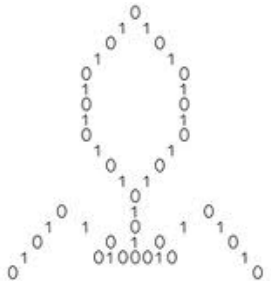
Estudio de la relación de divergencia en el uso de codones sinónimos entre virus-huésped y presencia de microRNA.

Riberi Franco G.

15 de octubre de 2013

Universidad Nacional de Río Cuarto
Departamento de Computación



FuDePAN
Fundación para el desarrollo de la programación en ácidos nucleicos

Trabajo Final

Licenciatura en Ciencias de la Computación

Estudio de la relación de divergencia en el uso de codones sinónimos entre virus-huésped y presencia de microRNA.

Autor:
Franco G. Riberi

Directora:
Laura
Tardivo

Co-Director:
Daniel
Gutson

Resumen

El presente proyecto encuadra en una de las áreas de la Biología, conocida como Virología, la cual estudia la acción de los virus en el organismo y cómo pueden ser aplicados ciertos fármacos para su tratamiento.

El código genético está organizado en tripletes o codones (conjunto de tres nucleótidos que determinan un aminoácido). Al menos uno de estos tres nucleótidos puede ser distinto y sin embargo codificar para el mismo aminoácido, por lo cual se dice que el código genético esta *degenerado*. Los tripletes que codifican para un mismo aminoácido se denominan *sinónimos*.

Se conoce que cada especie tiene un uso de codones sinónimos característico que le permite optimizar la síntesis proteica. Para algunos organismos patógenos intracelulares, como los virus, existe una divergencia entre el uso de codones sinónimos utilizados por el virus y el huésped correspondiente. El origen de esa divergencia no está suficientemente esclarecido.

En el presente trabajo se estudiará si la divergencia en el uso de codones sinónimos entre virus y huésped contribuye a disminuir la interferencia de los microRNA en la expresión de los RNA mensajeros de origen viral. De esa manera se contribuirá a comprender mejor la relación virus-huésped y la evolución viral.

Se desarrolló un software que realiza diversas comparaciones masivas entre secuencias de ${}_{mi}$RNA y ${}_{m}$RNA, además de comparaciones a nivel estructural entre ${}_{m}$RNA. El software sigue una implementación distribuida, a través de la utilización de varios procesos que cooperan para realizar las tareas. Por otra parte, se implementaron diversas librerías con funcionalidades anexas, que se acoplan con otros paquetes de software relacionados con el área de estudio.

Palabras claves: ${}_{mi}$RNA, ${}_{m}$RNA, aminoácidos, nucleótidos, FuD, Remo, fideo.

Agradecimientos

A todos los que la presenten vieren y entendieren.
Inicio de las Leyes Orgánicas.
Juan Carlos I.

A Dios y la vida.
Por haberme permitido llegar hasta este punto y haberme dado salud para lograr mis objetivos.

A mi madre Alejandra Riberi.
Por haberme inculcado la ética de trabajo y superación, por la motivación constante que me ha permitido ser una persona de bien, pero más que nada, por su amor.

A mi padre Ariel Buffa.
Por haberme apoyado en todo momento, por sus consejos, por el valor mostrado para salir adelante.

A mi directora Laura Tardivo.
Por dirigir este trabajo, por su constante ayuda, su predisposición, sus conocimientos y apoyo.

A mi co-director Daniel Gutson.
Por ser el mentor de este proyecto y quien confió mí para llevarlo adelante. Por su paciencia y dedicación. Por compartir su experiencia y conocimientos, tanto académicos como humanos.

Al Dr. Daniel Rabinovich.
Por su predisposición, dedicación y entusiasmo. Por haberme otorgado la responsabilidad de llevar a cabo su intelecto.

A los miembros de FuDePAN.
Por estar siempre presentes, por el trabajo en conjunto. Especialmente al

Lic. Emanuel Bringas por su constante apoyo y ayuda.

Al Departamento de Computación.
A todos los docentes que me capacitaron a lo largo de la carrera. Especialmente al Mg. Marcelo D. Arroyo por su colaboración cuando fue requerida.

A mis compañeros y amigos.
Quienes de diversas maneras estuvieron a mi lado durante este proceso compartiendo añecdotas, frustraciones y alegrías.

A aquellas personas que ya no están a mi lado.
Por haberme marcado de una u otra forma.

A tí querido lector.
Por tomarte el tiempo y la dedicación de leer este trabajo.

Franco G. Riberi

Índice general

Índice de figuras

Introducción

La vida sin pruebas y desafíos
no sería provechosa, sino.
¿Como aprenderíamos?

