Pegasus: Protein Prediction Service in the Web

Harris V. Georgiou, Dimitris A. Karras

Department of Informatics
University of Athens
Athens, Greece

Abstract

Protein prediction is one of the most commonly referenced topics in the area of gene analysis by molecular biologists and genetic engineers. Since the introduction of the Genome project initiative, huge amounts of data have been gathered in gene databases throughout the world, requiring efficient automatic analysis and classification systems. Exploiting the spreading of new Internet technologies and the World Wide Web global community, the Pegasus system provides an easily accessible, efficient, on-line tool, for scientists who wish to use a sophisticated protein prediction service for their work.

Keywords: protein prediction, artificial neural networks, client-server, distributed computing, Java language, Java applets, World Wide Web, Internet services.

1 Introduction

With the introduction of the Genome Project Initiative, a huge amount of data have been gathered throughout the world, regarding DNA sequences and gene properties. These data have been systematically stored and updated for years in databases, both conventional and electronic. Unfortunately, the nature of the gene analysis issue and the size of the relative available data does not permit a simple solution to the problem. Human efforts by skilled scientists are not enough to cover the entire range of available information, while to complexity of the problem cannot be confronted simple by the computational power of the current technology.

Protein prediction is one important aspect in gene analysis, regarding property classification and identification. However, although extensive gene data have been gathered for years, both for human and non-human organisms, not direct generic relationship have been established between gene properties and protein classifications. The introduction of an efficient model to define this connection is vital for the understanding and control of all the basic biologic functions of living organisms.

The current technology level in the area of Informatics, especially information sharing and sophisticated computational methods (heuristics, artificial intelligence, parallel processing), have opened whole new possibilities both in system implementation and human-computer interaction. Distributed and parallel computation have permitted the implementation of heavyweight algorithms, heuristics and artificial intelligence have introduced more efficient human-like processing techniques, while the outburst of Internet technologies and the World Wide Web community propose the ideal media for information and ideas exchange.

Pegasus system is a whole new electronic service for protein prediction. It is a robust application, deployed as an Internet service publicly available via the World Wide Web. Any scientist can use it as a tool for amino-acid sequence analysis and classification, accomplished through sophisticated artificial neural network architectures along with standard statistical properties evaluation. It is hoped to

establish a general framework for similar on-line services and tools, not only in the prospect of availability and usability, but also as a prototype for distributed Internet applications and Java programming.

1.1 HISTORY

The Pegasus system is now at its second version, not including the two initial pilot programs (Darwin, Flamingo) that were implemented and tested at the first stages of the development. The first version (Tomcat) was the predecessor of the current system, while Pegasus is the first pure Java implementation of the service.

The development and testing of the first version was completed during the late summer of 1997, as part of the graduate study "Protein Prediction using ANN Techniques - A Java-based approach on the development of a distributed service for the World Wide Web", designed and implemented by Harris Georgiou with Ph.D. Dimitris Karras as supervisor professor. It included an extensive study on distributed and client/server network services, and also the construction of a usable software package for implementation of the prediction service. The final package included some of the features of the Pegasus model, like cache buffers and profiling, but was a single-client implementation that used an external prediction system, programmed entirely in ANSI C. The study was conducted and presented in the Department of Computer Science, University of Ioannina, Greece.

The current version is the first pure Java implementation of the entire application, including the client applet, the remote manager and the main server - including all internal subsystems. Based on previous work, the project was the natural extension of the prediction service, an effort that started immediately after the completion of the first version, cooperatively by Harris Georgiou and Dimitris Karras. Current Pegasus package is a complete protein prediction client-server application, easy to configure and deploy on almost any operating system and network type.

2 General architecture overview

In order to create a usable, easy to access, publicly available service as an on-line tool for every molecular biologist and genetic engineer, the whole system had to be encapsulated in a distributed application that should run efficiently in any available computer platform. The implication of factors like distributed software, platform independence and network connectivity point to some form of an Internet application level service, accessed and exploited via the World Wide Web. Moreover, the latest achievements in Software Technology, specifically Sun’s JAVA programming environment and object-oriented programming, dictate the way to construct systems of high efficiency and high quality in structure and services.

The Pegasus system is designed and implemented as a pure client-server application, similar to many others existing in modern distributed operating systems, also known as daemons. The service is provided through a server program, running in some low (operating system) or high (application) level, and any client who wishes to use it, it has to connect and interact with the server using a special predefined protocol. Pegasus acts like an ordinary application level daemon, providing the service

of protein prediction, in a similar way as the file server or the mailer daemon, although the hole system is implemented in JAVA as individual programming modules and runs under a special runtime environment, the Java Runtime Environment (JRE).

In the core of the Pegasus system lies the protein prediction service (PPS). It is based on artificial neural network protein prediction through a pre-trained back-propagation network. Based on previous work regarding the issue of protein prediction via neural network techniques, the PPS comprises of three distinct neural networks, executing concurrently and producing combined prediction results. Furthermore, each network can be activated firing each neuron subsequently or in parallel with others, using windowing techniques. However, as concurrent activation schemes result in excess performance only when the underlying hardware offers multiple processing units, while on the contrary in a single CPU machine can cause additional overhead, the activation scheme is fully configurable via server setup parameters.

The Pegasus service includes three main modules that act as network nodes in the system's architecture model. The server is a stand-alone Java application that acts like an ordinary daemon, offering the service of protein prediction to anyone in the network. Although internally the server program implements sophisticated analysis techniques, client process scheduling, extensive data caching and user identification procedures, including multiple internal services, from the user's point of view it acts just like standard network services like SMTP or DNS services.

In order to properly access the Pegasus service, a special client application is used, analogous to the standard E-mail or FTP client programs. The main difference in case of the Pegasus service is that the client program is in fact a Java applet, a fully functional network application that can be accessed via a normal Web page in a browser. The program acts as an embedded module in the Web page and uses normal HTTP procedures to be downloaded to the user's machine as part of the Web page, but interacts directly with the Pegasus server, using an application-specific protocol. That way, the system exploits the advantages of Web services, combined with the effectiveness of a pure distributed application.

The Pegasus system also integrates a remote managing package, including special programs for easy server administration and monitoring. The accredited user can access and monitor the server program using the remote manager, from any node in the network, while the service is on-line. Unlike most network services that can only be configured before they start up, remote manager can supervise the operation of the service, get analytical statistics report or send a shutdown command, without interfering with the server's main routine. Remote managing programs are also stand-alone Java applications, implementing simple, yet effective, command-line and interacting text-mode user interface, simplifying the task of writing automated shell scripts by the system operators.

The whole Pegasus system composes a complete distributed application that can be employed either as a specialized analysis tool for skilled user groups, or as a publicly available network service for the entire Web community. The modularity and hardware independence of Java as the programming basis, along with the advanced networking techniques implemented by the standard socket libraries, ensure the usability and effectiveness of the system, on nearly any kind of hardware, operating system or network type.

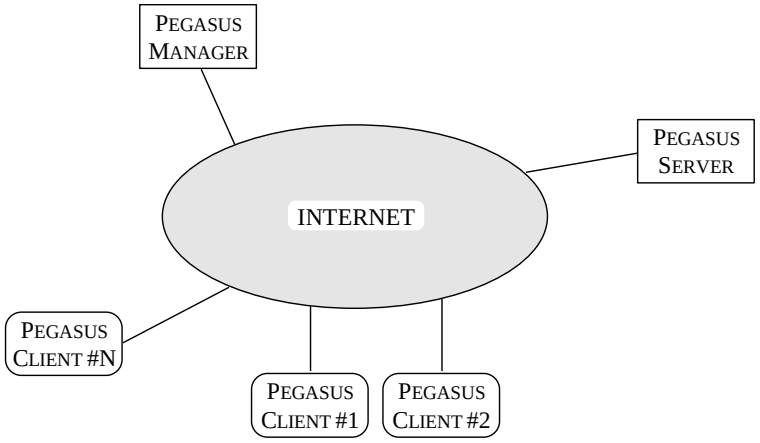


Figure 1: The Pegasus system architecture model

3 System design and functionality

The Pegasus system consists of three separate programs: the server, which is responsible for the data processing and protein prediction, the client that runs as a Java applet on the web browser of every end user, and a remote manager, which is a stand-alone Java application for the management and configuration of the server by an accredited user.

One of the main issues in system design and implementation was the availability and deployment of the prediction service in the World Wide Web. The case of client program distribution and installation on every user's local system, prior to the actual service usage, was simply an unacceptable solution for the whole system design. Besides the fact that everyday users would -unwillingly- install the new package, only if they valued the usefulness of the system high enough, it is most likely that most of the client-side machine might not have been efficient for the stand-alone version of the application, especially if there was not a version of the Java Runtime Environment already installed on their system.

On the other hand, the only other feasible solution for the implementation for such a service was a set of CGI script programs on the Web server side. This solution surely transfers the burden of the whole application in the server side, sending only pre-formatted results to the client browser, but overloads the server machine with CGI scripts and local data transfers. Additionally, CGI scripts provide only means for dynamic data manipulation and format, while data validation and user interface must be dealt with using other Web programming tools like HTML on-line scripts.

The potential use of a complete programming language for the implementation of both the server and client side of the system, lead early in JAVA, which was just emerging as a promising programming environment for Internet distributed applications.

Besides the completely object-oriented architecture of the language, an essential feature for large scale and extensible applications, the extremely complete API, even on the first versions (1.0.2) of the language, provided a very reliable and complete basis for the system implementation. The use of efficient API functions for data manipulation and storage, effective exception handling and standard BSD socket library wrapper objects, affected not only the implementation of the individual system

programs but the whole system architecture itself. The most evident element is the implementation of the client program as a Java applet, combining both the efficiency of a typical program and the functionality of the Web page.

3.1 THE CLIENT PROGRAM (APPLET)

The accessibility of the service to the end user does not only depend upon the design and implementation of the server, but rather on the ease of access and use of the corresponding client program. One of the main reasons for choosing JAVA as the programming language to be used for the implementation, was the new features that brings to the application design level, especially the ability to use the network to download and run an applet module on any machine with a Java Virtual Machine (JVM) implementation. This means that the system can not only provide the specific service to the users, but also the means to access and exploit it, i.e. the client program. The ability to access the client program through a Web page, probably residing on the same site where the server program runs, and then use it to access the service, opens up some new potentials and suppress many limitations as for the need of pre-installed and pre-configured software.

The client program is an integrated application that acts like a user agent, in order to make the service available to the end-user. Using the Java applet technology, the client program is downloaded dynamically through the network, from the main system to the local machine of every user, as part of the associated web page. The program builds a graphical user interface (GUI) in the form of multiple data input fields, interacting with the user in an efficient way.

According to the process stage and user input, the data form is updated dynamically to ensure data validation and integrity. The GUI also feeds the user with understandable information about the request processing progress, or any errors that prevent the completion of the task (network or server failure). The final results are displayed both in summarily and analytic form, providing the user with overall results of the prediction and details regarding specific factors of the analysis process.

The whole process consists, for the client-side, of three distinct stages: the data input stage, the process stage and the output stage. Each stage is connected with an associated "card" in the multi-card layout of the client interactive form. As the process continues form one stage to another, the form's layout changes automatically between cards, adapting the GUI to the user's expectations.

The first card is the input data form. The user provides the system with a used-id and an e-mail address, used to identify the user to the server and to provide the necessary information to classify user access level, along with an electronic address for the result data to be sent to. The choice of interactive or batch mode request processing defines whether the client program keeps the connection alive until the analysis results are received and displayed to the user, or the connection is closed after the server receives the request data and starts the analysis. On this case, the results are sent via e-mail to the address specified by the user. There is also a choice of both receiving the results via e-mail, usually for storage or forwarding purposes, while watching the same results in the output card of the client program's GUI. The user must also provide a valid sequence to be analyzed and whether the server program should conduct a single or windowed analysis on this sequence, along with upper and lower acceptance tolerance levels. The input form is updated dynamically so that only

the proper data fields are displayed at any time. There is also interactive data checking and validation to ensure a correct input data set, before it is sent to the server for analysis.

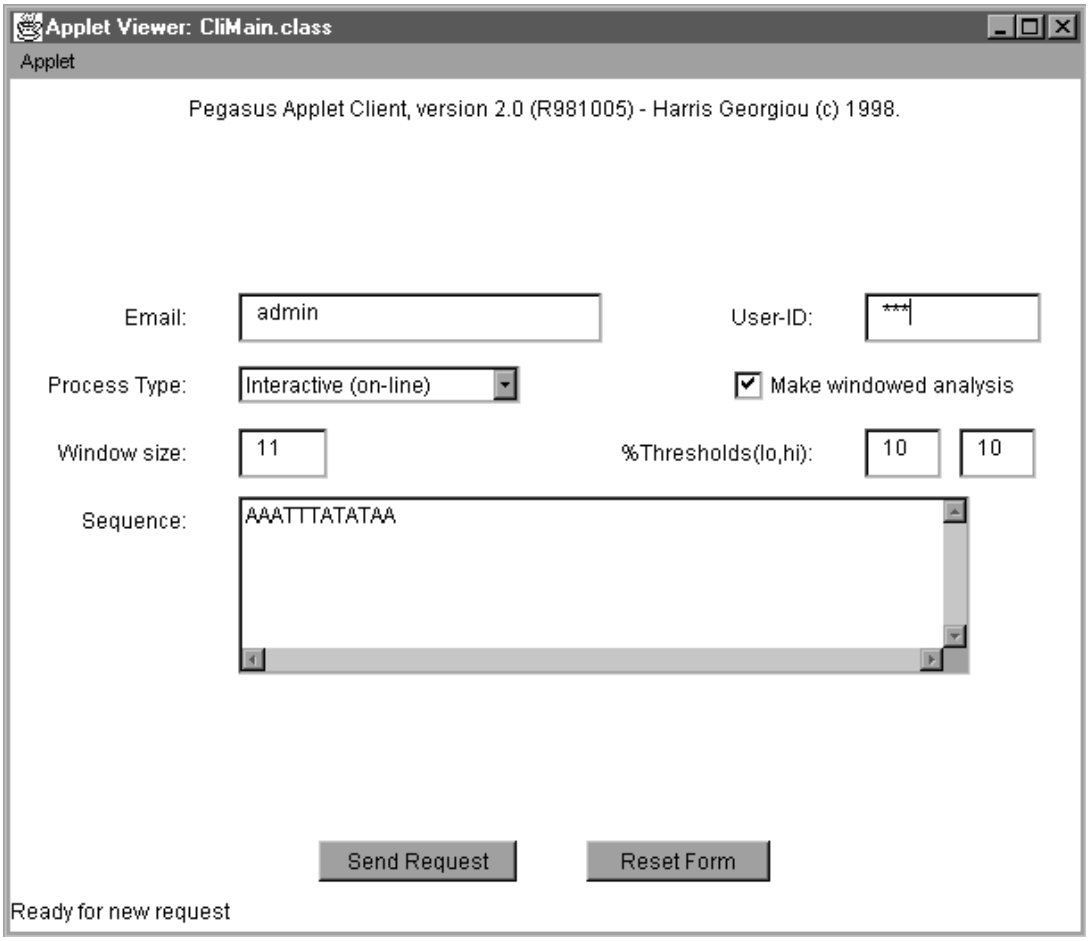


Figure 2: Client applet - Input stage

The second card is only an informative data form, where the user can watch the progress of the analysis. After the input data have been sent to the server, progress and status indicators explain the actual stage of the session, the status of the data communication streams to and from the server, along with informative messages.

