UNIVERSIDAD NACIONAL AUTÓNOMA De México

FACULTAD DE INGENIERÍA

"DISEÑO DE UN SERVIDOR PARA LA PREDICCIÓN DE LA ESTRUCTURA DE LAS PROTEÍNAS"

T E S I S

QUE PARA OBTENER EL TÍTULO DE:
INGENIERO EN COMPUTACIÓN

P R E S E N T A :
CLAUDIA MORALES ALMONTE

DIRECTOR DE TESIS:

Dr. GABRIEL DEL RÍO GUERRA

CIUDAD UNIVERSITARIA, MÉXICO, D.F., 2005

Agradecimientos

Quiero agradecer a Dios por cada día en el que nos llama a vivir la alegría de nuestra vocación.

Gracias a mis padres, Francisco Morales Meléndez y Ángeles Almonte Sánchez, por todo su apoyo, sacrificio y confianza, por el amor y la fe que me transmitieron; a mis hermanos y a toda mi familia.

Gracias a Adelia, a toda la comunidad de Misioneras Seculares Scalabrinianas y a
P. Gabriele, por todo lo que han compartido conmigo.

Grazie a tutto "il corpo" di cui siamo parte come umanitá e che ci rivela la bellezza della diversità nell'unità. Grazie perché camminiamo insieme su questa strada.

Agradezco también al Físico Raymundo Hugo Rangel Gutiérrez, pues esta tesis no se hubiera podido realizar sin su apoyo y sus consejos.
Gracias por los conocimientos que me transmitió y por el impulso a incursionar en nuevos campos, como lo es la bioinformática.

Gracias a mi director de tesis,
Doctor Gabriel del Río Guerra,
por su dedicación y apoyo en la realización de este trabajo de tesis,
por la confianza y la motivación durante este tiempo.

Índice

Introducción

Con el Proyecto del Genoma Humano (PGH), cuyo propósito es determinar la secuencia completa del genoma humano, se crearon infraestructuras tecnológicas entre las que destacan nuevas herramientas de hardware y software destinadas a automatizar tareas, a procesar enormes cantidades de datos y a extraer la máxima información biológica y médicamente significativa. Con este proyecto surgió también un campo nuevo, la bioinformática.
El propósito de este campo es desarrollar la infraestructura que requiere la biología moderna (para esto aplica herramientas y métodos computacionales) y además interpretar los resultados obtenidos al aplicar estos métodos.

En las áreas de investigación la computadora juega un papel muy importante pues se vuelve un instrumento indispensable para manejar los datos que se generan durante la experimentación. En la biología uno de los problemas actuales, de gran importancia, es la predicción de la estructura de proteínas.

Existen diferentes métodos de predicción pero éstos no han tenido avances sustanciales en los últimos cuatro años, esto se debe, por una parte, a que los métodos de alineamiento y de evaluación de modelos no han mejorado.

El objetivo de este trabajo de tesis es diseñar e implementar un servidor que permita la evaluación continua de la predicción de la estructura tridimensional de las proteínas, usando para ello un nuevo método de predicción basado en el análisis de redes.

Contar con un servidor, en el contexto de Internet, permitirá dar un servicio a la comunidad internacional interesada en realizar predicción de la estructura de proteínas.

Durante el desarrollo de este trabajo se explica la metodología que se utilizó para cumplir con los objetivos planteados.

En el capítulo 1, *Antecedentes*, se introduce de forma general el método de predicción, además de algunos conceptos sobre proteínas y su estructura tridimensional.

En el capítulo 2, *Definición del proyecto*, se plantean los propósitos, requerimientos y objetivos del sistema.

En el capítulo 3, *Herramientas para el desarrollo del servidor*, se explican los requisitos que deben cumplirse para crear una aplicación Web y se definen las herramientas que se utilizaron en la aplicación.

En el capítulo 4, *Diseño del servidor*, se realiza el análisis y el diseño del sistema, se definen los paquetes que deben crearse para cada etapa de la predicción.

En el capítulo 5, *Implementación del servidor*, se muestra la interfaz gráfica del sistema y los paquetes creados para la aplicación.

En el capítulo 6, *Pruebas y resultados*, se explica como se probó el funcionamiento del sistema y los resultados obtenidos.

Capítulo 1

Antecedentes

El funcionamiento del servidor, desarrollado en este tema de tesis, se basa en un nuevo método, sustentado en el análisis de redes, para la predicción de la estructura de las proteínas.

En este capítulo se explica, de manera general: ¿qué son las proteínas? ¿cómo es su estructura tridimensional? ¿cuál es su importancia? y ¿cuáles son los métodos que existen para predecir la estructura de estas? Se explica también la importancia de las ciencias computacionales en la biología (la bioinformática), y el papel de la bioinformática en la predicción de la estructura de las proteínas.

La predicción, que ofrece el servidor, consiste en realizar un análisis de las proteínas modeladas como redes. El método para la predicción es largo y complejo, en este capítulo se describen las etapas, formadas por los algoritmos MIN (*Minimun Interacting Networks*) y NIM (la idea inversa de MIN).

1.1 PROTEÍNAS Y SU ESTRUCTURA TRIDIMENSIONAL

(véase apéndice A)

1.1.1 ¿Qué son las proteínas?

Una proteína es una cadena de aminoácidos unidos mediante un enlace peptídico (Figura 1.1).

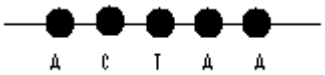

Figura 1.1 Secuencia de aminoácidos de una proteína.

La secuencia de aminoácidos (AA) de una proteína se representa como una cadena de símbolos pertenecientes a un alfabeto. El número de aminoácidos en la cadena representa su longitud.

Este alfabeto es un conjunto de letras, cada una de estas representa a un aminoácido diferente. Podemos definir este conjunto como sigue:

AA = { A,C,D,E,F,G,H,I,K,L,M,N,P,Q,R,S,T,V,W,Y }

Aminoácidos

Existen 20 aminoácidos y sus nombres se pueden abreviar por medio de tres letras o de una sola letra.

Así, el aminoácido alanina se abrevia Ala ó A, de la misma forma:

A = Ala (Alanina)	M = Met (Metionina)
C = Cys (Cisteína)	N = Asn (Asparagina)
D = Asp (Ac. Aspártico)	P = Pro (Prolina)
E = Glu (Ac. Glutámico)	Q = Gln (Glutamina)
F = Phe (Fenilalanina)	R = Arg (Arginina)
G = Gly (Glicina)	S = Ser (Serina)
H = His (Histidina)	T = Thr (Treonina)
I = Ileu (Isoleucina)	V = Val (Valina)
K = Lys (Lisina)	W = Trp (Triptófano)
L = Leu (Leucina)	Y = Try (Tirosina)

¿Cuál es la importancia de las proteínas?

Las proteínas son moléculas sintetizadas por los seres vivos y constituyen uno de los componentes fundamentales para la vida. Están involucradas en diversos fenómenos biológicos, incluyendo el crecimiento y la reproducción, entre otros (Branden, C., Tooze, J., 1991).

1.1.2 Estructura tridimensional de las proteínas

Los aminoácidos, a medida que van siendo enlazados, adquieren una disposición en el espacio, gracias a la capacidad de giro de sus enlaces, hasta llegar a la estructura tridimensional. A este proceso se le llama el plegamiento de las proteínas, la figura 1.2 muestra la estructura de una proteína.

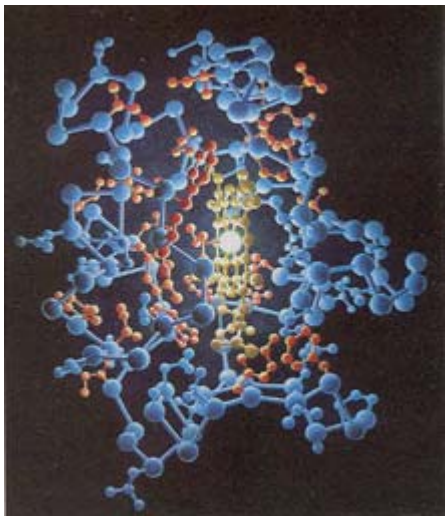

Figura 1.2 Estructura tridimensional de una proteína.

Se asume que las proteínas deben tomar una determinada conformación tridimensional estable para desempeñar su función biológica. Cuando una proteína pierde su estructura tridimensional nativa también pierde su función.

Conocer la estructura tridimensional de una proteína es esencial para entender cómo los factores genéticos (mutaciones) y los factores ambientales (por ej. pH, temperatura, concentración de sal, tratamientos químicos) pueden modificar las proteínas, reduciendo, destruyendo o elevando su actividad biológica.

Obtención de la estructura tridimensional mediante técnicas experimentales

Para obtener la estructura tridimensional de una proteína, es decir la posición en el espacio tridimensional de los átomos que la constituyen, se aplican algunas técnicas tales como la cristalografía de rayos X y la Resonancia Magnética Nuclear (RMN) (Branden, C., Tooze, J., 1991).

Estas técnicas se han aplicado para obtener las estructuras de miles de proteínas. Hasta el momento se han resuelto más de 26000 estructuras, mismas que están disponibles en bases de datos públicas (PDB - *Protein Data Base*).

1.2 PREDICCIÓN DE LA ESTRUCTURA DE LAS PROTEÍNAS

1.2.1 El problema de la predicción

La velocidad de obtención de las secuencias de las proteínas (secuenciación de genomas) sobrepasa la de la determinación de su estructura; se conocen aproximadamente 10^8 secuencias y 10^3 estructuras tridimensionales de proteínas.

Esta diferencia ha crecido exponencialmente en la última década, por lo que hasta el día de hoy resulta prohibitivo el poder determinar experimentalmente la estructura de las proteínas conocidas. Esta diferencia es el incentivo para desarrollar métodos de predicción de la estructura tridimensional de las proteínas (véase liga: bioSec).

La predicción de la estructura de las proteínas es uno de los problemas vigentes más importantes a los que se enfrenta la biología molecular. En términos muy simples, se puede formular como sigue: Dada una secuencia de aminoácidos de una proteína, ¿cuál es la estructura tridimensional asociada? (véase liga: bioPred)

1.2.2 Métodos de predicción

Se han desarrollado métodos que, basados en los conocimientos físicos, químicos, biológicos y con ayuda de herramientas computacionales, tratan de responder al problema de la predicción de la estructura de las proteínas.

Estos métodos de predicción se dividen en dos: los basados en la comparación de la secuencia de una proteína objetivo (o proteína problema) con aquellas secuencias cuya estructura tridimensional se conoce (métodos por homología), y los no comparativos (métodos *ab initio*) (véase liga: bioMétodos).

1) Métodos por homología

Existen estructuras que se repiten en diversas proteínas; basados en esta observación, muchos modelos teóricos de plegamiento buscan homología (similitud*)* con estructuras conocidas, para predecir la forma tridimensional de nuevas proteínas.

La predicción de la estructura de una proteína por homología es usualmente realizada mediante la identificación de una proteína de estructura conocida (proteína plantilla) que comparte *similitud* de secuencia con la secuencia de la proteína objetivo.

Las etapas necesarias para el modelado por homología, partiendo de la secuencia de la proteína objetivo, son básicamente cuatro:

a) Búsqueda de proteínas homólogas
El primer paso es la búsqueda de proteínas plantillas parecidas a las proteínas objetivo utilizando sistemas computacionales basados en similitud de secuencia.

b) Alineamiento de secuencias (véase apéndice B)
El objetivo de alinear dos secuencias de proteínas es encontrar la posición relativa de ambas, en la que se produzca un mayor número de coincidencias entre sus aminoácidos. Esta información es importante para inferir las relaciones (estructurales, funcionales o evolutivas) entre las secuencias.

c) Generación de modelos
La construcción de las coordenadas espaciales de la secuencia objetivo, se basa en el alineamiento de las secuencias y en las estructuras conocidas de las proteínas plantilla.

d) Elección y evaluación del modelo
Se requiere un sistema de puntaje para evaluar la calidad de las predicciones, en el caso de la predicción por homología el sistema de puntuación contiene el grado de conservación de la secuencia.
Después de haber generado los modelos debe elegirse uno, basados en algún criterio de decisión.

2) Métodos ab initio
Cuando la similitud entre la secuencia objetivo y la secuencia plantilla es demasiado baja no se puede aplicar con éxito el modelado por homología, pues el alineamiento no es significativo. En estos casos se emplean otros métodos como el *ab initio*, que se basa solo en la secuencia para realizar la predicción (emplea algoritmos de mecánica molecular y una función potencial). Los métodos *ab initio* a la fecha son computacionalmente costosos y su confiabilidad disminuye conforme el tamaño de la proteína aumenta.

Existe una gran variedad de herramientas que nos ayudan a realizar la predicción (la búsqueda de secuencias similares, el alineamiento y la generación de los modelos) (véase liga: bioHerram) y algunas de ellas están disponibles en la red (bases de datos públicas y software libre). Sin embargo aunque ha habido grandes avances en esta materia aún no se han alcanzado niveles de confiabilidad mayores al 80% en ningún método de predicción.

1.3 EL PAPEL DE LAS CIENCIAS COMPUTACIONALES EN LA BIOLOGÍA

En la biología, una ciencia donde se manejan grandes cantidades de datos, la computadora es esencial para el análisis de estos. Además de procesar datos, las computadoras son dispositivos poderosos en el análisis de sistemas descritos mediante modelos matemáticos, físicos, biológicos, etc.

La investigación que antes comenzaba en el laboratorio ahora empieza en las computadoras. Los científicos buscan información que pueda sugerirles nuevas hipótesis en las bases de datos. En la actualidad existen sofisticados laboratorios tecnológicos que permiten colectar datos más rápido de lo que se pueda interpretar.
Las computadoras y la *World Wide Web* están cambiando rápidamente la investigación biológica. En las últimas dos décadas es común el almacenamiento de datos biológicos (secuencias, publicaciones, etc.) en bases de datos publicas, estas bases han crecido exponencialmente.
La biología computacional ha existido por décadas, sin embargo se ha mantenido al margen de las ciencias biológicas. Con el desbordamiento de los datos biológicos, producidos por los esfuerzos de la genómica, se hizo necesario contar con la ayuda de herramientas de cómputo para el análisis de los datos genómicos, de esta manera surge un nuevo campo: la Bioinformática (Gibas, C., Jambeck, P., 2001).

1.3.1 Bioinformática

La Bioinformática (la convergencia de las Biociencias con las Ciencias de la Computación*)* es una disciplina científica emergente en rápido crecimiento que se encarga de todos los aspectos relacionados con el almacenamiento, mantenimiento y procesamiento de datos biológicos, mediante la aplicación de técnicas y herramientas matemáticas, estadísticas, químicas, físicas e informáticas, con el propósito de extraer información útil. Está dedicada

al desarrollo y aplicación de métodos de cómputo y analíticos para manejar la información (véase liga: bioinfNCBI).

La cantidad de datos de origen biológico (secuencias, estructuras, expresión de genes) es actualmente tan grande que el análisis manual es imposible. Una forma de analizarlos es mediante el uso de programas computacionales capaces de:

Organizar datos y hacerlos accesibles
Comprender procesos celulares complejos
Predecir fenómenos biológicos de múltiples variables

Los bioinformáticos deben tener los conocimientos biológicos y computacionales que les permitan dar solución a problemas de tipo biológico elaborando sus propias herramientas (Claverie, J-M., Notredame, C., 2003).

Campos de la Bioinformática

Una de las definiciones de bioinformática divide esta disciplina en dos campos, entendiéndola por un lado como un área técnica y por el otro como una disciplina científica.

- En sentido técnico, se encarga de desarrollar la infraestructura y los sistemas de información y comunicaciones que requiere la biología moderna. Aplica herramientas y métodos computacionales para adquirir, almacenar, organizar, archivar, analizar o visualizar los datos (redes y bases de datos del genoma, estaciones de trabajo para procesamiento de imágenes, etc.).

- Como disciplina científica, la bioinformática se entiende como la computación aplicada a las cuestiones biológicas, los métodos teóricos y técnicas de análisis de datos, modelado matemático y simulación (sistemas de vida artificial, modelos fisiológicos, etc.).

Actualmente esta diferencia va desapareciendo pues el bioinformático, al desarrollar sus propias herramientas, debe tener necesariamente conocimientos de ambos campos.

Tecnología Bioinformática

La tecnología Bioinformática involucra (véase liga: bioinfo):

* Minería de datos y análisis estadístico
* Alineación de secuencias de ADN y proteínas
* Generación y ensamblaje de secuencias (fragmentos de ADN y ARN)
* Predicción de estructura de proteínas
* Árboles filogenéticos empleados para conocer la evolución molecular
* Software para visualización de datos biológicos
* Técnicas de Inteligencia Artificial

1.3.2 El papel de la bioinformática en la predicción

El crecimiento de los datos referidos a las secuencias biológicas es espectacular. La primera proteína secuenciada (la insulina humana, 50 aminoácidos) requirió, con la tecnología disponible en los años 50's alrededor de 10 años, en los 60's este trabajo podría haber requerido 3 años, en los 70's solo 1 año y en los 80's una semana.

Los recursos computacionales crecen con tasas exponenciales y sin embargo no son suficientes para cubrir la demanda que generan muchas aplicaciones, como el análisis de secuencias biológicas. Para cerrar esta brecha tecnológica es necesario optimizar los algoritmos de análisis de datos biológicos.

En la predicción de la estructura de las proteínas se requiere de muchos recursos de procesador para analizar las secuencias. La bioinformática proporciona métodos para la predicción de estructuras aún no resueltas; para esto aplica métodos estadísticos e informáticos (desde redes neuronales, sistemas expertos, modelos de Markov ocultos, programación dinámica, árboles de decisión, etc.) facilitando la predicción o aproximación de modelos.

1.4 UN NUEVO MÉTODO PARA LA PREDICCIÓN

Los métodos actuales para la predicción de la estructura de proteínas no han tenido avances sustanciales para hacer predicciones confiables, en los últimos cuatro años. Como alternativa se desarrolló un nuevo método por homología basado en dos algoritmos: MIN y NIM. Estos algoritmos se basan en el análisis de redes para construir y evaluar los modelos generados a partir de las estructuras tridimensionales de las proteínas conocidas.

1.4.1 Los algoritmos MIN y NIM

Las redes como sistemas biológicos

Las redes pueden ser usadas para modelar sistemas complejos cuando tales sistemas pueden reducirse en sus componentes e interacciones. La teoría de gráficas nos ayuda a analizar estos sistemas, representados por nodos (vértices) y aristas.

Las redes están surgiendo como modelos naturales para representar los datos acumulados en las ciencias biológicas. A diferencia de muchos sistemas biológicos complejos, en las proteínas cuya estructura tridimensional ha sido determinada, los componentes y las interacciones están casi completamente definidos (Thibert, B., Bredesen, D-E., del Rio, G., 2005).

MIN (Minimun Interacting Networks)

El algoritmo MIN se basa en la estructura tridimensional de la proteína para generar una red de contactos entre los aminoácidos y poder identificar los *aminoácidos críticos* (fundamentales para que la proteína cumpla su función) (Thibert, B., Bredesen, D-E., del Rio, G., 2005).

Construcción de la red:

Las redes se derivan de la estructura de las proteínas, teniendo como nodos los aminoácidos y como aristas la relación de distancia entre nodos.

El criterio de distancia considera vecinos a dos aminoácidos si están a cinco Angstrom (Å) de distancia.

Obtención de los aminoácidos críticos:

Al recorrer la red por la ruta más corta que conecta todos los nodos de la gráfica utilizando el algoritmo de Dijkstra, se obtienen los aminoácidos centrales para la comunicación de la red (los más transitados). Estos aminoácidos son considerados como críticos para la función de la proteína, pues al ser removidos la conectividad de la red se afectaría sensiblemente.

NIM (la idea inversa de MIN)

Este método fue llamado NIM, porque es la idea inversa de MIN, es decir que a partir de los aminoácidos críticos se buscará predecir la estructura tridimensional de proteínas (Figura 1.3).

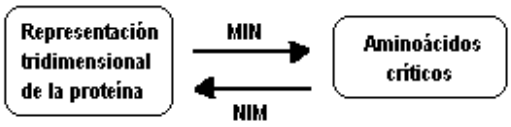

Figura 1.3 Método MIN y NIM

La predicción de la estructura de proteínas genera múltiples modelos, por eso es necesario un criterio de decisión para elegir un modelo. Se requiere también un sistema de puntaje que evalúe la confiabilidad y la calidad de las predicciones; en los métodos por homología el sistema de puntaje involucra el grado de conservación de la secuencia.

NIM es un algoritmo que proporciona un criterio para la elección de un modelo que se basa en los resultados del MIN y en los *aminoácidos conservados*.

Aminoácidos conservados:
Los aminoácidos conservados se obtienen al realizar un alineamiento múltiple de la secuencia objetivo con secuencias de proteínas homólogas (proteínas similares a nivel de secuencia). De este alineamiento se detectan aquellos aminoácidos que se conservaron en un 65% o más, en todas las secuencias alineadas. Las proteínas muestran huellas de su función y de su plegamiento en los aminoácidos conservados de la secuencia.

Funcionamiento de NIM:
Se ha observado que los aminoácidos conservados tienen un papel crítico en la función de la proteína.
NIM compara los aminoácidos centrales (obtenidos por MIN) y los aminoácidos conservados (obtenidos en el alineamiento) para elegir un modelo. El modelo que tenga mayor correspondencia entre los aminoácidos centrales y los aminoácidos conservados será considerado como el mejor. La confiabilidad de esta aproximación depende de la correspondencia y de la identificación correcta de los aminoácidos conservados predichos en el alineamiento (análisis filogenético).

La sensibilidad (S) es una medida de la correspondencia de la predicción. Si la sensibilidad es 1, significa que todos los aminoácidos conservados fueron considerados críticos en el MIN; si es 0, indica que los aminoácidos no coincidieron.

$$S = M / C \qquad 0 \leq S \leq 1$$

M: número de aminoácidos centrales que tienen correspondencia con los aminoácidos conservados.

C: número total de aminoácidos conservados.

1.4.2 El método de predicción

Como se mostró anteriormente, el proceso de predicción está formado por 4 etapas básicas: la búsqueda de proteínas homólogas, el alineamiento de secuencias, la generación de modelos y la elección y evaluación del modelo.

Los pasos para realizar la predicción con el nuevo método son los siguientes:

1. Se buscan las proteínas similares (en secuencia) a la proteína problema.
2. De las proteínas obtenidas en el paso anterior se eligen aquellas que tienen estructura tridimensional conocida.
3. Se realiza un alineamiento con las secuencias obtenidas en el paso 2.
4. Se generan modelos de estructuras tridimensionales para la proteína problema, basándose en las estructuras conocidas.
5. Se aplica MIN para obtener los aminoácidos centrales, se generan redes con los modelos del paso 4.
6. Se alinean todas las secuencias obtenidas en el paso 1 (sin importar si no se conoce la estructura tridimensional) y se obtienen los aminoácidos conservados.
7. Se aplica el algoritmo NIM para elegir uno de los modelos generados, basándose en los resultados del paso 5 y 6.

Capítulo 2

Definición del proyecto

En este capitulo se explica como surge el proyecto 'Diseño de un servidor para la predicción de la estructura de las proteínas' y los propósitos, requerimientos y objetivos del mismo. También se explica la metodología que se sigue para cumplir con los objetivos planteados.

2.1 ANÁLISIS DE REQUERIMIENTOS

Partiendo de los propósitos del cliente se propusieron los requerimientos necesarios para cumplirlos.

Propósito: Participar en el CASP
Requerimientos:

1. Contar con un entorno de trabajo para someter a prueba los algoritmos MIN y NIM.
2. Implementar los algoritmos MIN y NIM.

Propósito: Ofrecer un servicio de predicción de la estructura de proteínas
Requerimientos:

3. Prestar un servicio, libre para la comunidad académica y de investigación, restringido para empresas farmacéuticas y otras relacionadas. Ofrecer posibilidad de acceso a este servicio a través de Internet.

2.1.1 CASP (*Critical Assessment of techniques for protein Structure Prediction*)

La evaluación crítica de los métodos para la predicción de la estructura de proteínas (CASP) tiene como objetivos establecer el estado actual de la biología en la predicción de la estructura de proteínas, medir el progreso que se ha hecho y enfocar los esfuerzos futuros para que sean lo más productivos posible (véase liga: CASP).

CASP es un proyecto que surge para contribuir al avance de los métodos de predicción a través de pruebas y evaluaciones profundas y objetivas de los métodos existentes.

Para realizar la evaluación de los métodos se obtiene experimentalmente la estructura de una proteína, desconocida hasta ese momento, y antes de hacerla pública se invita a la comunidad a predecir la estructura de esa proteína. La confiabilidad del método se medirá a partir de la similitud observada entre el modelo generado y el modelo experimental.

¿Porqué participar en el CASP?
Participar en el CASP permitirá hacer una evaluación objetiva (aunque no estadística) de la confiabilidad del método de predicción que utilizará este servidor, además de dar a conocer el método.

2.2 EL PROYECTO

2.2.1 Definición del problema

El problema puede plantearse desde dos perspectivas, la biológica y la computacional.

Desde el punto de vista biológico:

Los métodos actuales para la predicción de la estructura de proteínas han permitido avanzar en el entendimiento de los mecanismos moleculares de la vida, sin embargo, estos métodos han alcanzado un nivel de confiabilidad que no ha mejorado en los últimos cuatro años.
Se ha propuesto, a partir del CASP, que es necesario mejorar los métodos de alineamiento y de evaluación de modelos para alcanzar predicciones confiables.

Alternativa:

En respuesta a esta demanda se proponen dos algoritmos nuevos (MIN y NIM) para la predicción de modelos, un método basado en el análisis de redes (capítulo 1).
Estos algoritmos fueron realizados por el Dr. Gabriel del Río Guerra del Instituto de Fisiología Celular (IFC, UNAM) y hasta ahora no existe otro método que realice la predicción con el enfoque de redes.

Desde el punto de vista computacional:

Los algoritmos MIN y NIM no cuentan con una interfaz que permita su ejecución de principio a fin. La predicción, por este método, se realiza etapa a etapa ejecutando cada programa, desde el *shell* de Linux.

Cuando se han obtenido los resultados de un programa se modifica el formato: si es necesario, este proceso se hace manualmente o a través de un programa y los datos obtenidos pasan como parámetros de otros programas. Este proceso es largo y tedioso, pues se debe estar pendiente durante todo el proceso para ejecutar uno u otro programa, además no se cuenta con una interfaz gráfica que facilite el proceso.

Es importante destacar que hasta ahora, en México, no se ha dado a conocer ningún servidor para la predicción de la estructura tridimensional de proteínas. Este trabajo busca desarrollar el primero, basado en un nuevo método de predicción.

2.2.2 Objetivos

Los objetivos planteados, de acuerdo a los propósitos, son los siguientes:

Participar en el experimento CASP:
Crear un entorno de trabajo que permita la predicción mediante los algoritmos MIN y NIM.
Evaluar y dar a conocer el método de predicción.
Implementar los algoritmos MIN y NIM
Hacer compatibles los formatos de los programas involucrados en la predicción.
Agilizar el proceso de predicción.

Dar un servicio a la comunidad internacional, interesada en la predicción:
Desarrollar un servidor para la predicción.
Crear una interfaz de usuario amigable.
Controlar la entrada de los usuarios a la aplicación.
Permitir el acceso al servidor a través de Internet.

2.2.3 Solución

Crear un entorno de desarrollo integral, que agrupe las herramientas necesarias para la predicción de estructura de proteínas, implementando los algoritmos MIN y NIM.
Generar una interfaz gráfica que facilite el uso de este método.

Para resolver un problema se aplican una gran variedad de métodos, herramientas, procedimientos y paradigmas (Lawrence, S., 2002).
Los pasos que se proponen para resolver el problema se describen a continuación:

Método
El método o técnica es la secuencia de pasos que se siguen para obtener un resultado; en este problema, para cumplir los requerimientos se realizó lo siguiente:

1. Para crear el entorno de trabajo
   Recopilar las herramientas necesarias para la predicción: software, bases de datos, etc.
   Crear una interfaz gráfica que permita accesar a la aplicación.

2. Para implementar los algoritmos de la predicción
   Entender el funcionamiento del método de predicción.

Dividir en etapas el proceso de la predicción (identificando los datos de entrada y de salida).

Crear los paquetes y las clases necesarias para el funcionamiento de cada etapa.

Hacer un programa que conjunte todos los paquetes necesarios para realizar la predicción y que verifique el resultado de cada etapa durante todo el proceso.

Verificar el funcionamiento del software necesario para la predicción, identificar sus parámetros de entrada y los formatos de entrada y de salida.

3. Para que este servicio pueda ser accesible al público

Desarrollar un servidor en el contexto de Internet, una aplicación Web del lado del servidor.

Buscar una herramienta que permita la comunicación entre el servidor y los usuarios.

Crear la interfaz gráfica que se mostrará al usuario.

Diseñar un sistema de admisión, que controle el acceso al sistema.

Herramientas

Para desarrollar el servidor se requiere una plataforma:

De software: Servidor de páginas, servidor de correo, contenedor Web, JVM, j2se, MySQL, etc.

De hardware: Ordenadores con gran capacidad de almacenamiento en disco duro, alta velocidad, memoria, etc.

(Las herramientas utilizadas se describen en el capítulo 3).

Procedimiento:

El procedimiento (la combinación de herramientas y técnicas) utilizado para construir el servidor se describe en los capítulos 4 y 5. Durante el diseño se describen las etapas de la predicción, los pasos que se siguieron y los programas utilizados. En la implementación del sistema también se explica como se manejaron las herramientas para la creación del sitio.

Paradigma:

El paradigma representa el enfoque particular o filosofía para la construcción del software. Para el diseño de este servidor se adoptó el enfoque orientado a objetos.

Capítulo 3

Herramientas para el desarrollo del servidor

En las siguientes páginas se explican algunos conceptos sobre el desarrollo de aplicaciones Web del lado del servidor, sus características y sus requerimientos. Conocer como funcionan estas aplicaciones nos ayuda a definir las herramientas que debemos utilizar para crear el servidor para la predicción de la estructura de proteínas.