Josue Alvarez. Guatemala

Debido al gran número de datos generados por los experimentos biológicos actuales, una nueva clase de híbridos ha surgido entre informáticos y biólogos. Los mismos se encargan de desarrollar métodos y programas para el análisis masivo de la información biológica. Dentro de esta línea, la *Bioinformática* es considerada una de las disciplinas con mayor índice de expansión y crecimiento en la actualidad.

No se puede abordar la Bioinformática sin describir inicialmente algunos conceptos elementales de la *Biología*. En realidad son los biólogos y los bioquímicos quienes hicieron su primer acercamiento a la tecnología computacional como elemento fundamental para su trabajo diario. La tecnología proporciona un elemento teórico y las herramientas prácticas necesarias para que los científicos puedan explorar, analizar y establecer conclusiones al respecto.

La Biología

La Biología es aquella ciencia que estudia a los seres vivos, ya sean estos animales, plantas o seres humanos. Principalmente se preocupa de los procesos vitales de cada ser, tales como su nacimiento, desarrollo, muerte y procreación. Por lo que estudia el ciclo completo de los mismos[Curtis H., 2006]. La palabra como tal, proviene del griego, *bios* (vida) y *logos* (estudio). Es decir, *Estudio de la vida*.

En particular, una de las áreas de aplicación de la Biología es la Virología, la cual estudia la acción de los virus y cómo pueden ser aplicados ciertos fármacos para su tratamiento.

Todos los seres vivos existentes sintetizan proteínas. De hecho, los tipos celulares (procariotas o eucariotas) están determinados por los tipos de proteínas

que cada célula puede sintetizar. Por este motivo el material genético debe controlar los tipos y cantidades de proteínas que sintetiza una célula. Prácticamente todos los procesos biológicos dependen de la presencia o la actividad de proteínas. Dentro de las diversas funciones que desempeñan, se pueden mencionar las siguientes: función defensiva (crean los anticuerpos y regulan factores contra agentes extraños o infecciones), función reguladora (colaboran en la regulación de la actividad de las células, por ejemplo en la división celular), función estructural o de construcción (forman parte de las estructuras corporales suministrando el material necesario para el crecimiento y la reparación de tejidos y órganos del cuerpo), función energética (cuando el aporte de hidratos de carbono y grasas resulta insuficiente para cubrir las necesidades energéticas, las proteínas se emplean como combustible energético), entre otras.

Las *proteínas*, también denominadas polipéptidos, están constituidas por cadenas de un surtido de sólo 20 *aminoácidos*. A su vez, cada aminoácido se encuentra conformado por una sucesión de tres *nucleótidos* consecutivos, los cuales se los denominan *codón* o *triplete*, y codifican a un determinado aminoácido. A pesar de ello, existen aminoácidos que son codificados por más de una combinación de codones, a los cuales se los conoce como *codones sinónimos*. Desde una perspectiva computacional, esta secuencia de nucleótidos se trata de una cadena de caracteres pertenecientes al alfabeto de los 4 nucleótidos del *DNA*(Ácido Desoxirribonucleico) conocidos como *A*denina, *C*itosina, *G*uanina y *T*imina.

El nexo entre el DNA que se encuentra en el núcleo celular y las proteínas, que se encuentra en los ribosomas(citoplasma), lo conforma el RNA(Ácido Ribonucleico), aunque el mismo puede ser funcional por sí mismo (figura 1). Este flujo de la información que lleva a la producción de proteínas se denomina *Dogma Central de la Biología*.

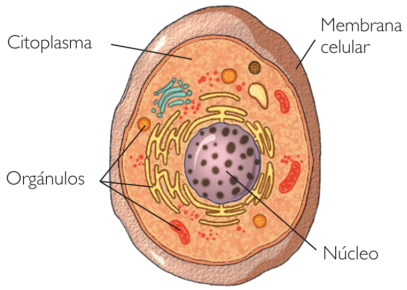


Figura 1: Estructura de un célula [1].

Básicamente, el DNA transfiere la información al RNA, el cual controla directamente la síntesis de proteínas. Este flujo es unidireccional, aunque existen excepciones. En este proceso entran en juego dos etapas importantes conocidas como *transcripción* y *traducción*. La transcripción es el proceso mediante el cual el RNA es sintetizado a partir de un molde de DNA, esto significa, que la información del DNA es reescrita, pero básicamente en el mismo lenguaje de nucleótidos. Por otro lado, la traducción es el proceso por el cual el RNA controla la síntesis de proteínas. En términos generales la información en el lenguaje de nucleótidos es traducida al lenguaje de aminoácidos[Tamarin, 1996]. El proceso mencionado anteriormente se exhibe gráficamente en la figura 2.