As the process continues, the user can watch the exchange of data between the client and the server program, the completion percentage over the whole analysis stage and finally the transfer and formatting of the result data, before they can be displayed on the output card. If the user has chosen a batch mode only analysis, the current form shows only the initial connection and transfer of the input data to the server, before it returns again to the input stage for a new request.

In case of an error, informative messages are displayed on top an and at the bottom, explaining the type and severity of the error. If the error originates from the server, the detailed message contains additional information provided by the server program itself. In case of a broken connection or total network failure, the client program attempts to reconnect several times before it aborts. Of course there is always the potentiality for the user to voluntary retry to establish the connection after some time manually, or cancel the whole session and return to the previous input card.

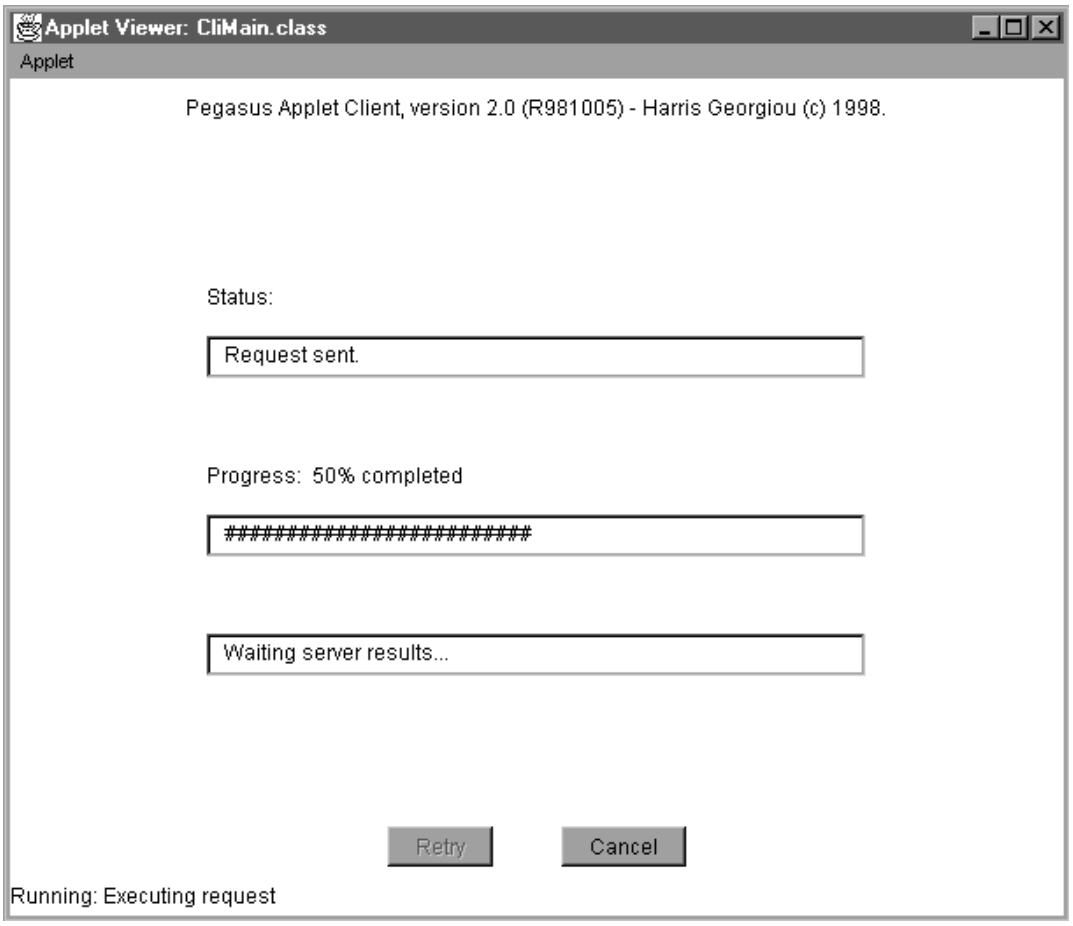


Figure 3: Client applet - Process stage

The output card consists of several data fields, containing the summarized and detailed results of the analysis, customized according to the user's choices regarding single or windowed analysis, window size, acceptance thresholds, etc. This complete result data set is referred to as "profile" of the current sequence, as it identifies the prediction.

The displayed data are extracted from the result data package sent by the server upon the completion of the analysis. The output data set contains the user analysis parameters and sequence for completeness reasons. There is a total prediction associated with single analysis, which is always conducted, and a total prediction from combining the individual window analysis results, if the user has chosen to conduct such one. Along with the total window prediction, a detailed set of individual predictions is provided, containing the sub-sequences and predictions for each one.

The result data also contain information about amino-acid frequencies, correlation, and near or far rejections on lower or upper acceptance limits. On this case only total sequence data are provided due to excess amount of data produced during a windowed analysis. It is believed that these data sets provide enough information about the content of the sequence and that additional data on segments of it would provide no important additional information related to the specific prediction. Having this complete set of result data in hand, the user can not only get a final prediction -single or windowed- about the specific amino-acid sequence, but also see the specific parameters affecting this prediction and characterizing the sequence itself.

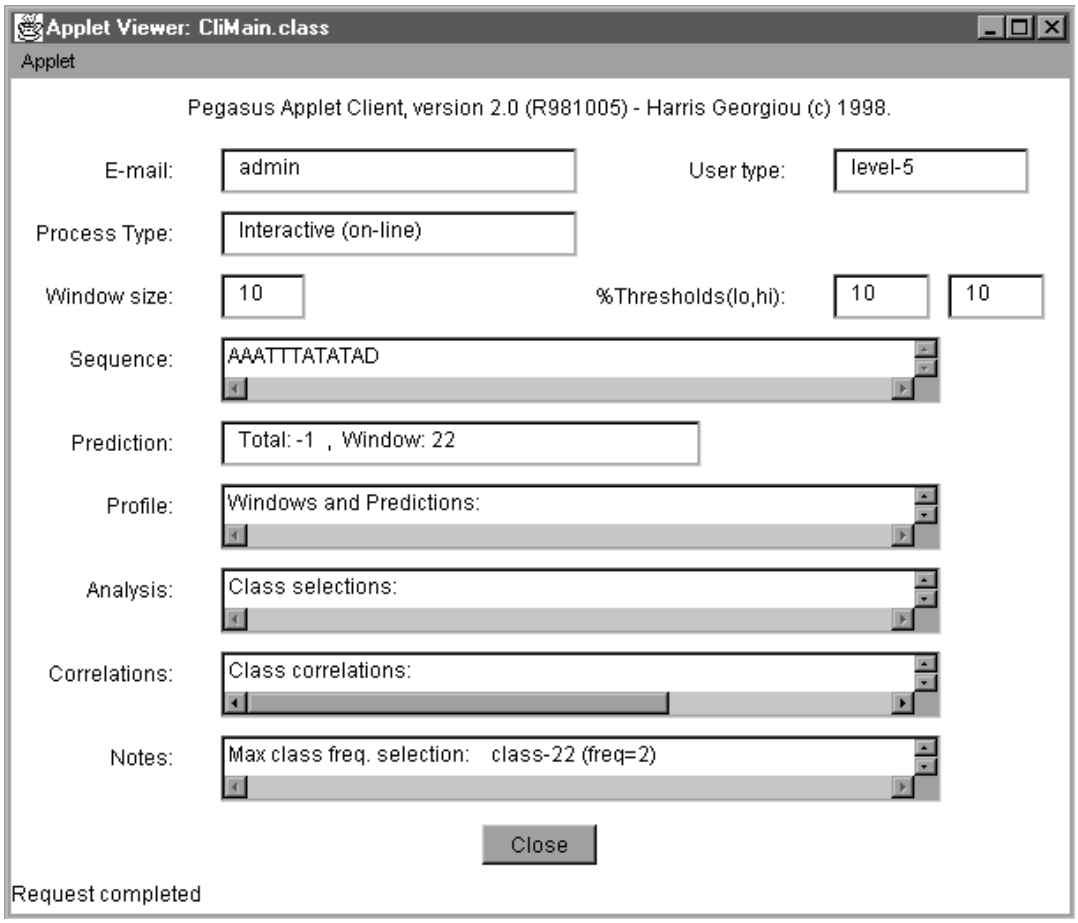


Figure 4: Client applet - Output stage

3.2 THE SERVER PROGRAM

The core of the Pegasus system is, of course, the PPS subsystem, but in order to run as a network service it is wrapped into a complete server program, which in fact is the main node of the whole architecture. The server program runs passively as a normal daemon and executes protein predictions for the input data provided by the clients connected to it. As any ordinary daemon, it binds a specific local system port and listens for requests. The accepted requests are then processed according to their content, as input data or administration commands, according to the Pegasus Communication Protocol (PCP).

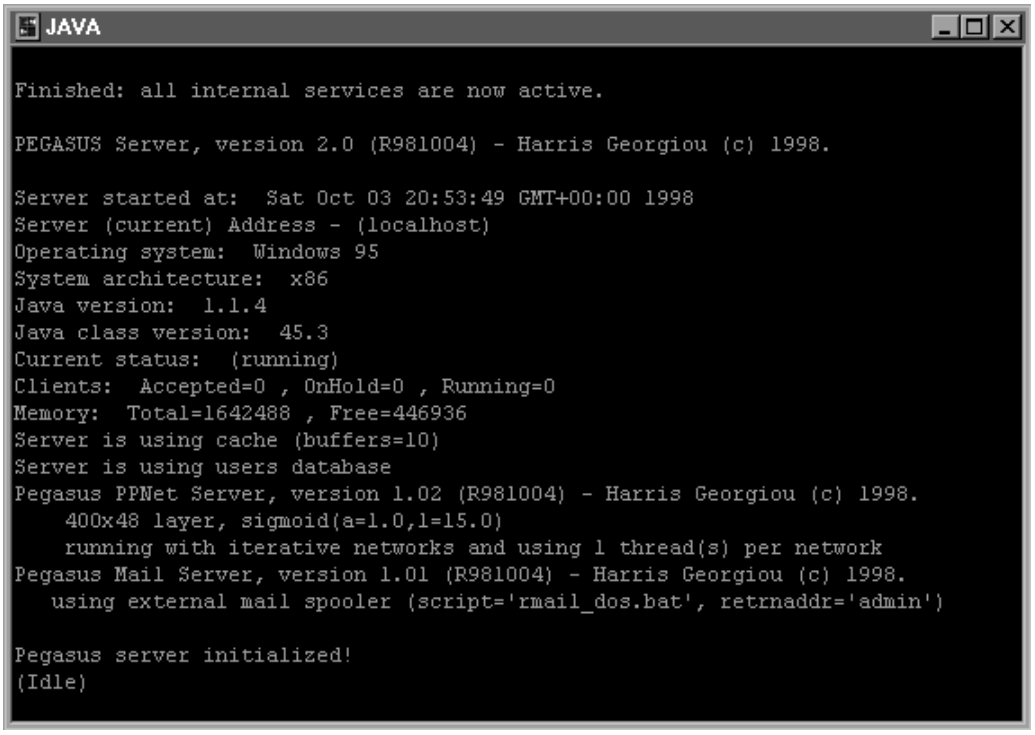


Figure 5: Server program - Initialization

Although Pegasus server can generally be considered a stateless, it uses a serious amount of system resources (CPU time, memory, disk files, network connections) and requires specific steps for activation and deactivation procedures. The execution of the program on a virtual runtime environment (JVM) complicates the task of signal passing, used often to as a mean of communication and manipulation between the daemon process and the underlying operating system. Using a multiple content type protocol (PCP) for the application it is easy to exchange command messages besides the normal request data messages. This way, the server communicates with the remote manager, ensuring normal and efficient procedures of activation and deactivation.

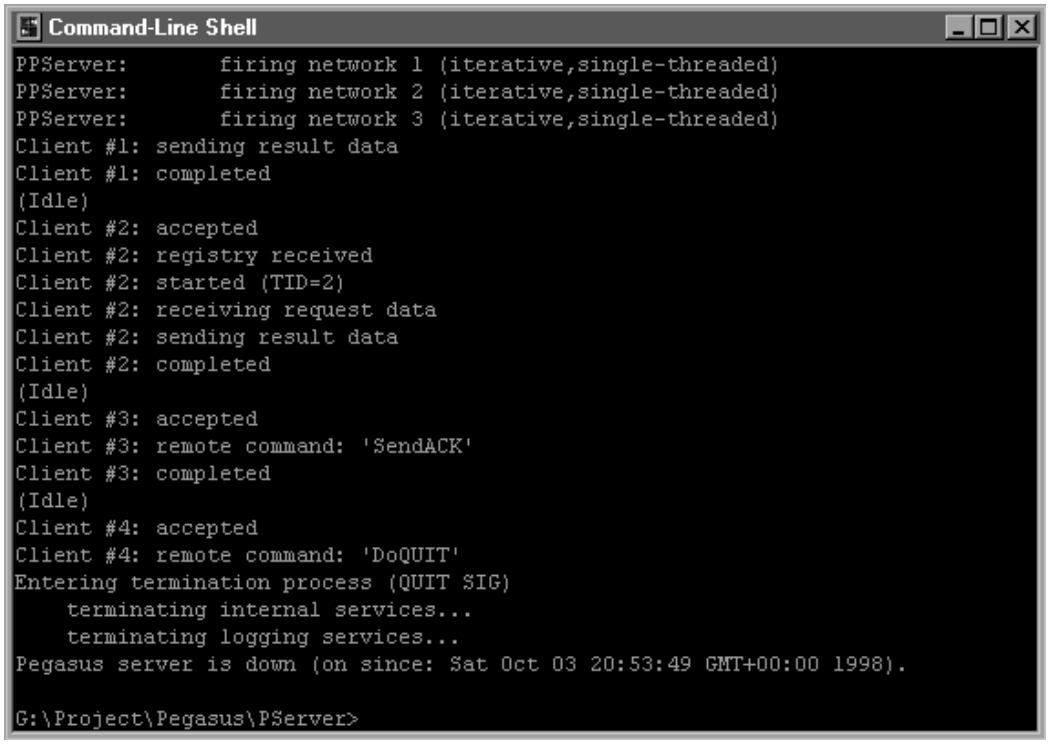


Figure 6: Server program - Request processing and shutdown.

3.2.1 The J-GANNO package

The J-GANNO package is the product of an undergoing project, named "Daedalus", started some time after the release of the first version of Pegasus. It is an autonomous programming framework of object libraries, written entirely in Java as well, for the implementation of neural networks of generic structure and architecture, adaptable to any kind of ANN processing easily and efficiently.

After the implementation of the main framework of the Pegasus system and the deployment of a separate stand-alone application for the prediction system, it was apparent that the next version would have to integrate a full functional internal prediction module. Typical profiling tests showed that an integrated analysis and prediction service would boost the overall performance of the server up to an average factor of 100, regarding mainly data analysis, prediction evaluation and network activation times. There was the need for a robust programming framework, written or closely cooperating with Java, that could be used for the implementation of such a service as an internal module in the main server.

However, our research showed no indication of a ready-to-use product at that time, available as a commercial package or the result of some previous work on Java technologies and programming. The only apparent choice was to design and build such a software package ourselves from the start, relying only on the basic architecture and API provided by the Java language. It was decided that such a project should not be directly connected with the work done on the Pegasus package and that the final result should be an autonomous, complete and generic package that could efficiently implement any kind of artificial neural network architecture and design.

The significant factors for the design and implementation of such a software package are primarily the abstract and modular nature of the design, the generality in implementing any well-known neural network architecture and the efficiency of the implementation, regarding both hardware exploitation (parallel processing) and system independence. As with the main Pegasus system, Java programming language proved flexible and efficient enough to satisfy these specifications.