3.1 APLICACIONES DEL LADO DEL SERVIDOR

Las aplicaciones del lado del servidor (*server side applications)* suelen ser la elección por la que se opta, en la mayoría de las ocasiones, para realizar aplicaciones Web (véase apéndice C), pues al ejecutarse en el servidor y no en la máquina del cliente, no necesitan ninguna capacidad adicional, como ocurre en el caso de querer ejecutar aplicaciones *javascript* y *applets*. De esta forma cualquier cliente dotado de un navegador Web básico puede utilizar este tipo de aplicaciones.

Una aplicación Web es igual a cualquier otro programa, sólo con pequeñas excepciones. Para que un programa sea accesible desde un servidor Web, se debe cumplir lo siguiente (Danny, A. et Al., 1999):

- El programa debe estar habilitado para ser invocado por el servidor Web. Cuando el navegador envía una petición el servidor debe estar habilitado para localizar y ejecutar el programa requerido.
- Cuando el servidor invoca el programa, necesita un camino para pasar la petición HTTP y los datos que requiere el programa.
- Después de que el programa ha procesado los datos de entrada, este debe empaquetar los resultados y regresarlos al servidor, el cual los envía al navegador.

Se han desarrollado diferentes tecnologías que permiten la comunicación entre los programas y el servidor Web. Se debe elegir la opción que permita direccionar el potencial de estas tecnologías para ofrecer una solución tanto al cliente como al servidor.

Hasta hace poco tiempo la única solución real para traer datos dinámicos de la Web fueron los CGI (*Common Gateway Interface*). Los programas CGI proporcionan un camino simple para crear aplicaciones Web. Las últimas tecnologías Web del lado del servidor son ASP (*Actives Server Pages*), *JavaServlet* y JSP (*Java Server Pages*).

3.1.1 Requisitos de una aplicación Web

Los requisitos más esenciales para el desarrollo de una aplicación Web son los siguientes:

- Un modelo de programación y un API que especifiquen cómo desarrollar aplicaciones.
- Soporte de ejecución del lado del servidor: recursos para ejecutar las aplicaciones y apoyo para servicios de red.

- Soporte de implementación que es el proceso de instalación de la aplicación en el servidor. Este proceso también incluye la configuración de los componentes de la aplicación, como especificar los parámetros de inicialización y especificar cualquier base de datos.

3.2 DESARROLLANDO LA APLICACIÓN DEL SERVIDOR

Para crear la aplicación que ofrece el servidor desarrollado en este trabajo de tesis se debe cumplir con los requisitos de una aplicación Web.

Para construir y ejecutar la aplicación del servidor se tienen los siguientes elementos (Beust, C. et Al., 2001):

- Modelo de programación*:* Para desarrollar la aplicación se eligió la tecnología *JavaServlet.*
- Aplicación Web: Son los programas que realizan la predicción y muestran la interfaz gráfica. Es una colección de *servlets*, clases auxiliares, librerías de clases y recursos estáticos como documentos HTML, XML o imágenes.
- Soporte de ejecución e implementación: Para que la aplicación pudiera ejecutarse se creó un marco de trabajo (con las herramientas que necesita la aplicación). El contenedor Web alberga la aplicación Web y es el encargado de gestionar los programas de la aplicación. El servidor utiliza el contenedor *Jakarta-tomcat*.

3.2.1 Un modelo de programación

El lenguaje Java

Java es un lenguaje de programación de alto nivel. Las principales características de Java son que es un lenguaje simple de codificar, orientado a objetos, distribuido, seguro, portable y multitarea.

Java es compilado e interpretado al mismo tiempo; el compilador es el encargado de convertir el código fuente de un programa en un código intermedio llamado *bytecode,* que es independiente de la plataforma en que se trabaje y que es ejecutado por el intérprete de Java.

La plataforma Java

Una plataforma es el ambiente de hardware o software en el cual se ejecutan los programas. En general, la mayoría de las plataformas pueden ser descritas como una combinación de hardware y sistema operativo. La plataforma Java está basada únicamente en software y corre por encima de las plataformas basadas en hardware (véase liga: java).

La plataforma Java consta de dos componentes:

- La Máquina Virtual de Java (JVM)
- La Interfaz de Programación de Aplicaciones de Java (API Java)

Tipos de programas en Java:

Los programas en Java suelen estar en una de las siguientes categorías:

- *Applets*: Los *applets* son programas que a menudo se descargan junto con una página HTML desde un Servidor Web y se ejecutan en la máquina cliente.
- *Programas standalone:* son aplicaciones de propósito general, que normalmente se ejecutan desde la línea de comandos del sistema operativo.
- *Servlets:* Los *servlets* al contrario de los *applets* son programas que están pensados para desarrollar aplicaciones Web y trabajar del lado del servidor.
- *JSP (Java Server Pages):* Los *JSP* combinan código HTML y código Java en un solo programa. Cuando un *JSP es* solicitado el servidor Web lo compila y lo convierte en *servlet*, para ejecutarlo.
- *Beans* y *Java Beans*: Es un modelo para el desarrollo de unidades de software reusables, seguras, reajustables y transaccionales o multi-usuario.

Java Development Kit (JDK) es el paquete de desarrollo de Java que contiene las herramientas y librerías necesarias para crear y ejecutar applets y aplicaciones en Java.

Algunas de las utilidades que se pueden encontrar en el JDK son:

- *javac*: Es el compilador de Java, convierte el código fuente escrito en Java a *bytecode*.
- *java*: Es el intérprete de Java. Ejecuta el *bytecode* a partir de los archivos class.
- *appletviewer*: Es un visor de applets que puede utilizarse en lugar de un Navegador Web.
- *javadoc*: Se utiliza para crear documentación en formato HTML a partir del código fuente Java.

- *javap*: Es un desensamblador de Java.
- *jar*: Es una herramienta que permite comprimir archivos.

Servlets

Los *servlets* son programas que se utilizan para desarrollar aplicaciones Web (Danny A. et Al., 1999).

Los *servlets* son invocados por una PC remota a través de peticiones HTTP, se ejecutan en el servidor (en la máquina virtual de java) y mandan los resultados a la maquina remota, la interacción con los usuarios se realiza a través de páginas HTML.

Cuando un usuario requiere un *servlet* el servidor crea un proceso que corre independiente a la conexión hecha con el cliente, de esta manera el *servlet* puede responder de manera continua a las siguientes peticiones. Los *servlets* son eficientes debido a un modelo de hilos (*threading model*), en el cual cada petición es atendida por un *thread*. La ventaja de este modelo es que un hilo necesita menos recursos que un proceso nuevo.

Cuando se habla de la tecnología *JavaServlet* se debe pensar en el marco de trabajo (*framework*) formado por las clases que implementan los servlets y las clases que ayudan a establecer el dialogo Web (mantenimiento de sesiones, manejo de *cookies*, etc.).

¿Porqué JavaServlets?

Las tecnologías Java del lado del servidor (*servlet* y *JSP*) son independientes de la plataforma, eficientes, accesibles desde otra API de Java, reusables y modulables.

- Independencia de la plataforma: El código de los *servlets* se compila y se interpreta por una plataforma específica: JVM.
- Eficiencia: Para manejar cada petición del usuario los *servlets* generan un *thread,* esto hace posible que cientos de usuarios utilicen el mismo *servlet* simultáneamente.
- Acceso a los APIs de Java: Ya que los *servlets* son una extensión de la plataforma Java pueden utilizar todos los APIs de Java (*JavaMail*, *RMI*, *CORBA*, *JNDI*, *EJB*).
- Reusabilidad: Java, un lenguaje orientado a objetos, provee mecanismos de reusabilidad al separar una aplicación en partes.
- Modularidad: En una aplicación de servidor los programas suelen ampliarse y complicarse rápidamente por eso es mejor separar la aplicación en módulos donde cada uno sea responsable de una tarea específica.

Algunas desventajas de esta tecnología es que si un *servlet* es muy especializado o de poco uso, estará vivo en memoria consumiendo los recursos del ordenador, por esta razón es mejor seleccionar los programas o aplicaciones que se deberán construir con esta tecnología. Otra desventaja es que si se quiere mostrar información en formato HTML se debe escribir el código línea a línea.

Implementación de los servlets
La interfaz *servlet*: *public interface servlet*

El API *JavaServlet* (*javax.servlet.Servlet)* es la interfaz que el contenedor Web utiliza para referenciar a los *servlets*. Cuando se escribe un *servlet* se debe implementar esta interfaz (de forma indirecta con *javax.servlet.GenericServlet* o *javax.servlet.http.HttpServlet*) (Beust, C. et Al., 2001).

Con esta interfaz se implementan los métodos:

- *init():* Cuando se crea una instancia del *servlet* este método realiza cualquier inicialización requerida, el método se ejecuta una sola vez independientemente del número de veces que se llame el *servlet*.
- *service():* Es el punto de entrada para ejecutar el servlet, el contenedor invoca este método en respuesta a las solicitudes entrantes (GET, POST, PUT, etc.).
- *destroy():* El contenedor invoca este método antes de eliminar la instancia de un *servlet* que está fuera de servicio, de está manera se libera espacio en memoria.
- *getServletInfo*(): Devuelve un objeto con la información sobre el *servlet* (autor, fecha de creación, descripción, etc.).

La clase *HttpServlet* amplía la clase *GenericServlet* y proporciona una implementación específica de HTTP de la interfaz *servlet*. Esta clase implementa los métodos *doGet() y doPost()* que atienden las peticiones GET y POST.

3.2.2 Aplicación Web

Para que el servidor realice su función se crearon paquetes con los programas que implementan los algoritmos para la predicción, así como los archivos que constituyen la interfaz gráfica de la aplicación.
La descripción de cómo se diseñó la aplicación se muestra en el capítulo 4.

3.2.3 Soporte de ejecución e implementación

Para gestionar los programas de la aplicación y para comunicar al usuario, con los *servlets* y con el servidor se utilizó un *contenedor de servlets***.**

Contenedor de servlets

Ofrece un ambiente donde habitan los *JSP* y los *servlets*, es ahí donde se tiene una gran cantidad de funcionalidades como: *threading*, manutención de sesiones, conexión con el servidor de páginas, etc. El programador no tiene que preocuparse por estas tareas (Beust, C. et Al., 2001).

El contenedor es el responsable de manejar las peticiones del cliente: recibe una petición la pasa al *servlet* y regresa la respuesta al cliente en el formato correcto. La interfaz entre el contenedor y los *servlets* está especificada en el API *JavaServlet*.

El contenedor gestiona los *servlets*, una de sus responsabilidades es el ciclo de vida de un *servlet*. El ciclo de vida empieza cuando el contenedor invoca el método *init()* y termina cuando invoca *destroy();* las etapas fundamentales son:

- *Instanciación*: El contenedor Web crea una instancia del *servlet*.
- *Inicialización*: El contenedor invoca el método *init()* de la instancia del *servlet*.
- *Revisión*: Si el contenedor tiene una solicitud para el *servlet*, invoca el método *service()* de la instancia del *servlet*.
- *Destrucción*: Antes de destruir la instancia, el contenedor invoca el método *destroy()* de la instancia del *servlet*.
- *No disponible*: La instancia es destruida.

Para que la aplicación se ejecute se construyó un marco de trabajo con las herramientas de software, hardware y bases de datos, a continuación se describen esas herramientas.

Herramientas para el desarrollo del servidor

Las herramientas computacionales necesarias para desarrollar la aplicación son principalmente: las que dan soporte a la interfaz gráfica y a la aplicación (predicción de estructuras de proteínas).

Las herramientas elegidas para desarrollar el servidor se describen a continuación:

1) Lenguaje de programación

Para desarrollar la aplicación, se eligió la tecnología *JavaServlets* por las ventajas que presenta la plataforma Java. Se requiere de un entorno de programación para compilar y

ejecutar los *servlets,* existen diversas versiones del paquete de desarrollo Java (JDK) de acuerdo al sistema operativo de la máquina en la que se desarrolla la aplicación Web. Algunos JDK son: J2SE (*Java 2 Standard Edition*) de Sun y JDK de IBM.
El JDK que se utilizó para desarrollar la aplicación fue el *j2sdk1.4.2_03* (véase liga: j2seAPI).
El API *JavaServlet* que se utilizó para construir la aplicación fue el *JavaServlet 2.4* (véase liga: javServer).

2) Contenedor de servlets
Apache Jakarta Tomcat es un contenedor de *servlets* y *JSP* y es uno de los servidores más utilizados ya que es confiable, rápido, gratuito y soporta las tecnologías *JavaServlet*.
La versión utilizada para la aplicación fue *jakarta-tomcat-5.0.28* (véase liga: jakTomcat).
Esta versión implementa las especificaciones *JavaServlet 2.4* y *JSP 2.0*

3) Servidor de bases de datos
Para crear la base de datos se utilizó MySQL.
MySQL es un manejador de bases de datos relacionales que ha adquirido gran importancia en la creación de aplicaciones Web, debido a su rapidez de procesamiento, confiabilidad, flexibilidad y principalmente por ser software libre. Este manejador utiliza el lenguaje de consultas estructurado, SQL (*Structure Query Language*), para realizar las consultas a la base de datos.
La versión instalada en el servidor fue *MySQL 3.23.58* (véase liga: MySQL).

Para acceder a la base de datos a través de los procedimientos desarrollados en java se utilizó el API JDBC (*Java Database Connectivity*) que es un estándar que aporta una interfaz de programación que permite la conexión con manejadores de bases de datos que soporten el lenguaje SQL. Cada fabricante de bases de datos implementa un controlador JDBC para el acceso a la base de datos desde Java.
La versión utilizada del JDBC de mysql fue *mysql-connector-java-3.0.11-stable-bin.jar* (véase liga: javMySQL).

4) Servidor de correos
Para conectarnos con el servidor de correos, se descargó el API *JavaMail*, una interfaz de Java que permite enviar y recibir correos electrónicos. *JavaMail* se ajusta a los protocolos de correo más utilizados en Internet, como IMAP4, POP3 y SMTP.

La versión utilizada en el servidor fue *javamail-1.3.2 y jaf-1.0.2* (véase liga: javMail).

El software se instaló en el servidor donde se desarrolló la aplicación, en el Instituto de Fisiología Celular (IFC, UNAM). El IFC cuenta con un servidor de correos (*send mail*) y todas las máquinas conectadas en esa red tienen acceso al servidor, por lo que no fue necesario instalar el software para usar este servicio.

Las herramientas para la predicción fueron las siguientes:
1) Búsqueda de proteínas homólogas
Para realizar la búsqueda de proteínas homólogas a la proteína objetivo se empleó un método de comparación de secuencia llamado *BLASTPGP* (véase liga: Blast).

2) Alineamiento de las secuencias de las proteínas
Para realizar el alineamiento de las secuencias se utilizó *TCOFFEE.* La *versión utilizada fue T-COFFEE_distribution_Version_1.37* (véase liga: Tcoffee).

3) Generación de modelos
Para construir las coordenadas espaciales de la secuencia objetivo, a partir del alineamiento realizado se empleó *MODELLER,* versión *modeller6v* (véase liga: Modeller).

4) Base de datos
Se descargó una base pública con información sobre las proteínas, *PDBFiles y PDBFilesModeller* (véase liga: DBase).

La herramienta de hardware disponible es un servidor, del IFC, con las siguientes características:

Sistema Operativo - Red Hat Linux release 9
Disco duro - Intel(R) Xeon(TM), 60 GB, velocidad - 2.80GHz
Memoria - 1 GB

La instalación de estas herramientas da soporte a la implementación del servidor y permite la ejecución de los algoritmos del método de predicción. El proceso de implementación se describe en el capítulo 5.

Capítulo 4

Diseño del servidor

Antes de explicar el diseño del servidor se debe efectuar un análisis, desglosando el problema en piezas que se puedan comprender y abordar. De esta forma se describe un problema grande como una serie de problemas más pequeños y sus interrelaciones.

En el diseño se trata de construir cada componente, es decir, definir el método que se seguirá para resolver una tarea. El conjunto de estos elementos deben solucionar el problema y cumplir con los requerimientos del sistema.

El diseño de este servidor pretende solucionar un problema: crear un entorno integral para la predicción de la estructura tridimensional de las proteínas.

En este capítulo se explican los componentes del sistema, con énfasis en el procedimiento para realizar la predicción, las etapas y las clases que deben implementarse para el funcionamiento de esta aplicación.

4.1 ANÁLISIS DEL SISTEMA

En el análisis se desglosa el problema en piezas que se puedan comprender y abordar (Lawrence, S., 2002).

Es necesario identificar los usuarios que solicitan el servicio y definir las tareas (o funciones) que debe realizar el sistema.
El sistema se representa, de forma general, en la siguiente figura.

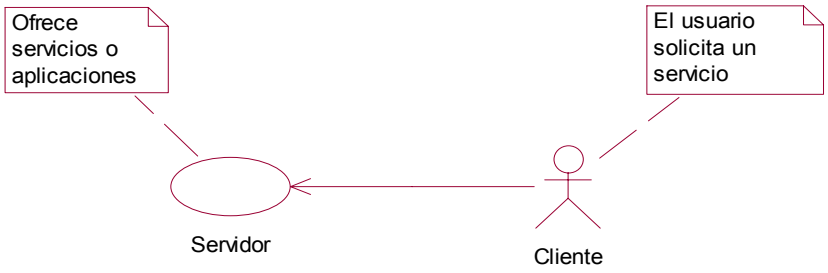

Figura 4.1 Modelo cliente servidor.

Diagrama de actores
El siguiente diagrama muestra los usuarios (actores) involucrados en el sistema.

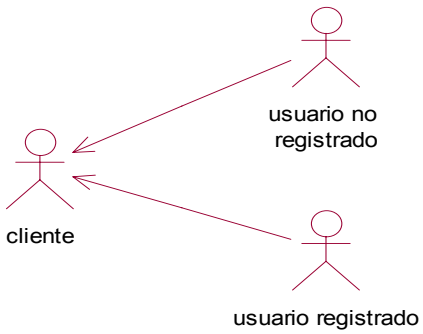

Figura 4.2 Diagrama de actores.

Descripción
cliente: Se refiere al actor más general del sistema, es una persona cualquiera.
usuario registrado: Son los usuarios que se han registrado en la base de datos y que tienen una cuenta en el sistema.
usuario no registrado: Son los usuarios que nunca han ingresado al sistema, no tienen una cuenta.

Diagrama de casos de uso

En este modelo se describen las interacciones entre los actores y el sistema. Cada caso de uso especifica las acciones (funciones) que pueden ser ejecutadas en el sistema (Figura 4.3).

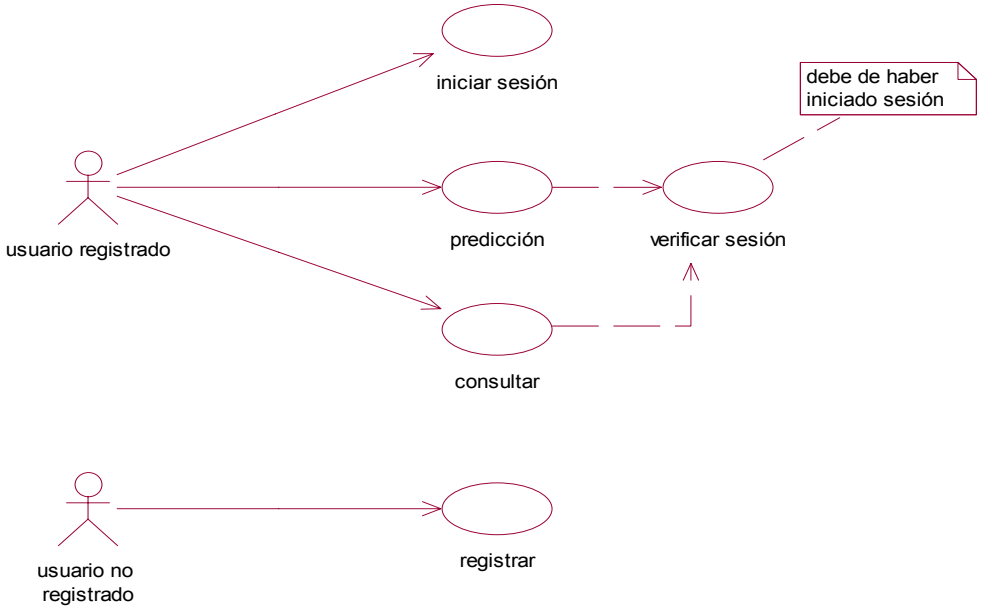

Figura 4.3 Diagrama de casos de uso.

Descripción

iniciar sesión: Un usuario registrado puede acceder al sistema proporcionando su nombre y su contraseña.

predicción: Un usuario con sesión iniciada puede realizar predicción de la estructura de proteínas (es la aplicación que ofrece el servidor).

consultar: Un usuario con sesión iniciada puede obtener documentos y consultar la información contenida en el sitio.

registrar: Un usuario no registrado puede obtener una cuenta para tener acceso al servidor.

Para realizar el análisis dividimos el sistema en tres partes: **los servicios, la interfaz y los datos**.

- Los servicios del sistema (casos de uso) son cuatro: iniciar sesión, consultar, predicción y registrar.
- La interfaz es el medio por el que el usuario se comunicará con el sistema.
- Los datos se refieren a la organización de la información que el usuario ingresa al sistema, por ejemplo: la cuenta para iniciar sesión, los datos para registrarse, etc.

4.1.1 Los servicios del sistema

A continuación se realiza el análisis de cada servicio del sistema.

'iniciar sesión'

Para que un usuario ingrese al sistema debe tener una cuenta (Figura 4.4).

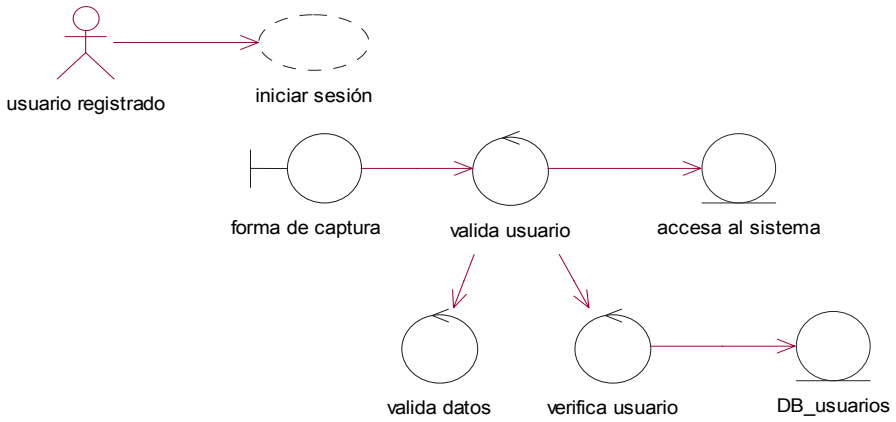

Figura 4.4 Análisis del servicio 'iniciar sesión'.

Los pasos para iniciar la sesión son los siguientes:

1. Introducir nombre y contraseña.
2. Validar que los datos sean correctos (campos no vacíos y tipos de datos).
3. Verificar que el usuario exista en la base de datos.
4. Permitir el acceso al sistema.

'registrar'

Para tener control del acceso al sistema (según los requerimientos, capítulo 2) es necesario registrar a los usuarios (Figura 4.5).

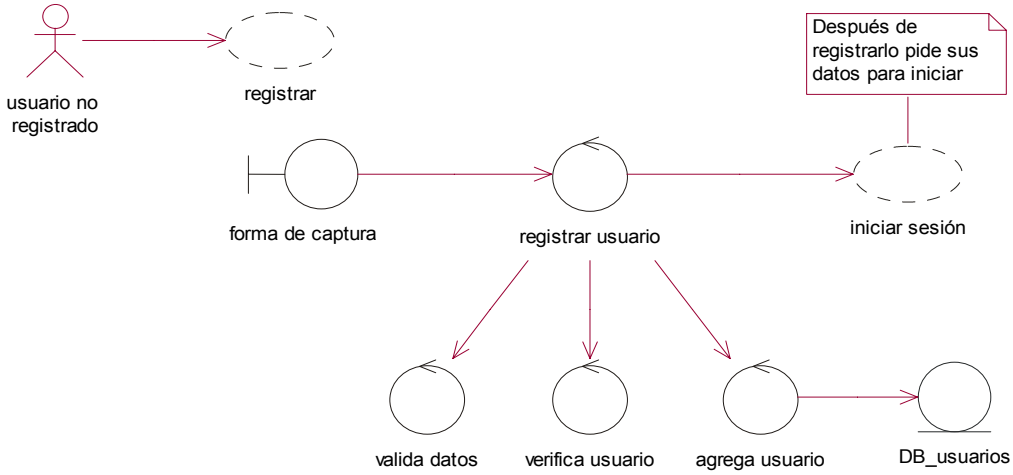

Figura 4.5 Análisis del servicio 'registrar'.

Los pasos para registrarse son los siguientes:

1. Solicitar el registro al sistema.
2. Proporcionar datos personales (nombre, nombre de acceso, contraseña, país, institución, correo electrónico).
3. Validar los datos (campos no vacíos y tipos de datos).
4. Verificar que no exista otro usuario con el mismo nombre y contraseña.
5. Agregar usuario a la base de datos.
6. Iniciar sesión.

'consultar'

En el diagrama se describe el proceso de consulta; el usuario tiene acceso a la información publicada en el servidor (documentos, ligas a otros sitios Web, etc.) (Figura 4.6).

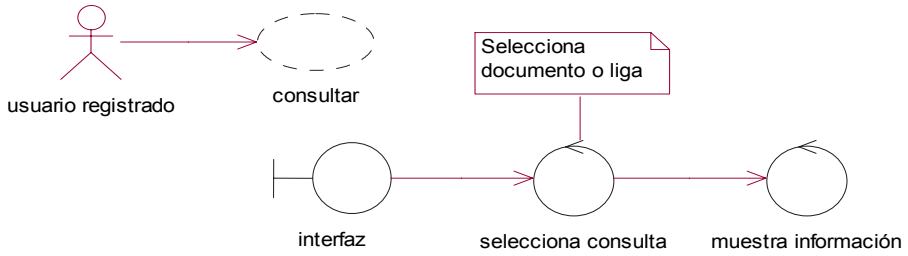

Figura 4.6 Análisis del servicio 'consultar'.

El proceso de consulta es el siguiente:

1. Seleccionar documento o liga.
2. Mostrar información.

'predicción'

El proceso de la predicción se realiza como sigue:

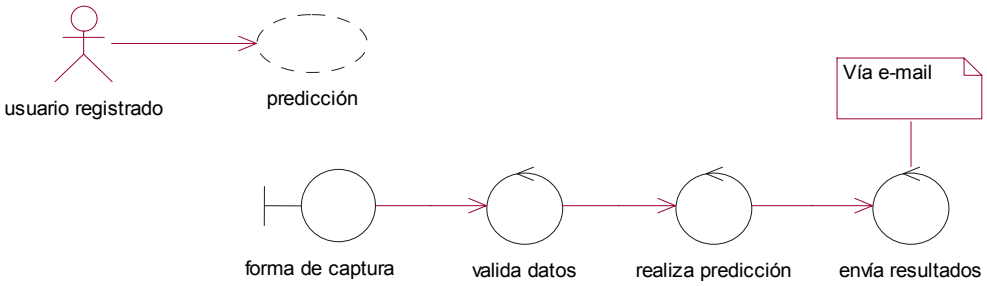

Figura 4.7 Análisis del servicio 'predicción'.

Los pasos para realizar la predicción son los siguientes:

1. Llenar el formulario con los parámetros para la predicción.
2. Enviar la solicitud.
3. Validar datos (campos no vacíos, tipo y rango).
4. Recibir los parámetros e iniciar el *proceso de predicción*.
5. Enviar los resultados al usuario.

En el capítulo 1 se explican los algoritmos que utiliza el método de predicción. Para implementar el servicio de predicción se crearán paquetes con los programas de cada etapa de predicción.

Las etapas son las siguientes (Figura 4.8):

1. Obtención de las secuencias de las proteínas similares.
2. Obtención de los aminoácidos centrales (aplicando MIN).
3. Obtención de los aminoácidos conservados.
4. Obtención de un modelo (aplicando NIM).
5. Generar archivo con los resultados.

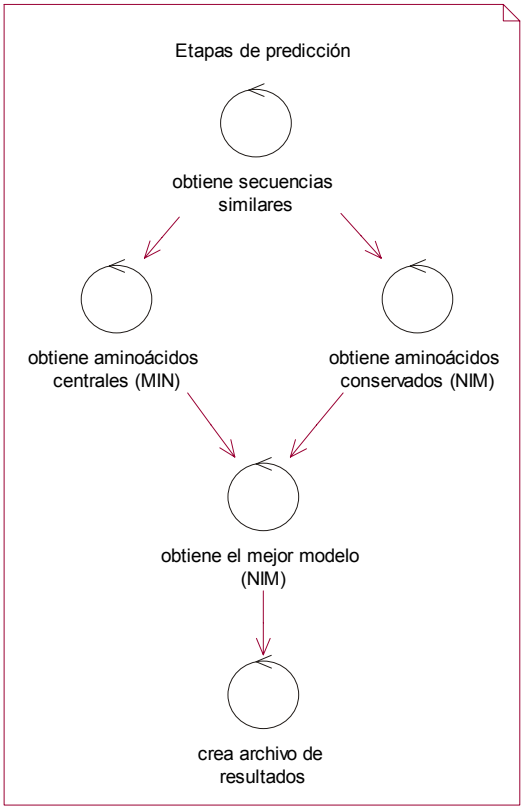

Figura 4.8 Proceso de predicción.

Los parámetros que se requieren para la predicción son los siguientes:

1. Para el algoritmo MIN:
   Número de moldes
   Número de modelos
2. Para el algoritmo NIM:
   Número de secuencias a alinear
   Valor de redundancia
   Porcentaje de conservación (superior e inferior)
3. Para la obtención de las secuencias similares:
   e_value (parámetro del programa blastpgp)
4. Para realizar el proceso de predicción:
   La secuencia problema, de la cual se quiere conocer su estructura

4.1.2 La interfaz

Es el conjunto de páginas Web y archivos electrónicos que presentan el sistema, incluye una página inicial de bienvenida y un nombre de dominio o dirección en Internet específicos.
Se deben crear formularios para ingresar datos y comunicarse con el servidor. Los datos que el usuario debe enviar son el nombre y la contraseña (para iniciar sesión), datos del usuario (para registrarlo) y parámetros de la predicción (para realizar la predicción).

4.1.3 Los datos

Para organizar los datos que se obtienen del usuario (en el registro, la predicción, etc.) se propone la creación de una base de datos que permita el almacenamiento, la recuperación y la consulta de esta información.