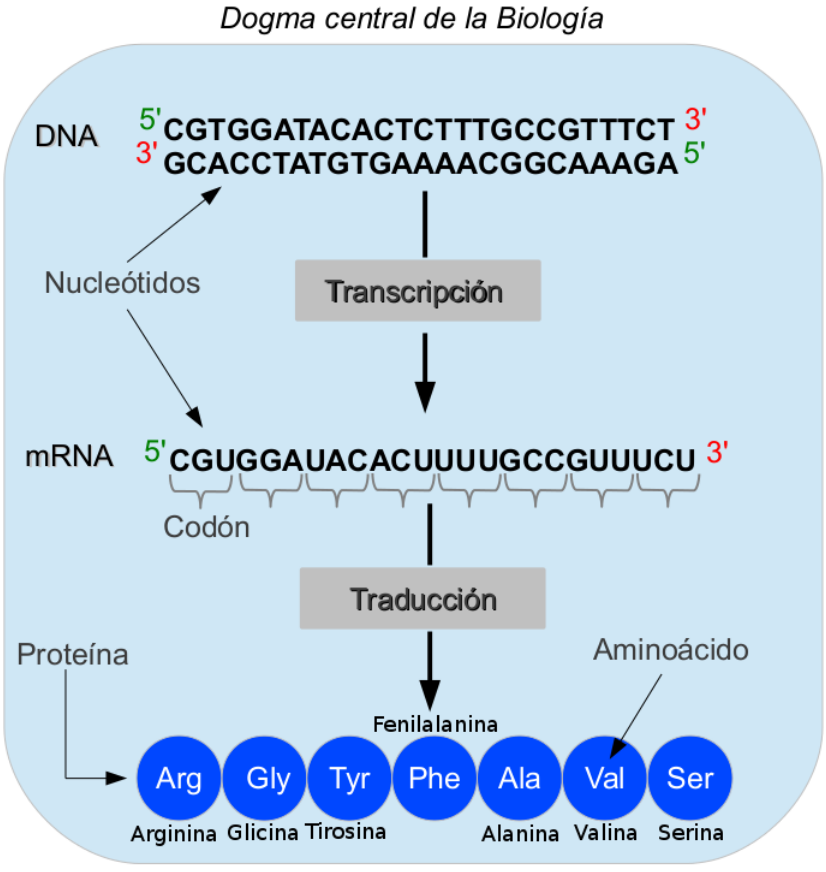


Figura 2: Integrando conceptos iniciales [2].

Las secuencias mencionadas poseen diversos tamaños lo cual dificulta su procesamiento y análisis de forma manual. Las secuencias de proteínas contienen en promedio entre 100 y 5000 aminoácidos. Por otro lado, las secuencias de DNA son mucho más grandes que las de proteínas. Por ejemplo, el

DNA de la bacteria E. Coli tiene 4.7 millones de nucleótidos y el del humano 3.3 billones.

La Bioinformática

Si bien no existe una única definición, puede decirse que *la Bioinformática* es la aplicación de las Ciencias de la Computación a la Biología Molecular. Este término hace referencia a campos de estudios interdisciplinarios muy vinculados, que requieren el uso o el desarrollo de diferentes técnicas que incluyen informática, matemática aplicada, estadística, Ciencias de la Computación, Inteligencia Artificial, Química y Bioquímica para solucionar problemas, analizar datos, o simular sistemas o mecanismos, todos ellos de índole biológica, y usualmente de nivel molecular. El núcleo principal de estas técnicas se encuentra en la utilización de recursos computacionales para solucionar o investigar problemas sobre escalas de tal magnitud que sobrepasan el discernimiento humano.

Según la definición del Centro Nacional para la Información Biotecnológica ("National Center for Biotechnology Information", NCBI por sus siglas en Inglés, 2001):

> *"La Bioinformática es un campo de la ciencia en el cual confluyen varias disciplinas tales como: biología, computación y tecnología de la información. El fin último de este campo es facilitar el descubrimiento de nuevas ideas biológicas así como crear perspectivas globales a partir de las cuales se puedan discernir principios unificadores en biología."*

Históricamente, el uso de las computadoras para resolver cuestiones biológicas comenzó con el desarrollo de algoritmos y su aplicación en el entendimiento de las interacciones de los procesos biológicos entre diversos organismos. La Biología Molecular es una ciencia rica en datos que crecen a tasas exponenciales[Genebank, 1999]. Sin embargo, desde la perspectiva computacional la característica clave de los datos biológicos no es tanto su volumen sino su diversidad, heterogeneidad y dispersión, lo que impide o dificulta la explotación integrada de esta información. **Los "grandes volúmenes de datos" se suelen citar como una de las características mas relevantes de la Bioinformática debido a sus tasas exponenciales de crecimiento.** Este crecimiento se debe en buena medida al desarrollo de nuevas tecnologías habiendo pasado del estudio de genes individuales a genomas completos y ahora a la interacción entre organismos.

Remo

Luego de mencionar brevemente algunos conceptos importantes a continuación se presenta el trabajo titulado: *"Estudio de la relación de divergencia en el uso de codones sinónimos entre virus-huésped y presencia de microRNA"*. El mismo tiene como principal objetivo contrastar una teoría postulada y encomendada por el Dr. Roberto Daniel Rabinovich[1], que involucra principalmente la molécula de RNA. Dicho estudio surge como propuesta de la fundación FuDePAN[2].