The J-GANNO package comprises of a variety of interconnected objects that construct the main system. The basic idea is to implement a generic neural network framework, able of representing a large variety of multi-layered feedback architectures, and a robust, usable software package at the same time, the whole development has adapted a fully extensible but complete implementation of the system's modules.

Design and capabilities

Common network procedures like neuron or layer construction, interconnection and activation already exist. Besides the extremely important issue of completely dynamic data structure handling, the programmer can easily construct quite complex network designs, from simple perceptrons to multi-dimensional neocognitron and modular architectures.

The data handling layer of the system can implement and automatically manage up to three dimensional weight arrays per neuron, enabling easy 3-D image processing in each neuron, and three dimensional neuron arrays per layer, providing the capability of constructing clustered and multi-dimensional layers. Each network can contain multiple neuron layers, interconnected with one or more others in any level, enabling subsequent, feed-forward or feed-back data flows between them. The basic network framework also contains two additional data layers, one for the input and one for the output, acting both as data buffers and transformation pools. The data container objects of the standard Java API has been used extensively, to provide an efficient and dynamic way to handle variable data types and network infrastructure representations.

The main objects' methods include default and customized constructors for all object types, functioning both as data initialize procedures and network construction processes. Basic structure information is passed through the constructor parameters to define object instances customized for a specific network architecture. Additional factors, like feedback connections or data transformation parameters, can be defined afterwards when the basic network model has been constructed.

A special set of activation function objects have been defined as part of the system model. Each neuron can use one of the activation functions in this set, but the set itself can be easily extended with customized functions for a specific application. The initial set contains a variety of activation functions, used in most neural network architectures. Every neuron contains a reference to an existing instance or a new copy of a function, while the reference itself is dynamic and can be redirected at runtime.

The same scheme of customizable functions is used for the input and output data layers of the network. It is often useful to send raw input data (text characters, image segments, sound patterns, etc.) directly to the network and receive the post-processed data from the output, without the intervention of wrapper functions outside the whole network structure. This is extremely useful in cases where the data conversion includes pattern matching and lookup in variable-sized matrixes. Transformation functions can be defined as part of the network itself, ensuring the proper data interfacing between the layers and the correct activation scheme for the data

synchronization procedures. As with the activation functions, the references are dynamic, allowing customization and modification at runtime. The programmer is free to define transformation functions, ranging from null-conversion (deactivated) to multiple lookup array hierarchies.

One of the most important aspects of the model is the ability to dynamically interconnect the layers of the network. In fact, the exact dimensions and sizes of the neuron weight arrays are defined from the number and type of the layers with which it is connected. Each neuron feeds the current layer with its output, when it is activated, but its inputs may come from various layers in the network structure, in the previous, next or the same layer level. Every layer contains an output data container and a pool of references to other layers. The output of every neuron can be directly related to a specific position at the layer's output array, but inputs are not stored inside the structure of the same layer. All neurons get only references to input values, enabling easy manipulation and instant update in case of network's structure modification (neuron addition or removal from a layer) or feedback from multiple sources (output values of the same layer from previous activation).

For such a generic package to be instantly usable, it has to at least implement construction and activation functions in any case. Construction procedures for the most part involve the translation of user definitions to real software structures, while the activation processes have to deal with the generic network infrastructure and optimum hardware exploitation. The issue of network structure has to be compromised with some sort of maximum expected dimensions, due to programming limitations regarding non-retrospective implementations of the actual methods. The current package can deal network structures with up to three-dimensional weight and neuron arrays, for vector layered network.

The activation scheme, however, is much more flexible. Network layers, of course, are fired subsequently, according to the specific ANN model and architecture. Each layer can contain neurons in one, two or three dimensional levels, which can be fired in an iterative loop or concurrently, including a group of adjacent units in a sliding activation window or even the all the units in the layer in parallel. Activation efficiency depends heavily on the underlying hardware, which means that concurrent activation in single-CPU hardware can overload the system, as neural network processes mainly propose computational (not I/O) factors and firing neurons in parallel has meaning only in multiple CPU machines. The package supports all kinds of activation schemes, varying from iterative neuron firing to sliding window of concurrent firing, with customizable window size. Although it is very unlikely for each neuron to receive a dedicated execution hardware unit, very large networks with heavy data transfers between the layers might operate more efficiently with a concurrent scheme, even on systems with single CPU. Obviously, network activation and processing times are greatly affected, not only by the number of the underlying execution units, but also by the overall hardware architecture (main memory access and time, data bus, caching).

Besides the main methods for network construction and activation, many other additional procedures have been integrated into the basic framework of the package. Some of them extend abstract methods inherited from parent objects from the Java API on which the system's modules are based and they are mainly used internally by other portions of the Java Runtime Environment (garbage collection, hashing functions, serialization, etc.). Others implement automated cloning and mirroring functions used in special cases where the application requires network duplicates or

bi-directional network structures. Cloning is supported for nearly all the objects in the framework, while mirroring refers to layers and entire networks. A current issue in the implementation and programming of the whole package is an efficient way to support cloning in networks with multiple feedback interconnections. The problem lies in the fact that feedback creates graph loops in the network's model, which are difficult to identify in the programming level without introducing new complexities and overhead to the basic data handling scheme.

The following figure shows class hierarchies and dependencies between the objects that comprise the J-GANNO package. The project is currently under constant development and there has been some code optimization in the version that the Pegasus system uses, but as this architecture is fairly fundamental and no changes are expected to this model scheme. Solid lines represent hierarchy dependencies, while dashed lines represent reference (inline variables) dependencies.

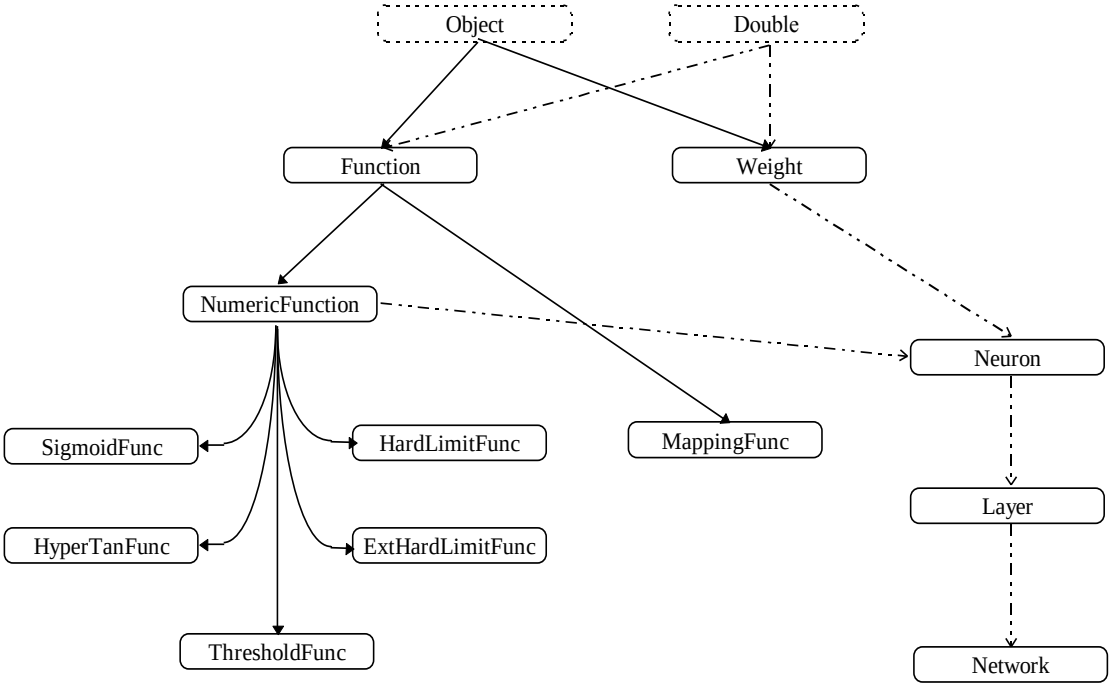


Figure 7: The J-GANNO framework architecture

3.2.2 The PPS server

Based on previous works concerning the issue of the definition of an optimal architecture and parameter optimization for an ANN-based protein prediction system, Pegasus implements a simple, yet effective, network of a single back-propagation perceptron layer, with 400 input and 48 output nodes. During the training process, three distinct optimal parameter (weight) sets have been defined and it was decided that all three of them should be used in a combined way, in order to produce accurate and more precise predictions. Weight values for all three networks are stored externally in a local file, in a compressed binary format.

The protein prediction system is implemented as a separate service that handles the task of identifying a suitable protein category (one of the 48 available) for any random

amino-acid (DNA) sequence given as input. In fact, it can actually function as a separate program, using file-based or interactive input and output data manipulation. However, Pegasus initiates an internal version of the service, using it to process data effectively and in a transparent way to the end-user. When multi-client process is enabled, many instances of the same service remain active in an execution pool, processing prediction requests concurrently or in parallel, depending on the specific underlying machine architecture where Pegasus resides (number of processors).

Every instance of the PPS service consists of an input data interface, an output data interface and the multiple network prediction core. Due to the triple set of parameter (weight) values, PPS contains three identical and distinct prediction networks. Every input set contains an amino-acid sequence in special text format, as it has been given by the user. During the preprocess level (input data interface) it is determined if the given sequence is valid, i.e. it only contains characters from the valid set of 20 amino-acids. If the input sequence is valid, a first order 20x20 correlation array is constructed for the given data. The new 400 (20x20) values of the correlation vector, constructed by appending the array's lines starting from the first, compose the raw input data to be passed onto the networks of the prediction core.

Network execution method is determined through a series of preset configuration parameters. The tree prediction networks share the same input data pool and can be activated one by one (iterative mode) or all together at the same time (concurrent mode). Furthermore, network neurons can be activated in a similar way. Activation can contain one neuron (iterative), a cluster of neurons in a sliding window fashion (windowed) or all the -48- neurons of the network layer simultaneously (concurrent). The exact activation scheme is defined via a configuration file, read upon PPS activation, or via direct parameter manipulation by the main Pegasus server at run-time.

After the input data have been entered to the prediction core and the networks have been activated, the raw prediction data are available and separated into three segments according to the network that made each prediction. Each networks has given a independent result as one of the 48 possible protein categories, so the output data consist of three protein category values. These values are then passed to the output data interface to combine them and determine the final result. A simple majority rule is used to decide whether the prediction result is certain (3 of 3), probable (2 of 3) or vague (1 of 3). In case of windowed analysis, each sub-sequence is analyzed the same way, storing the subsequent results. The protein class selection counters are then used to determine the final prediction (maximum selection frequency), along with some other statistical properties (class frequencies, selections correlation, etc.).

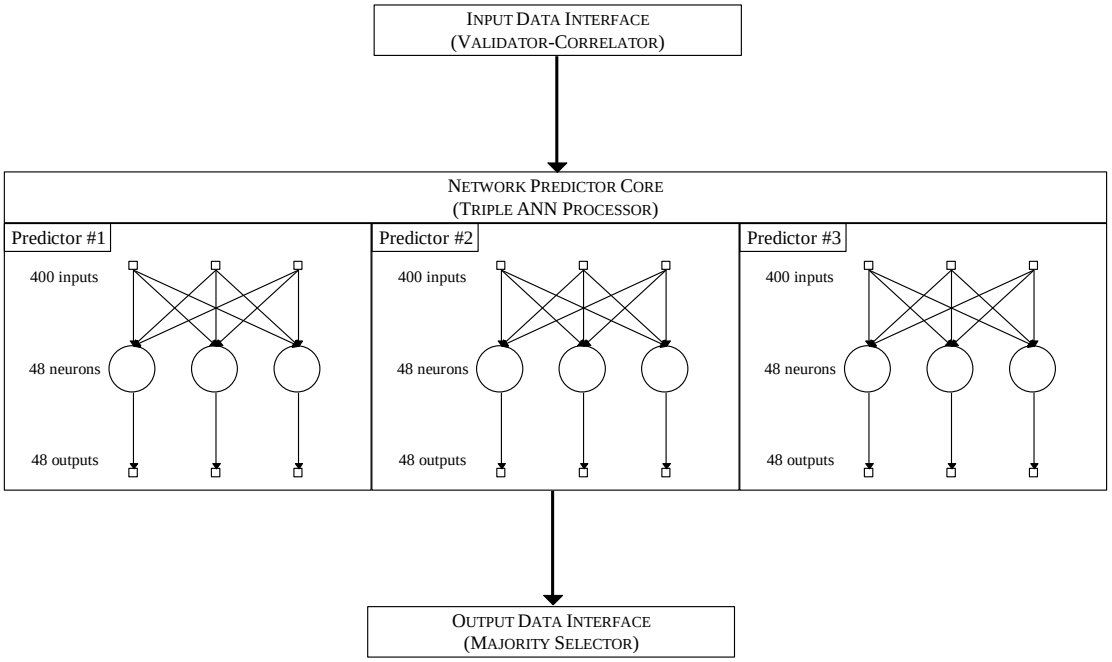


Figure 8: The PPS architecture model

3.2.3 Special design issues

Apart from the main development issues like the PPS module and the J-GANNO neural network framework, a number of other subsystems have been integrated into the main server, as part of its functionality and modular design. These subsystems are implemented as internal services of the general server design, while maintaining the generic character of independent objects able to function properly in any other similar application. These objects include a simple mailer (SMTP server), a data cache manager, a user identification database manager and a priority queue data structure, along with a special client scheduling scheme.

Client Scheduling

As the application design dictated a robust implementation for the whole system and especially the server, it was apparent that an efficient client scheduling algorithm was necessary. It was decided that the common first-in-first-out (FIFO) scheme implemented by the operating system's socket buffers was not enough for the original specifications, so a whole new scheduling manager had to be constructed from the start.

The first version of Pegasus did not include an internal prediction service like PPS, but instead it used external system programs for this purpose. The process time even for average analysis data packs was considerably large (minutes or so), too much for any simple scheduling scheme to function efficiently with many clients. Unfortunately, the execution of external programs did not permit the implementation of any round-robin (R-R) time sharing scheduling, so it was decided to implement a generic priority scheduling algorithm, of customizable queues size and priority levels. The general idea was to divide the requests into categories of descending "importance", always processing the most important to the full.

The two important factors considered during the design of the scheduling scheme were:

- the justice in client selection, according to the arrival time
- the preference of some clients over others, regarding various factors

The first criteria is a common issue in most scheduling algorithms, as it defines the system's behaviour when many client of equal importance have arrived. The second criteria is the contrary part, when the system tries to be more efficient in some way, like maximizing throughput or minimizing the delay time of specific "important" requests. As the Pegasus system also implements a user validation database subsystem, for exclusive or privileged access to some users, it was decided that client categorization should be the first factor in scheduling.

The Pegasus system implements a fairly easy scheduling scheme, utilizing client class information and arrival time, in that order of importance. Using a standard priority queue data structure, incoming clients are put on hold in one of the queues of the system, according to their priority number obtained from the user database manager. Each priority level has its separate conventional queue, where the ordering criteria between equal clients is only the arrival time. When the system becomes available, the scheduling algorithm checks the queue pool, starting from the highest priority level. Only when the current queue is empty the next -lower priority level- queue is examined for suspended clients. The priority levels and queue sizes are fully adjustable and at the current version they are consistent with the thread execution priority levels provided by the Java Runtime Environment. The reason for such a choice is that client priority levels are adjusted to a thread execution priority, assigned accordingly to the process thread of that client when it is picked up by the scheduler.