* Información del usuario - La información que se solicita en el registro de un usuario es la siguiente: Nombre, Contraseña, Correo electrónico, País, Institución o Grupo de trabajo.

* Información del proceso – La información de los procesos es la siguiente:
Procesos en espera de ser ejecutados: Número de proceso, Nombre del proceso.
Procesos ejecutados: Número de procesos ejecutados, Fecha.

* Información de los documentos – De los documentos solo se requiere el Nombre o URL.

4.2 DISEÑO DEL SISTEMA

Una vez analizado el problema se puede construir una solución a partir de los componentes (que se ocupan de diversos aspectos) del sistema. En esta etapa se trata de realizar la composición del sistema mediante pequeños bloques (Lawrence, S., 2002).

¿Qué es una arquitectura?
La arquitectura de un sistema de cómputo, comprende los procesos del sistema (funciones del software) y sus relaciones. El propósito de una arquitectura es preparar las especificaciones generales del sistema.

Principios de una arquitectura

1. Con conocimiento parcial del sistema se puede predecir el resto del sistema.
2. Las funciones son independientes, separadas en sus especificaciones.
3. Aceptar solo las funciones esenciales para el sistema.
4. No se deben repetir funciones en la descripción de la arquitectura.
5. Las funciones introducidas deben satisfacer las necesidades y los deseos del cliente.

¿Cuál es la importancia de una arquitectura?
Proyectar la organización de un sistema nos permite conocer las especificaciones y necesidades significativas para el usuario.
Proporciona una vista general del todo en la que se omiten los detalles, éstos pueden derivarse de la estructura.

¿Cómo es una arquitectura?
Muestra las funciones principales del sistema y sus interrelaciones, usualmente mediante diagramas y símbolos.

4.2.1 Arquitectura del sistema

El diseño del sistema se basa en la arquitectura de una aplicación Web de tres capas (Danny A. et Al., 1999) (Figura 4.9).

- La primera capa, contiene la interfaz gráfica que es el medio por el cual el cliente interactúa con el sistema.
- La segunda capa está formada por los servicios del sistema.
- La tercera capa contiene la base de datos con la información del sistema.

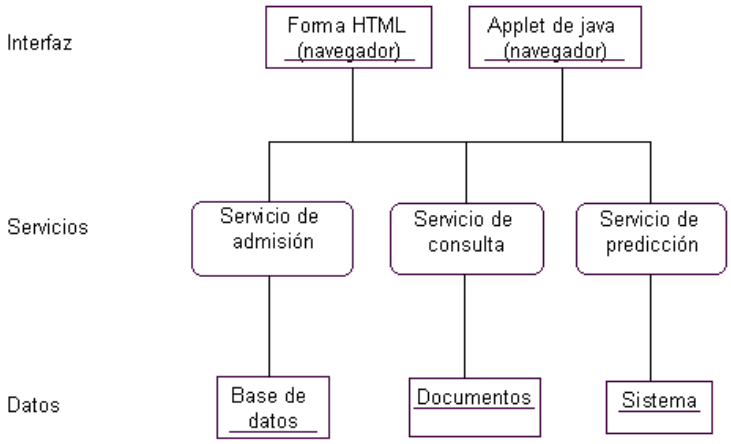

Figura 4.9 Capas del sistema

Durante el análisis se identificaron las funciones del sistema: 'iniciar sesión', 'registrar', 'consultar' y 'predicción'. Ya que la consulta no es una función esencial del sistema, desaparece de la arquitectura.

A continuación se muestra el diagrama con la arquitectura propuesta para el sistema.

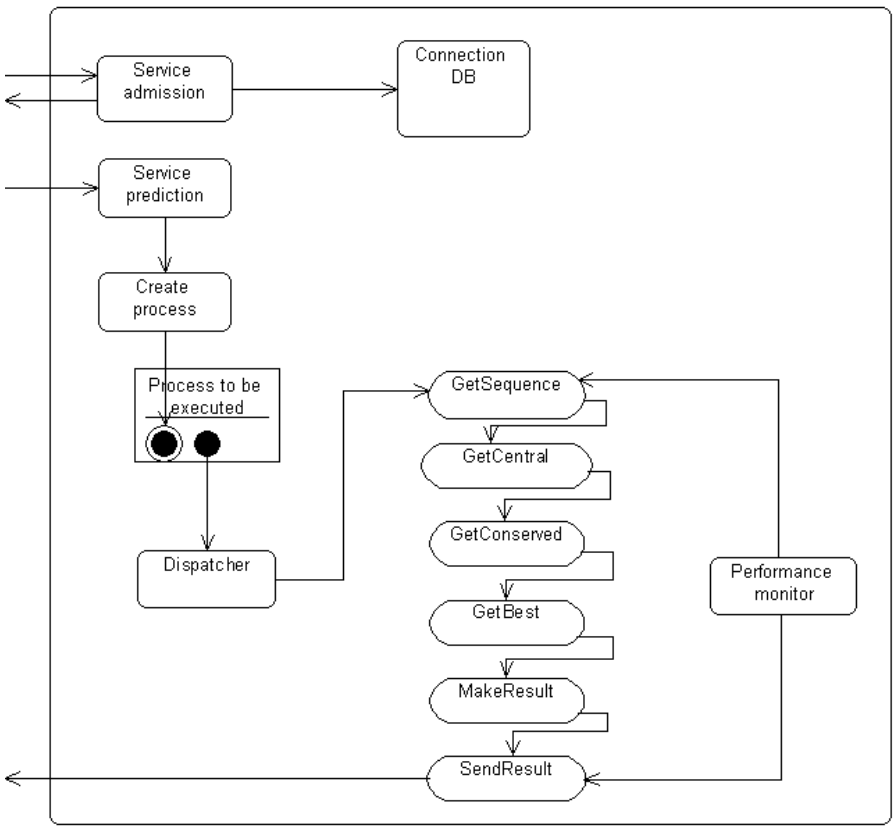

Figura 4.10 Arquitectura del sistema.

Las capas del sistema son las siguientes:

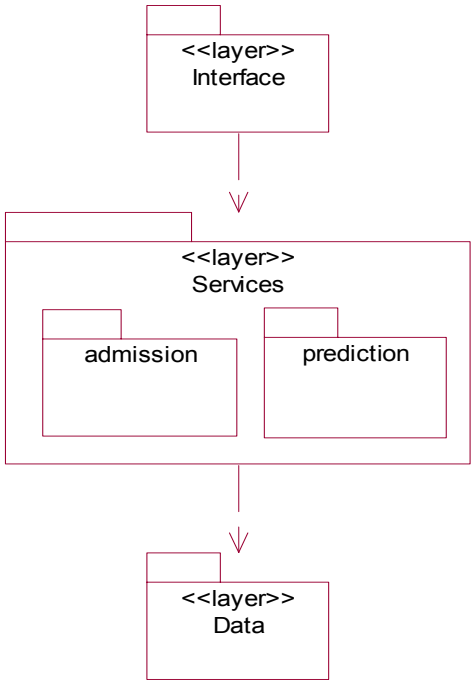

Figura 4.11 Diagrama con las capas de la arquitectura.

En esta sección se describe la arquitectura del sistema y los elementos que la componen, así como los diagramas (de actividades y de clases) necesarios para implementar los componentes: Interface, Services, Data.
El diagrama de actividades muestra los pasos que se siguen para realizar una tarea.
El diagrama de clases ilustra las especificaciones de las clases (los métodos, atributos, sus relaciones, dependencias etc.) que participan en la solución de una tarea.

4.2.1.1 Interface

Está formada por un conjunto de páginas HTML y un *applet* que muestra los resultados. Es el medio por el cual el usuario se comunica con el servidor, para ingresar, consultar información o enviar datos para la predicción.

Descripción de la interfaz:
La organización jerárquica del sitio Web se muestra a continuación:

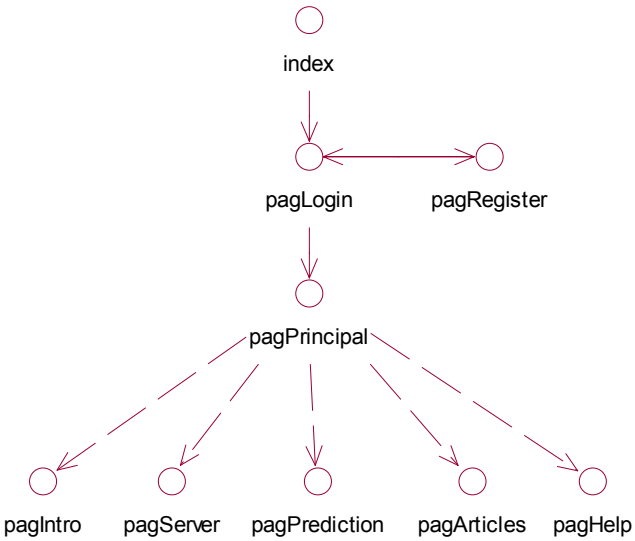

Figura 4.12 Organización jerárquica del sitio.

1. La primera página (*index.html*) muestra la presentación del sitio (Bioinformatics for Protein Structure Prediction), desde aquí se tiene acceso a la página de ingreso.
2. La página *pagLogin.html* solicita el login y el password y verifica que el usuario esté registrado; en caso de que el usuario no se encuentre en la base de datos, manda una página de error (*pagLoginError.html*) y pide nuevamente los datos. Aquí aparece una liga a la página de registro.
3. *pagRegister.html* solicita algunos datos del usuario y le asigna una cuenta de ingreso al sistema, después del registro manda la página *pagLogin.html* para que ingrese con su nueva cuenta.
4. La página principal muestra información general del sitio (*pagIntro.html*) y, desde el menú que aparece en la parte superior, tenemos acceso a las demás páginas.
5. Para obtener información sobre el servidor seleccionamos 'The server'.
6. *pagPrediction.html* muestra información sobre el proceso de la predicción (el método utilizado) y nos liga al formulario que solicita los parámetros para la predicción.
7. Para consultar artículos, documentos o ligas a otros sitios seleccionamos 'Articles'.
8. El sistema tiene una herramienta de ayuda, en la que podemos verificar información sobre la predicción, el formato de la secuencia, el valor y el significado de los parámetros de la predicción, etc.

Funcionamiento:
El navegador (Netscape, Explorer, etc.) es la interfaz que muestra las páginas HTML. Al enviar un formulario de datos se genera una petición HTTP (con un método definido: POST, GET, HEAD, ..) que solicita la ejecución de un programa del servidor.
El contenedor de *servlets* es el encargado de manejar las peticiones del cliente y pasarlas a los *servlets*, que reciben los datos y ejecutan una aplicación (véase apéndice C). Después se envía una respuesta al usuario.

Resultados:
Para presentar los resultados se utilizó un *applet* de *webmol* (software libre), que muestra la estructura tridimensional de una proteína. Este *applet* fue modificado, por Michael Cusack, para que los aminoácidos conservados aparezcan de un color diferente y para que al seleccionar un punto de la estructura se indique el aminoácido y su posición dentro de la secuencia.

4.2.1.2. Aplicación

En esta capa se agrupan los servicios que ofrece el sistema: *Service admission, Service prediction* (Figura 4.13).

Figura 4.13 Servicios que ofrece el sistema.

Service admission

Este servicio incluye tanto el inicio de sesión como el registro del usuario, pues son requisitos necesarios para ingresar al sistema. Un usuario que tiene cuenta en el sistema debe proporcionarla para ingresar, de lo contrario debe registrarse para obtenerla.
Las dos funciones del servicio de admisión son: 'Login system' y 'Register system' (Figura 4.14).

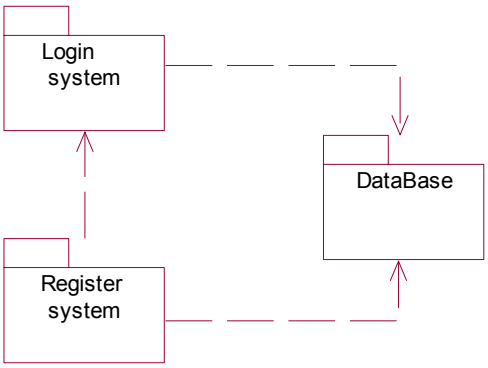

Figura 4.14

'Login system'

Se encarga de controlar el acceso de los usuarios al sistema. Para ingresar al sistema un usuario debe proporcionar su login y su password. El proceso para permitir o negar el acceso se muestra en el siguiente diagrama.

El diagrama de actividades es el siguiente:

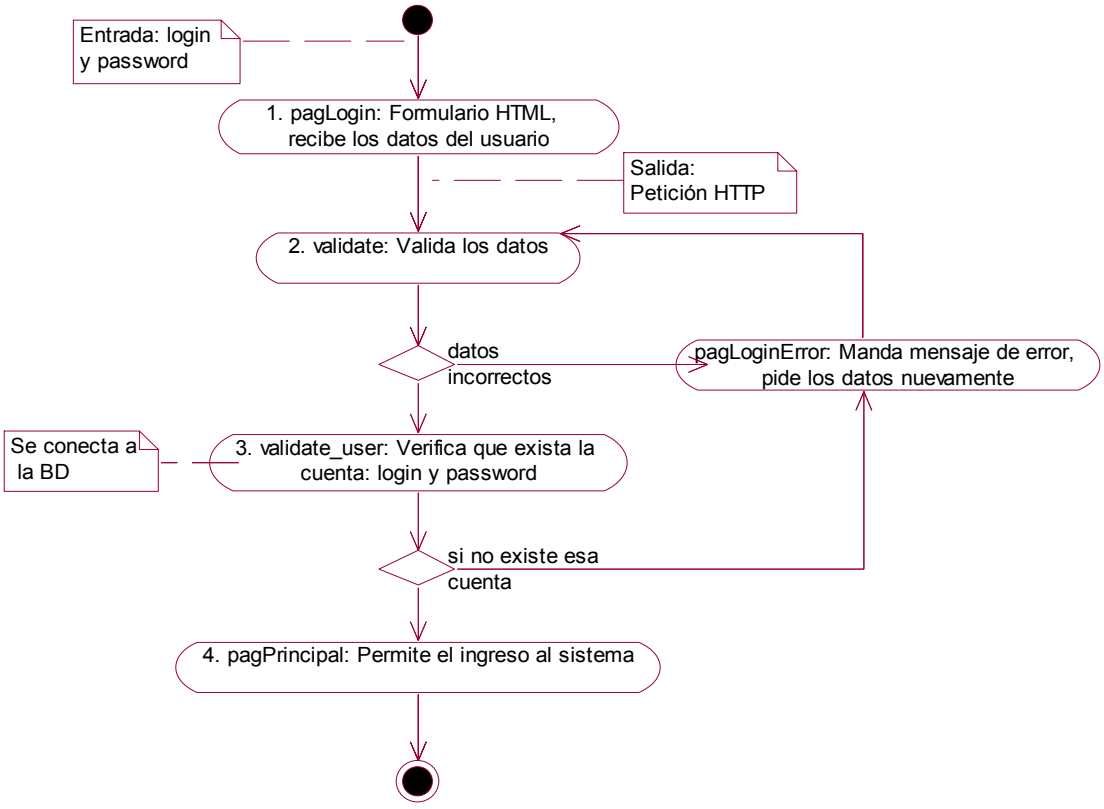

Figura 4.15 Diagrama de actividades de 'Login system'.

El diagrama de clases es el siguiente:

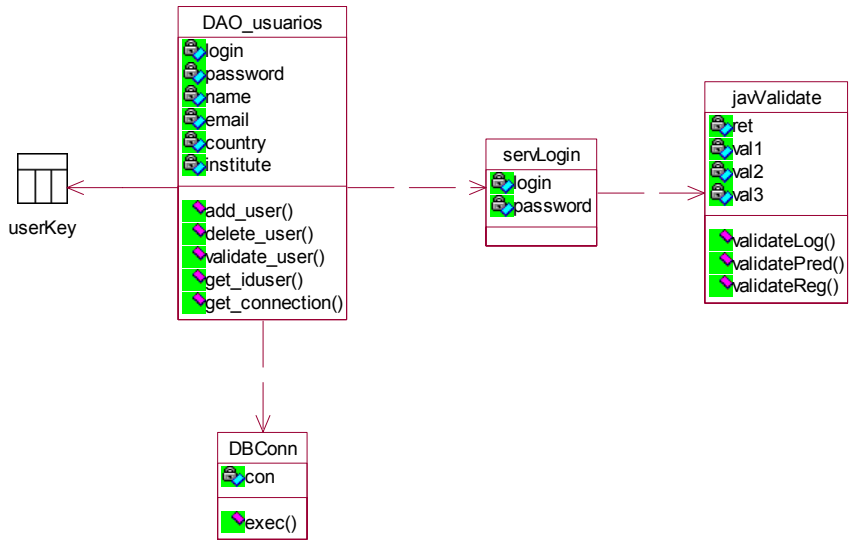

Figura 4.16 Diagrama de clases de 'Login system'.

Descripción

servLogin: Este *servlet* recibe los datos del formulario html, crea instancias de las clases DAO_users y javValidate.

javValidate: Valida los datos (campos no vacíos y tipo).

DAO_users: Contiene métodos para manipular la información de la base de datos. En esta función solo se utiliza *get_connection()*, *validate_user().*

DBConn: Se conecta a la base de datos a través del controlador JDBC de mysql.

'Register system'

Esta función se encarga de registrar un usuario al sistema, agrega sus datos a la base de datos.

El diagrama de actividades es el siguiente:

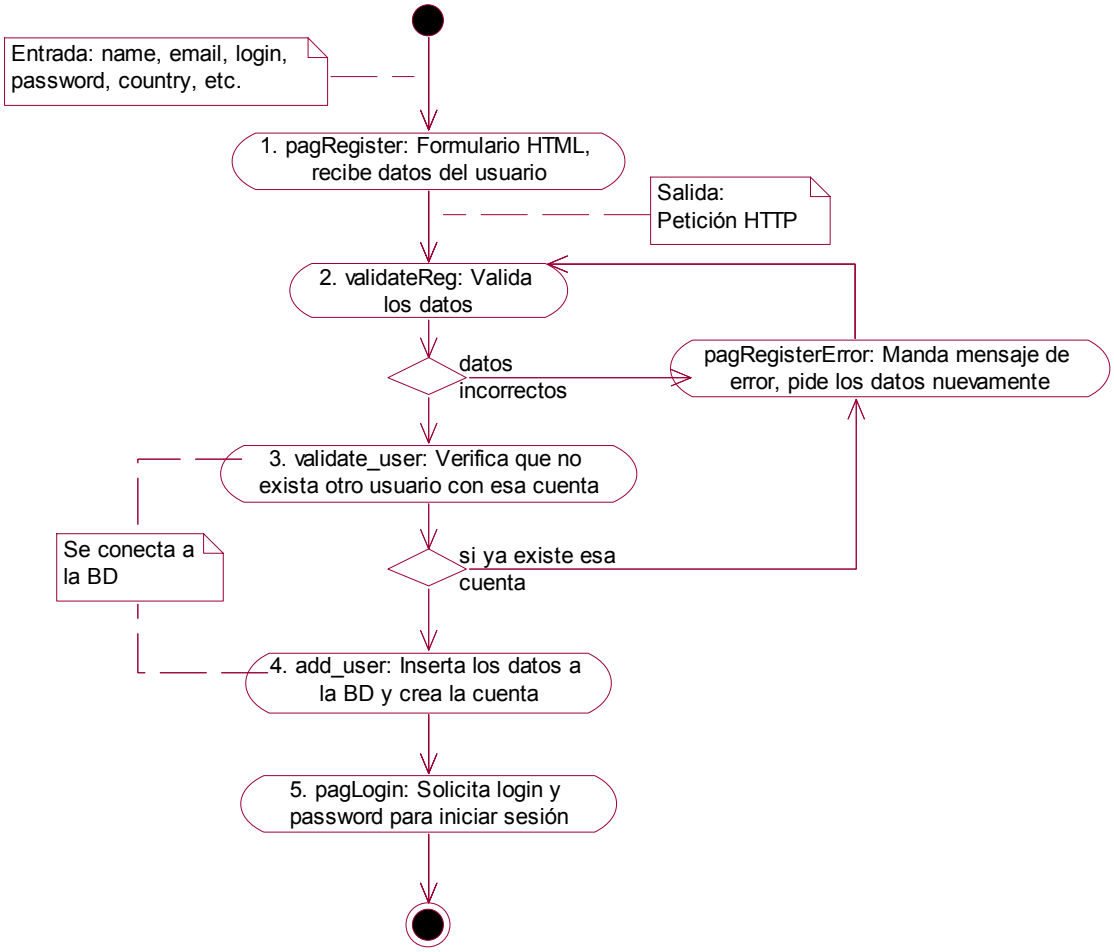

Figura 4.17 Diagrama de actividades de 'Register system'.

El diagrama de clases es el siguiente:

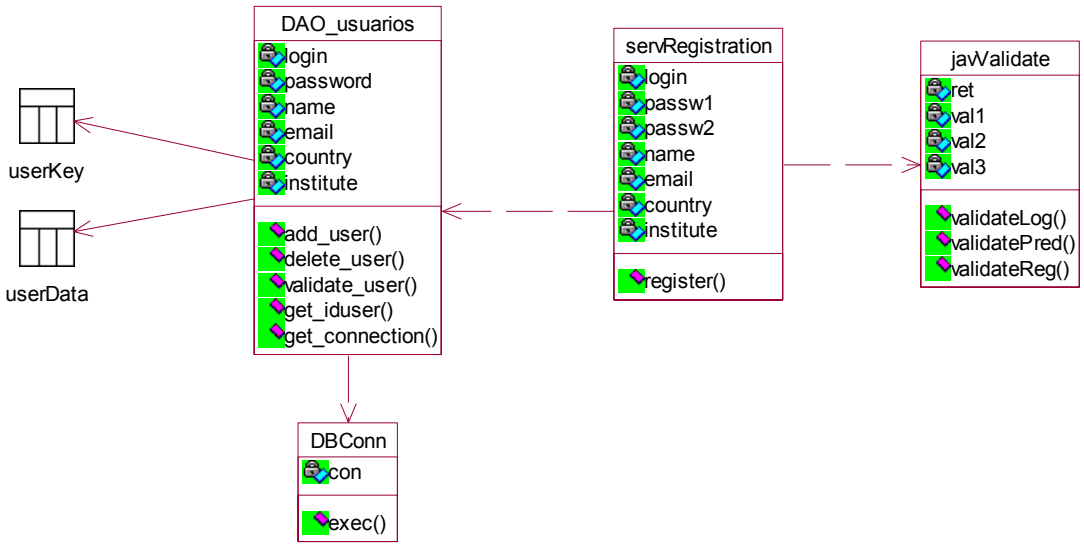

Figura 4.18 Diagrama de clases de 'Register system'.

Descripción

servRegistration: Recibe los datos del formulario HTML, crea instancias de las clases DAO_users y javValidate.

javValidate: Valida los datos del usuario (campos no vacíos y tipo).

DAO_users: Contiene métodos para manipular la información de la base de datos. En esta función solo se utiliza *get_connection()*, *validate_user()*, *get_iduser()*, *add_user().*

DBConn: Se conecta con la base de datos, a través del controlador JDBC de mysql.

Service Prediction

La arquitectura del sistema contiene los bloques en los que se divide el proceso de la predicción, cada bloque realiza una función distinta de la aplicación (crear un proceso, despacharlo o monitorear el proceso).

El siguiente diagrama muestra el proceso de la predicción:

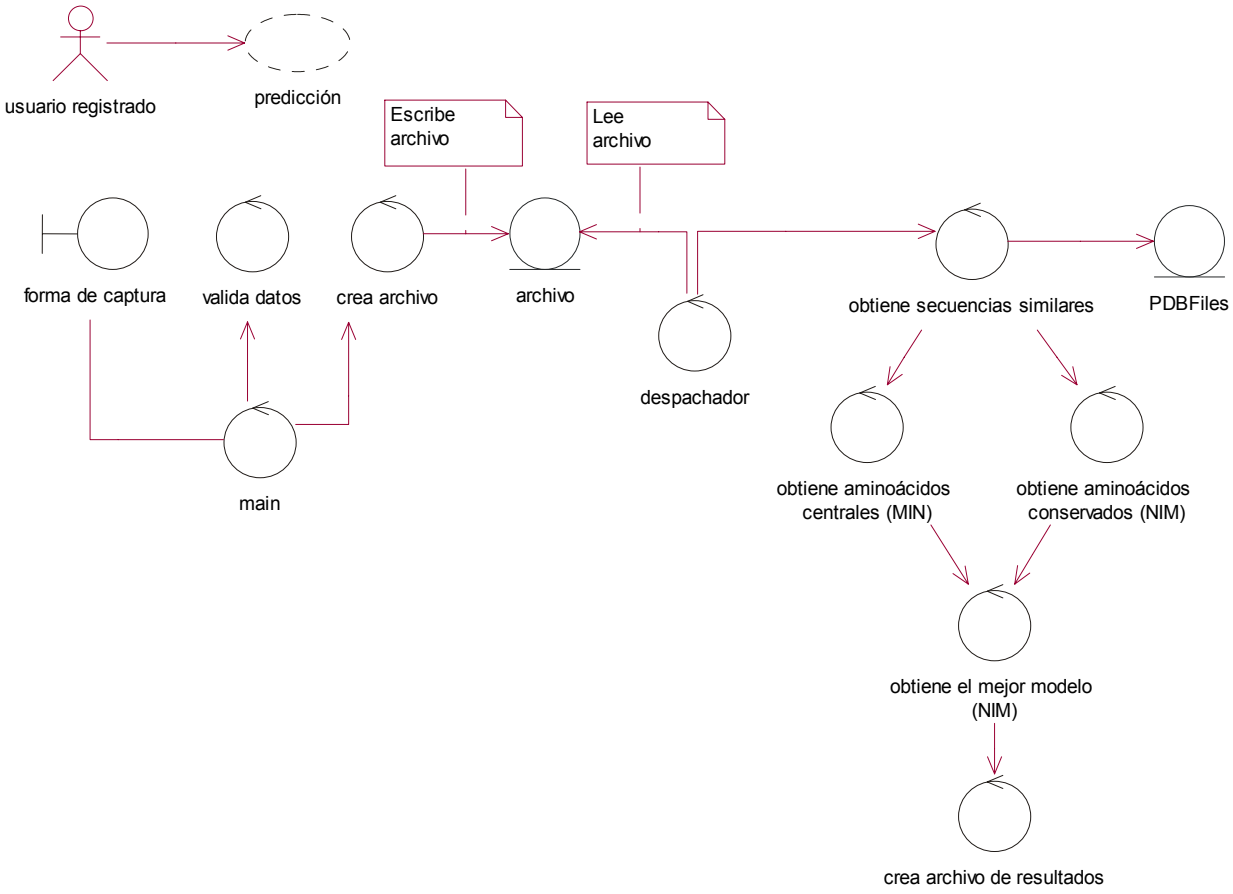

Figura 4.19 Proceso de predicción.

Los bloques de este servicio son los siguientes:

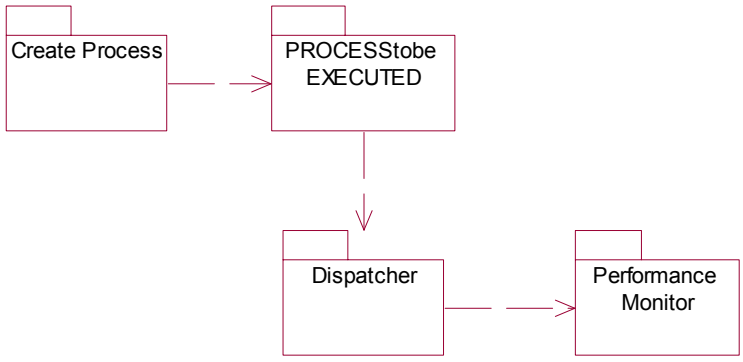

Figura 4.20 Bloques del proceso de predicción.

CreateProcess: Crea una solicitud de predicción.
PROCESStobeEXECUTED: Son los procesos que están en espera de ser ejecutados.
Dispatcher: Es el encargado de ejecutar los procesos.
PerformanceMonitor: Verifica que el proceso de predicción haya sido correcto.

- *CreateProcess*

Cuando un usuario realiza una petición de predicción, debe ingresar los parámetros de la predicción (para obtener los aminoácidos centrales y los conservados), así como la secuencia de la proteína, de la cual se desea conocer su estructura tridimensional.
Después de validar los datos se generan dos archivos, uno contiene los parámetros para la predicción (processx) y el otro con la secuencia de la proteína (processx.tfa).

Un *proceso* es un archivo que contiene los parámetros para realizar la predicción: el nombre del proceso se forma con el *id* de la sesión y con el número de procesos en cola (por ejemplo: process22lq6i615xz).

El diagrama de actividades es el siguiente:

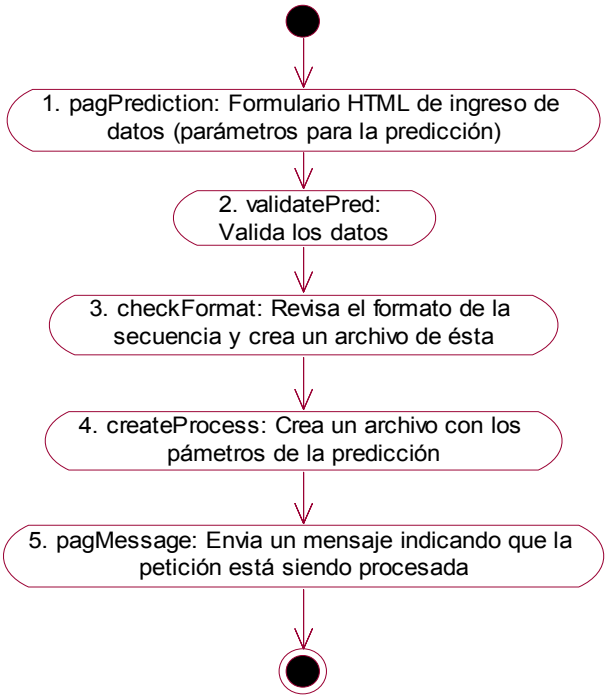

Figura 4.21 Diagrama de actividades de *Create process*.

