources. Our knowledge base servers always consist of natively compiled associative engines, but we plan to add the ability to interoperate with other types of servers as well. While Jinni uses a blackboard architecture for interprocess communication, our system is based on a message passing protocol for scalability and efficient client-server communication.

There have been several efforts to integrate the World Wide Web with the logic programming paradigm. Examples of these are the PiLLoW library [6] and the HTTP library for the ECLiPSe constraint logic programming system [12]. These libraries provide mechanisms for fetching data, including logic programs, from web sites. In LogicWeb [8], web pages are parsed and the structure of the page, as well as any embedded logic programs, are compiled into the knowledge base.

Our system accesses the resources of the World Wide Web via URLs, but does not make use of, or attempt to reason about, the HTTP protocol as these systems do. The knowledge that is in a distributed knowledge base may be stored or presented in an HTML format, but the display of that data would be delegated to an specialized browser engine.

6. CONCLUSIONS

The Java programming language was an appropriate tool for the development of this front end for a number of reasons:

1. It is an object-oriented language. This enables a straightforward representation of the logic program abstractions and the parsing process,
2. It offers a standard portable GUI toolkit. The Java Abstract Window Toolkit (AWT) provides standard visual components, including windows, buttons and menus, on all supported platforms. It is therefore not necessary to create a port, or even re-compile, the user interface for different target systems,
3. It provides mechanisms for easily accessing Internet content via URLs,
4. It is able to display multimedia content such as images and sound.

We believe that implementing the front end of the Distributed Associative Logic Programming System in Java while retaining the C++ implementation for the knowledge base engine is a good pairing of technologies. The strengths of the Java language in portability, network programming, and user interface construction were well suited to the user interface side of this application. On the other hand, for the knowledge base engine, where performance is paramount, there is no substitute for the native C++ implementation. Using PVM and C++ for the back end allows us to take advantage of the low latency and high performance of the local network.

A limitation in the described configuration is the need to transfer remote knowledge bases to the local network for processing. Especially in the case of large knowledge bases, this transfer time could be significant. It would be preferable to execute the query at the remote site where the knowledge is located and communicate only the results of the query. This can be accomplished by accessing the ALPServer via a Common Gateway Interface (CGI) application or Java servlet on the web site host.

Acknowledgments

This research is supported by a Graduate Student Research Program (GSRP) grant from the NASA Lewis Research Center in Cleveland, Ohio. We thank Greg Follen, other researchers, and the administration at NASA Lewis for their continued support.

References

[1] Arnold, K., and J. Gosling, The Java Programming Language, Addison-Wesley, Reading, Massachusetts, (1996).

[2] Bansal, A. K., and J. L. Potter, An Associative Model to Minimize Matching and Backtracking Overhead in Logic Programs with Large Knowledge Bases, *The International Journal of Engineering Applications of Artificial Intelligence*, *Volume 5, Number 3*, (1992) pp. 247-262

[3] Bansal, A. K., An Associative Model to Integrate Knowledge Retrieval and Data-parallel Computation, *International Journal on Artificial Intelligence Tools, Volume 3, Number 1*, (1994), pp. 97 - 125.

[4] Bansal, A. K., A Framework of Heterogeneous Associative Logic Programming, *International Journal of Artificial Intelligence Tools, Vol. 4, Nos. 1 & 2* , (1995), pp. 33 - 53.

[5] Bonnet Ph., S. Bressan, L. Leth, and B. Thomsen. Towards ECLiPSe Agents on the INTERNET, *Proceedings of the 1st Workshop on Logic Programming Tools for INTERNET Applications*, JICSLP'96, Bonn, (1996).

[6] Cabeza, D., M. Hermenegildo, and S. Varma. The PiLLoW/CIAO Library for INTERNET/WWW Programming, *Proceedings of the 1st Workshop on Logic Programming Tools for INTERNET Applications*, JICSLP'96, Bonn, (1996).

[7] Gropp, W., E. W. Lusk, and A. Skjellum, "Using MPI: Portable Parallel Programming with Message Passing Interface," *MIT Press*, 1994, also see *The Message Passing Interface (MPI) Standard*, http://www.mcs.anl.gov/mpi/index.html

[8] Loke, S.W., and A. Davison, Logic Programming with the World-Wide Web. *Proceedings of the 7th ACM Conference on Hypertext*, pages 235 - 245. ACM Press, (1996).

[9] Pontelli, E., and G. Gupta, "W-ACE: A Logic Language for Intelligent Internet Programming," Proceedings of the Ninth International Conference on Tools with Artificial Intelligence, Newport Beach, (1997)

[10] Potter, J. L., Associative Computing, Plenum Publishers, New York, (1992).

[11] Ryan, S. W., and A.K. Bansal, *A Distributed Logic Program Solver*, Proceedings of the Ninth International Conference on Tools with Artificial Intelligence, Newport Beach, (1997), pp. 37-44

[12] Sterling, L.S., and E. Y. Shapiro, The Art of Prolog, MIT Press, (1994).

[13] Sunderam, V. S., et. al., PVM: A Framework for Parallel Distributed Computing, *Concurrency: Practice and Experience, No. 2*, (1990), pp. 315 - 339.

[14] Tarau, P., Jinni: a Lightweight Java-based Logic Engine for Internet Programming, *Proceedings of JICSLP'98 Implementation of LP languages Workshop,* (1998)

[15] Thurman, D., jPVM: A native methods interface to PVM for the Java platform, http://www.isye.gatech.edu/chmsr/jPVM