Se desea estudiar si la divergencia en el uso de codones sinónimos entre virus y huésped contribuye a disminuir la interferencia de los microRNA ($_{mi}$RNA) en la expresión de los RNA mensajeros($_{m}$RNA) de origen viral. Los $_{mi}$RNA son pequeñas moléculas codificadas en el DNA de las distintas especies que regulan la expresión de una gran fracción de las proteínas[3]. Un $_{m}$RNA es un tipo particular de ácido ribonucleico monocatenario que contiene la información genética procedente del DNA(bicatenario) para utilizarse en la síntesis de proteínas. De esa manera se contribuirá a comprender mejor la relación virus-huésped y la evolución viral. Estos estudios tienen también una importancia potencial en la comprensión de la patogenia viral y en el desarrollo de herramientas terapéuticas como la terapia génica.

Para lograr tal objetivo se plantea desarrollar un software que, dadas ciertas secuencias de $_{m}$RNA y secuencias de $_{mi}$RNA, permita determinar si la divergencia en el uso de codones sinónimos entre virus y huésped contribuye a disminuir la interferencia de los $_{mi}$RNA en la expresión de los $_{m}$RNA de origen viral.

Dada la gran cantidad de datos a manipular y el tamaño de las secuencias, se requiere de un gran número de pequeños cálculos y comparaciones lo cual demanda un alto poder de cómputo. Por otro lado, para tener soporte de la ejecución distribuida se plantea utilizar el framework *FuD*[4][Biset, 2009] sin tener que desarrollar específicamente esta funcionalidad y sin perder de vista el problema original.

Algunos motivaciones que llevaron a tomar la decisión de realizar el presente proyecto refieren a la posibilidad de enfrentarse con un problema real,

[1]Miembro del Instituto Biomédico en Retrovirus y SIDA-INBIRS e investigador del Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET), Buenos Aires, Argentina.

[2]Fundación para el Desarrollo de la Programación en Ácidos Nucleicos, Córdoba, Argentina. `http://www.fudepan.org.ar/`

[3]Los $_{mi}$RNA reconocen secuencias parcialmente complementarias en un $_{m}$RNA correspondiente, dando como resultando la inhibición de la traducción de dicho mRNA.

[4]FuDePAN Ubiquitous Distribution

de índole biológico, manipulando datos muy complejos tales como secuencias de DNA, secuencias de virus, entre otras. Por otro lado, de cierto modo, estos estudios pueden servir para contribuir a la investigación de temas relacionados con el área de la salud.

En el resto del documento se empleará el término **R-emo** o **Remo** para referirse al software desarrollado como solución planteada.

Estructura del Documento

El documento esta compuesto por 8 capítulos, los cuales se detallan brevemente a continuación.

- *Capítulo 1: La Biología,* describe los conceptos biológicos elementales necesarios para comprender el trabajo realizado. Algunos temas específicos quedan fuera del alcance de este documento, debido a la complejidad de los mismos, aunque no dejan de ser importantes.
- *Capítulo 2: Descripción del Problema,* introduce el problema a resolver con una descripción general.
- *Capítulo 3: Metodología de Trabajo,* abarca la metodología de desarrollo seleccionada, detallando el modelo clásico y algunas consideraciones que se tuvieron en cuenta. Se describen las prácticas de software y herramientas empleadas.
- *Capítulo 4: Elicitación y Análisis de Requerimientos,* se describe brevemente la elicitación de requisitos y el análisis de los mismos, enunciando el estudio de las herramientas necesarias y búsqueda de datos mínimos.
- *Capítulo 5: Diseño,* se menciona la arquitectura global de **Remo** y algunas consideraciones de diseño.
- *Capítulo 6: Implementación,* abarca detalles de implementación, representación de datos biológicos, el framework empleado para distribuir el trabajo, métricas de código, etcétera.
- *Capítulo 7: Experimentación y Testing,* corresponde a las pruebas de **Remo** y las herramientas empleadas para tal fin.
- *Capítulo 8: Conclusiones del Trabajo,* se detalla la conclusión del presente trabajo, los resultados, aportes y futuros desarrollos.

A continuación de describe el presente trabajo.

Parte I

Preliminares

Capítulo 1

La Biología

> Essentially, all models are
> wrong, but some are useful.
>
> **George E. P. Box**

Dado a que el presente informe corresponde a un trabajo de Ciencias de la Computación, sólo se presentarán algunas definiciones y conceptos biológicos elementales que brindarán al lector la capacidad de comprender los temas abordados en los próximos capítulos, ya que se utilizará un lenguaje al cual el mismo puede no estar habituado. No se abordarán rigurosamente todos los conceptos biológicos involucrados, quedando como tarea del lector interesado.