El diagrama de clases es el siguiente:

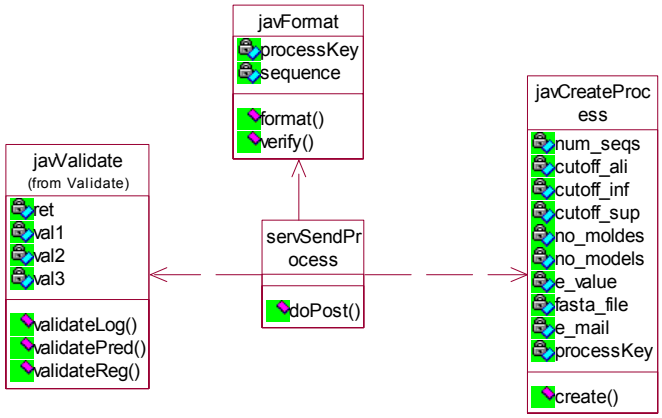

Figura 4.22 Diagrama de clases de *Create process*.

- *PROCESSStobeEXECUTED*

En el momento en el que se genera el archivo con los parámetros se puede decir que se ha creado un proceso, que está en espera de ser ejecutado. El nombre de este proceso se almacena en una base de datos, esto permite tener control sobre los procesos enviados y sobre los procesos ejecutados.

PROCESSStobeEXECUTED es una carpeta que contiene todos los procesos o solicitudes de predicción.

- *Dispatcher*

Es un demonio o programa que permanece en segundo plano ejecutándose continuamente. Este demonio (dispatcher.java) revisa cada tres segundos el directorio que contiene los procesos a ser ejecutados (/PROCESSStobeEXECUTED) y almacena el nombre de los procesos en un arreglo para ejecutarlos.

Cuando hay varios procesos en cola *Dispatcher* crea un hilo del programa *PerformanceMonitor* para cada proceso, y ejecuta los procesos simultáneamente, de forma paralela.

El diagrama de actividades es el siguiente:

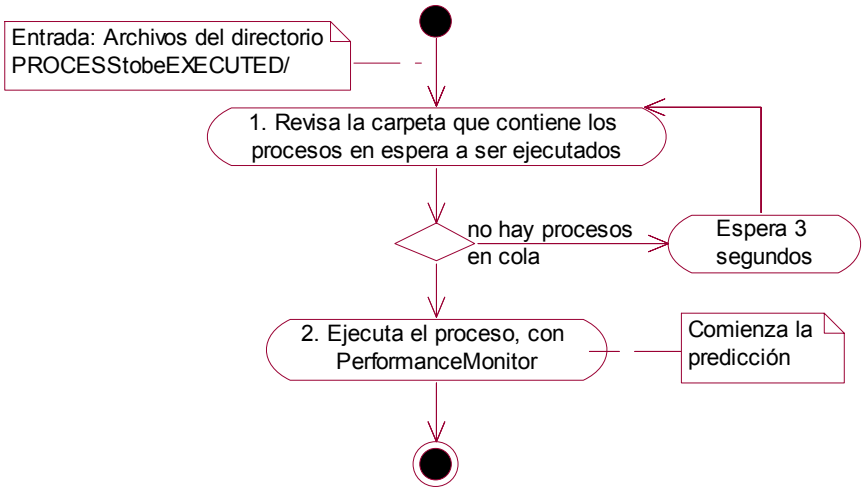

Figura 4.23 Diagrama de actividades de *Dispatcher.*

El diagrama de clases es el siguiente:

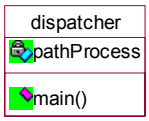

Figura 4.24 Diagrama de clases de *Dispatcher*.

- *PerformanceMonitor*

Ejecuta los programas que realizan la predicción. Se encarga de monitorear el proceso de la predicción, verifica que cada etapa se haya realizado exitosamente, en caso de que no sea así, termina el proceso y manda un mensaje de error al usuario.

PerformanceMonitor recibe el proceso a ser ejecutado (archivo con parámetros), toma la información que requiere y elimina el proceso de la cola de espera. Se encarga de enviar y recibir los datos necesarios para cada etapa de la predicción.

El diagrama de actividades es el siguiente:

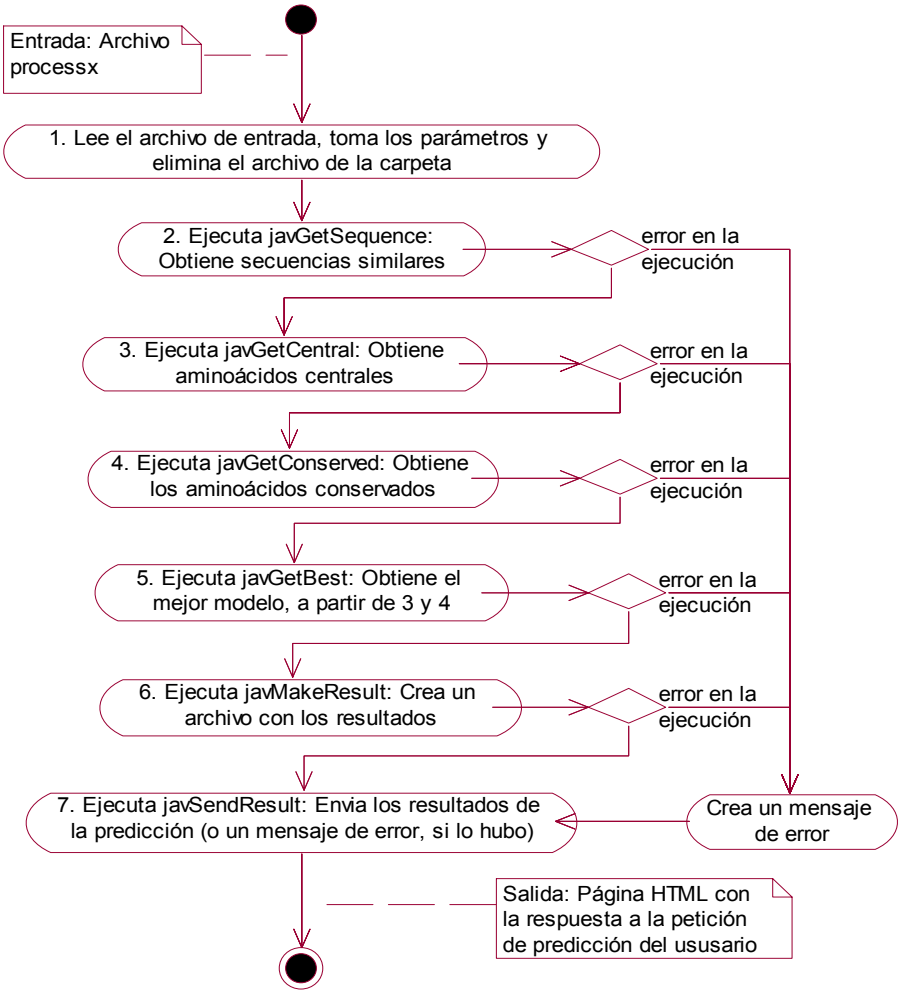

Figura 4.25 Diagrama de actividades de *PerformanceMonitor*.

Cada etapa de la predicción se implementó en un paquete: *GetSequence, GetCentral, GetConserved, GetBest, Result (MakeResult, SendResult)* (Figura 4.26).

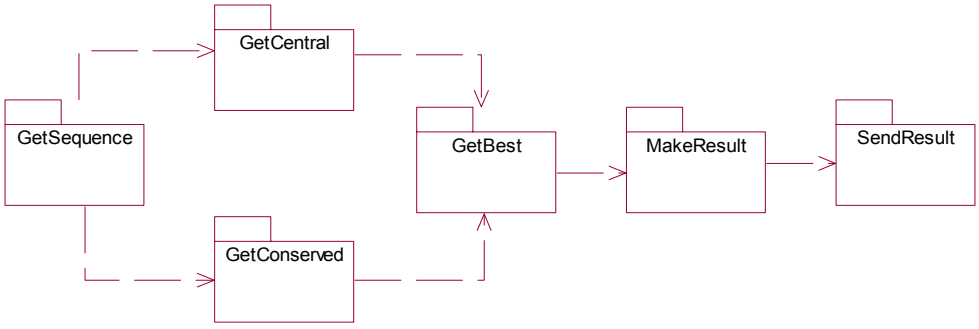

Figura 4.26 Etapas de la predicción.

A continuación se describe cada etapa de la predicción y los programas que deben implementarse para su funcionamiento.

I. GetSequence

GetSequence se conecta a la base de datos, PDBFiles o nr (*non redundant*), que contiene información de las proteínas, y realiza una selección de las proteínas que son similares a la proteína objetivo (de la que se pretende conocer la estructura).

Esta información sirve para las dos etapas siguientes: *GetCentral* y *GetConserved*.

El diagrama de actividades es el siguiente:

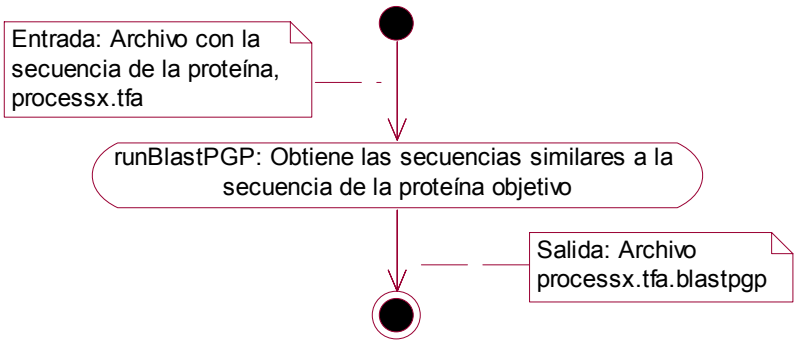

Figura 4.27 Diagrama de actividades de *GetSequence.*

El diagrama de clases es el siguiente:

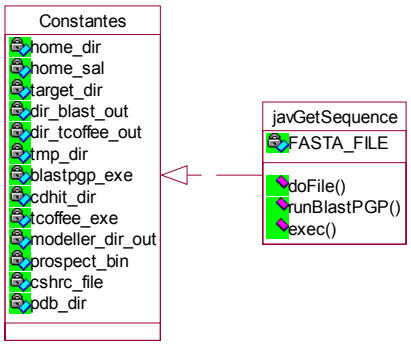

Figura 4.28 Diagrama de clases de *GetSequence.*

Descripción

javGetSecuence: Crea un archivo con la instrucción para ejecutar el programa blastpgp.

II. GetCentral

En esta etapa se obtienen los aminoácidos centrales, se implementa el algoritmo MIN (capítulo 1).

El diagrama de actividades es el siguiente:

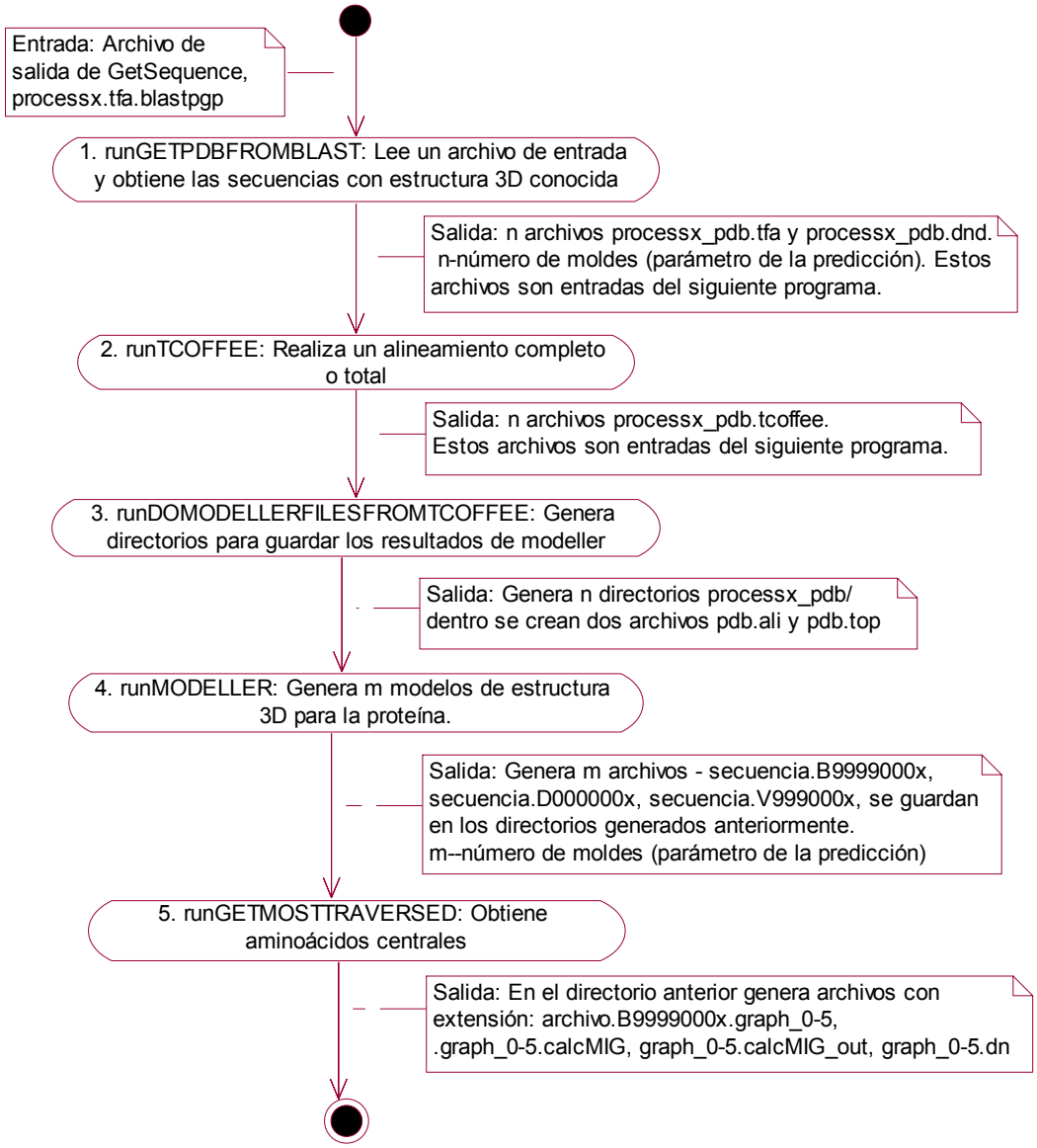

Figura 4.29 Diagrama de actividades de *GetCentral.*

El diagrama de clases es el siguiente:

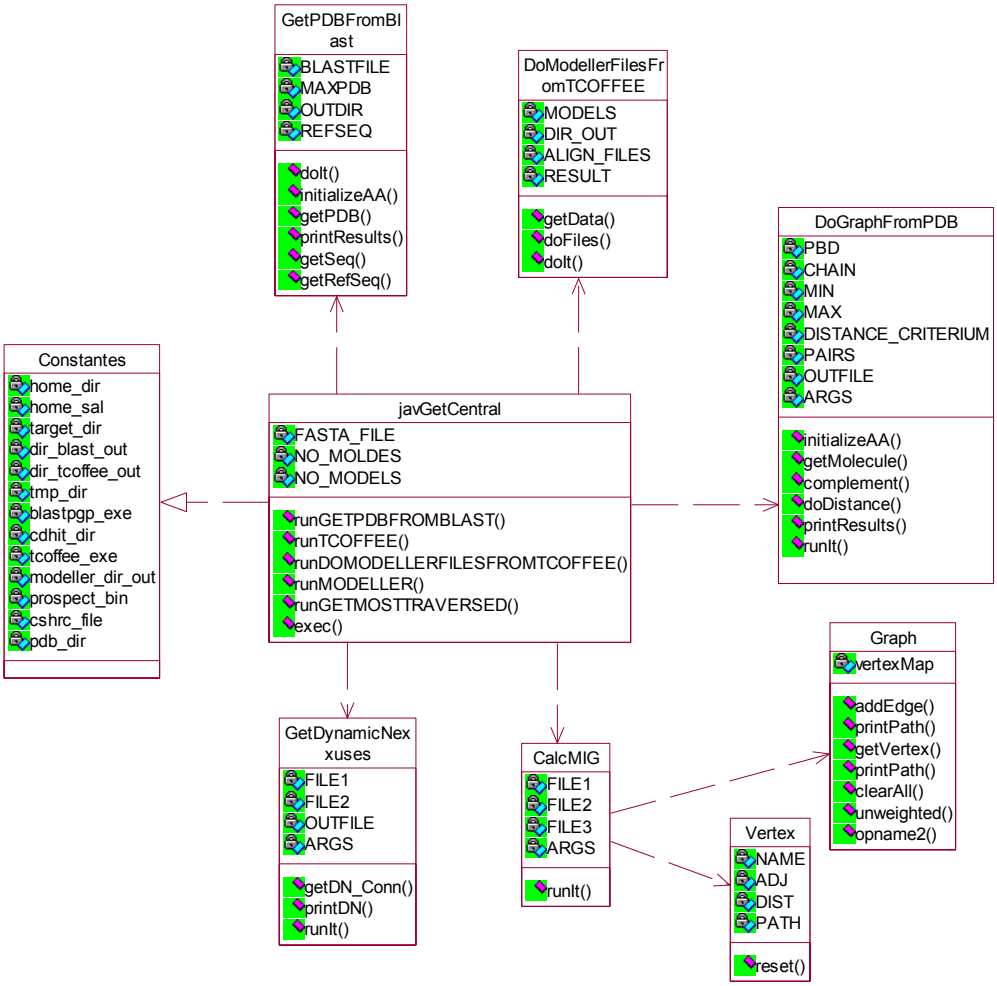

Figura 4.30 Diagrama de clases de *GetCentral.*

Descripción

javGetCentral: Ejecuta los métodos necesarios para obtener los aminoácidos centrales.
GetPDBFromBlast: Obtiene las secuencias con estructura conocida.
DoModellerFiles: Crea un sistema de directorios para almacenar los resultados de 'modeller'.
DoGraphFromPDB: Traza una gráfica con los aminoácidos de la proteína.
CalcMIG: Traza la ruta más corta que conecta cada elemento de la red (algoritmo de la ruta mínima).
GetDynamicNexxuses: Determina la conectividad de la red.

III. GetConserved

GetConserved obtiene los aminoácidos conservados de la proteína.

El diagrama de actividades es el siguiente:

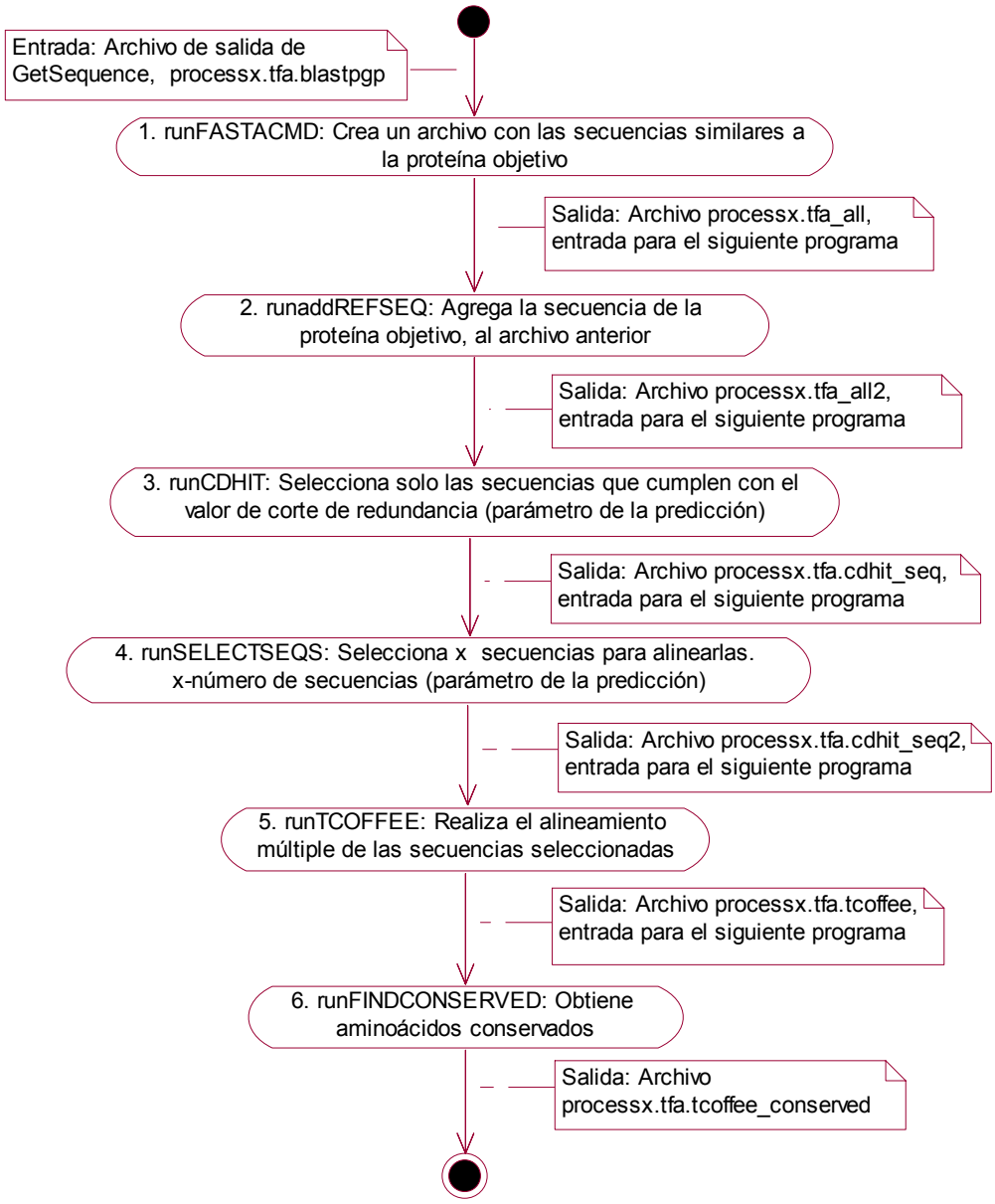

Figura 4.31 Diagrama de actividades de *GetConserved.*

El diagrama de clases es el siguiente:

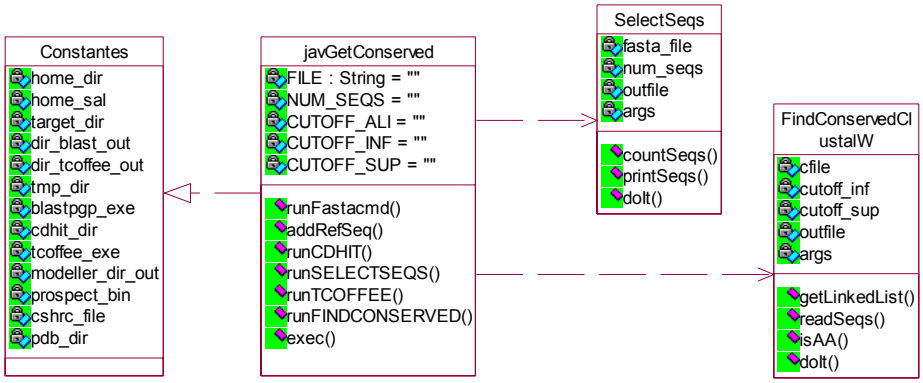

Figura 4.32 Diagrama de clases de *GetConserved.*

Descripción

javGetConserved: Ejecuta los métodos necesarios para obtener los aminoácidos conservados.

SelectSeqs: Obtiene el número de secuencias definido como parámetro de la predicción.

FindConservedClustalW: Realiza el alineamiento múltiple y obtiene los aminoácidos conservados.

IV. GetBest

En esta etapa se elige el mejor modelo generado, basándose en el algoritmo NIM (capítulo 1).

Este criterio compara los aminoácidos centrales (obtenidos en *GetCentral*) con los conservados (obtenidos en *GetConserved*), el modelo que tenga una mayor correspondencia entre estos dos valores es considerado como el mejor.

El diagrama de actividades es el siguiente:

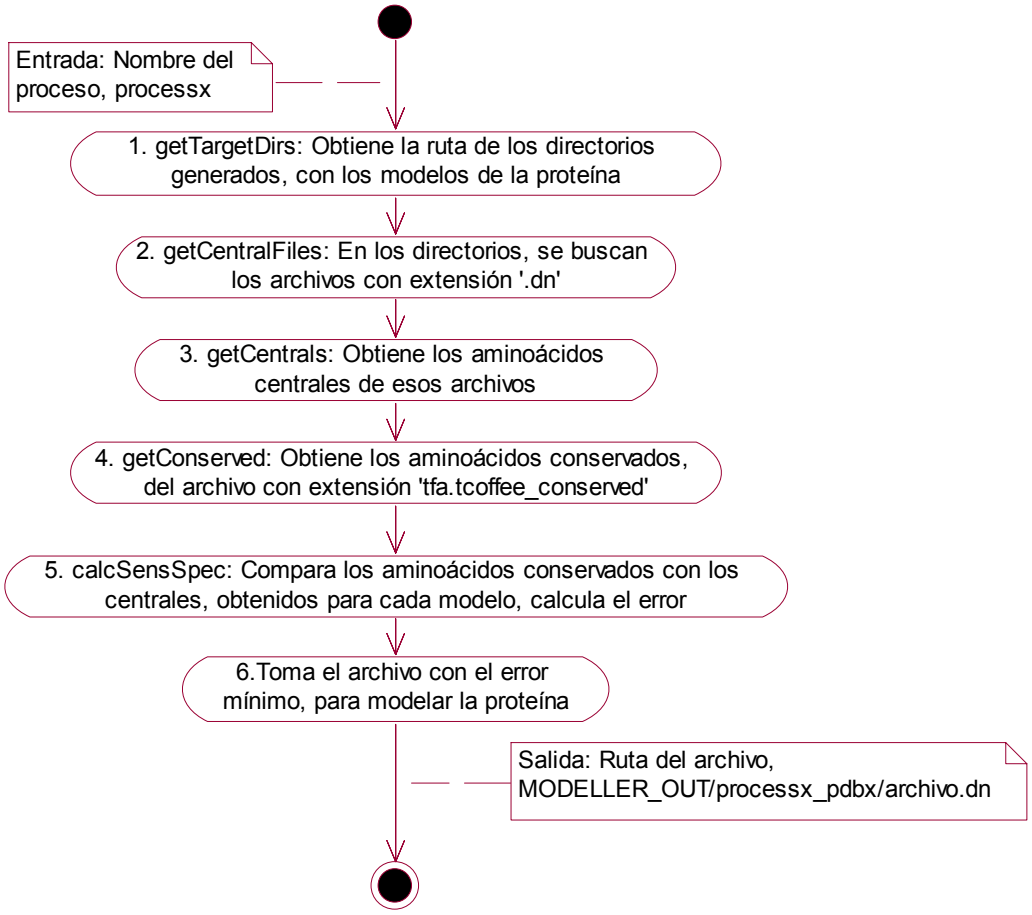

Figura 4.33 Diagrama de actividades de *GetBest.*

El diagrama de clases de *GetBest* es el siguiente:

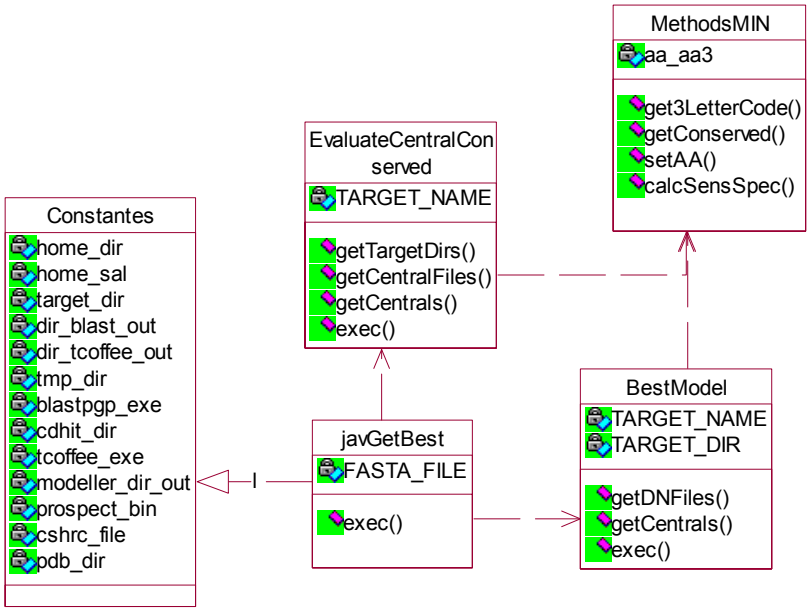

Figura 4.34 Diagrama de clases de *GetBest.*

Descripción

javGetBest: Crea instancias de EvaluateCentralConserved y de BestModel

EvaluateCentralConserved: Obtiene el directorio con los modelos de error mínimo

BestModel: Obtiene el mejor modelo del directorio anterior.

MethodsMIN: Contiene métodos necesarios para las clases anteriores

V. MakeResult

El programa *MakeResult* crea una página (archivo de código HTML) que mostrará los resultados de la petición de predicción. Esta página (processx.html) invoca un *applet* que muestra la estructura tridimensional de la proteína. Los parámetros de este *applet* son tres: el nombre de la proteína, los aminoácidos conservados y las coordenadas de sus átomos (obtenidas del modelo generado).

El diagrama de actividades es el siguiente:

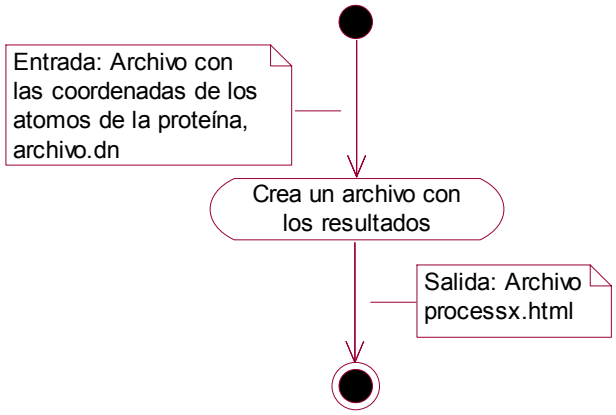

Figura 4.35 Diagrama de actividades de *MakeResult.*

El diagrama de clases es el siguiente:

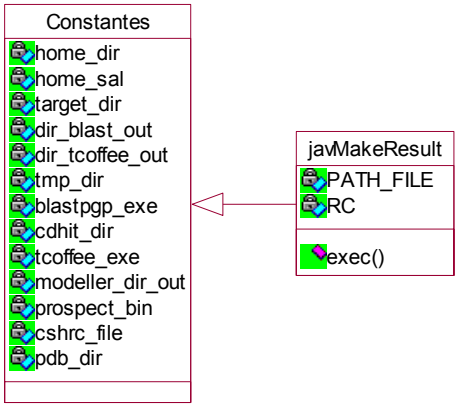

Figura 4.36 Diagrama de clases de *MakeResult.*

Descripción

javMakeResult: Crea un archivo con los resultados.