Request Processing

Even with an efficient scheduling scheme in use, large processing times can overload the server with long queues of suspended clients in a single processing server. The next obvious step towards server efficiency and availability improvement is the implementation of a multi-client processing system. Most Web or heavy-load servers adopt some kind of multi-access architecture so that more than one requests can be processed at any time. The main issue in this case is the techniques that will enable the server to listen for incoming requests on one commonly used port, which it binds statically at startup, and redirect them to other ports, which binds dynamically at need and assigns them to independent processes running concurrently with it.

The Pegasus system includes a robust client processing scheme as part of the scheduler, providing multiple requests to be processed concurrently as separate threads of the main server. The scheduling manager maintains an execution pool, customizable in size, of concurrent threads, waiting to be initialized and start the process of a new request. After the clients have been selected by the scheduler from the queues, the assignment of the execution threads to each one of them is random, but each thread receives the according execution priority corresponding to the clients' priority level.

The requests arrive at the main server port, declaring the presence of a new client which is about to enter the system. The server sends an acknowledge data pack, which informs the client that it has been accepted in the system and appended to a queue, according to the priority level retrieved from the user database using the given user-id. When an execution thread becomes available, the server dynamically binds a new port for the new processing thread and notifies the client, while the thread has initialized

and started, waiting for the specific client's request to arrive. After that, the client can directly interact with the dedicated processing thread, until the session is completed or aborted.

Client identification is based on the initial request data, including user-id, e-mail address, sequence data and analysis parameters, which are stored in the priority queue as the request's tag. Access restrictions integrity is ensured by the dynamic binding of new ports by the server and the request's matching with the data introduced to the system on the first contact of the client with the server. Each new request is introduced to the system the same way, i.e. requesting an available execution thread, appending to a queue and receiving a new port number -with a new processing thread connected to it-, even if the request comes from the same client. This is ensures that no client can overuse the system against other suspended ones, and that the server dynamically binds each port for a specific request, releasing it right after the process is complete.

The Pegasus execution scheduling can be quite effective, as the overhead for each thread is minimal in the Java Runtime Environment. The execution pool size is constrained purely on local system's resources, including main memory for thread stacks and the number of available ports. However, the system can take full advantage of such a scheduling scheme only when the client agents act like stand-alone applications and not as Java applets, due to several security restrictions (networking, file access, etc.) applied to them.

Priority Queue

As part of the entire scheduling algorithm, Pegasus server integrates a simple yet effective data structure for appending incoming clients to the waiting queue. Since the standard Java API (1.0.2) did not include any kind of simple queue or priority queue data container class, a new one had to be constructed from the start. Although the immediate target was the implementation of an efficient way to sort the incoming requests, it was apparent that such a module should act as a complement to the generic object container classes already provided by the Java API (Vector, Stack).

The priority queue is a general purpose object container, implementing the classic data structure with advanced functionality, efficiency and object-oriented design according to the Java programming environment. To avoid time consuming procedures for adding components into a single sorted queue, the entire priority queue is divided into priority levels, each level having a corresponding simple queue. Priority levels are provided as another object, which acts as a generic priority vector where the user can dynamically add or remove priority levels. Priority vector size is adjusted to the number of levels, while the level values (integer) are kept sorted in descending order, from higher to lower priority level.

Queue pool is kept as a hashing table, where the key is the priority value and the value is the entire queue of that priority level. When an object is to be removed from the head of the whole priority queue, the pop method uses the priority level vector to determine the priority level sequence and their values, from the highest to the lowest level. Using subsequent level values, it checks the queues form top to bottom layer, removing objects from the first non-empty one. The first object at the head of the entire priority queue is the first one in the highest non-empty queue, while the last object at the tail of the entire queue is the last one in the lowest non-empty queue. Appending a new object into the priority queue results in appending it to the correct queue the same way.

The priority queue container provides methods for conventional pop and push procedures, as well as priority-specific instructions. Any object can be popped from the general queue with no priority level specification, or specifically removed from its queue at any time. The flexibility of priority levels definition and the optimal internal structure provided by the hashing table, the priority queue container proves to be a very useful modeling tool, not only for the specific application, but also as a general purpose complement to the Java API classes.

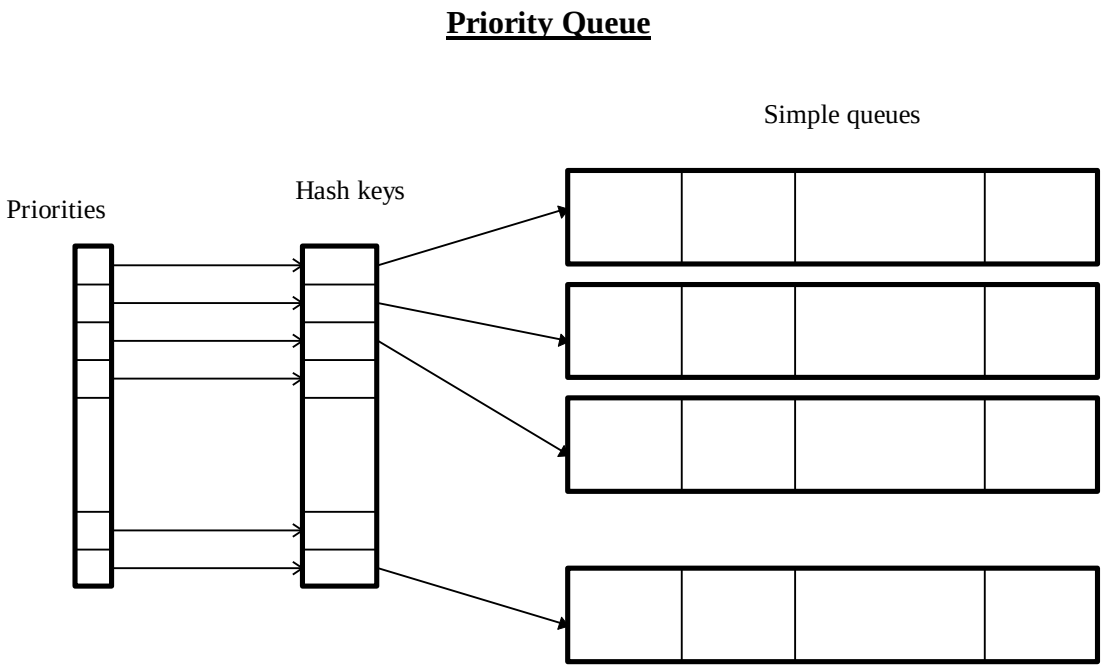


Figure 9: The priority queue - Infrastructure

3.2.4 The mailer subsystem

Sometimes, computational intensive applications present large processing times and low interactivity levels with the end user. Response time is one of the most important factors in interactive services, as users must be contented with the presentation and functionality of the system before it is accepted and used in extent. The first version of Pegasus presented very long processing times, something that might discourage novice users from utilizing the system often. Besides the need for interactive processing, implemented by the graphical user interface of the applet, it was apparent that a batch processing mode would be suitable for users that might want to send their request, without the need for waiting minutes or so before they could receive the analysis results. Of course the nature of this batch mode would still be connected with the interactive client program, at least for the input of the data to be processed.

The solution for this design concern was the implementation of a batch processing mode, which could be selected by the user, where the client program closes the connection with the server as soon as the input data have successfully been sent to it. The server would then process the request and send the results (plain text data) via

electronic mail to an e-mail address specified by the user on its initial request. The batch mode could function exclusively with the interactive mode, selection which could be made by the user on the first form (input stage) of the client program's GUI.

The first implementation used external shell scripts that read the result data of from local file and convert them into an e-mail message, using the standard mailing programs of the operating system. Although this technique usually ensured the mailing of the result data on any properly configured operating system, the whole procedure of writing the -usually large- text data blocks in a local file and executing external shell scripts produced additional overhead to the server and to the host system.

The current version of Pegasus server includes an integrated mailer, implementing both internal SMTP mailing services and external mailing scripts. The mailer supports both message sending using the standard SMTP protocol and the original scheme of external mailing scripts. The internal mailing service provides instant and efficient e-mail message transmission, interacting directly with an SMTP server already available in the network, while the external mailing scripts redirect the mailing requests to the operating system's programs.

Using a predefined host address, the mailer connects to the remote mail server and interacts directly with it, sending result data in multiple lines, formatted according to the SMTP protocol. The mailer connects to the client's local host at the common SMTP port by default, but host:port and return address is fully customizable.

SMTP mailer is, of course, more preferable in systems which already have an SMTP server installed in the network, but external scripts are more generic, as they do not need to connect to a specific host to conduct the transaction. Furthermore, all messages are transferred into local files systems, which can be collected for detailed logging.

As the e-mail message with the result data presented a compact and efficient way to package and store them by the user in a text file of it's system, current version of Pegasus supports both interactive and batch processing mode exclusively or concurrently, resulting in a total of three processing modes (interactive-only, batch-only or both) that the user can select. When batch mode is active, result data are always sent via e-mail to the user's address, even if the client already have them presented in the output form of the applet's GUI.

3.2.5 Prediction Analysis and Profiling

The issue of protein prediction with ANN techniques, together with windowed sequence analysis, is related with computational and data-generation intensive procedures. To provide an efficient way to categorize and handle the individual result data properties, Pegasus system used a special data structure which is called "profile" of the sequence and the whole request.

Profile is in fact a multi-level record object, organized in sections and sub-sections of slots, in the form of pairs of fields and values. Each field is used to identify a specific property of the session itself, while the value specifies some information related to the request or the result data. The more important data for the end user are the sequence, the analysis parameters and the detailed result information, total and per window. The pairs of fields and values, also referred to as slots, are grouped according

to their context with other related information and stored in specific sections of the profile.

The profile object is implemented as a multi-level hashing table, where the first level contains the main profile sections (server tag, client tag, request data, analysis data, etc.), the second level contains the subsections, etc. Each section or subsection is an individual hashing table, supporting key/value slot handling, I/O functions and cloning. In fact, profile object acts as a coupon that is exchanged between the client and the server. When a new request is made by the user, a new profile object is initialized on the client, using the current input data. The profile is then sent to the server for process. The server analyzes the request data, conducts a prediction and stores the results in the proper slots, filling the profile, before it sends it back to the client. From the higher connectivity level of application model, the profile is the compact unit which implements the transactions between a client and the server. The same unit is used throughout the system for data transfer (sockets) or storage (caching) of the sessions.

3.2.6 Cache buffers

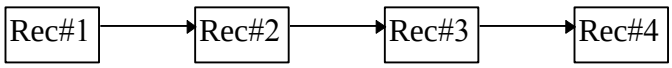
One of the most efficient and commonly used techniques for minimizing the response time in various kinds of network services is caching. Even if the process time for each request is quite small, the storage of previous results in buffers and the instant forwarding whenever a similar request is made, greatly increases the efficiency and throughput of the server. Proxy servers have been used for years for that purpose only in conjunction with Web servers, even in small LANs and enterprises.

The relatively intense processing nature of the protein prediction techniques used in Pegasus system, indicated the need of some sort of caching scheme for the result data. Especially with the longer processing times of the first version of the server, the issue of an caching service has been one of the main features early during the system's design phase. However, it was apparent that incoming requests had to be analyzed and checked against certain portions of previous analyses (sequence, analysis parameters, etc.), requiring some sort of sophisticated internal subsystem, rather than any other external low-level caching on the communication data packets.

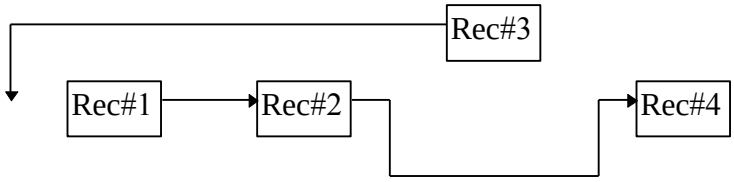
Pegasus server includes a complete caching subsystem, acting on the higher level of profile objects. When a new request has been successfully processed and the corresponding profile object has been filled with the analysis information, one copy of it is sent to the client and one copy is sent to the caching subsystem. The profile object is checked against the current contents of the cache buffers, using specific property fields - the ones related directly to some property of the analysis process (sequence, window size, rejection limits) and not the user identification data (used-id, e-mail address). The profile object provides a fairly easy mechanism for handling session information in a compact form, but as this structure integrates the key fields, a special checking and retrieving scheme is necessary to ensure, not only the proper function of the caching service, but also the integrity of a few security restrictions - preventing the transmission of user identification data of the cached profile along with the forwarded result data.

The main data container structure used in the caching subsystem is in fact a linked list of profile objects, arranged in most recently used fashion. The size of the list is set to a fixed value, configured as a server property, however the dynamic nature of such

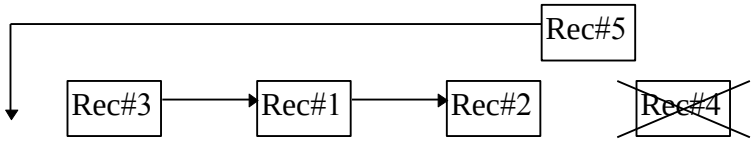
a structure ensures maximum efficiency at memory management and the ability to adjust the number of buffers at runtime. The container uses a typical Last-Referenced-First-Out (LRFO), adding new objects or moving references at the head and removing from the tail. Every new object is added at the head, removing the last object from the tail if needed, while a reference to an object already contained in cache moves it on the top without removing any old references. The entire subsystem is implemented optimally for the specific application's properties (data components are prediction profiling objects), but the underlying structure of the LRFO linked list is a generic object container, supplement to the standard Java API container classes. The caching service provides methods for "touching" objects, resulting their promotion to the top, for adjusting the cache size and for the list "freezing", allowing only rearrangement and shrinking of the buffers. This last feature is extremely useful, as when the cache size is adjusted to a smaller value, but the contents get truncated gradually (on every reference) rather than instantly, providing the chance of keeping some of the objects near the tail, if they are referenced in the following operations, until the requested size is reached.



1. Initial record sequence in cache buffers list



2. Accessing record #3, moving the record at the head.



3. New record addition, remove record from the tail.

Figure 10: Cache buffers - The LRFO list structure.

Usually, caching structures are volatile, initializing as empty at startup and loosing all their contents at shutdown. The Pegasus caching system provides the feature of storing the contents of the entire cache into a group of local files, using the serialization abilities of the base profile object. Each object is stored as a separate file,

similar to the actual data packet sent to the client through the network. At any time, the cache manager can dynamically load a stored set of buffer contents, flushing the current contents or merging the two data sets. The caching subsystem also provides consistency checking mechanisms to ensure the integrity and synchronization of the cache contents, both internal (volatile) and external (non-volatile). Although standard server operation does not utilize non-volatile cache contents, this feature may prove to be an important factor in high-end central servers with extremely large cache sizes, where the initial throughput depends heavily on the initial state of the server's cache.

3.2.7 User Database

Every service in a network can be regarded as a resource, available to the users according to a well specified access protocol and permissions distribution. Typical services, such as file or mail servers, commonly used by all users, there are no access restrictions regarding service availability, while other critical services, like system loggers, require some sort of user identification scheme before they process any request. But even when the service is not critical to the operating system environment, user validation can be used to ensure the availability of the service to a specific group of users and to them only.

The main Pegasus server implements a typical user identification subsystem, including a an access control list (ACL) in the form of an external user database. Although the character of the Pegasus system was, from the start, a publicly available protein prediction service, easily accessible via the World Wide Web, user classification and grouping into categories, provides a fairly easy way to implement multiple privilege levels. As in most special purpose network services, more attention should be drawn to the regular or well trusted users, rather than to new or guest users that wish to explore the system's properties. This does not contradicts the fairness and equal servicing of all incoming clients by the scheduling scheme, as long as the privilege levels are assigned to the registered clients in the proper way.