1.1. Introducción a la Biología

La *Biología* es una ciencia que tiene como objeto de estudio a los seres vivos. Se ocupa tanto de la descripción de las características y los comportamientos de los organismos individuales como de las especies en su conjunto, así como de la reproducción de los seres vivos y de las interacciones entre ellos y el entorno. De este modo, trata de estudiar la estructura y dinámica funcional comunes a todos los seres vivos, con el fin de establecer las leyes generales que rigen la vida orgánica y los principios explicativos fundamentales de la misma[Curtis H., 2006].

1.1.1. Niveles de Organización Biológica

Los *niveles de organización biológica* son eslabones organizados de forma jerárquica, es decir, están organizados desde lo más simple hasta lo más

complejo. Estos niveles se utilizan para clasificar la materia, de acuerdo a su tamaño y/o cantidad. A continuación se detallan los diferentes niveles de organización biológica:

- **Nivel Atómico:** constituido por átomos. Aquellos elementos que forman la materia viva se conocen con el nombre de *bioelementos*. Dentro de los mismos se encuentran el carbono, el hidrógeno, el oxígeno, el nitrógeno, el azufre y el fósforo.
- **Nivel Molecular:** este nivel está formado por las moléculas que se originan al unirse dos o más átomos. Las moléculas que constituyen la materia viva se denominan *biomoléculas*. Éstas pueden ser inorgánicas como el agua, las sales minerales o los gases, u orgánicas como los glúcidos, los lípidos, las proteínas y los ácidos nucleicos.
- **Nivel Celular:** se incluyen las células. Una célula es la menor porción de materia viva que conforman los organismo, dentro de la cual tienen lugar todas las funciones vitales. Toda célula está formada por los niveles inferiores, el molecular y el atómico. La complejidad de este nivel es mucho mayor, ya que la célula es una unidad anatómica y funcional, esto significa que es la estructura más pequeña que podría sobrevivir por si misma.
- **Nivel Pluricelular:** refiere a la asociación de varias células que pueden llegar a constituir un organismo completo. Este nivel se puede subdividir en los siguientes subniveles: *tejidos*, *órganos*, *sistemas* y *aparatos*. En conjunto forman el individuo pluricelular.
- **Nivel Población:** incluye al conjunto de individuos de la misma especie que viven en un lugar concreto y en un tiempo determinado pudiendo relacionarse entre sí.
- **Nivel Ecosistema:** abarca las relaciones que se establecen entre las poblaciones que viven en un determinado lugar y el ambiente en el que habitan.

Como se puede observar, la Biología abarca muchas áreas de estudio. El presente trabajo se encuentra a nivel molecular, por lo cual a continuación se desarrollarán conceptos pertinentes a este nivel, el cual es conocido dentro de la jerga como *Biología Molecular*.

1.1.2. Moléculas Orgánicas y Macromoléculas

Moléculas Orgánicas

Las moléculas orgánicas pueden definirse como aquellas que contienen carbono y se encuentran en los seres vivos. Hay dos tipos de moléculas orgánicas básicas:

1. **Nucleótidos:** corresponden a pequeñas moléculas constituyentes de macromoléculas de mayor complejidad tales como los ácidos nucleicos. Un nucleótido está formado por un azúcar, un fosfato y una base nitrogenada como se observa en la figura 1.1. Además de su papel en la formación de ácido nucleicos, tiene una función independiente y vital para la vida celular. Cuando un nucleótido se modifica por la unión de dos fosfatos, se convierte en un transportador de energía, la cual es necesaria para que se produzcan numerosas reacciones químicas celulares.

   Las bases nitrogenadas se clasifican en:

   - *Bases nitrogenadas purínicas:* encontramos la Adenina(**A**) y Guanina(**G**).
   - *Bases nitrogenadas pirimidínicas:* encontramos la Timina(**T**), Citocina(**C**) y Uracilo(**U**).
   - *Bases nitrogenadas isoaloxacínicas:* encontramos la Flavina(**F**).

   Las bases **A**, **G** y la **C** forman parte del DNA y RNA. La base **T** sólo forma parte del DNA, y en contrapartida, la base **U** sólo forma parte del RNA.

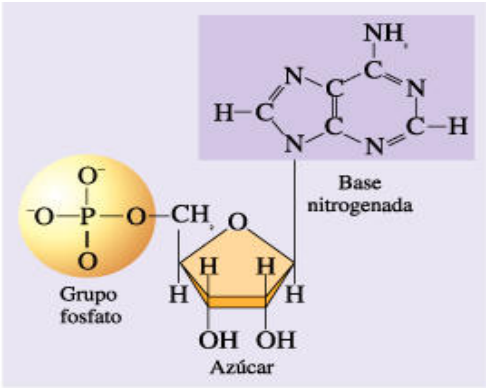


Figura 1.1: Estructura de un Nucleótido [4].