VI. SendResult

Finalmente se envía un correo al usuario indicándole la liga a la página donde puede ver los resultados de su consulta, esta página está almacenada en el servidor. Si hubo algún error durante la predicción se envía un mensaje de error.

El diagrama de actividades es el siguiente:

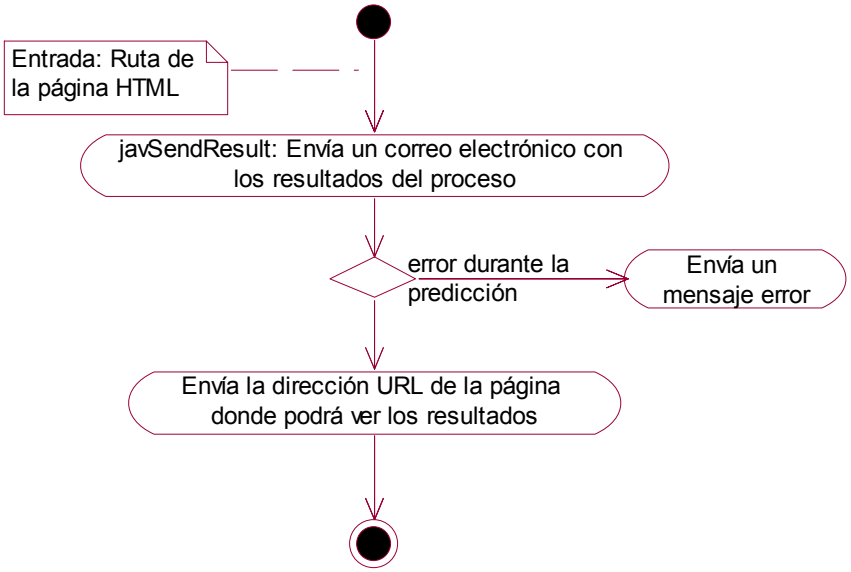

Figura 4.37 Diagrama de actividades de *SendResult.*

El diagrama de clases es el siguiente:

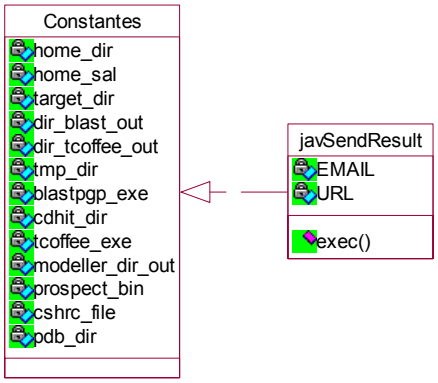

Figura 4.38 Diagrama de clases de *SendResult.*

Descripción

javSendResult: Envía un correo electrónico al usuario.

4.2.1.3. Datos

Datos del usuario:

userKey

Campo	Tipo	Null	Key	Default	*Ejemplo*
Id_user	int(4) unsigned	no	pk	0	*1*
login	varchar(15)	no			*bioifc*
password	varchar(15)	no			*user01*

userData

Campo	Tipo	Null	Key	Default	*Ejemplo*
id_user	int(4) unsigned	no	fk	0	*1*
name	varchar(25)	yes			*bioinformatics*
email	varchar(25)	no			*bioinfo@ifc.unam.mx*
country	varchar(25)	no			*MEXICO*
institute	varchar(25)	no			*IFC, UNAM*

Datos del proceso:

Procesos en espera de ser ejecutados

processQueue

Campo	Tipo	Null	Key	Default	*Ejemplo*
id_proc	int(4) unsigned	no	pk	0	*1*
process	varchar(20)	no			*process71lq6i615xz1*

Procesos ejecutados

processExecuted

Campo	Tipo	Null	Key	Default	*Ejemplo*
no_proc	int(4) unsigned	no	pk	0	*5*
date	varchar(20)	no			

Datos de los documentos:

documents

Campo	Tipo	Null	Key	Default	*Ejemplo*
no_doc	int(4) unsigned	no	pk	0	*1*
path	varchar(40)	no			*/INTERFACE/documents/MIN.doc*

Diseño de la base de datos:

Para almacenar y accesar a esta información se creó una base de datos, estos datos se manejan desde los *servlets* a través del conector *JDBC* de mysql.

Nombre de la base de datos: users

Nombre de las tablas:

userKey
userData
processQueue
processExecuted
documents

Diagrama de la base de datos:

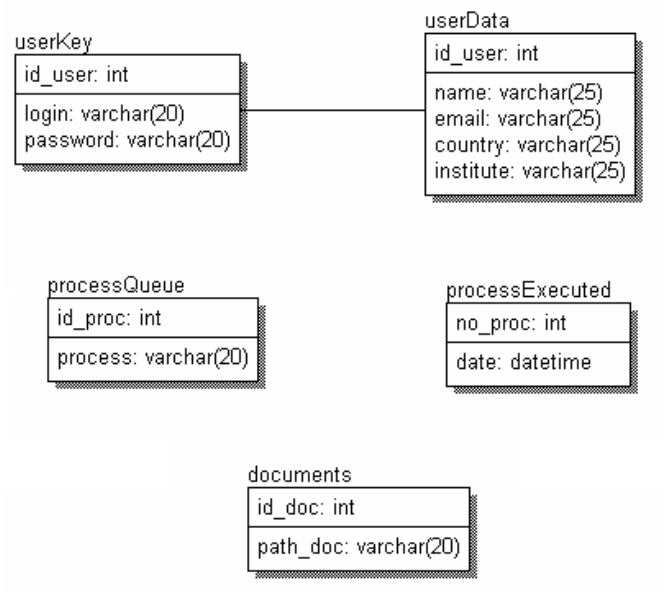

Figura 4.39 Base de datos.

Capítulo 5

Implementación del sistema

La implementación del servidor muestra los pasos que se siguieron para instalar la aplicación: el sistema de directorios, de archivos y los paquetes con los programas necesarios para la predicción de la estructura tridimensional de las proteínas. Incluye también la configuración de los componentes de la aplicación y la presentación del sitio.

Para explicar el proceso de implementación el sistema se divide en dos partes: la aplicación (los paquetes de la predicción) y la interfaz gráfica.

5.1 LA APLICACIÓN

La aplicación consta de 4 partes (Beust, C. et Al., 2001):

Un directorio público
Un archivo WEB-INF*/web.xml*
Un directorio WEB-INF*/classes*
Un directorio WEB-INF*/lib*

Modelo general de implementación (Figura 5.1)

Archivo / Carpeta	*Descripción:*
PSPrediction/	Carpeta raíz
index.html	Página de presentación del sistema
INTERFACE	Contiene los archivos de la interfaz gráfica
WEB-INF	Área privada de la aplicación, contiene los programas que se ejecutan para la predicción

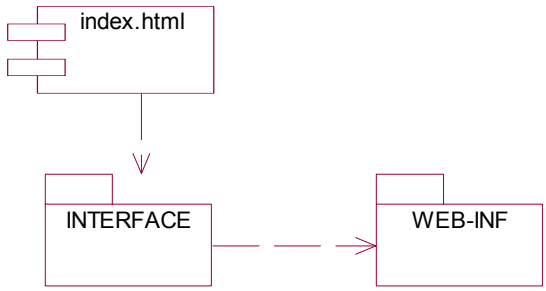

Figura 5.1 Modelo general de implementación.

El **área pública** es el directorios raíz donde se encuentra la aplicación (PSPrediction/), sin incluir el directorio WEB-INF. En esta aplicación el área publica consta de dos elementos: el archivo *index.html* y el directorio INTERFACE/.

El archivo *index.html* es la página de presentación del sistema.

Modelo de implementación de ***INTERFACE*** (Figura 5.2)

En este directorio se encuentran los archivos que forman la interfaz gráfica del servidor (documentos HTML, imágenes y *applets*).

Archivo / Carpeta	*Descripción:*
pagHTML/	Contiene las páginas HTML que se muestran al usuario
pagRESULT/	Contiene los archivos generados por MakeResult, también las páginas HTML que muestran el applet con los resultados de la estructura de la proteína
documents/	Contiene los documentos que se publican en el sitio y también un archivo con ligas a otros sitios
images/	Contiene las imágenes que las páginas utilizan

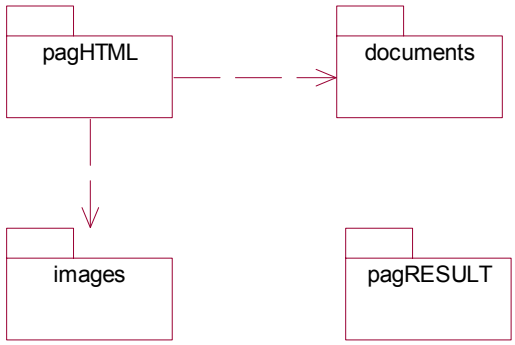

Figura 5.2 Modelo de implementación de *INTERFACE.*

El directorio WEB-INF es un *área privada*. Los archivos de este directorio están destinados únicamente al uso del contenedor de *servlets*. WEB-INF contiene, entre otras cosas, el descriptor de despliegue, un directorio *classes* y un directorio *lib*.

Modelo de implementación de ***WEB-INF*** (Figura 5.3)

Archivo / Carpeta	*Descripción:*
web.xml	Archivo de configuración de la aplicación
lib/	Librerías y archivos empaquetados
classes/	Contiene los programas necesarios para la aplicación

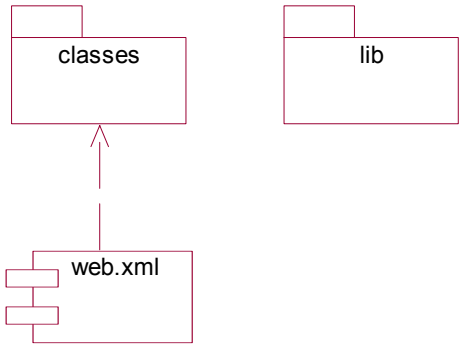

Figura 5.3 Modelo de implementación de *WEB-INF.*

El contenedor de *servlets* tiene un descriptor de despliegue, un archivo XML llamado *web.xml*. El descriptor ayuda a gestionar la configuración de la aplicación, una vez iniciado el servidor de páginas (Kurniawan, B., 2002).

Configuración de los *servlets*:
Todos los *servlets* que se generan en la aplicación deben ser declarados en el descriptor de despliegue. Debe definirse el nombre del programa (*archivo.java* y *archivo.class*) y la dirección URL donde se mostrará el *servlet*.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
"http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">

<web-app>
  <servlet>
   <servlet-name>Bioinformatics</servlet-name>
   <servlet-class>Bioinformatics</servlet-class>
  </servlet>
  <servlet-mapping>
   <servlet-name>Bioinformatics</servlet-name>
   <url-pattern>/Bioinformatics</url-pattern>
  </servlet-mapping>
</web-app>
```

En el directorio *lib* pueden ser copiados los archivos empaquetados (JAR) de una aplicación.

Modelo de implementación de ***classes*** (Figura 5.4)

El directorio *classes* es utilizado para almacenar *servlets* compilados y otras clases de utilidad.

Archivo / Carpeta	*Descripción:*
ServiceAdmission/	Contiene las clases necesarias para validar y registrar un usuario
PROCESSStobeEXECUTED/	Contiene la lista de procesos que están en espera de ser ejecutados
PerformanceMonitor.java	Programa encargado de monitorear el proceso de la predicción
prediction/	Carpeta que contiene los paquetes para realizar la predicción y las carpetas con los archivos de salida
MAKEFILER/	Carpeta que contiene las variables del sistema, como la ruta de las carpetas y los paquetes donde se encuentran los programas y las clases de la aplicación
PROGTEMP/	Contiene programas que sirven de prueba, para la predicción

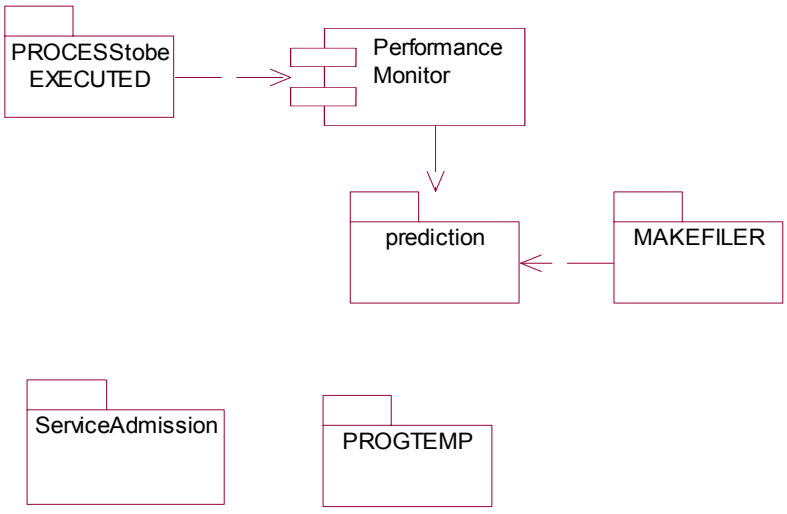

Figura 5.4 Modelo de implementación de *classes.*

Modelo de implementación de ***prediction*** (Figura 5.5)

Archivo / Carpeta	*Descripción:*
GetSequence/	Paquete con los programas que implementan GetSequence
GetCentral/	Paquete con los programas que implementan GetCentral
GetConserved/	Paquete con los programas que implementan GetConserved

GetBest/	Paquete con los programas que implementan GetBest
Result/	Paquete con los programas que implementan MakeResult y SendResult
TARGETS/	Carpeta que contiene archivos con las secuencias de las proteínas enviadas para la predicción (secuencia objetivo)
RESULTS_BLAST/	Carpeta que contiene archivos con los resultados de GetSequence y algunos archivos generados por GetCentral
RESULTS_TCOFFE/	Carpeta que contiene archivos con los resultados de GetConserved
MODELLER_OUT/	Carpeta que contiene archivos con los resultados de GetCentral
TMP/	Carpeta que contiene archivos con resultados parciales de GetConserved

Los paquetes *GetSequence*, *GetCentral*, *GetConserved*, *GetBest* y *Result* (*MakeResult* y *SendResult*) corresponden a las etapas de la predicción. Las clases que se crearon en cada paquete se describen en 'Diseño del servidor' (capítulo 4).

Las carpetas: TARGETS/, RESULTS_BLAST/, RESULTS_TCOFFE/, MODELLER_OUT/, TMP/, contienen los archivos generados durante el proceso de predicción.

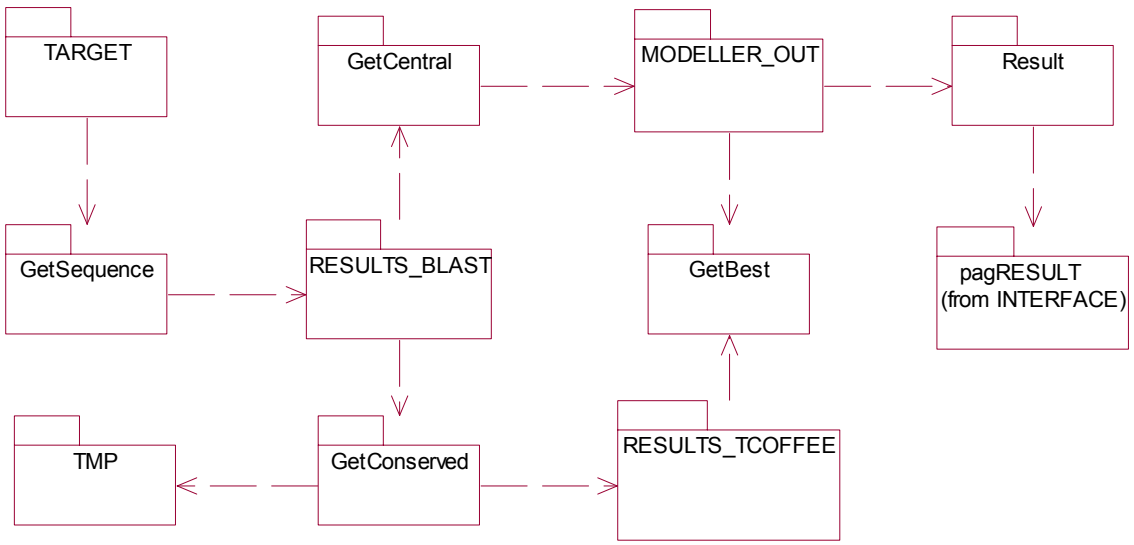

Figura 5.5 Modelo de implementación de *prediction.*

5.2 LA INTERFAZ GRÁFICA DEL SISTEMA

El conjunto de páginas generadas para presentar la aplicación se muestran a continuación.

1. Pagina inicial (Figura 5.6)
Muestra la presentación del sistema.

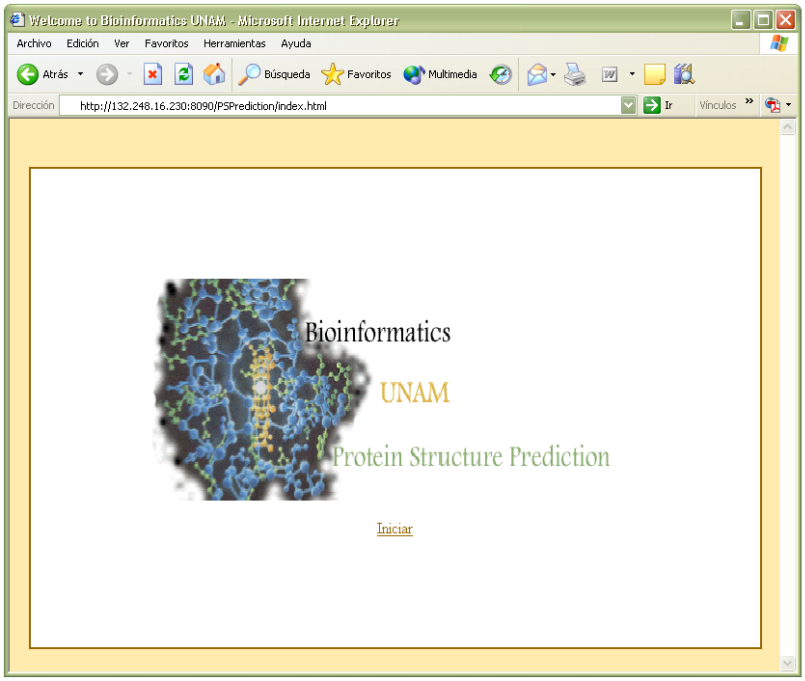

Figura 5.6 Página inicial.

El archivo de inicio debe definirse en el descriptor *web.xml*:

```
<welcome-file-list>
  <welcome-file>
    index.html
  </welcome-file>
</welcome-file-list>
```

2. Iniciar sesión (Figura 5.7)

Esta página solicita el nombre y la contraseña del usuario, para permitir el ingreso al sistema.

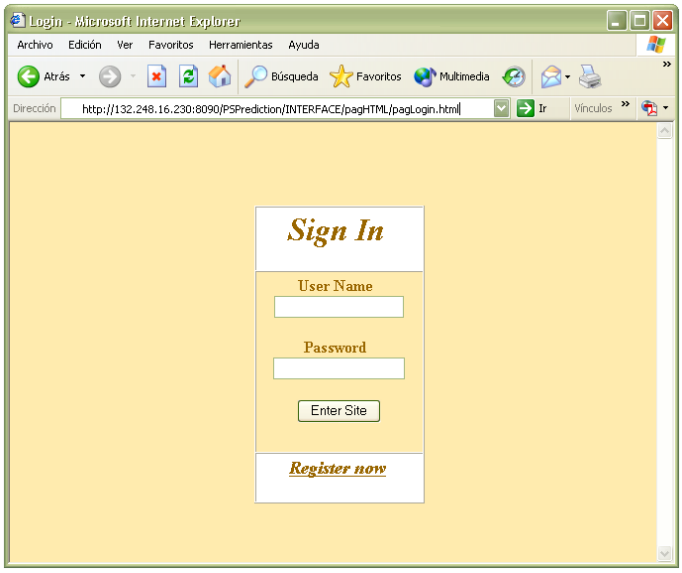

Figura 5.7 Página de ingreso al sistema.

3. Registrarse

La pantalla que solicita los datos, para registrar un usuario, es la siguiente:

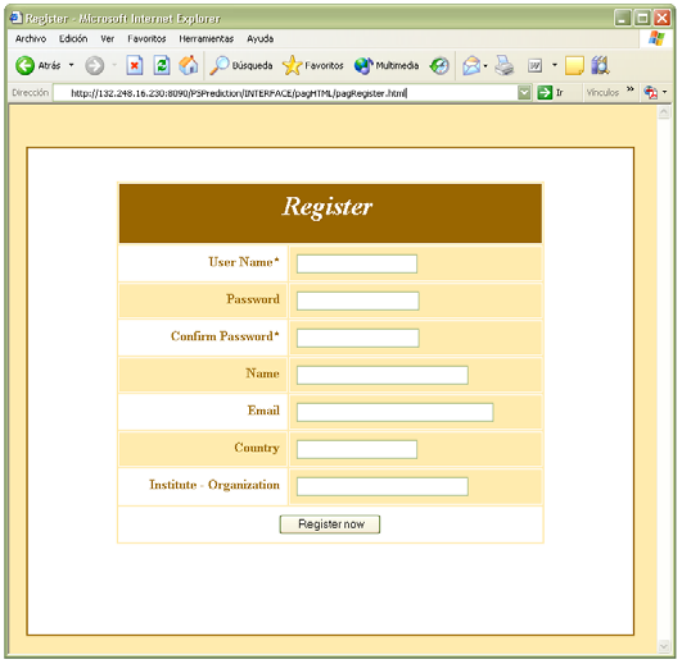

Figura 5.8 Página de registro.

El paquete S*ervice admission* permite validar los datos (de inicio de sesión y de registro) y conectarse a la base de datos.

4. Página principal (Figura 5.9)
Muestra información sobre el grupo dedicado a la predicción de estructuras de proteínas con el que se desarrolló el proyecto.

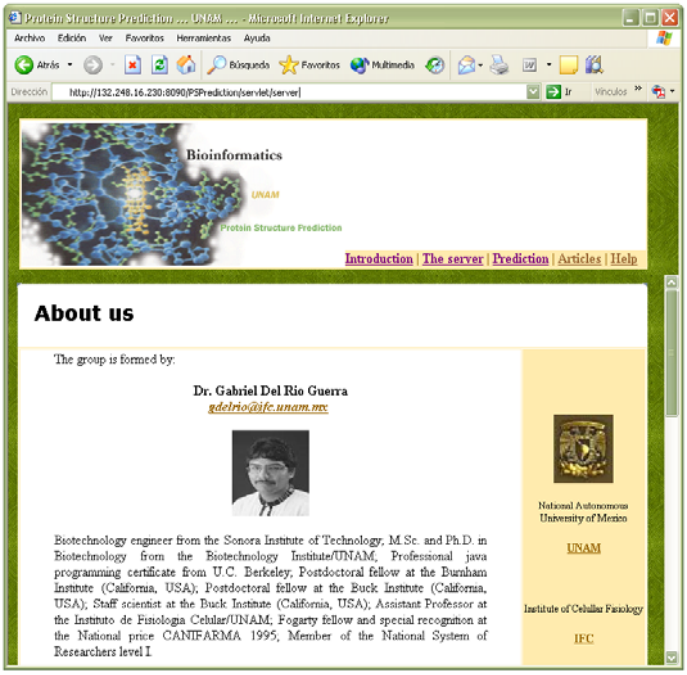

Figura 5.9 Página principal.

5. El servidor (Figura 5.10)
Muestra información sobre el servidor, su importancia, sus objetivos, etc..

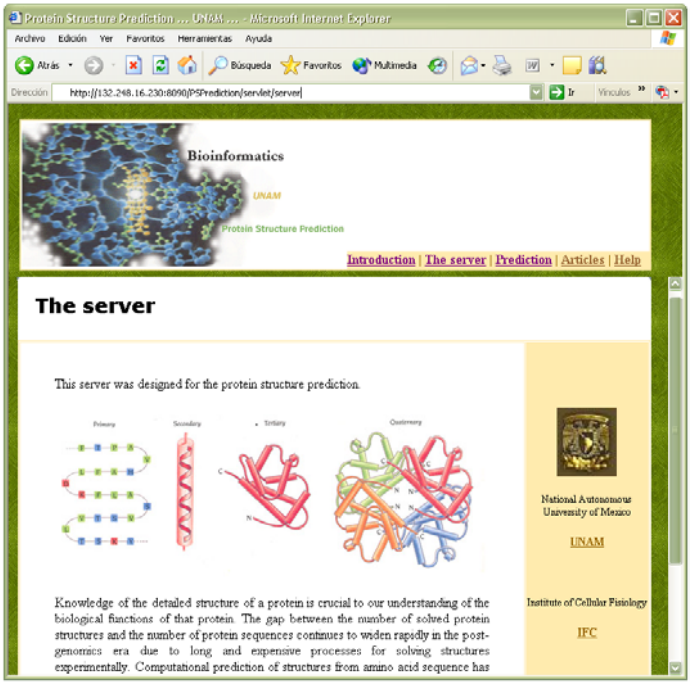

Figura 5.10 Página con información del servidor.

6. La predicción (Figura 5.11)

Esta página contiene información sobre el método utilizado para la predicción (algoritmos MIN y NIM).

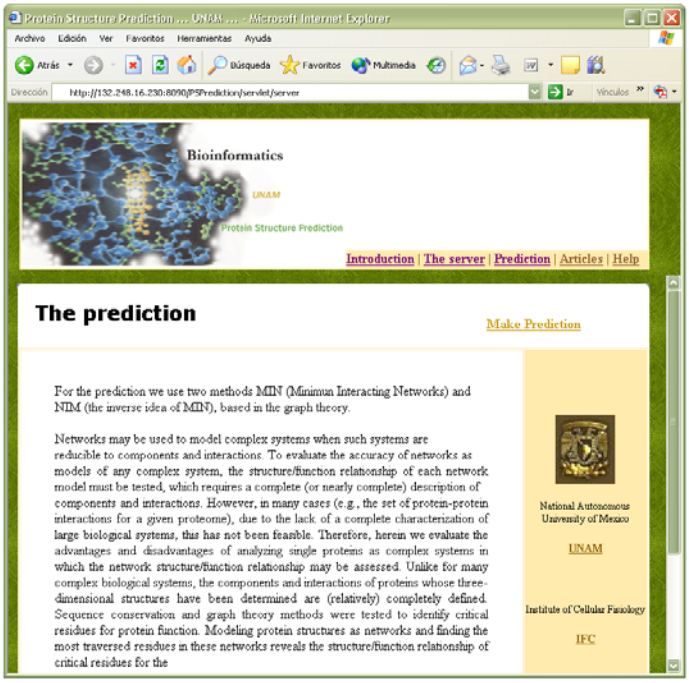

Figura 5.11 Página con información de la predicción.

7. Parámetros de la predicción (Figura 5.12)

Formulario que solicita los parámetros para la predicción.

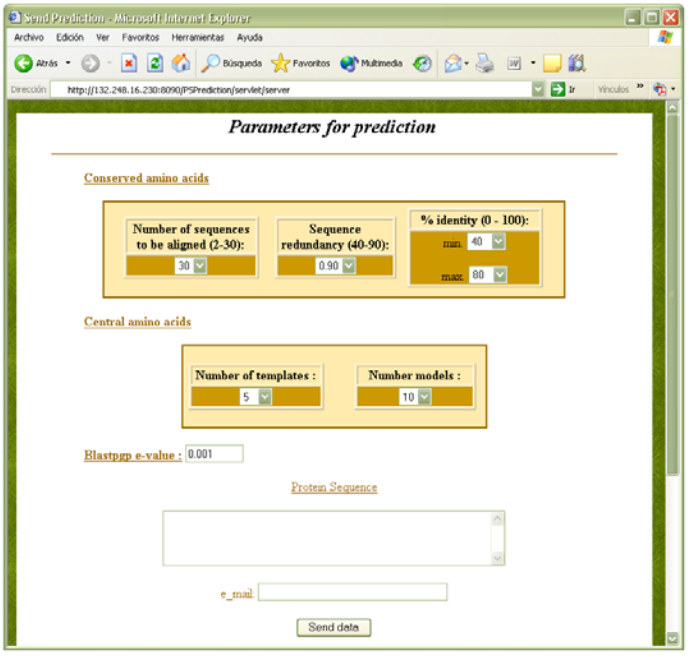

Figura 5.12 Página con los parámetros de la predicción.

Al enviar la petición de la predicción se genera un archivo (o proceso), al que se le asigna un nombre de acuerdo al *id* de la sesión y al número de procesos en cola.

El formato del archivo es el siguiente:

```
# process71lq6i615xz1

num_seqs            30
cutoff_ali      0.90
cutoff_inf      40
cutoff_sup          80
no_templates        5
no_models       10
eValue          0.001
fasta_file      process71lq6i615xz1.tfa
mail            bioinfo@ifc.unam.mx
```

El nombre del proceso, en este ejemplo, es: *process71lq6i615xz1*, este archivo se crea en PROCESStobeEXCECUTED/.

Se genera además un archivo con la secuencia de la proteína.
La ruta de este archivo es: prediction/TARGETS/*process71lq6i615xz1.tfa*

El formato de la secuencia es:
>GI|12643972|SP|P01315|INS_PIG INSULIN PRECURSOR
MALWTRLLPLLALLALWAPAPAQAFVNQHLCGSHLVEALYLVCGERGFFYTPKARREAENPQA
GAVELGGGLGGLQ_ALAL_EGPPQKRGIVEQCCTSICSLYQLENYCN

Después de mandar la solicitud de predicción el usuario recibe un mensaje donde se le informa que su petición se está procesando y que los resultados se le enviarán a su correo electrónico (Figura 5.13).

El mensaje que recibe el usuario contiene la dirección donde encontrará los resultados.
A continuación se presenta un ejemplo:

"Result PROTEIN STRUCTURE PREDICTION"

You can find the result in:
http://132.248.16.230:8090/PSPrediction/process71lq6i615xz1.html

La página HTML muestra la estructura tridimensional y los aminoácidos conservados de la proteína. El *applet* con la estructura aparece en la figura 5.14.

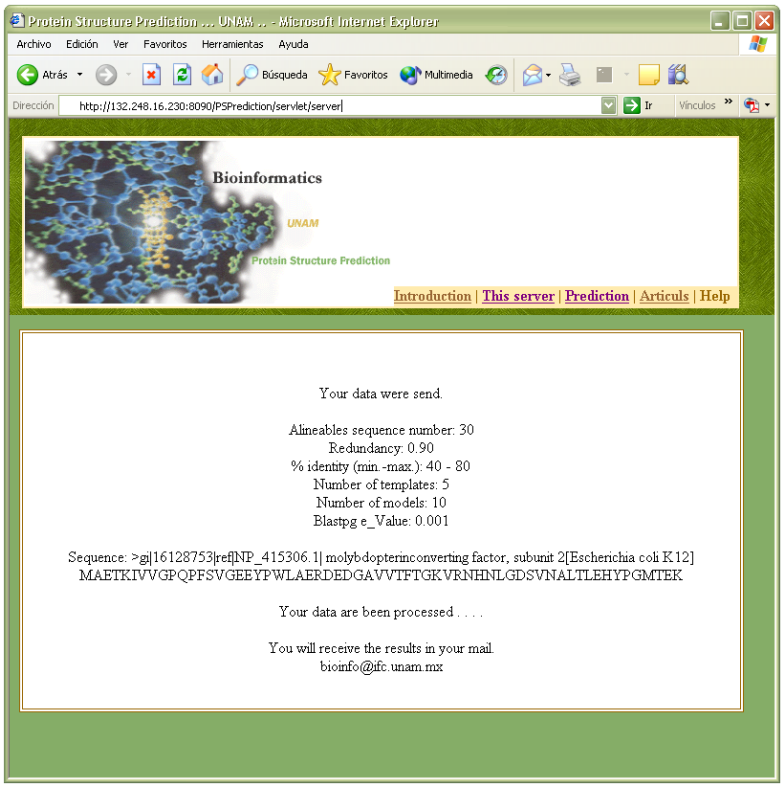

Figura 5.13 Mensaje de la petición de predicción.

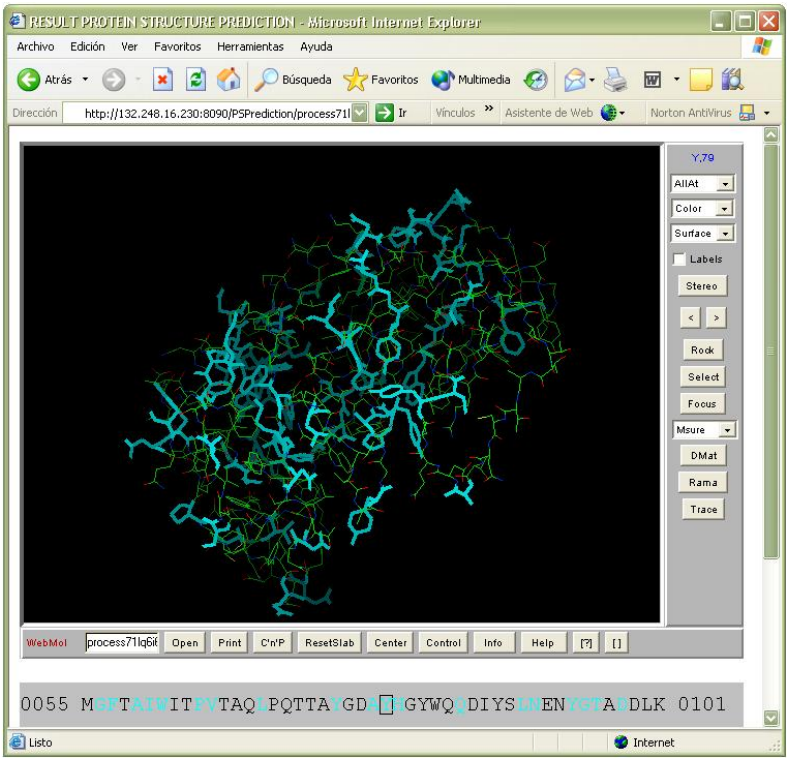

Figura 5.14 Resultado de la predicción.

8. Documentos (Figura 5.15)

Muestra la información disponible en el servidor y en otros sitios Web.

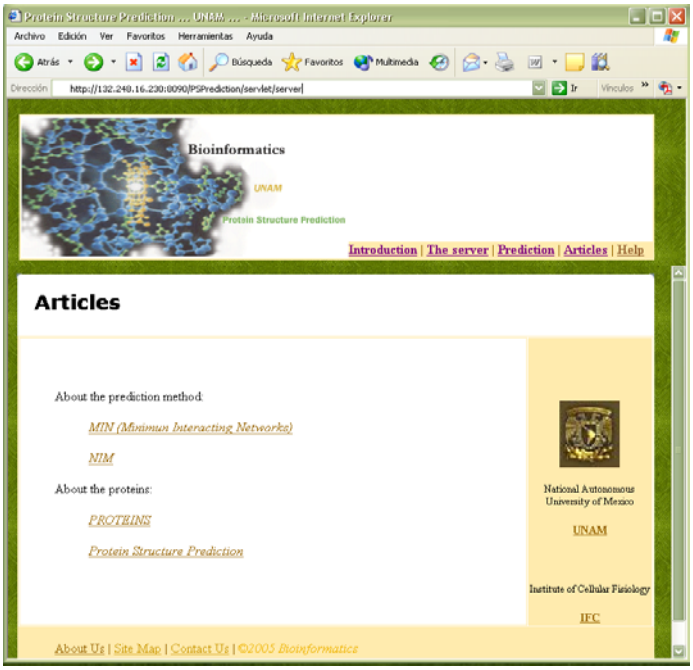

Figura 5.15 Página con documentos.

9. Ayuda (Figura 5.16)

El menú de ayuda proporciona información sobre la predicción de la estructura de proteínas y sobre los parámetros que utiliza.

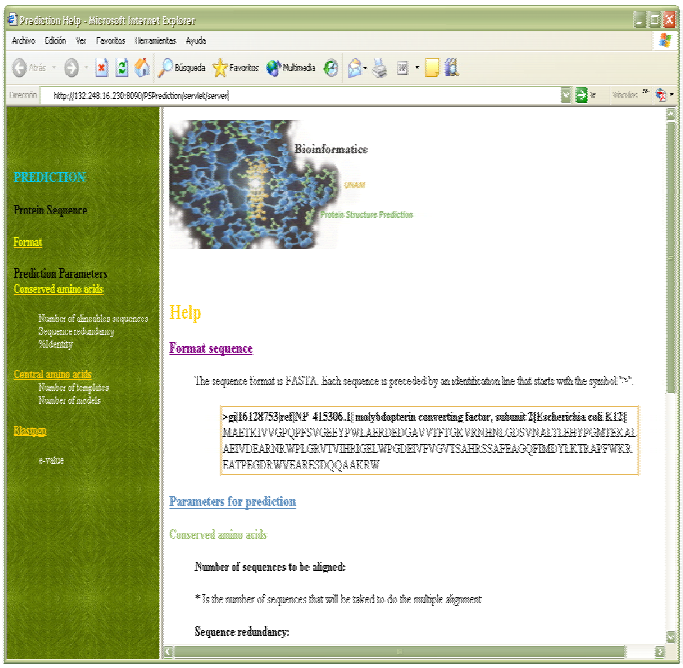

Figura 5.16 Página de ayuda.

Capítulo 6

Pruebas del sistema y resultados

Para probar el servidor se realizaron algunas pruebas sobre el servicio de admisión (iniciar sesión y registrar usuario) y el servicio de predicción.

Las pruebas sobre el servicio de admisión no tienen como objetivo mostrar la confiabilidad del método (o exactitud de los resultados), sino mostrar que el proceso de predicción se realiza correctamente, es decir, que los algoritmos MIN y NIM fueron implementados correctamente.

6.1 PRUEBAS DEL SERVICIO DE ADMISIÓN

El objetivo de la página *iniciar sesión* es controlar el acceso de los usuarios que utilizan el sistema. Las pruebas que se realizaron para verificar el funcionamiento de este servicio fueron proporcionar nombres y contraseñas de usuarios registrados y otros datos incorrectos (de usuarios no registrados). La validación del tipo de datos y la verificación del usuario fue correcta pues permitió el acceso solo a los usuarios que tienen cuenta registrada en la base de datos.
La función de *registrar usuario* sigue un proceso similar al de *iniciar sesión* pues valida los datos y el usuario, y después agrega la información a la base de datos. Las pruebas que se realizaron validaron correctamente los datos y la verificación del usuario (verificar que no exista otro usuario con la misma cuenta) y se agregó el usuario a la base de datos del sistema.

6.2 PRUEBAS DE LA PREDICCIÓN

Para comprobar el funcionamiento del servicio de predicción se hicieron algunas pruebas para comparar los resultados y para verificar que el proceso se haya realizado correctamente de principio a fin.

De los resultados de la predicción, la información que nos interesa es el tiempo que tarda en ejecutarse un proceso, el espacio (en disco duro) que ocupan los resultados y la generación correcta de los resultados.
Para saber como varían estos valores se realizaron pruebas cambiando los parámetros y las características de la secuencia (tamaño, formato, etc.).
Los resultados de la predicción dependen de las características de la máquina que ejecuta la aplicación.

Las características de la máquina donde se ejecuta la aplicación son:

Sistema Operativo - Red Hat Linux release 9
Disco duro - Intel(R) Xeon(TM), 60 GB, velocidad - 2.80GHz
Memoria - 1 GB

A continuación se presentan los resultados de dos pruebas de predicción, una con los parámetros máximos y otra con los parámetros mínimos.

PRUEBA 1:

(Prueba con los parámetros mínimos de la predicción)

Parámetros:

Alineables sequence number: 2
Redundancy: 0.40
% identity (min.-max.): 40 - 60
Number of templates: 5
Number of models: 5
Blastpgp e_Value: 0.001

Secuencia de la proteína:

Sequence: >GI|12643972|SP|P01315|INS_PIG INSULIN PRECURSOR
MALWTRLLPLLALLALWAPAPAQAFVNQHLCGSHLVEALYLVCGERGFFYTPKARREAENPQA
GAVELGGGLGGLQALALEGPPQKRGIVEQCCTSICSLYQLENYCN

Resultados:

Tiempo de ejecución: 6 min.

Archivos generados: 271

PRUEBA 2:

(Prueba con los parámetros máximos de la predicción)

Parámetros:

Alineables sequence number: 30
Redundancy: 0.90
% identity (min.-max.): 40 - 80
Number of templates: 15
Number of models: 20
Blastpgp e_Value: 0.001

Secuencia de la proteína:

Sequence: >GI|12643972|SP|P01315|INS_PIG INSULIN PRECURSOR
MALWTRLLPLLALLALWAPAPAQAFVNQHLCGSHLVEALYLVCGERGFFYTPKARREAENPQA
GAVELGGGLGGLQALALEGPPQKRGIVEQCCTSICSLYQLENYCN

Resultados:

Tiempo de ejecución: 56 min.

Archivos generados: 2317

Para comparar los resultados de la prueba 1 y 2 se realizó la siguiente tabla:

Parámetros	**Archivos generados**	**Tiempo (min.)**	**Tamaño (MB)**
Valores mínimos (prueba 1)	271	6	8.63
Valores máximos (prueba 2)	2317	56	58.81

La diferencia en el tiempo de ejecución del proceso de predicción y en el espacio en disco duro entre los archivos generados es grande. La cantidad de archivos generados depende del número de moldes y modelos que se generan para la proteína. La diferencia en tiempo es casi una hora, esto se debe a que la etapa más larga es cuando se generan los modelos de estructuras tridimensionales. El espacio en disco aumentó casi 50 MB, esto se debe al número de archivos, ya que cada archivo es individualmente pequeño.

Observamos que los aminoácidos conservados cambiaron en las dos pruebas, esto se debe a que en cada prueba se consideraron parámetros diferentes para seleccionar las secuencias que serían alineadas.

Los aminoácidos conservados fueron:

Prueba 1 6, 8, 10, 12, 17, 20, 21, 24, 25, 27, 33, 34, 37, 46, 59, 61, 63, 67, 68, 73, 82, 83, 88, 95.

Prueba 2 1, 8, 11, 13, 14, 16, 30, 34, 36, 38, 42, 44, 46, 47, 48, 56, 88, 89, 93, 102, 106.

El resultado de la predicción se guarda en un archivo y esta información es tomada por el *applet* que muestra la estructura tridimensional de la proteína. La figura 6.1 y 6.2 presentan las estructuras resultantes para la prueba 1 y 2, respectivamente.

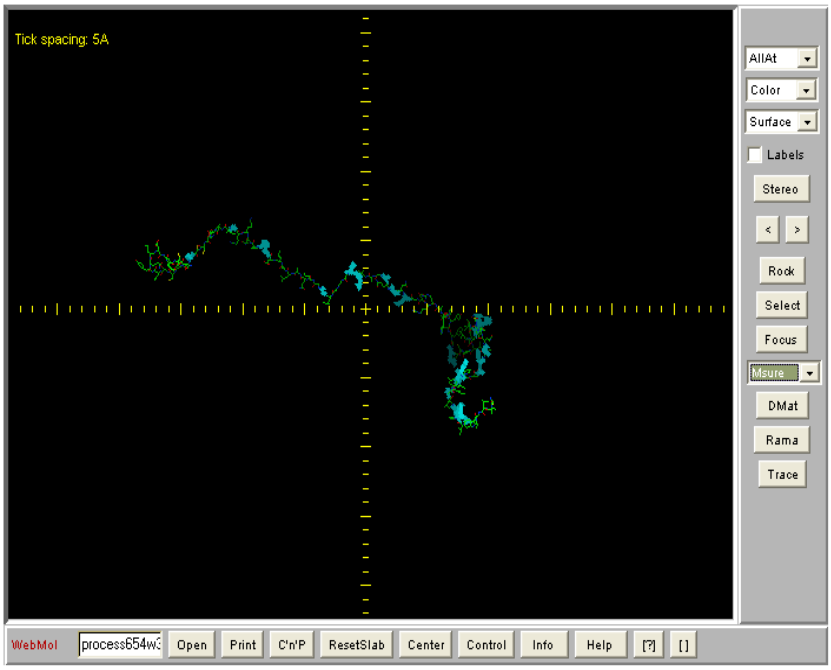

Figura 6.1 Resultado de la prueba 1.

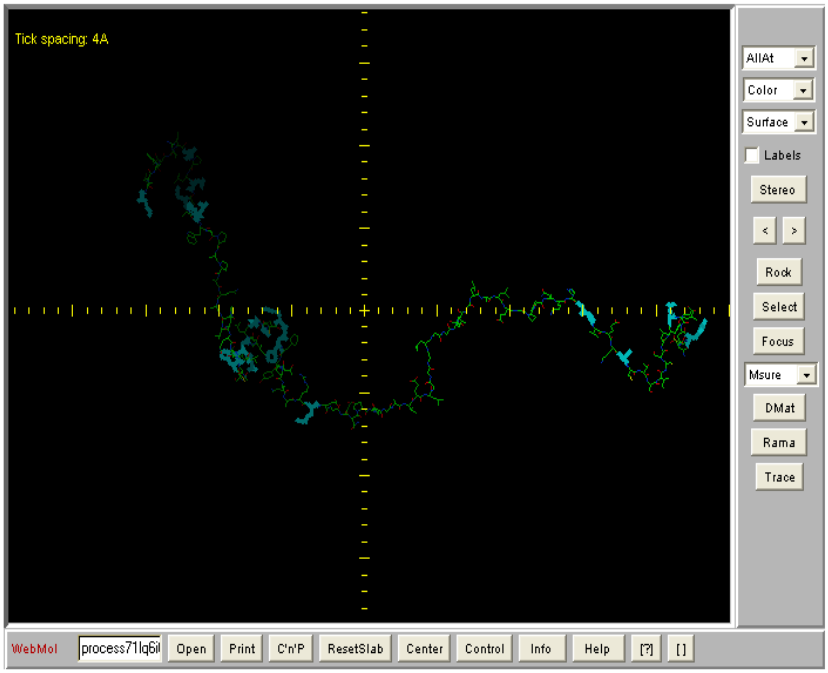

Figura 6.1 Resultado de la prueba 2.

PRUEBA 3, 4, 5:

Se realizaron otras pruebas para saber como se afectan los resultados de la predicción si se varía el tamaño de la proteína. El tamaño de las proteínas evaluadas fue 150, 200, 250 AA.

Parámetros (para las tres pruebas)

Alineables sequence number: 15
Redundancy: 0.90
% identity (min.-max.): 60 - 100
Number of templates: 5
Number of models: 10
Blastpgp e_Value: 0.001

Secuencia de la proteína 3:

>gi|16128753|ref|NP_415306.1| molybdopterin converting factor, subunit 2 [Escherichia coli K12]
MAETKIVVGPQPFSVGEEYPWLAERDEDGAVVTFTGKVRNHNLGDSVNALTLEHYPGMTEKAL
AEIVDEARNRWPLGRVTVIHRIGELWPGDEIVFVGVTSAHRSSAFEAGQFIMDYLKTRAPFWKRE
ATPEGDRWVEARESDQQAAKRW

Secuencia de la proteína 4:

>gi|16130139|ref|NP_416706.1| cytochrome c-type protein [Escherichia coli K12]
MGNSDRKPGLIKRLWKWWRTPSRLALGTLLLIGFVGGIVFWGGFNTGMEKANTEEFCISCHEMR
NTVYQEYMDSVHYNNRSGVRATCPDCHVPHEFVPKMIRKLKASKELYGKIFGVIDTPQKFEAHR
LTMAQNEWRRMKDNNSQECRNCHNFEYMDTTAQKSVAAKMHDQAVKDGQTCIDCHKGIAHK
LPDMREVEPGF

Secuencia de la proteína 5:

>gi|16129295|ref|NP_415850.1| transcriptional regulation of aerobic, anaerobicrespiration, osmotic balance [Escherichia coli K12]
MIPEKRIIRRIQSGGCAIHCQDCSISQLCIPFTLNEHELDQLDNIIERKKPIQKGQTLFKAGDELKSLY
AIRSGTIKSYTITEQGDEQITGFHLAGDLVGFDAIGSGHHPSFAQALETSMVCEIPFETLDDLSGKM
PNLRQQMMRLMSGEIKGDQDMILLLSKKNAEERLAAFIYNLSRRFAQRGFSPREFRLTMTRGDIG
NYLGLTVETISRLLGRFQKSGMLAVKGKYITIENNDALAQLAGHTRNVA

La tabla con los resultados se muestra a continuación:

Tamaño de la proteína (no. AA)	Tiempo (min)	Tamaño (MB)
150 (prueba 3)	11	7.17
200 (prueba 4)	39	20.99
250 (prueba 5)	45	25.76

En la prueba 3 el tiempo disminuyó demasiado porque sólo se encontraron 2 proteínas plantillas (homólogas), y no cinco como se solicitó. En estos casos se realiza la predicción con las plantillas encontradas y se envía un mensaje al usuario indicándole esto. De manera similar, cuando las plantillas son más de las que indicó el usuario, se manda un mensaje con esta información.

Si quisiéramos realizar la predicción para el genoma de una bacteria de 5000 proteínas con un tamaño promedio de 200 aminoácidos, el tiempo que tardaríamos sería, tomando como referencia los valores de la tabla, 195000 min. es decir 3250 hrs. que son 135.5 días correspondientes a casi cinco meses.

OTRAS PRUEBAS

Al realizar la predicción se encontraron algunos casos en los que la proteína objetivo no tenía homólogas; en estos casos el método no pudo utilizarse pues se requiere de una proteína plantilla para generar un modelo. Para notificar al usuario se le envía un correo electrónico comunicándole que no se pudo llevar a cabo la predicción.

En de las pruebas no se pudo leer la información de unos archivos de la base de datos, en este caso se terminó el proceso y se envió un mensaje al usuario comunicándole el error en la predicción.

El servidor puede detenerse también cuando se presentan errores en los programas o por alguna otra falla durante el proceso, el usuario recibe un mensaje de error en su predicción.

6.3 EVALUACIÓN DEL MÉTODO

Las pruebas que se realizaron anteriormente comprueban el funcionamiento de los algoritmos implementados MIN y NIM, sin embargo esto no muestra que los resultados de la predicción sean confiables.

Existen muchas formas de evaluar los resultados del método de predicción, una de ellas es CASP. CASP realiza la evaluación de los métodos de predicción comparando la estructura predicha con la estructura obtenida por técnicas experimentales.

El método de predicción que utiliza este servidor fue evaluado (con solo dos secuencias) en CASP5 y para el CASP6 se participó prediciendo la estructura de 14 secuencias de proteínas.

Los resultados de la evaluación del método fueron los siguientes:

Secuencia de la proteína	Diferencia geométrica (Å)
T0215	3.02
T0246	1.97
T0262	3.07
T0264	2.66
T0265	3.02
T0266	2.36
T0267	2.14
T0268	2.14
T0269	1.96
T0274	3.81
T0277	1.82
T0279	2.93
T0280	2.29
T0282	2.04

Secuencia de la proteína: Es el nombre de la secuencia objetivo por predecir
Diferencia geométrica: Indica la diferencia geométrica entre todos los átomos de las dos estructuras (la predicha y la experimental), en Angstroms. Si la posición de un átomo es la misma en las dos estructuras, la distancia entre ellos es cero, si no es así, se mide esa diferencia.

La interpretación de estos resultados depende del uso que se le dará al modelo tridimensional de la proteína, para un experimento en particular. En general, se dice que si la distancia entre el modelo y la estructura real de la proteína es menor a 4 Å las estructuras son muy parecidas.

Esta evaluación no es estadísticamente significativa, pues se evaluó un conjunto de 14 secuencias: para comprobar la validez del método se requieren miles de pruebas.

Conclusiones

Uno de los problemas fundamentales de la biología es la predicción de la estructura de las proteínas, en todo el mundo hay grupos de investigación que tratan de responder a este problema. En México también existen instituciones e investigadores del área biológica, química y bioinformática, etc. que se han involucrado en la búsqueda de la creación y la optimización de métodos que mejoren la confiabilidad de la predicción: en la UNAM se encuentra el Dr. Lorenzo Segovia y el Dr. Eduardo Horjales del Instituto de Biotecnología (IBT), el Dr. Gabriel del Río del Instituto de Fisiología Celular (IFC) y el Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas (IIMAS).

Algunos de los proyectos que se han desarrollado son: un software para la predicción de la estructura de péptidos (proteínas menores a 50 AA) del IIMAS, un método para la predicción de la estructura de proteínas basado en el reconocimiento del plegamiento (*fold recognition*) del IBT y un nuevo método de predicción, basado en el análisis de redes, del Dr. Gabriel del Río (del IFC). El primero de estos dos trabajos genera modelos de la estructura de péptidos, basándose en métodos de predicción existentes. Ninguno de ellos cuenta con una interfaz gráfica que muestre la estructura tridimensional de la proteína, ni cuenta con un servidor o sitio Web que permita el acceso a la aplicación.

Es precisamente en esa carencia (de accesibilidad y de interfaz) donde quiere incidir el proyecto que se desarrolló en este trabajo de tesis 'Diseño de un servidor para la predicción de la estructura de proteínas', que tiene el objetivo de implementar el nuevo método de predicción del Dr. Gabriel del Río Guerra.

Uno de los propósitos del proyecto fue, a través del desarrollo del servidor que proporciona el medio (un entorno integral) para presentar y ejecutar la aplicación, participar en el CASP para evaluar el nuevo método de predicción (basado en dos algoritmos MIN y NIM) y en diciembre del 2004 se logró participar en la predicción de 14 proteínas, pero el servidor no se presentó ya que aún no estaba concluido. Los resultados de la evaluación de las proteínas enviadas a CASP mostraron que las estructuras predichas fueron muy cercanas a las obtenidas experimentalmente pero esto no basta para decir que el método es confiable.

Lo que hasta la fecha se puede decir es que el método de predicción utilizado ofrece una alternativa a los métodos actuales, pero este servidor no se dará a conocer hasta que se hayan realizado las pruebas necesarias para demostrar la confiabilidad del método. Para

evaluarlo se tienen que realizar miles de predicciones que prueben la validez de los resultados. Una prueba podría ser como sigue:
Se toman 3000 proteínas con estructuras conocidas y se realiza la predicción para cada una. Se generan los alineamientos de secuencias, la secuencia 1 con la 1, la 1 con la 2, la 1 con la 3, … la 1 con la 3000, después la 2 con la 1, la 2 con la 2, … etc. Se generan $(3000)^2$ o sea 9 millones de alineamientos. Si se quieren realizar 10 modelos por cada proteína, se crean 90 millones de modelos.
El tiempo y los recursos computacionales necesarios para realiza una prueba como esta son grandísimos, esto no se lograría con un solo procesador tipo Intel.

En lo que concierne a la presentación del servidor, la interfaz es sencilla y amigable y la información que el usuario debe proporcionar es mínima (sólo los parámetros para la predicción). Un usuario envía una petición de predicción, el servidor le manda un mensaje notificándole que su petición se está procesando y que recibirá un mensaje con los resultados y en ese momento el trabajo del usuario termina; no tiene que preocuparse por esperar a que termine el proceso de predicción (pueden ser unos minutos, unas horas, o unos días si hay muchos procesos en espera de ser ejecutados); se puede decir que el proceso de predicción es transparente al usuario.

A la fecha el acceso al servidor es libre, el registro del usuario sólo permite tener control sobre el numero de personas que accedan al servidor pero posteriormente será necesario agregar un servicio que condicione el acceso al servidor (por ejemplo el número de predicciones permitidas por día y por usuario) pues demasiados procesos podrían saturar la máquina y detener el procesos de predicción.

El objetivo del servidor que se realizó no es aumentar la confiabilidad o la exactitud del método, sino implementar los algoritmos MIN y NIM para que el proceso de predicción pueda realizarse de forma rápida y sencilla.

Para implementar los algoritmos se conjuntaron todos los programas que intervienen en la predicción, de esta forma se logró que el proceso se ejecutara de principio a fin, sin que una persona interviniera para iniciar o detener los programas o para modificar los resultados de cada etapa del método.

La elección de aprovechar el modelo de los *servlets* (*threading model*) fue para agilizar el proceso de la predicción y de hecho de esta manera el servidor puede enviar simultáneamente varios procesos de predicción, sin consumir tantos recursos del ordenador. Gracias a este modelo los distintos procesos pueden desarrollarse de forma paralela (independientes y simultáneos) aunque las etapas que constituyen el método de predicción siguen realizándose de forma secuencial. Considerando que dos etapas del método (*GetCentral* y *GetConserved*) son independientes una de la otra, un objetivo futuro podría ser ejecutar de forma paralela esas dos etapas, mientras que las demás necesitan esperar a que una etapa termine para recibir los resultados y ejecutar el siguiente programa.

Entonces, se puede afirmar que esta parte del proyecto ha logrado su objetivo, pero después de haber realizado algunas pruebas del servidor nos damos cuenta de que el tiempo y los recursos de cómputo (memoria y disco) que se requieren para realizar predicciones son grandes. Si para analizar el genoma de una bacteria (de 5000 proteínas) tardamos casi cinco meses, para obtener la estructura de las proteínas del organismo humano (se infiere que son aproximadamente 30000) tardaríamos muchos años.

Los inconvenientes de tiempo y de recursos del procesador hacen necesario que el proyecto siga desarrollándose, eso significa que una vez realizado el servidor se debe buscar la forma para mejorar y agilizar el proceso de la predicción.

El futuro desarrollo del trabajo podría ser el de crear una malla (*grid*) que permita el acceso a computadoras y bases de datos remotas y también a los servicios que ofrecen (procesamiento, almacenamiento, datos y software) sin importar su ubicación. Crear una red integradora ofrecería grandes capacidades de procesamiento de datos y funciones no disponibles en máquinas individuales o de grupo (Foster, I., 2004).

El proyecto de crear esta malla ya se está formando: hasta el momento se cuenta con un cluster de 64 nodos del IBT, algunas máquinas del IFC, posiblemente un cluster del Centro de Ciencias Genómicas y la colaboración de DGSCA (Dirección General de Servicios de Cómputo Académico).

Apéndice A

Proteínas

Las proteínas tienen un significado especial en la biología pues están presentes en todos los organismos vivos y constituyen uno de los componentes fundamentales para la vida (Bruce, A. et Al., 2002).

1. PROTEÍNAS

Una proteína es una macromolécula formada por aminoácidos unidos mediante un enlace peptídico (Figura A.1). Los compuestos de una proteína son: carbono, nitrógeno, hidrógeno y oxigeno.

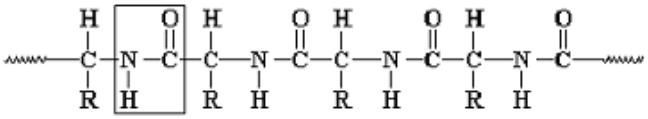

Figura A.1 Cadena de aminoácidos.

Las diferentes propiedades de los aminoácidos permiten formar un enorme número de proteínas con una amplia gama de funciones. No se conoce la función, ubicación e interacciones de todas las proteínas ni tampoco el número exacto de proteínas en nuestro organismo, aunque gracias a la secuenciación del genoma humano se infiere que son aproximadamente 30000.

Los aminoácidos, unidades de las proteínas

Los aminoácidos son moléculas que poseen un *grupo carboxilo* (-COOH) y un *grupo amino* ($-NH_2$) ambos unidos al mismo átomo de carbono (Figura A.2).

La fórmula química general de un aminoácido es:

NH_2-CHR-COOH

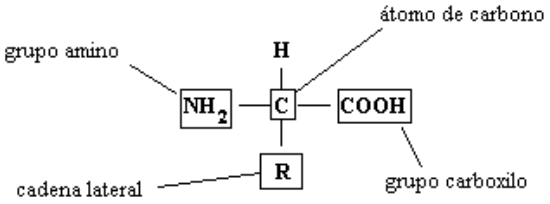

Figura A.2 Representación de un aminoácido.

Según el *grupo variable* denominado R (o cadena lateral), se distinguen 20 tipos de aminoácidos. Éstos difieren en sus propiedades físico-químicas así como en su tamaño y en la posibilidad de interacción entre ellos.

Nuestras células no poseen los mecanismos bioquímicos para fabricar todos estos aminoácidos, sólo 11; el resto deben ser ingeridos en la dieta y se denominan aminoácidos esenciales.

Los aminoácidos están unidos por un enlace amida (formado entre el grupo carboxilo de un aminoácido y el grupo amino de otro) denominado enlace peptídico (Figura A.3); los productos que se forman a partir de esta unión se llaman péptidos.

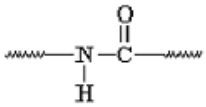

Figura A.3 Grupo amida.

Los aminoácidos están divididos en cuatro clases diferentes definidas por la naturaleza química de la cadena lateral: ácidas, básicas, polares, no polares.

Los nombres de los 20 aminoácidos se presentan a continuación:

A = Ala (Alanina)	G = Gly (Glicina)	M = Met (Metionina)	S = Ser (Serina)
C = Cys (Cisteína)	H = His (Histidina)	N = Asn (Asparagina)	T = Thr (Treonina)
D = Asp (Ac. Aspártico)	I = Ileu (Isoleucina)	P = Pro (Prolina)	V = Val (Valina)
E = Glu (Ac. Glutámico)	K = Lys (Lisina)	Q = Gln (Glutamina)	W = Trp (Triptófano)
F = Phe (Fenilalanina)	L = Leu (Leucina)	R = Arg (Arginina)	Y = Try (Tirosina)

Síntesis de las proteínas

El ADN (ácido desoxirribonucleico) es análogo al *software* de una computadora, es decir, las instrucciones que la célula recibe de su progenitor, mientras que las proteínas constituirían el *hardware,* o sea, la maquinaria que ejecuta los programas almacenados en la memoria.

La información de cuales aminoácidos deben asociarse para formar una proteína está en el ARN (ácido ribonucleico) y el mARN (ARN mensajero) lleva el mensaje desde el ADN hasta los ribosomas, donde se realizará el proceso de síntesis. Al proceso de trasladar la información del mARN a la proteína se le conoce como traducción (Figura A.4).

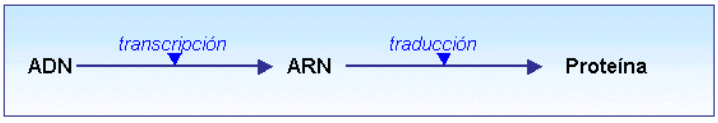

Figura A.4 Síntesis de las proteínas.

Una vez finalizada la síntesis se añaden, algunas veces, compuestos formados por fósforo, azufre y diversos minerales y grupos químicos.

2. ESTRUCTURA DE LAS PROTEÍNAS

Los aminoácidos a medida que van siendo enlazados durante la síntesis de la proteína y gracias a la capacidad de giro de sus enlaces, van adquiriendo una disposición estable, hasta llegar a la estructura tridimensional.

La organización de una proteína viene definida por cuatro niveles estructurales denominados: estructura primaria, estructura secundaria, estructura terciaria y estructura cuaternaria (Figura A.7).

A. Estructura primaria

La estructura primaria se refiere a la secuencia de aminoácidos que componen a la proteína. Esta secuencia de aminoácidos influye en la conformación final y en su función.
La secuencia de aminoácidos de proteínas con funciones similares en diferentes organismos vivos se encuentra altamente conservada. Más aún, la estructura tiende a estar más conservada que la secuencia exacta de los aminoácidos.

B. Estructura secundaria

La estructura secundaria está definida por las regiones de la secuencia de proteínas que presentan formas regulares de conformación de la cadena polipeptídica, como las ⟨ (alfa)-hélice o la conformación ® (beta).

Las estructuras más comunes son:

1. La ⟨ (alfa)-hélice

Esta estructura se forma al enrollarse helicoidalmente sobre sí misma la estructura primaria. Tiene una periodicidad de aproximadamente 3.6 aminoácidos por cada vuelta (360º) (Figura A.5).

Figura A.5

2. La conformación ® (beta)

En esta disposición los aminoácidos no forman una hélice sino una cadena en forma de zigzag, denominada disposición en lámina plegada. Tiene una

Figura A.6

periodicidad de aproximadamente 2 aminoácidos por cada nivel (Figura A.6).

C. Estructura terciaria

La estructura terciaria es la disposición tridimensional de todos los átomos que componen la proteína. Es la responsable de las propiedades de la proteína, ya que la disposición espacial de los distintos grupos funcionales determina su interacción con otras moléculas (por ejemplo: Proteínas, ADN, etc.).

Existen regiones diferenciadas dentro de la estructura terciaria que actúan como unidades autónomas de plegamiento y/o desnaturalización de las proteínas y reciben el nombre de dominios. Los dominios se pliegan por separado a medida que se sintetiza la cadena polipeptídica.

D. Estructura cuaternaria

Esta estructura informa de la unión, mediante enlaces débiles (no covalentes) de varias cadenas polipeptídicas con estructura terciaria, para formar un complejo proteico. Cada una de estas cadenas polipeptídicas recibe el nombre de protómero.

La estructura cuaternaria modula la actividad biológica de la proteína y la separación de las diferentes cadenas. Las fuerzas que mantienen unidas a estas cadenas son las mismas que estabilizan la estructura terciaria, las más abundantes son las interacciones débiles (hidrofóbicas, polares, electrostáticas y puentes de hidrógeno que son enlaces no covalentes), aunque en algunos casos la estructura cuaternaria se mantiene mediante puentes disulfuro (enlaces covalentes).

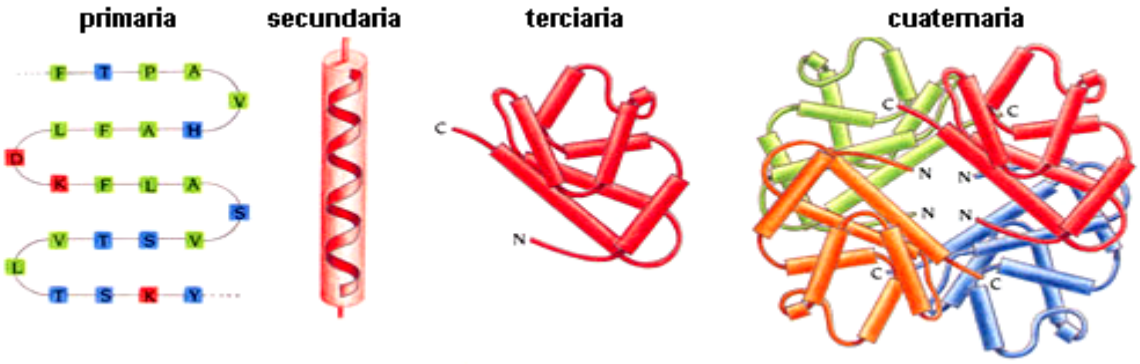

Figura A.7 Estructura de las proteínas.

Para obtener más información sobre este tema consultar: Bruce, A. et Al., 2002 y Branden, C., Tooze, J., 1991.

Apéndice B

Alineamiento de secuencias de proteínas

El objetivo de comparar dos secuencias de proteínas es encontrar la posición relativa de ambas en las que se produzca un mayor número de coincidencias entre sus aminoácidos (AA). Esta información es importante para inferir las relaciones estructurales, funcionales o evolutivas entre las secuencias.

Secuencia de una proteína: es una cadena lineal de aminoácidos, pertenecientes a un alfabeto. El número de símbolos de la cadena representa su longitud.
Alfabeto: Conjunto de aminoácidos diferentes usados para representar las secuencias.

AA = { A,C,D,E,F,G,H,I,K,L,M,N,P,Q,R,S,T,V,W,Y }

Se presentan diferentes casos en la tarea de encontrar las relaciones de semejanza existentes entre el grupo de secuencias {X1, X2, …., X_n} y las secuencias del grupo {Y1,Y2,….,.Y_m} .

n=1, m=1: Comparación de parejas de secuencias (pairwise comparison)
En este caso se compara una secuencia contra otra. Es una etapa parcial de los alineamientos múltiples.

n=1, m>>1: Alineamiento múltiple. Búsqueda en base de datos
Este caso es frecuente en la biología molecular, cuando se compara una secuencia de prueba con un conjunto de secuencias almacenadas en una base de datos. Se trata de encontrar semejanzas que permitan inferir algunas de las propiedades de la secuencia investigada.
La demanda computacional en este tipo de aplicaciones es muy elevada. Uno de los problemas que se presentan en este tipo de aplicaciones es que debido al alto volumen de datos y su crecimiento exponencial, la probabilidad de que se produzcan coincidencias debidas al azar aumenta. Este problema se conoce como el ruido de fondo de la búsqueda en las bases de datos y su efecto es la aparición de lo que se denomina falsos positivos (referido a la semejanza entre secuencias).

n>1, m>1: Varias contra varias
Si n>1 y m>>1 la búsqueda es similar al caso anterior, sino se deben hacer comparaciones de todas contra todas (X_j contra Y_j, j = 1….n).
El problema de la búsqueda de secuencias en bases de datos es, computacionalmente, interesante; cuando se trabaja con 40000 proteínas, hablamos de ½ millón de comparaciones en un alineamiento múltiple de 1000 secuencias.

Debido al gran volumen de datos de que se dispone, es frecuente que los algoritmos orientados al análisis de secuencias biológicas sean grandes consumidores de recursos

computacionales (CPU, E/S, memoria, etc.). Por esta razón, las tareas de optimización (técnicas de programación paralela y uso de multiprocesadores) son indispensables.

1. Comparación de parejas de secuencias (Figura B.1)

Existen distintas formas de comparar secuencias y su objetivo es encontrar el *alineamiento* que maximice el parecido.

1a. Comparación por identidad

- Representación con secuencias (Figura B.1)

El algoritmo desplaza una de las secuencias debajo de la otra y contabiliza los residuos que coinciden. El mejor alineamiento corresponde a la posición en que éste valor es máximo (indicado al lado derecho).