The user database is an ACL registry, containing registered users that should be treated by the scheduler in a special way. Each entry includes an e-mail address, a user-id and an integer number, indicating the priority level that should be assigned to any request coming from this client. The e-mail address can in fact be any valid identifier that can be used as a unique reference to this user, throughout the network. The user-id field is used as a validation password, known only by the specific user, to provide the necessary identification information. Priority level is retrieved from the user database automatically, as long as the name/password pair given by the user are valid. When the value of the priority level has been set, it can be used both for request scheduling and for the actual process thread priority level.

As the Pegasus service is considered a publicly available network service, no strict security restrictions have been applied during the system's development. The user database ACL is used exclusively for the special handling of some users' requests and not for the restriction of the access to a specific group of registered clients. The security model is fairly loose, as the privacy of the user database contents has been left to the integrity of the underlying file system. Not registered users still get accepted by the server, receiving the default (normal) priority level privileges, customized to the specific environment and incoming request load. The current version of the system uses a text-based ACL local file, maintained off-line by the server administrator.

However, if a more strict security scheme is to be used, encryption algorithms and certification techniques should be used, not only for the local ACL files, but also for all communications and client handling (disable non-registered users' requests processing).

3.3 THE REMOTE MANAGER

Although the server, in conjunction with the client program, compose the heart of the whole system, the operation of such a system in a networked environment requires the provision of the means for the management and administration of the hole application. Usually, the task of administrating such services resides on the management of the corresponding processes in operating system level. This is especially the case for lightweight or crucial system services, which are always activated on system boot and remain active ever after. However, this is not the case for most application level servers that use and manipulate huge amounts of critical data, and have the need for normal activation and deactivation procedures.

Stateless network servers tend to lack administration programs, mostly because the only activities involved in their management is starting and stopping. Even major network services, such as HTTP or mail servers, usually act as daemon programs in a system shell, communicating with the other processes and the user strictly via interrupts and signals, or in some modern operating systems via message queues. In most cases this scheme, especially the message queues, is efficient enough, but it does not provide any means for easy managing, administration and control from the user level, and nearly no means at all in case of on-line service monitoring.

In the case of Pegasus protein prediction service, the basic factors for system load and system efficiency are computational and main memory resources. Each sequence analysis usually implies many sub-sequence window analyses, generation a huge amount of internal data that has to be processed, concentrated and structured properly for the following stage. The core of the prediction system is composed of three individual neural networks of about 20K weights (floating point) each that are activated concurrently. In addition, the main system includes a user database and a very large cache buffer for the output results. The main concern regarding the proper service initialization and shutdown is not only the stability and integrity in the operating system environment. There is a need for gradual and fully controlled processes for initialization and shutdown, to preserve the stability and integrity of the server environment itself (data file structures, non-volatile cache buffers, etc.), as in any other statefull network service. For these reasons it was decided that a full package of management programs should be provided as part of the Pegasus system.

The administration package contains a number of stand-alone programs, intended to be used by a few privileged users. There is a simple socket connector program that interacts with a socket stream in raw text (ASCII) level. Its primary use is to test and validate the transactions of any network program that uses a text-level socket protocol. There is also a more sophisticated version that includes some PCP (Pegasus Communication Protocol) commands in a list of choices of menu type. These two programs are now regarded as deprecated and they are included in the package for completeness purposes only.

The main administration program is the Pegasus Manager, which is in fact a stand-alone Java application, interacting with the user in batch (command-line) or on-line

mode, used to manage and monitor the main server from a local or remote machine in the network. The Manager implements a smart multi-level interaction scheme, providing the user the ability to send remote commands to the server as parameters in the command-line, or using a simple text choice menu. The accredited user can operate the program from any network station, connect to the main server and conduct managing or monitoring activities as the service is on-line.

The current protocol (PCP) includes three main commands that the Manager uses:

- **ACK**: The Manager polls to the server machine:port to confirm that it is up and functioning properly.
- **INFO**: The Manager asks the server to send a detailed status report, including system characteristics and status, usage statistics and active internal services.
- **QUIT**: The Manager sends a shutdown command to the server. This is the only proper way to conduct such an action on the server.

For security reasons, the first version of the PCP scheme does not include an extensive command set for manipulating the systems internals, like adjusting the size of the cache buffers or updating the user database. The protocol provides some additional identification overhead information for the remote commands for security reasons, since a secure transfer protocol has not yet been implemented at this stage of development.

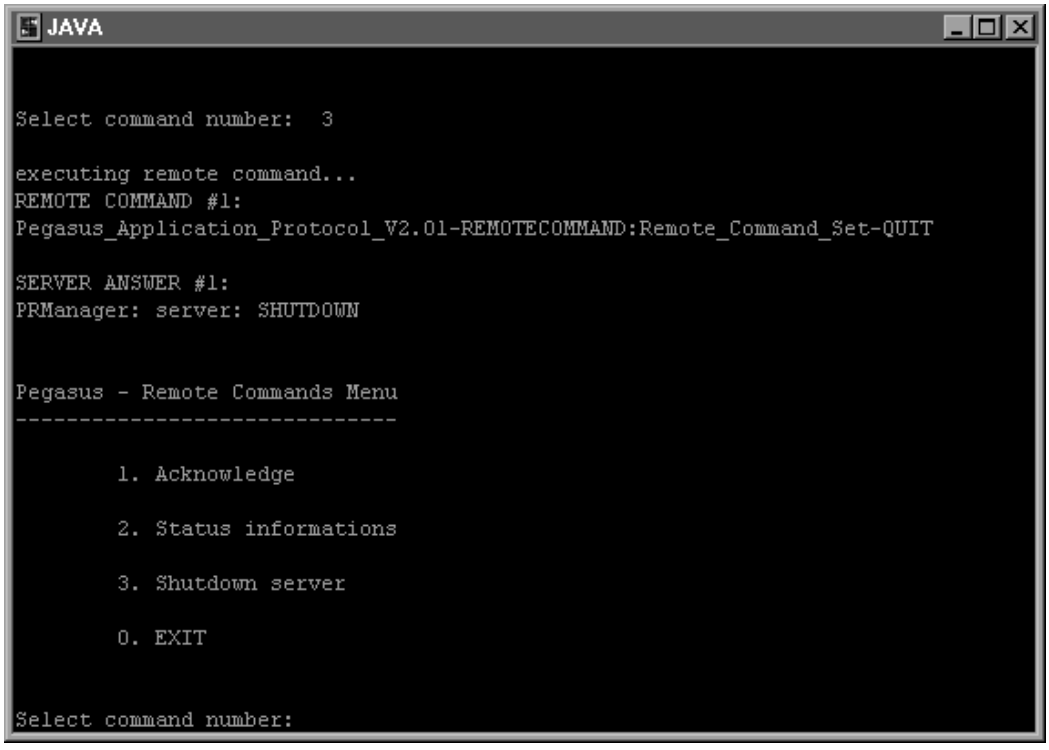


Figure 11: Remote Manager program - A simple session

4 Other server and communication issues

Except the main server and subsystems references, there are some other important issues regarding the entire Pegasus service architecture. Detailed description of the communication protocol, system deployment in various host environments and optimal configuration capabilities should complete the overall reference to the system's internals.

The session protocol

The Pegasus server is configured to listen and accept clients on a fixed port number. Incoming clients sent a request for acceptance on the specified -well known- port, containing all the necessary identification information, and wait for an acknowledge signal. When the server receives the request, it analyzes the identification data and checks them against the user database ACL. If the analysis for this request pattern (sequence and analysis parameters) has already been conducted and the results are still in one of the cache buffers, the server resolves the request instantly, sending a copy of the result data to the client and finishing the session. Otherwise, the proper priority level is acquired and a client registry is constructed for this specific request. The server sends an acknowledge signal to the client, containing the final registry data, informing that the request has been accepted and appended to the system's queue (in the proper position). The client remains idle, until the scheduler picks its request from the system queue. In theory the client becomes perfectly idle, waiting patiently for its turn. In fact, when the same client impatiently retransmits the same request, the server checks the queue to ensure that the request has already been accepted, sending the registry data to the client once more. When the scheduler does pick up the particular client's request from the queue, a new execution thread is allocated, binding dynamically a separate socket port for the process. The server informs the client that it is ready to receive the main request and notifies it for the new contact point (socket port). Only then the client sends the request and the server processes it in the normal way, transmitting the results interactively or via electronic mail.

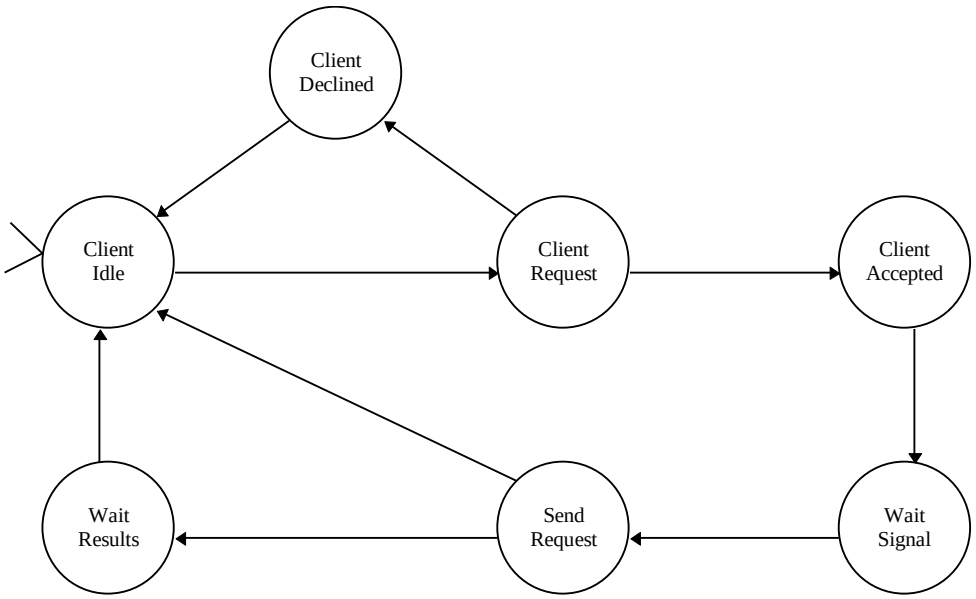


Figure 12: Client's state transition diagram.

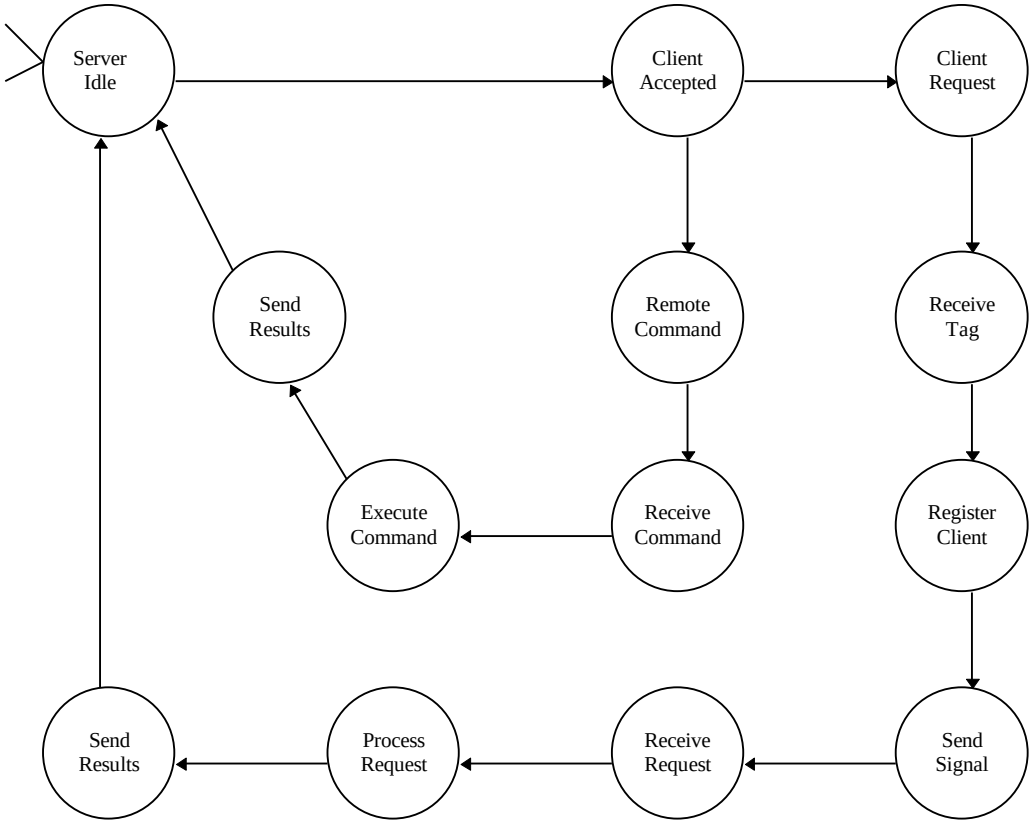


Figure 13: Server's state transition diagram.

The above session protocol corresponds to the complete communication scheme implemented by the Pegasus server and a stand-alone client program. In case of an applet client, Java security model proposes many access restrictions regarding local files and network communications, resulting in a more limited protocol. Mainly, the server has to act as a UDP server, although using session connections (TCP sockets), as the client cannot enter a passive "listening" mode in any case. However, the great improvement of the response times achieved by the new PPS internal service and high performance communication mechanisms (buffered/compressed socket data streams), show little to no deterioration in the performance of such a limited transaction scheme.

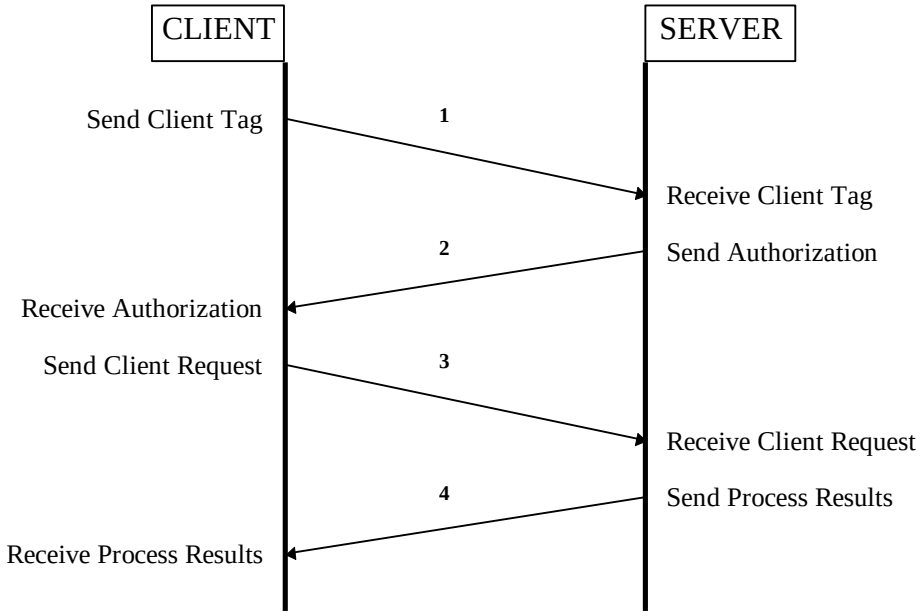


Figure 14: The client-server session protocol.