2. **Aminoácidos:** son las unidades estructurales que conforman las proteínas y sirven como materia prima para la formación de una gran variedad de compuestos nitrogenados con actividad fisiológica de vital importancia para la vida. Existen 20 aminoácidos que se exhiben en la figura 1.2 y que tienen codones específicos[1] en el código genético (figura 1.3). La unión de varios aminoácidos da lugar a cadenas llamadas polipéptidos o simplemente péptidos, que se denominan *proteínas* cuando la cadena polipeptídica supera los 50 aminoácidos.

Nombre	**Ab. 3 letras**	**Ab. 1 letra**
Alanine (Alanina)	Ala	A
Arginine (Arginina)	Arg	R
Asparagine (Asparagina)	Asn	N
Aspartic acid (Aspartato)	Asp	D
Cytesine (Cisteína)	Cys	C
Glutamic acid (Ácido glutámico)	Glu	E
Glutamine (Glutamina)	Gln	Q
Glycine (Glicina)	Gly	G
Histidine (Histidina)	His	H
Isoleucine (Isoleucina)	Ile	I
Leucine (Leucina)	Leu	L
Lysine (Lisina)	Lys	K
Methionine (Metionina)	Met	M
Phenylalanine (Fenilalanina)	Phe	F
Proline (Prolina)	Pro	P
Serine (Serina)	Ser	S
Threonine (Treonina)	Thr	T
Tryptophan (Triptófano)	Trp	W
Tyrosine (Tirosina)	Tyr	Y
Valine (Valina)	Val	V

Figura 1.2: Tabla de representación de Aminoácidos [2].

[1]Recordemos que un codón es un conjunto de tres nucleótidos.

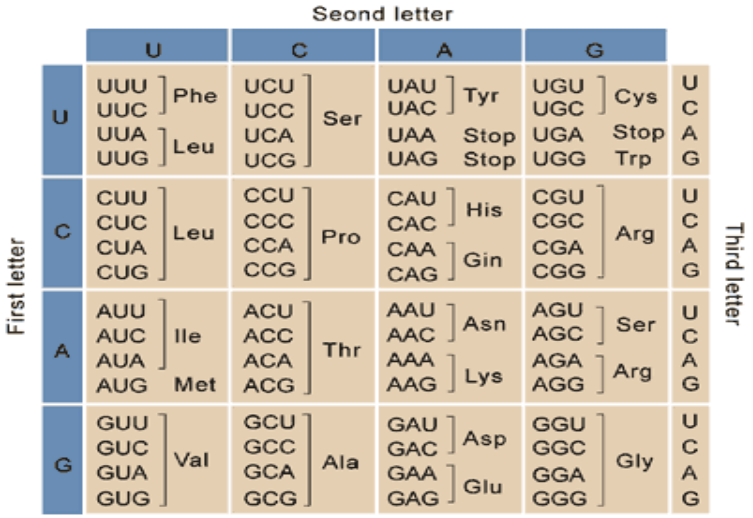


Figura 1.3: Código Genético [5].

Macromoléculas

Las macromoléculas son estructuras biológicas conformadas por un determinado número de moléculas orgánicas. Dentro de las macromoléculas podemos encontrar: *Glúsidos*, *Lípidos*, *Proteínas* y *Ácidos Nucleicos*. Las dos primeros escapan al presente trabajo, por lo cual se deja a criterio del lector su investigación.

- **Ácidos Nucleicos:** están conformados por secuencias de nucleótidos específicos, que son utilizados por todos los organismos para almacenar su información genética. La misma, está codificada mediante sucesivos codones, los cuales están conformados por tripletes de nucleótidos que codifican (traducen) a un determinado aminoácido. Existen dos tipos principales de ácidos nucleicos:

  1. **DNA (Ácido Desoxirribonucleico):** almacena y transmite la información genética para el desarrollo y el funcionamiento de los organismos vivos y de algunos virus, la cual se hereda de generación en generación. Está formado por una doble cadena de nucleótidos, donde las dos hebras están unidas a través de las bases complementarias (respetando una estructura helicoidal) según el modelo propuesto por Watson[2] y Crick[3] en 1953 (ver apéndice

[2]James Dewey Watson, biólogo estadounidense. Premio Nobel en Medicina.

[3]Francis Harry Compton Crick (1916-2004), físico, biólogo molecular y neurocientífico británico. Premio Nobel en Medicina.

A). El DNA contiene azúcares y fosfatos en su exterior y las bases en su interior. La doble hélice le proporciona al DNA una increíble estabilidad y permanencia. Las dos cadenas polinucleótidas se disponen en direcciones opuestas, es decir son antiparalelas, lo que significa que el extremo denominado *5'* de una cadena se ubica en oposición al extremo denominado *3'* de la otra cadena, tal como se exhibe en la figura 1.4.