```
                X:    TCAGACGATTG (n=11)
                Y:    ATCGGAGCTG  (m=10)

        TCAGACGATTG   (0)                          .
ATCGGAGCTG                                         .
                                                   .
        TCAGACGATTG   (1)              TCAGACGATTG          (3)
 ATCGGAGCTG                                  ATCGGAGCTG

        TCAGACGATTG   (0)              TCAGACGATTG          (1)
  ATCGGAGCTG                                   ATCGGAGCTG

        TCAGACGATTG   (2)              TCAGACGATTG          (0)
   ATCGGAGCTG                                    ATCGGAGCTG

        TCAGACGATTG                    TCAGACGATTG          (0)
    ATCGGAGCTG        (0)                        ATCGGAGCTG
```

Figura B.1 Comparación de dos secuencias por identidad.

- Representación mediante una matriz (Figura B.2)

En este algoritmo desplazar las secuencias equivale a colocar una de ellas en vertical y la otra en horizontal y recorrer cada una de las diagonales formadas en la matriz, acumulando el número de coincidencias que se produzcan. La diagonal en la que se produzca mayor número de coincidencias representará el desplazamiento relativo que mejor alinea las secuencias.

	0	1	2	3	4	5	6	7	8	9	10	11
0		T	C	A	G	A	C	G	A	T	T	G
-1	A			1		1			1			
-2	T	1								2	1	
-3	C		2				1					
-4	G				1			2				3
-5	G				3			1				1
-6	A			1		4			2			
-7	G				2			2				2
-8	C		1				1					
-9	T	1								3	1	
-10	G				2			3				2

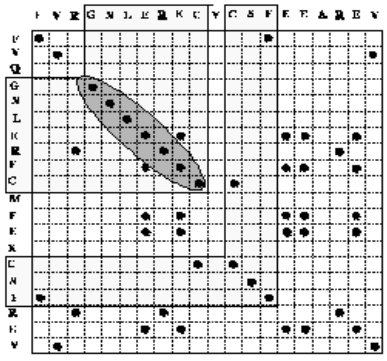

Figura B.2 Representación mediante una matriz.

En este ejemplo el mejor alineamiento por coincidencias es:

```
      12345678901
X:    TCAGACGATTG     (n=11)
      || ||
Y:  ATCGGAGCTG       (m=10)
    1234567890
```

Figura B.3 Resultados del alineamiento.

Una de las limitaciones de esta aproximación es que no tiene en cuenta la frecuencia de cada símbolo (si dos símbolos muy poco frecuentes coinciden, ha de tener más peso que si coinciden dos símbolos más abundantes).

1b. Comparación por semejanza

Los métodos actuales para comparar secuencias biológicas se basan en determinar una función de distancia entre las dos secuencias, que pretende informar de la cercanía entre ellas (por ejemplo: de acuerdo a los criterios será una distancia evolutiva o estructural, etc.).

La distancia puede representar el mínimo coste para transformar la secuencia X en la secuencia Y aplicando una serie de transformaciones (sustituir, insertar, eliminar), cada una tiene un coste.

2. Alineamiento múltiple. Búsqueda en base de datos

Las técnicas del alineamiento múltiple de secuencias son aplicadas comúnmente a las secuencias de proteínas. Se sabe que las proteínas con funciones parecidas tienen secuencia y estructura similar y que durante la evolución la secuencia tiende a cambiar más rápidamente que la estructura. En los alineamientos múltiples generados para una secuencia de datos las regiones que son similares en secuencia lo son también en estructura.

El alineamiento múltiple (de más de dos secuencias) puede ser visto como una generalización del alineamiento de parejas, donde la complejidad de esta aproximación crece exponencialmente con el número de secuencias que intervienen. Por analogía con la comparación de parejas es posible aplicar la misma técnica de programación dinámica (el caso exhaustivo multidimensional) pero el crecimiento exponencial de la complejidad (del orden de nxm) del método lo limita a no más de tres dimensiones. Si se quisieran alinear tres secuencias, la complejidad sería de orden nxmxl. O dicho de otra forma: si alinear dos secuencias de 300 residuos tarda un segundo, alinear tres secuencias tardaría 300 segundos, y alinear 10 secuencias tardaría 300^8 segundos (más que la edad del universo).

Apéndice C

Aplicaciones Web

1. ARQUITECTURA WEB

Para abrir una página Web en el navegador debe teclearse el URL correspondiente. Una vez que se ha enviado esta petición (mediante el protocolo HTTP) y que el servidor Web la ha recibido, éste localiza la página en su sistema de ficheros y la envía de regreso al navegador que la solicitó. El navegador recibe una respuesta y despliega la página en pantalla (Figura C.1).

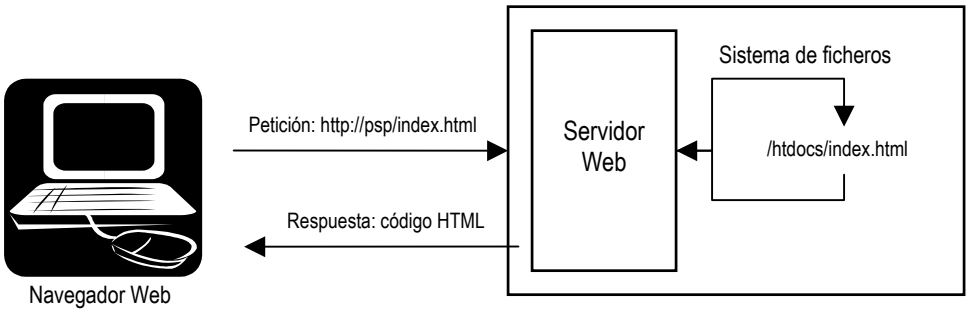

Figura C.1 Arquitectura Web.

2. APLICACIONES MULTINIVEL

Las aplicaciones Web se presentan generalmente dentro de las aplicaciones multinivel.

Los sistemas típicos *cliente/servidor* pertenecen a la categoría de las aplicaciones de dos niveles. La aplicación reside en el cliente y la base de datos se encuentra en el servidor. En este tipo de aplicaciones la carga de procesamiento recae sobre el cliente (generalmente es una máquina menos potente que el servidor), mientras que el servidor actúa como controlador del tráfico entre la aplicación y los datos. Uno de los principales problemas de estos sistemas reside en la actualización y el mantenimiento de las aplicaciones, pues estas modificaciones deben trasladarse a todos los clientes.

Una aplicación de tres niveles se divide en *interfaz de presentación, lógica de la aplicación* y *datos.* Esta división entre la capa de presentación y la de la lógica permite una gran flexibilidad a la hora de construir aplicaciones, ya que se pueden tener múltiples interfaces sin cambiar la lógica de la aplicación.

Una aplicación de n niveles (multiniveles) extiende la capa intermedia (la lógica de la aplicación), la divide por funciones (múltiples aplicaciones) y no físicamente.

3. ARQUITECTURA DE UNA APLICACIÓN WEB

La arquitectura de las aplicaciones Web suelen presentar un esquema de tres niveles.
El primer nivel consiste en la capa de presentación que incluye no sólo el navegador, sino también el servidor Web que es el responsable de dar a los datos un formato adecuado para su presentación.
El segundo nivel está referido habitualmente a algún tipo de programa o *script*.
Finalmente, el tercer nivel proporciona los datos necesarios al segundo nivel.

Una aplicación común recoge datos del usuario (primer nivel), los envía al servidor que ejecutará un programa (segundo y tercer nivel) y da formato a los resultados para presentarlos al usuario en el navegador (primer nivel) (Figura C.2).

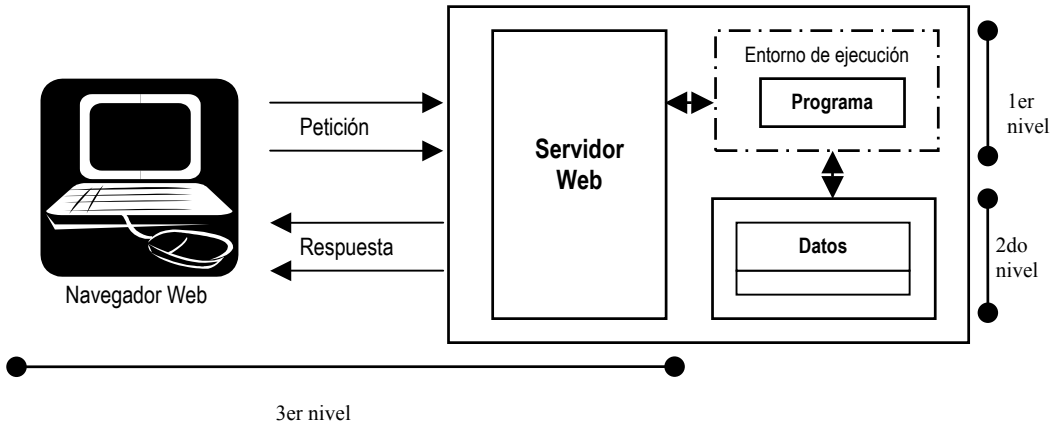

Figura C.2 Arquitectura de una aplicación Web de tres niveles.

A continuación se describe el proceso de ejecución de una aplicación Web.

- **Colectando datos**

El primer paso de una aplicación Web, usualmente, es la *toma de datos***.** Generalmente se utiliza un formulario HTML, el usuario teclea información en algunos campos del formulario, presiona un botón (*submit*) para enviar los datos y después espera los resultados.

- **Enviando la petición al servidor Web**

Para que el servidor Web realice una acción, como ejecutar un programa, el navegador Web necesita empaquetar los datos del usuario y enviar la petición al servidor Web. Una petición HTTP está formada por el URL de la página o el script al que el usuario desea accesar por el formulario de datos (si existe) y alguna información adicional (información de cabecera: sobre el navegador, longitud y tipo de la petición, etc.). El protocolo HTTP especifica como están estructuradas las solicitudes y las respuestas de una aplicación.
HTTP define los tipos de solicitudes que los clientes pueden enviar a los servidores y los tipos de respuestas que los servidores pueden enviar a los clientes. HTTP/1.0 especifica tres tipos de métodos de solicitud: HEAD, GET y POST.

El método HEAD regresa información acerca del documento y no el documento en sí.
Los métodos GET y POST se utilizan para enviar una petición de ejecución de un programa Web. Ambos realizan la misma tarea, pero la forma de hacerlo es diferente.

- **Ejecutando el programa o el script**

Una función del servidor Web es pasar la petición al programa o al script específico, para ser procesada. Primero determina el tipo de ambiente de operación que se requiere para cargar la aplicación y para esto revisa la extensión del archivo de la petición (o el directorio donde el archivo está localizado).

- **Regresando los resultados al navegador**

El paso final de una aplicación Web es regresar los resultados de la operación. Cuando el navegador recibe la respuesta, primero mira la cabecera para determinar como presentar los datos. El tipo de contenido más común es *text/html*, pero el servidor puede regresar XML, o texto sin formato, imagen o audio.

Para obtener más información sobre este tema consultar: Kurniawan, B., 2002 y Danny, A. et Al., 1999.

Apéndice D

Programas

En este apéndice se muestran algunos de los programas generados para la aplicación. Los paquetes se dividen en dos servicios: *Service prediction* y *Service admisión*.

1) Service prediction

Las etapas de la predicción se implementaron en paquetes con el mismo nombre, a continuación se describen sólo los programas principales de cada etapa:

DISPATCHER

```
import java.io.*;

/**
 *Descripcion de la clase
 * dispatcher
 **/

public class daemon{

 public static void main (String args[]) throws Exception {
  String ruta = "/apps/tomcat/jakarta-tomcat-
5.0.28/webapps/PSPrediction/WEBINF/classes/PROCESStobeEXECUTED";
  String name = "user";
  boolean x=true;
  String[] file;
  int i;

  File dir = new File(ruta);
  String file1;

  do{
    file = dir.list();
    for (i = 0; i < file.length; i++ ){
       file1 = file[i];
//-- Comienza el proceso de prediccion //
       PerformanceMonitor newproc = new PerformanceMonitor(file1);
       newproc.start();

    }
    Thread.sleep(3000);
  }while(x);

 } // Fin main
} // Fin daemon
```

PERFORMANCEMONITOR

```
import makeFiler.*;
import prediction.GetSequence.*;
import prediction.GetCentral.*;
import prediction.GetConserved.*;
import prediction.GetBest.*;
import prediction.Result.*;
import java.io.*;
import java.util.*;

class PerformanceMonitor extends Thread implements Constants{

 String CUTOFF_ALI="", NUM_SEQS="", CUTOFF_INF="", CUTOFF_SUP="";
 String FASTA_FILE="", NO_MOLDES="", NO_MODELOS="", E_VALUE="", MAIL="";
 String ruta = home_dir+"/PROCESStobeEXECUTED/";  //"/apps/tomcat/jakarta-tomcat-
5.0.28/webapps/PSPrediction/WEB-INF/classes/PROCESStobeEXECUTED/";
 String KEY="res";

 public PerformanceMonitor(String f) {
  try {
     Properties props = new Properties();
     FileInputStream fis = new FileInputStream(ruta+f);
     File file = new File(ruta+f);
     props.load(fis);
     FASTA_FILE = props.getProperty("fasta_file");
     CUTOFF_ALI = props.getProperty("cutoff_ali");
     NUM_SEQS = props.getProperty("num_seqs");
     CUTOFF_INF = props.getProperty("cutoff_inf");
     CUTOFF_SUP = props.getProperty("cutoff_sup");
     NO_MOLDES = props.getProperty("no_moldes");
     NO_MODELOS = props.getProperty("no_modelos");
     MAIL =  props.getProperty("mail");
    fis.close();
    file.delete();
  } catch(java.lang.Exception e) { System.err.println(e.getMessage());  }
 }

 public void run() {
 int res_PDB=0;
 String msg="";
 try {
//-- Comienza obteniendo las secuencias similares a la proteina
   System.out.println(".............GET SEQUENCE..........");
   javGetSequence getseq = new javGetSequence(FASTA_FILE);
   getseq.runBlastPGP();

//-- Obtiene los residuos conservados
```

```
   System.out.println("..............GET   CENTRALS..........");
   javGetCentral getcent = new javGetCentral(FASTA_FILE, NO_MOLDES, NO_MODELOS);
   res_PDB = getcent.exec();

 if (res_PDB!=3){
 if (res_PDB==1) { msg="There were less pdb files than the number specified to search for."; }
 if (res_PDB==2) { msg="There were more pdb files than the number specified to search for."; }

//-- Obtiene los residuos conservados
   System.out.println("..............GET  CONSERVED..........");
   javGetConserved gc = new
javGetConserved(NUM_SEQS,CUTOFF_ALI,CUTOFF_INF,CUTOFF_SUP,FASTA_FILE);
   String RC = gc.exec();

   System.out.println("..............EVALUATED BEST .........");
   javGetBest gb = new javGetBest(FASTA_FILE);
   String PATH_FILE = gb.exec();
   System.out.println(PATH_FILE);

   System.out.println("..............MAKE PAGE .........");
   RC = RC + '\n'+ msg;
   javMakeResult mr = new javMakeResult(FASTA_FILE,PATH_FILE,RC);
   String kEY = mr.exec();

   System.out.println("..............SEND MAIL..........");

   javSendResult sr = new javSendResult(KEY, MAIL);
   sr.exec();
   System.out.println("..............END...............");

  } else{
      System.out.println("..............SEND MAIL..........");
      javSendError serr = new javSendError(MAIL);
      serr.exec();
      System.out.println("..............END ERROR...............");
 }

  } catch(Exception e) {System.err.println(e.getMessage());}
 } // Fin run
} // Fin PerformanceMonitor
```

GETSEQUENCE

```
package prediction.GetSequence;
import makeFiler.*;
import java.io.*;
import java.util.*;
```

```
/**
 *Descripcion de la clase
 * javGetSequence <archivo_entrada>
 * <archivo_entrada> nombre del archivo con la secuencia objetivo
 * El programa ejecuta blastpgp, programa que identifica las
 * secuencias homologas a la secuencia problema, <archivo_entrada>.
 */

public class javGetSequence implements Constants{
 String FASTA_FILE="";

 public javGetSequence(String f)  { FASTA_FILE=f; }

 public File doFile(String f, String instruction, Runtime rt) throws Exception {
   String header="#!/bin/bsh\n\nexport BLASTDB=/home/gdelrio/CASP6/apps/BLASTDB\nexport
BLASTMAT=/home/gdelrio/CASP6/apps/BLAST/data\nexport
path=$path:/home/gdelrio/CASP6/apps/BLAST\n\n";
   File file = new File(f+".exe");
   if(file.exists()) file.delete();
   BufferedWriter outfile = new BufferedWriter(new FileWriter(file.toString()));

//-- Creando archivo //
   outfile.write(header+"\n");
   outfile.write(instruction+"\n");
   outfile.close();

   String[] cmd = new String[]{"/bin/chmod","+x",file.toString()};
   Process child=rt.exec(cmd);
   child.waitFor();

   return file;
  } // Fin doFile

 public void runBlastPGP () throws Exception
  {
   Runtime rt = Runtime.getRuntime();
   String[] cmd=null;
   Process child=null;

//-- Creando el archivo con la instruccion para correr el blastpgp //
   String instruction=blastpgp_exe+" -b 0 -j 3 -h 0.001 -i "+home_dir+target_dir+FASTA_FILE+" -o
"+home_dir+dir_
blast_out+FASTA_FILE+".blastpgp";
   File exe_file=doFile(FASTA_FILE,instruction,rt);

//-- Ejecuntado el archivo con la instruccion //
   cmd = new String[]{"/bin/bsh",exe_file.toString()};
```

```
   child = rt.exec(cmd);
System.err.println(FASTA_FILE+": blastpgp enviado...");
   child.waitFor();
System.err.println("blastpgp concluido.");

  } // Fin runBlastPGP
}
```

GETCENTRAL

```
package prediction.GetCentral;
import makeFiler.*;
import java.io.*;
import java.util.*;

/**
 *Descripcion de la clase
 * javGetCentral <archivo_entrada> <moldes> <modelos>
 *  <archivo_entrada> nombre del archivo con la secuencia objetivo
 *  <moldes> numero de moldes a usar para evaluar residuos centrales
 *  <modelos> numero de modelos a generar para cada molde
 * El programa va a ejecutar en orden:
 *   a)GETPDBFROMBLAST, para encontrar las proteinas con estructura conocida
 *   b)tcoffee, genera un alineamiento total entre las secuencias
 *   c)DoModellerFiles, crea un sistema de directorios, de acuerdo al numero de moldes
 *   d)modeller6v2, genera <modelos> para cada plantilla
 *   e)GetMostTraversed, crea las graficas y obtiene los vertices mas transitados, residuos centrales.
 */

public class javGetCentral implements Constants{
 String FASTA_FILE;
 int NO_TEMPLATES, NO_MODELS, resPDB=0;

 public javGetCentral(String fasta_file, String no_temp, String no_model) {
   FASTA_FILE=fasta_file;
   NO_TEMPLATES=Integer.parseInt(no_temp);
   NO_MODELS=Integer.parseInt(no_model);
  }

 public File doFile(String f, String instruction, Runtime rt, String header) throws Exception{
  //--- String header="#!/bin/bsh\n\nexport
CLUSTALW_4_TCOFFEE=/home/gdelrio/CASP6/apps/CLUSTALW/clustalw1.8/clustalw\nexport
path=$path:/home/gdelrio/CASP6/ apps/TCOFFEE/T-COFFEE_distribution_Version_1.37/bin\n";
   File file = new File(f+".exe");
   if(file.exists()) file.delete();
   BufferedWriter outfile = new BufferedWriter(new FileWriter(file.toString()));
```

```
//-- Creando archivo //
   outfile.write(header+"\n");
   outfile.write(instruction+"\n");
   outfile.close();

   String[] cmd = new String[]{"/bin/chmod","+x",file.toString()};
   Process child = rt.exec(cmd);
   child.waitFor();

   return file;
  } // Fin doFile

 public LinkedList runGETPDBFROMBLAST(String line) throws Exception {
   LinkedList result = null;

System.err.println("GetPDBFromBlast enviado...");

//-- Crea una instancia y ejecuta GETPDBFromBlast, para encontrar las proteinas similares con
estructura conocid
a //
   GetPDBFromBlast gp = new GetPDBFromBlast( (home_dir+dir_blast_out+line+".blastpgp"),
String.valueOf(NO_TEMPLA
TES), home_dir+dir_blast_out, (home_dir+target_dir+line) );
   result = gp.doIt();
   resPDB = gp.result();