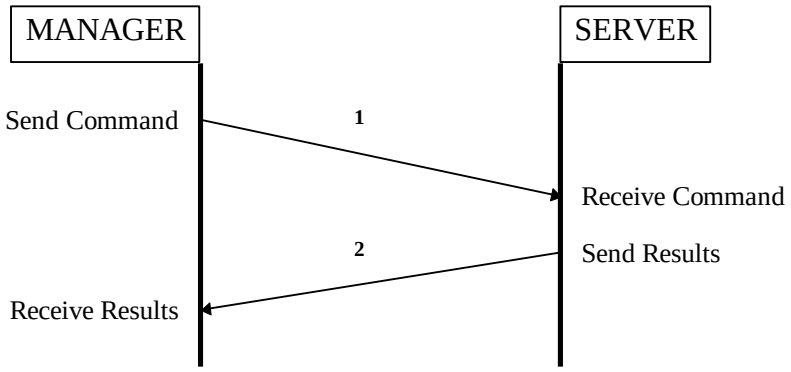


Figure 15: The (remote) manager-server session protocol.

Sequence profiling

During the analysis process, various data are gathered on the client and on the server side, regarding not only the specific sequence data and analysis properties, but the entire session itself. All these information are stored in a compact hierarchical data structure, easy to convert into a transmittable data packet or to store in a server local file, but also an efficient object for session handling by the internal subsystems of the service (cache).

The following figure shows the complete static form and infrastructure of the profile object. The individual sections are constructed upon object creation, but the various data fields are added by the client or the server dynamically, during the entire session.

profile	

header	*general information section*
• client	*client version tag*
• server	*server version tag*
• email	*user's e-mail address*
• usertype	*user's access level*
• proc	*process type (interactive/batch)*
• sendmail	*send results via mail (flag)*
• sendsock	*send results via socket (flag)*
• outcome	*general outcome indicator*

inpseq	*amino-acid sequence section*
• sequence	*amino-acid sequence data*

params	*analysis parameters section*
• winsize	*analysis window size*
• lorejlim	*lower "rejection" limit (%)*
• hirejlim	*higher "acceptance" limit (%)*

totalp	*total prediction section*
• prediction	*total prediction result*

windowp	*windowed prediction section*
• winseqs	*window sequences*
• results	*subsequent prediction results*
• prediction	*final prediction result*

clsfreq	*class frequencies section*
• selections	*class subsequent selections*
• freqs	*class selection frequencies*

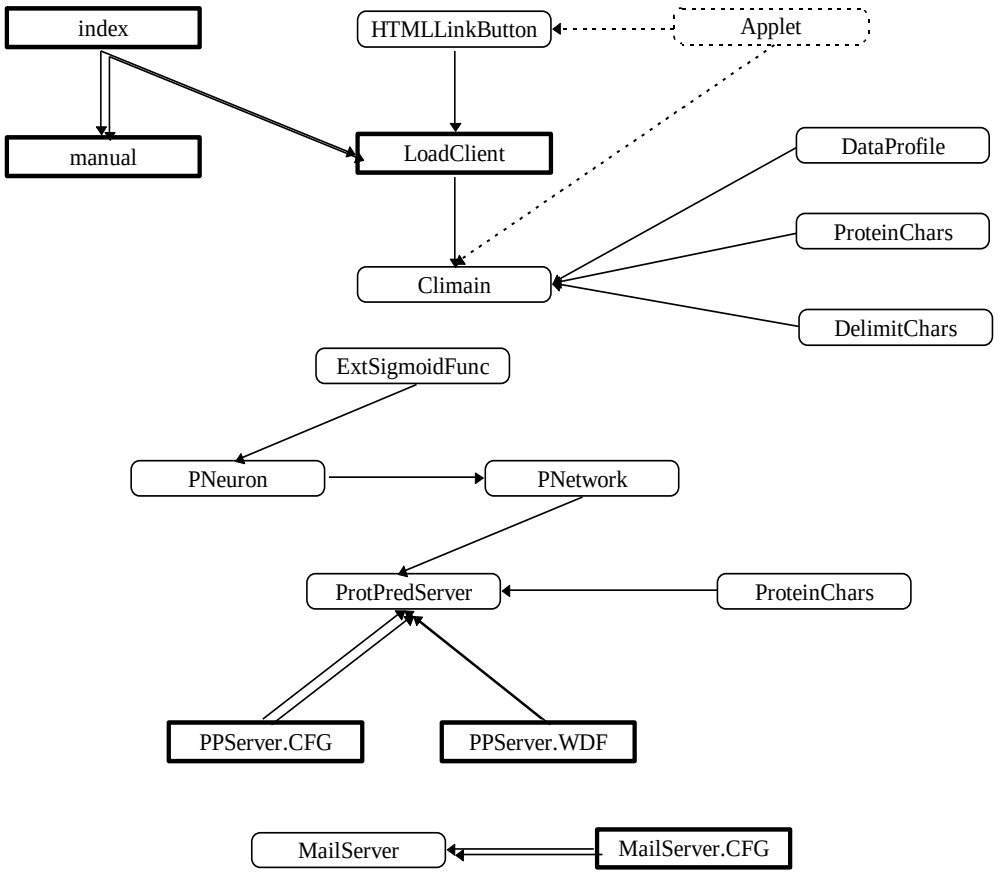
clscorrl	*class correlation section*
• corrls	*class correlation data*

clsnotes	*class general information section*
• lovalue	*lower "rejection" limit (freq. number)*
• hivalue	*higher "acceptance" limit (freq. number)*
• rejections	*total list of rejections & acceptances*

Module hierarchies and interconnections

The entire Pegasus system is composed of a large set of modules, containing application specific and general purpose Java classes, embedded Java applets, Web

pages (HTML) and other external data files. The extent of the whole set does not permit a detailed analysis of the infrastructure and usage of each one of these modules, but the names and position of most of them indicate their role in the Pegasus application model. Dotted lines show inheritance connection while solid lines indicate direct reference usage.



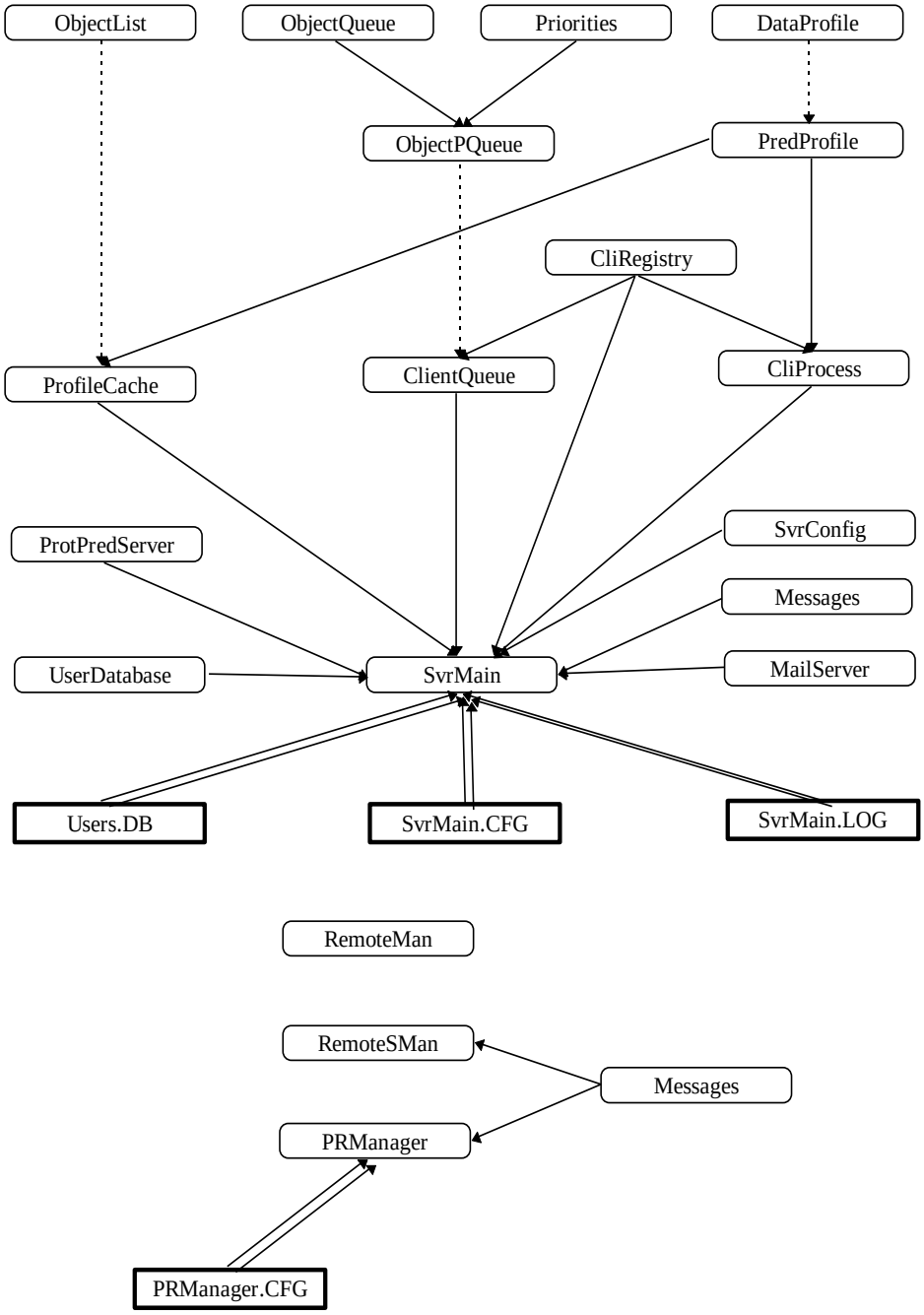


Figure 16: The complete Pegasus module hierarchy

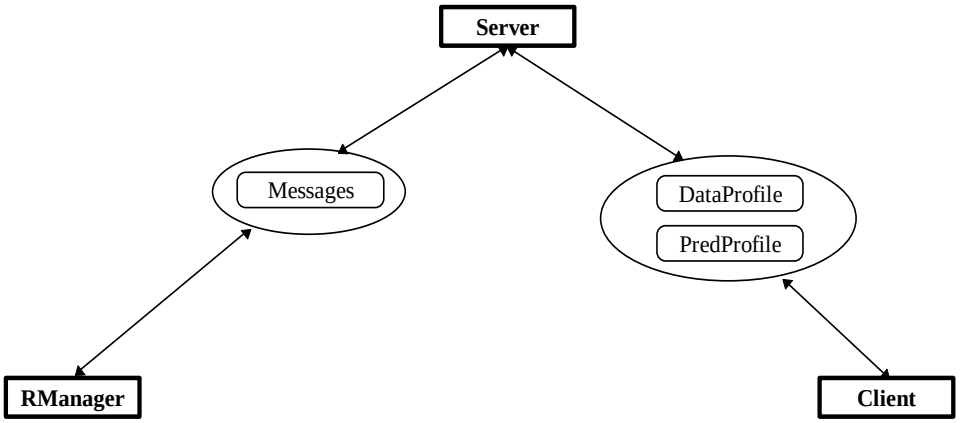


Figure 17: Module sharing in Pegasus model

System adaptability and property customization

For such a service to be usable and effective in various network and hardware environments, most of the operation properties had to be easily adjustable to the capabilities and limitations of the underlying system. Restrictions may be presented by various hardware or software factors, like the underlying hardware capabilities or the speed of the network connections, or functional factors like the administrative policy and service importance. Besides the main server, all internal subsystems and structures, depended more or less to any property of the underlying system (processing speed, available main memory, etc.) or other network service (SMTP server location) should be fully customizable in an easy and comprehensible way.

The Protein Prediction Server (PPS) is the heart of the entire Pegasus system, and one of the most demanding subsystem regarding system resources. However, due to the static nature (at runtime) of the neural network architecture used for the prediction service, little adjustments can be made to limit the memory resources used by it. On the other hand, multithreaded network activation can be configured from one execution thread for the whole activation to one thread per active neuron. Of course, the choice of the optimal configuration depends heavily on the underlying system (number of processors). Furthermore, as the research for an optimal neural network protein predictor continues, the network structure itself can be redefined and initialized by a new set of parameters, including network sizes, weight values, activation functions parameters, etc. All PPS properties are stored in a configuration local text file, easy for the service administrator to manipulate, while actual weight values set is stored in a separate file, in a compressed binary form.

The internal mailing service is another autonomous subsystem, depending heavily on external network services - SMTP server and/or local mailing scripts. Except the main property of the selection between the internal and external mailing services, an external configuration file is used to store current SMTP server location (host) and port number, as well as the filename of the currently active of the mailing shell script the should be used. The administrator can choose (off-line) between internal or external mailing, the network's SMTP server connection parameters and the return address for messages that fail to reach a specific client. It should be noted that the return address is used only with SMTP procedures, as the return address for external mailing scripts defaults to the current user (operator) that executes them in the local system's shell.

The main Pegasus server itself deploys a simple configuration text file, used to store various properties like the basic port number, the logging filename, the maximum number of cache buffers and a set of boolean fields, defining the active status of the TTY terminal interactivity, the logging services, the cache usage, the queue and user database usage, etc. The file is a simple key/value pair list, easily handled by the service administrators during off-line periods. As the current security model for the system is fairly loose, no severe administration commands (manipulation of server properties at runtime) can be issued remotely, except the shutdown command.

Acting both as a command-line interpreter and an interactive shell program, the Remote Manager can either retrieve its configuration parameters on each execution by the user or reading the default values from a local configuration file. This text file contains basic information regarding Pegasus server current location (host) and port number, as well as a boolean option indicating whether the program should execute only in command-line mode (effective for automatic scripting) or resume in interactive mode when command-line parameters are inadequate. Of course, connection parameters should be consistent with the current Pegasus system configuration, that is the Remote Manager should be aware of the location of the main server prior to execution, as the two programs (server, manager) act independently as stand-alone network applications.

Security Issues

Although the nature of the service does not require a strict security scheme for the transactions and information storage, the introduction of a remote manager for proper server identification and shutdown proposed some security concerns regarding the integrity of the server itself.

As the overhead of data encryption and certification mechanisms was not justified for such an open service, it was decided that the current version of Pegasus should not implement a fully functional remote manager, other than dealing with server status query and shutdown. All other properties regarding internal services and process should be configured statically via a set of external configuration files, in a presumably secure file system. The issue of integrity and validation of the small set of remote commands should be dealt within the PCP protocol.

PCP is a context-sensitive protocol, used both for client serving and remote management. The header information dictate the content type and the way that the packet should be analyzed. If the client "request indicator" is set, the packet contains normal request data that are forwarded to the internal subsystems of the service. On the other hand, if the "remote command" indicator is set, the data packet is parsed and the remote command ID is handled directly by the main server, transmitting acknowledge/status information or issuing a general shutdown. To ensure a basic validation mechanism, the remote command packets contain a fairly long identification code like a serial number, which is checked by the server every time a new command arrives. The message set is implemented as a separate object module, available both to the server and the remote manager program, and it can be modified (in the programming level) as frequently as it is needed. As all communication sessions provide embedded compression mechanisms, connections cannot be directly intercepted by eavesdropping, as in most network services using simple text-context protocols. Furthermore, the secrecy of the identification codes in the storage level can

be ensured by a secure file system, while the applet client uses a subset of the message set, regarding client requests only.

The security level of the current Pegasus system is surely not enough for a strict access scheme, but quite enough for such an open, publicly available network service. The nature and concept of the entire application does not justify a more strict access scheme, as most commonly used network services do not do so. However, the object-oriented and completely modular architecture of the system and the Java programming language itself, ensure the easy transition from an open (securely loose) public service to a closed (securely strict) limited access specialized service.