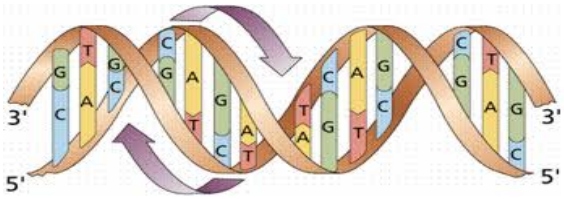


Figura 1.4: DNA: cadenas antiparalelas [6].

2. **RNA (Ácido Ribonucleico):** consiste en una hebra de cadena simple, la cual usualmente es trascripta a partir de una porción de DNA y se utiliza posteriormente en la célula para la síntesis de proteínas (sección 1.1.5).

Aunque el DNA y RNA son muy semejantes, desempeñan papeles biológicos muy diferentes. El DNA es el constituyente principal de los cromosomas de las células y es el portador del mensaje genético. En cambio, en RNA tiene como función transcribir el mensaje presente en el DNA y traducirlo a las proteínas. Además, el DNA se encuentra principalmente en el núcleo celular, mientras que el RNA se encuentra en el citoplasma, donde se produce la síntesis de proteínas. Otra diferencia entre estos dos tipos de ácidos nucleicos radica en las propias diferencias naturales existentes entre sus nucleótidos y residen en el tipo azúcar (ribosa en el caso del RNA y desoxirribosa en el caso del DNA) y las bases nitrogenadas características de cada uno. Otro rasgo a destacar, es que a pesar de que en la mayoría de las especies el DNA lleva la información genética, existen casos donde el RNA sirve como material genético (por ejemplo algunos virus tales como el TMV[4]). En la figura 1.5 se puede observar gráficamente la cadena doble característica del DNA (parte derecha de la figura) y la cadena simple del RNA (parte izquierda de la figura).

[4]Virus del mosaico del tabaco.

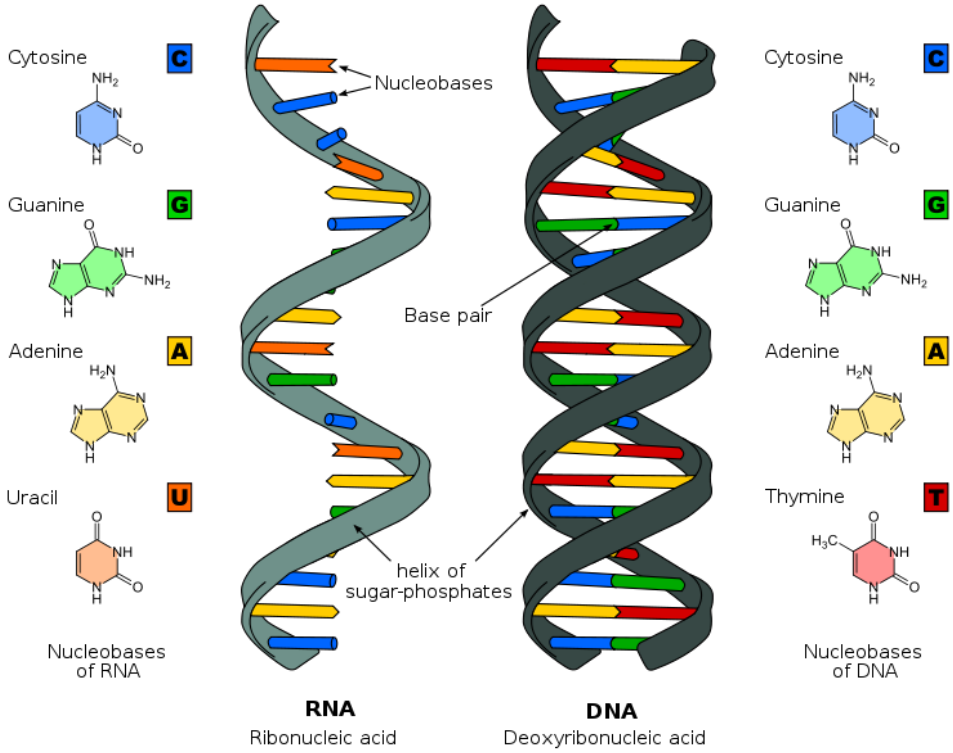


Figura 1.5: Representación gráfica del RNA y DNA [7].