System.err.println("GetPDBFromBlast concluido.");

   return result;
  } // Fin runGETPDBFROMBLAST

 public String runTCOFFEE(String line, String filename, Runtime rt) throws Exception
  {
   String result = "", name=filename.substring((filename.lastIndexOf("/")+1)),
name_ext=name.substring(0,name.in
dexOf("."));
    //--- System.out.println(name_ext);
   String[] cmd=null;
   Process child=null;

//-- Creando el archivo con la instruccion para correr el t_coffee //

   String instruction="/home/gdelrio/CASP6/apps/CLUSTALW/clustalw1.8/clustalw -
INFILE="+filename+" -OUTFILE="+ho
me_dir+dir_tcoffee_out+name_ext+".tcoffee";
   String header="#!/bin/tcsh\n\nsource /home/cmorales/.cshrc\n";

   File exe_file=doFile(line,instruction,rt,header);
```

```
//-- Ejecutando el archivo //
   cmd=new String[]{"/bin/bsh",exe_file.toString()};

   child=rt.exec(cmd);
System.err.println("tcoffee enviado...");
   child.waitFor();
System.err.println("tcoffee terminado.");

   result=home_dir+dir_tcoffee_out+name_ext+".tcoffee";

// Borrando el archivo generado //
   exe_file.delete();

   return result;
  } // Fin runTCOFFEE

 public File[] runDOMODELLERFILESFROMTCOFFEE(String line, LinkedList ll, Runtime rt) throws
Exception {
//-- Creando una instancia y ejecutando DoModellerFiles, para crear el sistema de directorios con los
archivos para ejecutar modeller //
   DoModellerFiles dmf = new DoModellerFiles(ll, String.valueOf(NO_MODELS),
(home_dir+modeller_dir_out),line );

System.err.println("doing modeller files...");
   File[] result= dmf.doIt();

   return result;
  } // Fin runDOMODELLERFILES

 public void runMODELLER(String line, Runtime rt, File[] modeller_dirs) throws Exception{
   int i=0;
   String[] cmd=null;
   Process child=null;
   File modeller_dir=null;

//-- Para cada archivo generado en el paso anterior, se ejecuta modeller, en ese directorio ///
   for(i=0; i<modeller_dirs.length; i++)
    {
     modeller_dir=modeller_dirs[i];

//-- Generando un archivo con las instrucciones para correr el modeller6v2 //

String name = modeller_dir.toString();
String nameF = name.substring(name.lastIndexOf("_")+1);    //,name.lastIndexOf("_"));
String header="#!/bin/tcsh\n\nsource /home/cmorales/.cshrc\n";
String instruction="cd"+modeller_dir.toString()+"\n/home/gdelrio/CASP6/apps
/MODELLER6v2/bin/mod6v2 "+ name +"/"+nameF+".top";
```

```
    File exe_file = doFile(line, instruction, rt, header);

//-- Ejecutando el archivo generado //
    cmd = new String[]{"/bin/bsh",exe_file.toString()};
    child = rt.exec(cmd);
System.err.println("modeller enviado...");
    child.waitFor();
System.err.println("modeller terminado.");

// Borrando el archivo generado //
   exe_file.delete();
   }
  } // Fin runMODELLER

 public Hashtable runGETMOSTTRAVERSED(File[] modeller_dirs) throws Exception{
   Hashtable resultados = new Hashtable();
   int i=0, j=0;
   File dir = null;
   String ref_dir="";
   String line="", token="", pdb="", chain="none", min="0", max="5", pairs="all",
distance_criterium="once", out
file1="",outfile2="",outfile3="";
   String[] contents = null;
   StringTokenizer st = null;
   LinkedList centrales = null;
   BufferedReader infile = null;
   DoGraphFromPDB dgfp = null;
   CalcMIG cm = null;
   GetDynamicNexxuses gdn = null;

   for(i=0; i<modeller_dirs.length; i++)
    {
     dir=modeller_dirs[i];
     //--System.out.println(dir.toString());
     ref_dir=dir.toString();
     if(!ref_dir.endsWith("/")) ref_dir=ref_dir+"/";
     contents=dir.list();

     for(j=0; j<contents.length; j++)
      {
       if(contents[j].indexOf(".B")>-1)
        {
         pdb=contents[j];
         outfile1=ref_dir+pdb+".graph_0-5";
         dgfp=new DoGraphFromPDB((ref_dir+pdb),chain,min,max,pairs,distance_criterium,outfile1);
         dgfp.runIt();
          System.out.println(outfile1);
```

```
        outfile2=outfile1+".calcMIG";
        outfile3=outfile1+".calcMIG_out";
        cm = new CalcMIG(outfile1,outfile2,outfile3);
        cm.runIt();

        outfile1=ref_dir+pdb+".graph_0-5.dn";
        gdn = new GetDynamicNexxuses(outfile2,outfile3,outfile1);
        gdn.runIt();

        centrales = new LinkedList();
        infile=new BufferedReader(new FileReader(outfile1));
        while((line=infile.readLine())!=null)
         {
          st = new StringTokenizer(line);
          token=(String)st.nextElement();
          if(!centrales.contains(token)) centrales.add(token);
          while(st.hasMoreElements()) st.nextElement();
         }
        infile.close();
        resultados.put(dir.toString(), centrales);
        //---System.out.println(dir.toString());
        System.out.println(centrales);
       }
     }
   }

  return resultados;
 } // Fin runGETMOSTTRAVERSED

 public int exec() throws Exception
 {
  Hashtable results_dn = null;

    LinkedList moldes=null;
    LinkedList align_files = new LinkedList();
    File[] modeller_dirs = null;
    int i=0;

    Runtime rt = Runtime.getRuntime();
    String line="", align_file="";
    if(moldes!=null) moldes=null;
    if(align_files.size()>0) align_files.clear();

//-- Ejecuta runGetPDBFromBlast //
    moldes = runGETPDBFROMBLAST(FASTA_FILE);

   if (resPDB != 3) {
```

```
//-- Ejecuta runTCOFFEE, genera una LinkedList con los nombres de los archivos y los alineamientos
generados con
 las plantillas detectados por BLAST //
     if(moldes.size()>0)
      {
       for(i=0; i<moldes.size(); i++)
        {
         align_file=runTCOFFEE(FASTA_FILE, (String)moldes.get(i), rt);
         align_files.add(align_file);
        }

//-- Ejecuta DoModellerFilesFromTCOFFEE //
       modeller_dirs=runDOMODELLERFILESFROMTCOFFEE(FASTA_FILE, align_files, rt);

//-- Ejecuta modeller6v2 //
       runMODELLER(FASTA_FILE,rt,modeller_dirs);

//-- Ejecuta GetMostTraversed //
       results_dn=runGETMOSTTRAVERSED(modeller_dirs);
      }
   }

   return resPDB;
  } // Fin exec
}
```

GETCONSERVED

```
package prediction.GetConserved;
import makeFiler.*;
import java.io.*;
import java.util.*;

/**
 *Descripcion de la clase
 * javGetConserved <archivo_entrada> <cutoff_ali> <num_seqs> <cutoff_inf> <cutoff_sup>
 *  <archivo_entrada> nombre del archivo con la secuencia objetivo
 *  <cutoff_ali> el valor limite de similitud no redundante permitido
 *  <num_seqs> numero de secuencias maximo a utilizar en los alineamientos
 *  <cutoff_inf y _sup> el porcentage de conservacion (inferior y superior),
 *                      para generar lista de residuos conservados
 * El programa va a ejecutar en orden:
 *   a) fastcmd, para obtener las secuencias
 *   b) cd-hit, para eliminar las secuencias <cutoff_ali>% parecidas
 *   c) t_coffee para generar un alineamiento multiple
 *   d) FindConservedClustalW para producir una lista de residuos conservados entre <cutoff_inf> y
<cutoff_sup>
```

```
 **/

public class javGetConserved implements Constants
{
String FILE, CUTOFF_ALI, CUTOFF_INF, CUTOFF_SUP, NUM_SEQS;

 public javGetConserved(String num_seqs, String cutoff_ali, String cutoff_inf, String cutoff_sup,String
file){
  NUM_SEQS = num_seqs;
  CUTOFF_ALI = cutoff_ali;
  CUTOFF_INF = cutoff_inf;
  CUTOFF_SUP = cutoff_sup;
  FILE = file;
 }

 public static File doFile(String f, String instruction, Runtime rt) throws Exception
  {
   String header="#!/bin/tcsh\n\nsource /home/cmorales/.cshrc\n\n";
   File file = new File(f+".exe");
   if(file.exists()) file.delete();
   BufferedWriter outfile = new BufferedWriter(new FileWriter(file.toString()));

//-- Crea el archivo
   outfile.write(header+"\n");
   outfile.write(instruction+"\n");
   outfile.close();

   String[] cmd = new String[]{"/bin/chmod","+x",file.toString()};
   Process child=rt.exec(cmd);
   child.waitFor();

   return file;
  } // Fin doFile

  public static void runFastacmd(String line, Runtime rt) throws Exception
  {
   String[] cmd=null;
   Process child=null;

//-- Ejecutando el parser6.pl para generar archivo processx.getseqs //
   String instruction="/usr/bin/perl "+(home_dir+"prediction/GetConserved/parser6.pl ")+(home_dir+
tmp_dir+line+"_all ")+(home_dir+dir_blast_out+line+".blastpgp
")+(home_dir+tmp_dir+line+".getseqs");
   File exe_file=doFile(line,instruction,rt);
   cmd=new String[]{"/bin/tcsh",exe_file.toString()};
   child=rt.exec(cmd);
System.err.println("parser6.pl enviado..");
   child.waitFor();
```

```
System.err.println("parser6.pl terminado.");

//-- Haciendo ejecutable el archivo generado por parser6.pl //
   cmd = new String[]{"/bin/chmod","+x",(home_dir+tmp_dir+line+".getseqs")};
   child=rt.exec(cmd);
   child.waitFor();

// Ejecutando el archivo, que contiene la instruccion fasta //
    cmd=new String[]{(home_dir+tmp_dir+line+".getseqs")};
    child=rt.exec(cmd);
System.err.println("fastacmd enviado..");
    child.waitFor();
System.err.println("fastacmd terminado.");

//-- Borrando archivo //
    exe_file.delete();
  } // Fin runFastcmd

 public static void addRefSeq(String target_seq) throws Exception
  {
System.err.println("Adding ref seq...");
   String line="", ref_seq="";
   BufferedReader infile_ref = new BufferedReader(new FileReader(home_dir+target_dir+target_seq));
   BufferedReader infile_seqs = new BufferedReader(new
FileReader(home_dir+tmp_dir+target_seq+"_all"));
   BufferedWriter outfile = new BufferedWriter(new
FileWriter(home_dir+tmp_dir+target_seq+"_all2"));

   while((line=infile_ref.readLine())!=null)
    {
     if(ref_seq.equals("")) ref_seq=line+"\n";
     else ref_seq=ref_seq+line;
    }
   infile_ref.close();

//-- Escribe la secuencia objetivo en el archivo de salida //
   outfile.write(ref_seq+"\n");
   while((line=infile_seqs.readLine())!=null)
    {
//-- Agrega a ese archivo las secuencias obtenidas por fasta //
     outfile.write(line+"\n");
    }
   infile_seqs.close();

   outfile.flush();
   outfile.close();

System.err.println("Ref seq added.");
```

```
  } // Fin addRefSeq

 public static void runCDHIT(String line, Runtime rt, String cutoff) throws Exception
  {
   String[] cmd=null;
   Process child=null;

//-- Definiendo las instrucciones para correr CD-HIT //
   double similitud_limite = Double.parseDouble(cutoff);
   System.out.println(similitud_limite);
   if(similitud_limite>=0.70)
    {
     cmd=new String[]{(cdhit_dir+"cd-hit"),"-n","5","-c",cutoff,"-i",(home_dir+tmp_dir+line+"_all2"),"-
o",(home_dir+tmp_dir+line+".cdhit_seq")};
    // instruccion = cdhit_dir+"cd-hit "+"-n 5 -c,cutoff,"-i",(home_dir+tmp_dir+line+"_all2"),"-
",(home_dir+tmp_dir+line+".cdhit_seq")};

    }
   else if(similitud_limite<0.70 && similitud_limite>=0.55)
    {
     cmd=new String[]{(cdhit_dir+"cd-hit"),"-n","4","-c",cutoff,"-i",(home_dir+tmp_dir+line+"_all2"),"-
o",(home_dir+tmp_dir+line+".cdhit_seq")};
    }
   else if(similitud_limite<0.55 && similitud_limite>=0.50)
    {
     cmd=new String[]{(cdhit_dir+"cd-hit"),"-n","3","-c",cutoff,"-i",(home_dir+tmp_dir+line+"_all2"),"-
o",(home_dir+tmp_dir+line+".cdhit_seq")};
    }
   else if(similitud_limite<0.50 && similitud_limite>=0.40)
    {
     cmd=new String[]{(cdhit_dir+"cd-hit"),"-n","2","-c",cutoff,"-i",(home_dir+tmp_dir+line+"_all2"),"-
o",(home_dir+tmp_dir+line+".cdhit_seq")};
    }
   else
    {
     System.err.println("El nivel de redundancia especificado para las secuencias, es mas bajo de lo que
este programa puede calcular");
     System.exit(0);
    }

//-- Ejecutando las instrucciones para correr el cd-hit //
   child = rt.exec(cmd);
System.err.println("cd-hit enviado...");
   child.waitFor();
System.err.println("cd-hit terminado.");
  } // Fin runCDHIT
```

```
 public static void runSELECTSEQS(String line, String num_seqs) throws Exception
  {
//-- Selleciona solo un numero de secuencias, es un parametro de la prediccion //
   SelectSeqs ss = new
SelectSeqs((home_dir+tmp_dir+line+".cdhit_seq"),num_seqs,(home_dir+tmp_dir+line+".cdhit_seq2"
));
   ss.doIt();
System.err.println("reducido el numero de seqs a: "+num_seqs);
  } // Fin runSELECTSEQS


 public static void runTCOFFEE(String line, Runtime rt) throws Exception
  {
   String[] cmd=null;
   Process child=null;

//-- Creando el archivo con la instruccion para correr el t_coffee //

   String instruction = "cd " +home_dir+"\n"+tcoffee_exe+" -in=./"+tmp_dir+line+".cdhit_seq2
slow_pair lalign_id_pair -dp_mode myers_miller_pair_wise -cle
an_aln 0 -tree_mode slow -outfile=./"+dir_tcoffee_out+line+".tcoffee";

   File exe_file=doFile(line,instruction,rt);
   cmd=new String[]{"/bin/tcsh",exe_file.toString()};

//-- Ejecutando el archivo //
   child=rt.exec(cmd);
     outProg s1 = new outProg ("stdin", child.getInputStream ());
     outProg s2 = new outProg ("stderr", child.getErrorStream ());
     s1.start ();
     s2.start ();
   child.waitFor();
   System.err.println("tcoffee terminado.");

//-- Borrando archivo generado //
   exe_file.delete();
  } // Fin runTCOFFEE

 public static String runFindConservedClustalW(String line, String lim_inf, String lim_sup) throws
Exception
  {
   String x = "";
   String infile = home_dir+dir_tcoffee_out+line+".tcoffee",
outfile=home_dir+dir_tcoffee_out+line+".tcoffee_conserved";
   FindConservedClustalW findaa = new FindConservedClustalW(infile, lim_inf, lim_sup, outfile);
System.err.println("FindConservedClustalW enviado...");
   x = findaa.doIt();
System.err.println("FindConservedClustalW terminado.");
```

```
   return x;
  } // end runFindConservedHSSP

 public String exec() throws Exception
  {
   String xret = "";

     Runtime rt = Runtime.getRuntime();

//-- Ejecuta fastacmd usando parser6.pl //
       runFastacmd(FILE, rt);

//-- Anade la secuencia objetivo al principio de las secuencias a alinear, esto indica que es la secuencia
a la cual se le detectaran los residuos conservados //
       addRefSeq(FILE);

//-- Ejecuta cd-hit //
       runCDHIT(FILE, rt, CUTOFF_ALI);

//-- Ejecuta SelecSeqs, reduce el numero de seqs obtenidas en CDHIT a num_seqs //
       runSELECTSEQS(FILE,NUM_SEQS);

//-- Ejecuta t_coffe //
       runTCOFFEE(FILE, rt);

//-- Ejecuta FindConservedClustalW, este genera una lista de residuos conservados //
       xret = runFindConservedClustalW(FILE, CUTOFF_INF, CUTOFF_SUP);

   return xret;
  } // Fin exec
}
```

GETBEST

```
package prediction.GetBest;

public class javGetBest{
 String FASTA_FILE;

public javGetBest(String file){
 FASTA_FILE = file;
}

public String exec() throws Exception{

   EvaluateCentralConserved  e_cc = new EvaluateCentralConserved(FASTA_FILE);
   String x = e_cc.exec();
```

```
   BestModel gbm = new BestModel(x);
   String PATH = gbm.exec();

return PATH;

}//Fin exec
```

MAKERESULT

```
package prediction.Result;

import makeFiler.*;
import java.io.*;
import java.lang.String.*;
import java.util.*;

public class javMakeResult implements Constants{
 LinkedList LL;
 String PROTKEY;
 String PATH;
 String RC;

  public javMakeResult(String key, String path, String rc){
   PROTKEY = key.substring(0,key.lastIndexOf("."));
   PATH = path;
   RC = rc;
  }

  public String exec(){
   try{
     String x = "", line = "";
     char com = '"';
     FileOutputStream FileO = new
FileOutputStream(home_sal+"/INTERFACE/pagRESULT/"+PROTKEY+".html");

     x= "<HTML>"+'\n'
       +"<BODY>"+'\n'
       +"<applet code=proteinViewer.class codebase=http://www.mpcusack.com/ width=700
height=500>"+'\n'
       +"<PARAM NAME=PROTEIN VALUE="+PROTKEY+">"+'\n'
       +"<PARAM NAME=PDB_STRING VALUE="+ com +'\n';

     BufferedReader infile = new BufferedReader(new FileReader(PATH));
     while((line=infile.readLine())!=null)
     {
```

```
      if (line.charAt(0)!='A'){

      }else{
       x = x + line + '\n';
      }
     }
     infile.close();

     x = x + "END" + '\n' + com + ">" + '\n'
        +"<PARAM NAME=CriticalResidues VALUE=" + com + RC + com +">"+ '\n'
        +"</applet>"+'\n'
        +"<br><br>"+ '\n'
        +"<applet code=DisSeq.class codebase=http://www.mpcusack.com/ width=700 height=80
name=DisSeq>" +
 '\n'
        +"<param name=text value=Please Wait While Applet Loads>"+ '\n'
        +"</applet>"+ '\n'
        +"<br><br>"+ '\n'
        +"<p>The Following Residues Were Identified As Critical:</p>"+ '\n'
        +"<p>"+RC +"</p>"+ '\n'
        +"</body>"+ '\n'
        +"</html>";

      FileO.write(x.getBytes());
    }catch(Exception e) {System.err.println("Error"); }
   // }catch(java.io.IOException ie){ }
    return PROTKEY;
   }// end exec()
}
```

SENDRESULT

```
package prediction.Result;

import java.util.Properties;
import javax.mail.*;
import javax.mail.internet.*;

public class javSendResult {
  String KEYPAG = "";
  String TO = "";

  public javSendResult(String key, String to){
   KEYPAG = key;
   TO = to;
  }
```

```
  public void exec() throws Exception{

// public static void main (String args[]) throws Exception {

    String smtpHost = "mail.ifc.unam.mx";
    String from = "bioinfo@ifc.unam.mx";
    String to = TO;

    // Get system properties
    Properties props = System.getProperties();

    // Setup mail server
    props.put("mail.smtp.host", smtpHost);

    // Get session
    Session session = Session.getDefaultInstance(props, null);
    session.setDebug(true);

    // Define message
    MimeMessage message = new MimeMessage(session);
    message.setFrom(new InternetAddress(from));
    message.addRecipient(Message.RecipientType.TO,
      new InternetAddress(to));
    message.setSubject("Result PROTEIN STRUCTURE PREDICTION");
    message.setText("!!You can find the result in :
http://132.248.16.230:8090/server/"+KEYPAG+".html");

    // Send message
    Transport.send(message);
  }
}
```

2) Service admission

LOGIN SYSTEM

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;
import java.sql.*;

public class servLogin extends HttpServlet {

  protected void doPost(HttpServletRequest request, HttpServletResponse response)
  throws ServletException, java.io.IOException {
```

```
   response.setContentType("text/html");
   java.io.PrintWriter out = response.getWriter();

   String user = request.getParameter("username");
   String passw = request.getParameter("password");
   String argm[] = {user, passw};

   //---Valida datos
      Validate valida = new Validate(argm);
      int res = valida.validateLog();
      //System.out.println("res es"+res);
        if ( res == 1){   // user.length() == 0 || passw.length() == 0) {
            //out.println("faltan  datos!!!"+ user+"," );
            RequestDispatcher
rd=request.getRequestDispatcher("../../../INTERFACE/pagHTML/pagLoginError.html");
             rd.forward(request,response);
         }
        else {

    //---Valida usuarios
          // Connection con;

          // DBCon conex = new DBCon();
           //con = conex.exec();
           DAO_usuarios val = new DAO_usuarios();
           val.getConnection();
           boolean x = val.validate (user,passw);
           //val.closeConnection();
           if (x) {
              RequestDispatcher
rd=request.getRequestDispatcher("../../../INTERFACE/pagHTML/pagPrincipal.html");
              rd.forward(request,response);
           }else{
              RequestDispatcher
rd=request.getRequestDispatcher("../../../INTERFACE/pagHTML/pagLoginError.html");
              rd.forward(request,response);
           }
            val.closeConnection();
        }
}
}
```

Glosario

A

Aplicaciones Web — Son fragmentos de código que se ejecutan cuando se realiza una petición HTTP.

Alineamiento de secuencias — Es la posición relativa de dos secuencias en las que se produce el mayor número de coincidencias entre sus componentes.

Aminoácidos (AA) — Son moléculas que poseen un *grupo carboxilo* y un *grupo amino,* ambos unidos al mismo átomo de carbono.

Aminoácidos centrales — Después de generar la red de una proteína se aplica el algoritmo de la ruta mas corta, para obtener los aminoácidos mas transitados, éstos son considerados centrales para la comunicación de la red.

Aminoácidos conservados — Son aquellos que tienden a ser evolucionadamente conservados en la secuencia de la proteína y que se conservan a través del tiempo.

Aminoácidos críticos — *S*on aquellos que se consideran fundamentales para mantener la estructura de la proteína y la actividad biológica (al ser sustituidos cambia la función de la proteína).

Angstrom (Å) — 1 Angstrom = 10^-8 m

Arquitectura de las aplicaciones Web — *S*uelen presentar un esquema de tres niveles. El primer nivel es la capa de presentación (incluye al navegador y al servidor Web) que es el responsable de dar a los datos un formato adecuado. El segundo nivel está referido habitualmente a algún tipo de programa o *script*. Finalmente, el tercer nivel proporciona al segundo nivel los datos necesarios para su ejecución.

B

Bioinformática — Aplicación de métodos y herramientas matemáticas, estadísticas, químicas, físicas, biológicas e informáticas para adquirir, almacenar, organizar, archivar, analizar o visualizar los datos.

C

CASP — *Critical Assessment of techniques for protein Structure Prediction*. Evaluación crítica de los métodos para la predicción de la estructura de proteínas.

Cristalografía de rayos X — En la Cristalografía los rayos X interaccionan con la materia. Cuando encuentra átomos el rayo se desvía, en caso contrario tenemos ausencia de materia. La magnitud de esta desviación depende de la densidad de la materia que encuentre a su paso.

D

Diseño arquitectónico — Es un proceso, manera o estilo que inicia con la abstracción del sistema y que finaliza con una forma o estructura de este.

H

HTML — *Hyper Text Markup Language*
El Lenguaje de Marcación de Hipertexto es un método para codificar la información de los documentos y sus enlaces.

HTTP — *Hyper Text Transfer Protocol*
El Protocolo de Transferencia de Hipertexto especifica cómo el navegador y el servidor intercambian información en forma de peticiones y respuestas.

I

Internet — Internet es la red física (basada en los protocolos TCP/IP) que nos permite acceder a otras computadoras, aún de entornos diferentes como UNIX, MS-DOS, MacOs, etc. Algunos de los servicios disponibles en Internet aparte de la Web son el acceso remoto a otras máquinas (telnet y ssh), transferencia de archivos (FTP), correo electrónico (e-mail), boletines electrónicos (news o grupos de noticias), conversaciones en línea (chat), mensajería instantánea (ICQ, YIM, etc.), entre otros.

M

MIN — *Minimun Interacting Network*
Este algoritmo obtiene los aminoácidos centrales de una proteína.

N

Navegador Web — El navegador o explorador Web sirve para extraer elementos de información, llamados documentos o páginas Web, de los servidores Web (o sitios) y mostrarlos en la pantalla del usuario. El navegador puede considerarse como una interfaz de usuario universal. Dentro de sus funciones están la petición de las páginas Web, la representación adecuada de sus contenidos y la gestión de los errores que se puedan producir.

NIM — Este algoritmo es la idea inversa de MIN. A partir de los aminoácidos críticos selecciona un modelo de la estructura de una proteína.

P

Procedimiento — Es la combinación de herramientas y técnicas que, juntas, dan un resultado particular.

Proteína — Una proteína es una macromolécula formada por aminoácidos unidos mediante un enlace peptídico.

R

RMN — Resonancia Magnética Nuclear
La RMN puede usarse para analizar moléculas en solución, es decir, en su ambiente natural. Los datos que se obtienen son una serie de distancias y ángulos entre los átomos, a partir de esta lista se construye un modelo molecular que satisfaga las medidas observadas experimentalmente.

S

Servidor

Un servidor en computación tiene dos significados: la aplicación informática que realiza una tarea en beneficio de otras aplicaciones llamadas clientes y el ordenador en el que se ejecutan dichos programas.

Servidor de bases de datos

Es un programa que escucha y atiende todas las peticiones que el usuario realiza para acceder a la información de la base de datos. El servidor de la base de datos recoge las peticiones del cliente, a través del servidor Web, y resuelve las mismas. El resultado de las peticiones del usuario es devuelto al servidor Web, el cual se comunica de nuevo con el cliente para mostrárselo. La comunicación con el servidor de bases de datos debe realizarse a través de un lenguaje que el mismo reconozca, como SQL.

Servidor de correo

Un servidor de correo, o MTA (*Mail Transfer Agent*), abre una conexión para establecer la comunicación con el servidor de correo; recibe, almacena y distribuye archivos o información en respuesta a las peticiones enviadas vía e-mail. SMTP recibe los mensajes de correo electrónico y los entrega a los destinatarios; existen dos métodos para que el usuario final revise el correo, POP (*Post Office Protocol*) e IMAP (*Interactive Mail Access Protocol*).

Servidor Web

Es un programa que implementa el protocolo HTTP, para transferir hipertextos, o páginas HTML. Un servidor Web se encarga de mantenerse a la espera de peticiones HTTP llevadas a cabo por un cliente HTTP, que solemos conocer como navegador. El navegador realiza una petición al servidor y éste le responde con el contenido que el cliente solicita. El cliente es el encargado de interpretar el código HTML, es decir, de mostrar las fuentes, los colores y la disposición de los textos y objetos de la página; el servidor tan sólo se limita a transferir el código de la página sin llevar a cabo ninguna interpretación de la misma.

U

URL	*Uniform Resource Locutor* El Localizador Uniforme de Recursos permite localizar o acceder a cualquier recurso de la red (pues lo asocia con una dirección única)

W

World Wide Web (Web)	Es el universo de información accesible a través de Internet. La funcionalidad elemental de la Web se basa en tres estándares: URL, HTTP y HTML.

Bibliografía

Beust, C. et Al. (2001) *Programación Java Server con J2EE edición 1.3.* Anaya multimedia. Capítulo 1,5,6.

Branden, C., Tooze, J. (1991) *Introduction to protein structure.* Garland Publishing. Parte 1.

Bruce, A. et Al. (2002) *Biología Molecular de la Célula.* Ediciones omega. Capítulo 5.

Claverie, J-M., Notredame, C. (2003) *Bionformatics for dummies.* Wiley Publishing. Capítulo 1.

Danny, A. et Al. (1999) *Professional Java Server Programming*. Wrox. Capítulo 1,2,8.

Foster, I. (2004) *Computación sin fronteras*, Scientific American México, p.61-67.

Gibas, C., Jambeck, P. (2001) *Developing Bioinformatics Computer Skills.* O'Reilly. Capítulo 1.

Kurniawan, B. (2002) *Java for the Web whith Servlets, JSP, and EJB. A Developers Guide to Scalable J2EE Solutions.* New Riders. Capítulo 1.

Lawrence, S. (2002) *Ingeniería de software. Teoría y práctica*. Prentice Hall. Capítulo 1.

Thibert, B., Bredesen, D-E. del Rio, G. (2005) *Improved prediction of criticalresidues for protein function based on network and phylogenetic analyses*. Sometido a BMC Bioinformatics.

Referencias en Internet:

CASP: http://predictioncenter.llnl.gov/casp6/

java: http://java.sun.com/overview.html

j2seAPI: http://java.sun.com/j2se/1.4.2/index.jsp

javServer: http://java.sun.com/products/servlet/index.jsp

jakTomcat: http://jakarta.apache.org/site/downloads/downloads_tomcat-5.cgi

MySQL: http://dev.mysql.com/downloads/index.html

javMySQL: http://www.mysql.com/products/connector/j/

javMail: http://java.sun.com/products/javamail/downloads/index.html

bioinfNCBI: http://www.ncbi.nlm.nih.gov/About/primer/bioinformatics.html

bioMétodos: http://www.ciencias.uma.es/publicaciones/encuentros/encuentros87/prediccion.htm

bioHerram: http://www.cab.inta.es/~LBIOINFO/courses/CBMSO2004/pages/practica/

bioSec: http://www.cab.inta.es/~LBIOINFO/courses/Barcelona2004/pages/pred1d/teoria/

bioPred: http://decsai.ugr.es/~dpelta/docuFANS/node29.html

bioPred: http://decsai.ugr.es/~dpelta/docuFANS/node29.html

bioinfo: http://www.monografias.com/trabajos14/bioinforma/bioinforma.shtml

Blast: ftp://ftp.ncbi.nih.gov/blast/

Tcoffee: http://igs-server.cnrs-mrs.fr/~cnotred/Projects_home_page/t_coffee_home_page.html

Modeller: http://salilab.org/modeller/

DBase: http://www.rcsb.org/pdb/