4 Technical Reference

4.1 General Information

The Pegasus application is a client-server application, implementing an on-line service for amino-acid sequence analysis and protein prediction. The analysis included various statistical properties of the sequence data, while the prediction is conducted by multiple artificial neural networks, embedded as internal server subsystems. The final software package implements a complete prediction service, accessible via the World Wide Web, easily adapted on any Java-enabled operating system, efficiently utilizing the underlying hardware in any case.

The final package is the result of gradual development and pilot programs. During most stages of the development process, the service was utilized and tested both for usability and proper functioning, by a wide range of user types (novice users, system administrators, molecular biologists, etc.). Currently, the service is under the final evaluation stages and documentation, and it soon will be activated in a permanent host machine, available to the entire Internet community.

4.2 Environment Reference

The following list contains detailed technical information about the software and hardware environments involved in the system's development and testing process.

- Hardware environments:
  1. SunSPARC (station/4, station/5, IPX, Xterminal), with SunOS 5.5.1 (Generic) and Solaris 2.4 (classic X Window system, CDE version 1.0.2).
  2. IBM-PC compatible (amd486DX2, i486DX, intel Pentium, PentiumPro and Pentium II processors), with Microsoft Windows 95, 98 and NT (workstation and server version 4.0), Linux (RedHat 4.2).

- Software environments:
  1. Sun's Java Development Kit, version 1.0.2 and 1.1.4, for SunSPARC and Intel based systems.
  2. Microsoft Visual J++, version 1.0 for MS-Windows 95.
  3. Sun's Java Workshop, version 1.0 for SunSPARC based systems.

4. Sun's standard C compiler, SunOS 5.5.1 version.
5. GNU C/C++ standard compiler for Linux and MS-DOS systems.
6. Borland C++, version 3.0 and 4.5 for MS-DOS systems.
7. GNU Java compiler and interpreter (Kaffe) for Linux.

- Testing environments:
  1. SunSPARC (station/4, station/5, IPX, Xterminal) machines on local Ethernet (10MBps), using Sun's JDK (1.0.2 and 1.1.4), Netscape Navigator 2.02 and 3.01.
  2. IBM-PC (486, Pentium, PentiumPro and Pentium II based systems) machines on local Ethernet and dynamic PPP/SLIP connections, using MS-Windows 9x/NT (w/s and server versions 4.0) and Linux (RedHat 4.5), Sun's JDK (1.0.2 and 1.1.4 for MS-Windows 32-bit and Linux), MS-Explorer 3.0/4.0, Netscape Navigator 2.02/3.0x, GNU Java tools for Linux (Kaffe, RedBaron).

4.3 TESTING AND PERFORMANCE DATA

Due to the nature of the Java programming language, the Pegasus software package can be easily adapted and utilized in almost any computing environment implementing a version of the standard Java Runtime Environment (JRE). The server can be deployed on any networked machine with an HTTP daemon, dedicated or not, for the access and fetching of the client package (HTML pages and embedded program applet). The client can be hosted on any Java enabled Web browser or the AppletViewer utility program, contained on every version of the Sun's JDK. The remote manager can be hosted on any Java enabled machine, supporting an implementation of the standard JRE for stand-alone Java applications.

Although thorough testing and evaluation is still conducted, the current version of the Pegasus package has been tested on various operating system and network environments, running on commonly used hardware. Specific efficiency factors depend heavily on the provider's implementation of the JRE, however the hardware independent nature of Java permits a generic evaluation about process and -especially- storage management requirements.

Storage requirements

The client (applet) program allocates about 50K in the JRE memory space, including code and data structures, but excluding GUI components, as the actual elements are allocated directly on the local operating system's resources. The remote manager is perhaps the most lightweight program of the package, requiring a total of 36K of main memory. The server program is, of course the largest module, containing various subsystems and external data storage. The memory requirements depend on the current configuration, especially the size of the cache pool and the user database. The PPS service requires about 400K for all networks, plus an additional 260K of external weight value data in a compressed binary form. The rest of the server's core and subsystems take about 80K in addition, while each user database entry adds about 0.5K and each new profile entry in cache adds about 2K-10K (depending whether it is a simple or windowed analysis).

Networking requirements

Network packets vary in size, as the transmitted profile's data fields during each session are filled gradually, but in general each session contains about two transactions, each containing the transmission of two full profiles in a compressed binary form. Thus, network load can be estimated roughly (using common text compression factor with LZ-compatible methods) about 15K-20K at most.

Process requirements

The requirements in process time can be referenced only in conjunction with specific hardware configurations, indicating only a generic quality feature of the system and not a quantitative measurement.

A sample case study (typical remote user) is this:

- **H/W**: amd486DX4/100MHz on VESA Local Bus architecture, with 24MB main memory (70ns DRAM), connecting with the local (typical) Ethernet network via dynamic PPP, using an internal (V42bis/MNP4) modem at 14.4Kbps.
- **S/W**: MS-Windows 95 (4.00B) with Netscape Navigator 3.04 and/or MS-Internet Explorer 3.0.

The server is installed on the local system, but the connection is made through the local network (not locally on the same machine), while all the programs (the server program and the Web browser with the client applet) run concurrently on the same machine.

Extensive processing measurements showed that each session on this particular machine required some 0.3 seconds of processing time for the prediction plus 10%-30% for data formatting. Cache hits reduce the overall time by 100%-150%, requiring only a clone function and limited data formatting. Remote commands cannot be cached, but the relatively primitive nature of this session requires the 10%-15% of the normal time.

Server initialization depends heavily on the PPS service, as it constructs the ANN structures and reads the initial weight values from an external file. On the particular machine, initialization time measurements averaged to 11.5 seconds, plus 1.5-2 seconds for the entire server program (including a user database of 20 records and non-volatile cache buffers dynamically allocated).

The PPS service was tested both as a stand-alone ANN environment and integrated subsystem to evaluate performance under various concurrent activation schemes. Thorough tests have been conducted only on the particular host configuration as described above, that is a single CPU machine, so no reliable conclusions can be derived regarding the general behaviour of the system in a pure multiprocessing environment. It should be noted that testing concurrent activation schemes (per network and per individual neuron) produced significant overhead due to thread handling functions, nearly 150% of the normal processing time. The overhead factor is caused by the intermediate generic interface between JRE thread functions and the real operating system's kernel routines. The thread creation, activation and destruction overhead is added to the total processing load of the single CPU, creating delay times directly proportional to the total number of threads used during the networks activation procedures. However, even these delay times are far less than the initialization and execution times for any external prediction program - application or

CGI (measured, in the first version, at about 45 seconds on the same host machine). Although thread handling functions in the JRE are proportional to the overall network activation time, it is expected that process time in a pure multiprocessing environment should be much smaller than in any single CPU machine.

5 Discussion

5.1 Conclusions

The Pegasus was designed and implemented as a modular and adaptable network service for the entire Internet community and especially as a useful analysis tool for molecular biologists. So far, only previous versions of the applications have been deployed for free use, mainly for evaluation and testing purposes, and not for long periods of time. Current version is to be deployed permanently for the first time, probably in a dedicated server machine in Athens, Greece.

People that have used the Pegasus protein prediction system, mainly molecular biologists and genetic engineers, have shown an increasing interest for such a service, not only regarding the particular gene analysis area (protein prediction), but on Web based analysis tools in general. Until now, all similar computer assisted gene analyses had to be conducted in various application specific tools, usually implemented for a limited range of hardware or software environments, mainly due to the excess resource requirements of such applications. Furthermore, most of these tools were limited in performing various calculations of some statistical properties about amino-acid sequence data (frequencies, base distributions, Fourier transformations, etc.), while difficult problems like protein prediction or stereoscopic analysis were usually the subject of more sophisticated AI or ANN processing models.

Since the prediction service has been deployed in a limited scale for evaluation purposes, many requests have been sent by people all over the world. The comments and suggestions of test users have often been the reason for implementing specific features like batch processing or user classification. The overall image of the service in its current form was commented as efficient and adequate to the requirements of the users, including skilled personnel that used similar gene analysis applications.

5.2 Future Improvements

Although current version can be considered complete in many ways, there are still areas and features that can be further extended or improved, not only regarding the functionality and efficiency of the service, but also the overall application architecture and design.

The main areas concerning design or implementation improvement are the security model, the communication protocol, the remote management and the application distribution model. All these issues are the main development goals for the next version of Pegasus, currently under design.

Security model

The fairly simple security model implemented in the current version of Pegasus, both for data communications and data storage, can not be sufficient for a more strict

service access and data secrecy scheme. The general security model includes data source authentication, data encryption and secure data storage.

The 1.1.4 release of the Java API introduced an embedded security model, including certification methods, key management and generic encryption framework. Certification includes data source authentication using digital signatures created by symmetric or asymmetric cryptosystems. Key management and generic data encryption framework proposes an efficient way to introduce a transparent security layer in all data transactions, using the commonly used data streaming scheme implemented throughout the Java API classes. The user can add new features and algorithms to implement customized cryptosystems or key providers, extending the functionality of the basic Java security model.

Communication security can be ensured by exploiting the Secure Socket Layer (SSL) standard in combination with digital certificates, as long as this scheme is supported by the operating system. Otherwise, embedded cryptosystems can be implemented directly by the application programs, including access validation and communication data encryption.

Data storage is a security concern mainly for the server, as the client and remote manager do not feature any major local file transactions. Server's local data files can be protected either by a secure file system or by implementing encryption techniques similar to the ones used for data communications over the network.

Communication protocol

As part of the overall security model, the protocol defines the entire access scheme for the client and for the remote management programs. A strict security model should implement a secure transaction protocol, including access validation and certification stages, along with secure session initialization and closing procedures. Of course the most efficient way to implement a secure protocol is to use integrated services of a secure operating system, like Kerberos, however the introduction of digital certification and advanced cryptosystems in the application level should ensure a sufficient security level for system like Pegasus.

Remote management

In order to provide a complete remote management system, security model and service integrity is the first issues to be ensured. The introduction of a relatively high security level can permit the implementation of a complete remote management program, including full server administration like operation properties manipulation (cache buffers, execution pool, priority queue) or on-line user registry transactions in the ACL database. Such an administration tool might justify an efficient GUI environment, a feature not necessary for the current -simple- version of the remote management program.

Application distribution model

Pegasus is a pure client/server application, as most of the commonly used network services. Although this model has been successfully used for a wide range of applications, it introduces some limitations for a completely distributed architecture. The control remains in a central node of the network, making the whole system vulnerable to network or server program failures. A more complete distribution model should be able to provide multiple access points for the service, probably deploying many servers in different machines in the network. Communication protocols should

be adjusted properly to provide an efficient method for the competing clients to access one of the available service entry points, and also to deal with unexpected session shutdown and recovery situations. Many of these features are provided by the Remote Method Invocation (RMI) model introduced by the core Java API, which is a major extension of the classical Remote Procedure Call (RPC) scheme in an pure object oriented computing environment. Implementing some additional features in the standard RMI model, like session recovery and automatic module distribution via agent programming, a complete loosely coupled parallel processing environment should be easily constructed in the application level. In that case, client programs would be full featured stand-alone applications, activating on users demand and automatically connecting to one of the service entry points as processing network agents.

6 Related work

Protein prediction via amino-acid analysis is one of the main topics in gene classification procedures. Much research has been done, not only regarding the protein prediction problem, but also stereoscopic analysis of proteins and DNA sequences. The proposed methods include both analysis of statistical properties of the sequences and advanced artificial intelligence and heuristic techniques.

The prediction system implemented by the Pegasus service is based on a similar stand-alone application, developed in the NEURAL environment, written entirely in ANSI C. NEURAL is the product of the RDP-14-DHMO-1 project (RDP 14 DHMO 1: "Neural Networks", Neural Network Laboratory - Institute of Informatics and Telecommunications (NNL-IIT), NCSR "Demokritos") by Dimitris Karras, a complete ANN simulation environment, implementing some of the most efficient and commonly used architectures and algorithms. The protein prediction ANN architecture and configuration were specified after extensive research and tests. The result was the definition of three efficient configurations for amino-acid sequence analysis, based on amino-acid frequencies. The first version of Pegasus (Tomcat) used a simplified version of the Neural environment as an external program, configured for the protein prediction ANN set, while the current version implements the same architecture and configurations as an internal subsystem. Research is continued both to identify more efficient ANN sets for protein prediction and to provide new solutions to similar problems related to amino-acid analysis procedures.

One of the most commonly used applications for gene identification in amino-acid sequences is the "Gene Recognition Analysis Internet Link (GRAIL)". It is an XWindow application for SunSPARC systems, developed by a group of scientists at ORNL as part of the Genome Project Initiative. Using remote connections to on-line protein databases and central analysis servers, it is used to analyze about two million DNA sequences every month by molecular biologists throughout the world. Analysis is produced using pattern recognition techniques.

An on-line system similar in function with the Pegasus service is the "PredictProtein" by Protein Design Group, European Molecular Biology Laboratory, Heidelberg, Europe. This application introduces a different approach to the problem, providing a World Wide Web service for sequence analysis using various coding measures. The user enters the input data and parameters in a static HTML form and

the analysis is conducted by multiple CGI programs on the Web server machine. Output results are introduced in various text formats used for generic categorization or by other analysis programs.

In the field of stereoscopic analysis, much research has been conducted both in the area of artificial intelligence and typical grammar definition. Artificial intelligent techniques include the exploitation of various aspects and abstract characteristics of proteins, regarding the relations between chemical composition and stereoscopic form of protein molecules. On the other hand, a typical grammar should be able to produce the complete stereoscopic model of any protein, given its chemical composition. Although some initial specifications have been reached by researchers, both these approaches (as various ANN models, also) have not yet been able to provide an efficient framework for the solution of the general issue.

7 References

The following references include resources used both for ANN research and application development. There are also references to related organizations, on-line services and company products.

[01] «Data Networks» (2nd/ed.)
Dimitri Bertsekas, Robert Gallager
Prentice-Hall International Editions - 1992

[02] «Internetworking with TCP/IP, volume I» (2nd/ed.)
Douglas Comer
Prentice-Hall International Editions - 1991

[03] «Internetworking with TCP/IP, volume II»
Douglas Comer, David Stevens
Prentice-Hall International Editions - 1991

[04] «Internetworking with TCP/IP, volume III»
Douglas Comer, David Stevens
Prentice-Hall International Editions - 1994

[05] «Network flows - Theory, Algorithms and Applications»
Ravindra Ahuja, Thomas Magnanti, James Orlin
Prentice-Hall - 1993

[06] «Data Communications and Distributed Networks» (3rd/ed.)
Uyless Black
Prentice-Hall Int. Ed. - 1993

[07] «Data Networks - Concepts, Theory and Practice»
Uyless Black
Prentice-Hall Int. Ed. - 1989

[08] «Solaris 2.4 - OpenWindows Reference Manual»
SunSoft
Sun Microsystems - 1994

[09] «Solaris 2.4 - SunOS Reference Manual: Section 3. Library Routines (N-Z)»
SunSoft
Sun Microsystems - 1994

[10] «Solaris 2.4 - Miltithreaded Programming Guide»
SunSoft
Sun Microsystems - 1994

[11] «Solaris 2.4 - TCP/IP Network Administration Guide»
SunSoft
Sun Microsystems - 1994

[12] «Handbook of Human-Computer Interaction»
Martin Helander
Elsevier Science Publishers B.V. - 1991

[13] «Engineering the Human-Computer Inteface»
Andy Downtown
The Essex series in Telecommunication and Information systems
McGraw-Hill Book Company

[14] «Distributed Computing - A practical synthesis»
Amjad Umar
PTR Prentice-Hall - 1993

[15] «Protocols for Application Communication - A top-down guide to OSI, TCP/IP and SNA»
Keith Bearpark
McGraw-Hill Book Company - 1995

[16] «The SLIP/PPP Connection»
Paul Gilster
John Wiley & Sons, Inc. - 1995

[17] «TCP/IP Network Administration»
Craig Hunt
O'Reilly & Associates, Inc. - 1994

[18] «The Web server book»
Jonathan Magid, R. Douglas Matthews, Paul Jones
Ventana Press - 1995

[19] «DNS and BIND»
Paul Albitz, Cricket Liu

O'Reilly & Associates, Inc. - 1992

[20] «Netscape 2 Unleashed»
Dick Oliver
Sams.net Publishing - 1996

[21] «Software Engineering Handbook»
Series on Software Engineering and Technology
General Electric Company
McGraw-Hill - 1989

[22] «Software Engineering for large software systems»
B. A. Kitchenham
Elsevier Applied Science - 1990

[23] «Using Visual J++»
Mark Culverhouse
QUE Corporation - 1996

[24] «Java in a Nutshell»
David Flanagan
O'Reilly & Associates, Inc. - 1996

[25] «The comprehensive guide to Visual J++»
Danial Joshi, Ramesh Chandak
Ventana Comm. Group, Inc. - 1997

[26] «Writing Java Applets»
John Rodley
The Coriolis Group - 1996

[27] «Java Programming with Visual J++»
Martin Rinehart
M & T Books - 1997

[28] «Security in Computing»
Charles P. Pfleeger
Prentice-Hall International Editions - 1989

[29] «Distributed Systems - Concepts and Design» (2^{nd}/ed.)
George Coulouris, Jean Dollimore, Tim Kindberg
Addison-Wesley Publishing Company

[30] «Ασφάλεια Πληροφοριακών Συστημάτων»
Δημήτρης Αρ. Γκριζάλης
Ελληνική Εταιρία Η/Υ και Πληροφορικής (ΕΠΥ) - Αθήνα, 1989

[31] «Ασφάλεια Πληροφοριακών Συστημάτων - Εισαγωγική μελέτη θεμάτων ασφάλειας σε περιβάλλον κατανεμημένων πληροφοριακών συστημάτων»
Εργασία εξαμήνου στο μάθημα «Κατανεμημένα Συστήματα»
Χάρης Γεωργίου, Ηλίας Μακρής, Δημήτρης Παπαδόπουλος
Πανεπ. Ιωαννίνων, Τμ. Πληροφορικής - 1997

[32] «Μέθοδοι Εκπαίδευσης και Μοντέλα Τεχνητών Νευρωνικών Δικτύων»
Εργασία εξαμήνου στο μάθημα «Ανάλυση και Σχεδίαση Αλγορίθμων»
Χάρης Γεωργίου, Σπύρος Μελισσόβας, Δημήτρης Παπαδόπουλος
Πανεπ. Ιωαννίνων, Τμ. Πληροφορικής - 1995

[33] «Δίκτυα Υπολογιστών - Μετάδοση δεδομένων, Επικοινωνιακό Υποδίκτυο»
Θεόδωρος Αποστολόπουλος
Αθήνα, 1994

[34] «The Java Class Libraries: An annotated reference»
The Java Series
Patrick Chan, Rosanna Lee
Addison-Wesley Publishing Company - 1997

[35] «The Java Application Programming Interface - Volume I»
The Java Series
Frank Yellin
Addison-Wesley Publishing Company - 1997

[36] «The Java Application Programming Interface - Volume II»
The Java Series
Frank Yellin
Addison-Wesley Publishing Company - 1997

[37] «Concurrent programming in Java: Design principles and patterns»
The Java Series
Doug Lea
Addison-Wesley Publishing Company - 1997

[38] «The Java Tutorial: Object-oriented programming for the Internet»
The Java Series
Mary Campione, Kathy Walrath
Addison-Wesley Publishing Company - 1997

[39] «The Java Language Specification»
The Java Series
James Gosling, Bill Joy, Guy Steele
Addison-Wesley Publishing Company - 1997

[40] «The Java Virtual Machine Specification»
The Java Series
Tim Lindholm, Frank Yellin

Addison-Wesley Publishing Company - 1997

[41] «Artificial Intelligence and Molecular Biology»
Lawrence Hunter
AAAI Press / MIT Press - 1993

[42] «Τεχνολογία Λογισμικού - Τόμος Α'»
Εμμ. Α. Γιακουμάκης
Σταμούλης - 1994

[43] «Τεχνολογία Λογισμικού - Τόμος Β'»
Εμμ. Α. Γιακουμάκης
Σταμούλης - 1994

[44] «Τεχνικές για την Ανάλυση και Σχεδίαση Συστημάτων»
Ευγενία Μανωλοπούλου
Anubis - 1994

[45] «Δομές Δεδομένων - Τόμος Α'» ($2^{η}$/έκδ.)
Μανωλόπουλος Ιωάννης
ArfOfText - Θεσσαλονίκη, 1992

[46] «Δομές Δεδομένων - Τόμος Β'» ($2^{η}$/έκδ.)
Μανωλόπουλος Ιωάννης
ArtOfText - Θεσσαλονίκη, 1991

[47] «Δομές Δεδομένων, Αλγόριθμοι - Τόμος Α'»
Μανόλης Λουκάκης
Θεσσαλονίκη, 1993

[48] «Σύγχρονα Λειτουργικά Συστήματα - Τόμος Β': Κατανεμημένα Συστήματα»
A. S. Tanenbaum
Παπασωτηρίου - Αθήνα, 1994

[49] «Συστήματα Υπολογιστών, Τόμος ΙΙ: Λειτουργικά Συστήματα» ($3^{η}$/έκδ.)
Ι. Κ. Κάβουρα
Κλειδάριθμος - Αθήνα, 1995

[50] «Java Programming Basics»
Edith Au, Dave Makower, Pencom Web Works
MIS Press - 1996

[51] «Hacking Java: The Java Professional's Resource Kit»
Mark Wutka
QUE Corporation - 1997

[52] «Data Structures Using C»
Aaron Tenenbaum, Yedidyah Langsam, Moshe Augenstein

Prentice-Hall International Editions - 1990

[53] «User Interface Design» (2nd/ed.)
Kevin Cox, David Walker
Prentice-Hall - 1993

[54] «Operation System Concepts» (3rd/ed.)
Abraham Silberschatz, James Peterson, Peter Galvin
Addison-Wesley Publishing Company - 1990

[55] «Τεχνητά Νευρωνικά Δίκτυα - Ι»
Γ. Παπαδουράκης, Γ. Μπέμπης
Ενημερωτικό Άρθρο, Τεχν. Χρον. Β, 1991, Τόμος 11, Τεύχος 3

[56] «Τεχνητά Νευρωνικά Δίκτυα - ΙΙ»
Γ. Παπαδουράκης, Γ. Μπέμπης
Ενημερωτικό Άρθρο, Τεχν. Χρον. Β, 1992, Τόμος 12, Τεύχος 1

[57] «Δίκτυα Υπολογιστών» - Πανεπιστημιακές Σημειώσεις
Στράτος Θ. Πάσχος
Πανεπ. Ιωαννίνων, Τμ. Πληροφορικής - 1995

[58] «Object Oriented Programming»
Tim Rentsch
BYTE Magazine, Aug. 1981, vol. 6, No. 8

[59] «Data abstraction, data encapsulation and object-oriented programming»
A. Toni Cohen
SIGPLAN Notices, V19 #1, Jan. 1984

[60] «Προηγμένος Προγραμματισμός» - Πανεπιστημιακές Σημειώσεις
Δημήτρης Ι. Φωτιάδης
Πανεπ. Ιωαννίνων, Τμ. Πληροφορικής - 1994

[61] «Algorithms and Data Structures»
Niklaus Wirth
Prentice-Hall International, Inc. - 1986

[62] «Δομές Δεδομένων» - Πανεπιστημιακές Σημειώσεις
Νικόλαος Γλυνός
Πανεπ. Ιωαννίνων, Τμ. Πληροφορικής - 1994

[63] «Borland C++ 3.0 - Library Reference»
Borland International
Borland International - 1991

[64] «Βασικές Αρχές Γλωσσών Προγραματισμού» (2η/έκδ.)
Ellis Horowitz
Κλειδάριθμος - 1993

[65] «Java Quick Reference»
Michael Afergan
QUE Corporation - 1996

[66] «An introduction to genetic analysis» (3rd/ed.)
David Suzuki, Anthony Griffiths, Jeffrey Miller, Ri Lewontin
Freeman and Company - 1986

[67] «Fundamentals of genetics»
Peter Russel
HarperCollins - 1984

[68] «Genetics» (2nd/ed.)
Robert Weaver, Philip Hedrick
WCB - 1992

[69] «Creating Web applets with Java»
David Gulbransen, Kenrick Rawlings
Sams - 1996

[70] «Java programmer's library»
Suleiman Lalani, Kris Jamsa
Jamsa - 1996

[71] «Computer Networks» (2nd/ed.)
Andrew S. Tanenbaum
Prentice-Hall - 1989

[72] «Network Security»
Richard Baker
McGraw-Hill - 1994

[73] STD 47 / RFC 1055: «A nonstandard for transmission of IP datagrams over serial lines: SLIP»
J. Romkey - 1988

[74] RFC 1144: «Compresing TCP/IP packets for low-speed serial links (CSLIP)»
V. Jacobson - 1990

[75] RFC 1134: « The Point-to-Point Protocol: A proposal for Multi-Protocol Transmission of datagrams over point-to-point links»
D. Perkins - 1989

[76] STD 51 / RFC 1661: « The Point-to-Point Protocol (PPP)»
W. Simpson - 1994

[77] FYI0008 / RFC 1244: «Site Security Handbook»

P. Holbrook, J. Reynolds - 1991

[78] STD 5 / RFC 791: «Internet Protocol (IP)»
DARPA Internet program protocol specification - 1981

[79] STD 6 / RFC 768: «User Datagram Protocol (UDP)»
J. Postel - 1980

[80] STD 7 / RFC 793: «Transmission Control Protocol (TCP)»
DARPA Internet program protocol specification - 1981

[81] STD 43 / RFC 1042: «A standard for the transmission of IP datagrams over IEEE 802 networks»
J. Postel, J. Reynolds - 1988

[82] STD 53 / RFC 1939: «Post Office Protocol, version 3 (POP3)»
J. Myers, M. Rose - 1996

[83] RFC 1856: «The Opstat client-server model for statistics retrieval»
H. Clark - 1995

[84] Java White Papers: «Java Language Overview»
JavaTeam, SunSoft
Sun Microsystems - 1997

[85] Java White Papers: «The Java Language Environment White Paper»
JavaTeam, SunSoft
Sun Microsystems - 1997

[86] Java White Papers: «Java Security»
JavaTeam, SunSoft
Sun Microsystems - 1997

[87] Java White Papers: «The Java Platform White Paper»
JavaTeam, SunSoft
Sun Microsystems - 1997

[88] Java White Papers: «Java OS™: A Standalone Java Environment»
JavaTeam, SunSoft
Sun Microsystems - 1997

[89] ISO/IEC 9126: «Information technology - Software product evaluation - Quality characteristics and guidelines for their use»
ISO/IEC 9126: 1991(E)

[90] ISO/IEC 9126: «Software Quality Assurance - Questionnaire»
ISO/IEC 9126: 1991(E)

[91] «Assessment of protein coding measures»

James W. Fickett, Chang-Shung Tung
Nucleic Acids Research, 1992, Vol. 20, No. 24
Oxford University Press - 1992

[92] «Client-Server» - Άρθρα σχετικά με την εξέλιξη του μοντέλου πελάτη-εξυπηρέτη, τα ανοικτά συστήματα και τις σύγχρονες εφαρμογές.
Περιοδικό CHIP (ελληνική έκδοση), Τεύχος 14, Ιούνιος 1995

[93] ANSI/IEEE std. 830: «Έγγραφο Περιγραφής Απαιτήσεων Λογισμικού»
ANSI/IEEE std. 830-1984

[94] ANSI/IEEE std. 1016: «Έγγραφο Περιγραφής Σχεδίου Λογισμικού»
ANSI/IEEE std. 1016-1987

[95] «Μαθήματα Γενικής Βιολογίας» (ΙΓ'/έκδ.)
Κ. Κρίμπα, Ι. Καλοπίση
Οργανισμός Εκδόσεων Διδακτικών Βιβλίων - 1989

[96] «Οργανική Χημεία» (2η/έκδ.)
Νίκος Στ. Αναστασιάδης
Gutenberg - Αθήνα, 1989

[97] «An integrated Environment for Experimenting with Feedforword Neural Network Algorithms»
S. J. Perantonis, D. A. Karras
Artificial Neural Network Laboratory Institute of Informatics and Telecommunications
NCSR Demokritos - 1994

[98] «Πρόβλεψη γονιδίων σε ακολουθίες DNA» - Σεμινάριο
Διάλεξη σχετικά με το πρόβλημα αναγνώρισης γονιδίων σε αλυσίδες νουκλεοτιδίων μορίων DNA, από τη Άρτεμη Χατζηγεωργίου.
Πανεπιστήμιο Ιωαννίνων, Τμήμα Πληροφορικής - Οκτώβριος 1996

[99] «Εισαγωγή στην τεχνητή νοημοσύνη και τα Έμπειρα Συστήματα» - Τεύχος Α', 2η/έκδ.
Σπύρος Τζαφέστας
Εθνικό Μετσόβιο Πολυτεχνείο - Αθήνα, 1996

[100] «Η γλώσσα προγραμματισμού C» (2η/έκδ.)
Brian W. Kernighan, Dennis M. Ritchie
Prentice Hall Software Series
Κλειδάριθμος - 1990

[101] «Το περιβάλλον προγραμματισμού UNIX»
Brian W. Kernighan, Rob Pike
Prentice Hall International, Inc.
Κλειδάριθμος - 1989

[102] «Το βιβλίο της C++»
Α. Σταμούλης
Learning Plan - 1992

[E01] «LBNL Human Genome Center»
Lawrence Berkeley National Laboratory - Human Genome Center
http://www-hgc.lbl.gov

[E02] «Maize Genome Database WWW Server»
Maize Genome Database Project
http://www.agron.missouri.edu

[E03] «The PredictProtein server»
European Molecular Biology Laboratory
http://www.embl-heidelberg.de/predictprotein/predictprotein.html

[E04] «GRAIL - Gene Recognition Analysis Internet Link»
Oak Ridge National Laboratory - Computational Sciences
http://www.ornl.gov/ornl94/computational.html

[E05] «Eugenides Foundation on the WWW»
Ευγενίδιο Ίδρυμα - Δημόσια Βιβλιοθήκη
http://www.eugenides_found.edu.gr

[E06] «Sun Microsystems»
Sun Microsystems, Inc. - Palo Alto, USA
http://www.sun.com

[E07] «Free On-line Documentation for Sun Products»
Sun Microsystems, Inc. - Palo Alto, USA
http://docs.sun.com

[E08] «Java Computing»
Sun Microsystems, Inc. - Palo Alto, USA
http://java.sun.com
http://www.javasoft.com

Acknowledgements

The authors wish to thank all their cooperatives, researchers and test users that contributed in many ways in the design and development of the Pegasus system. We hope that our work will help your own efforts as well as you did for ours.

Trademarks