- **Proteínas:** son las moléculas más abundantes y de máxima importancia biológica porque la ordenación de los distintos monómeros[5] en su estructura permite que sean utilizados como vehículo de la información celular. Están formadas por secuencias de aminoácidos específicas (se considera proteína a aquellas cadenas de aminoácidos enlazados cuyo peso molecular es superior a 6000 Daltons), que adoptan una conformación tridimensional determinada debido a las interacciones electrostáticas existentes entre los residuos de sus aminoácidos constituyentes. La estructura molecular de las proteínas (figura 1.6) tiene varios niveles de organización, al igual que los ácidos nucleicos. La *estructura primaria* consiste en una secuencia de aminoácido. A través de las interacciones entre los aminoácidos vecinos, una cadena polipeptídica se pliega y se enrosca para formar una *estructura secundaria*. Las estructuras secundarias interactúan y se pliegan adicionalmente para

[5]Un monómero es una molécula de pequeña masa muscular que unida a otros monómeros por medio de enlaces químicos forman macromoléculas denominadas *polímeros*.

forma lo que se conoce como *estructura terciaria*, que es la forma completa, tridimensional de la proteína. Finalmente, algunas proteínas consisten en dos o más cadenas polipeptídicas que se asocian para producir una *estructura cuaternaria.*

Las proteínas no son capaces de portar información genética y transmitirla a la descendencia, como ya se mencionó, este papel es realizado por los ácidos nucleicos, en particular por el DNA. Éste necesita de las proteínas para replicarse, y a su vez, las proteínas necesitan de la información que provee el DNA su síntesis.

Desde el punto de vista funcional, juegan un rol fundamental, dado que todo proceso biológico depende de la presencia y/o actividad de estas sustancias. Son proteínas casi todas las *enzimas* (catalizadores de reacciones químicas en los organismos), *hormonas* (reguladores de actividades celulares), *hemoglobina* (transporte de sangre), *anticuerpos* (encargados de acciones de defensa natural contra infecciones o agentes extraños), entre otras.

Cabe destacar que existe una relación colineal con los genes[6], es decir, hay una correspondencia directa entre la secuencia de nucleótidos del DNA y la secuencia de aminoácidos de una proteína. El concepto de colinealidad sugiere que el número de nucleótidos de un gen debería ser proporcional al número de aminoácidos de la proteína codificada por ese gen.

Las primeras investigaciones a cerca del material genético y su relación con DNA y proteínas mostraron que un cromosoma[7] estaba formado por DNA y proteínas en cantidades aproximadamente iguales. Por lo cual, ambos eran candidatos para desempeñar el papel de material genético. Las proteínas parecían ser la opción correcta dada su mayor complejidad química, aunque esta hipótesis era lógica fue errónea. El descubrimiento de la sustancia que puede transmitir características genéticas de una célula a otra resultó de los estudios con neumococos[8][Curtis H., 2006].

1.1.3. Estructura del DNA

- *Estructura Primaria*: consiste en una cadena de nucleótidos unidos entre sí por uniones fosfodiéster.

[6]Un gen se define como un factor hereditario que determina una característica.

[7]Se denomina cromosoma a cada uno de los pequeños cuerpos en forma de bastoncillos en que se organiza la cromatina del núcleo celular durante las divisiones celulares. La cromatina es el conjunto de DNA y proteínas que se encuentra en el núcleo de las células.

[8]Bacterias que causan la neumonía.

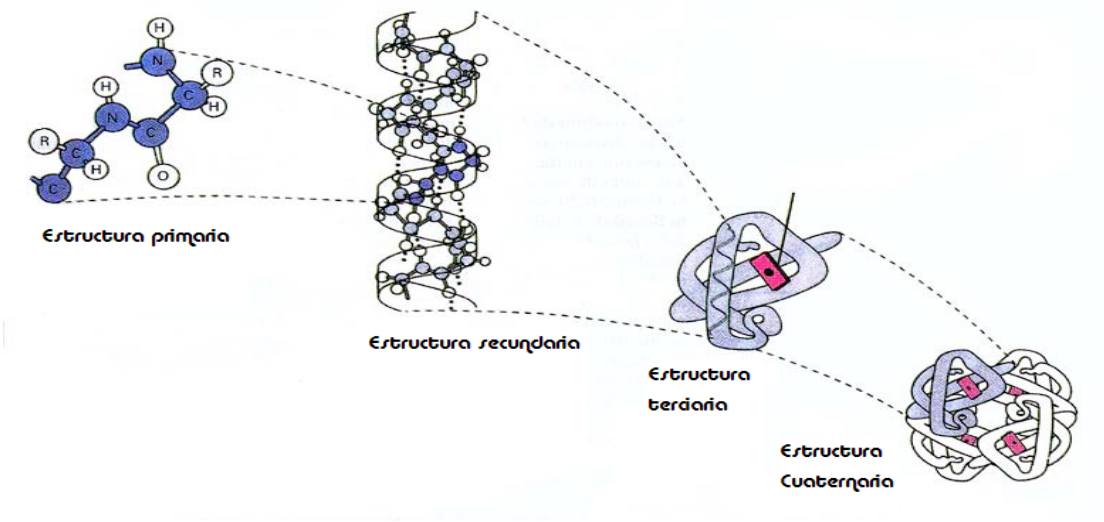


Figura 1.6: Estructura molecular de las proteínas [8].

- *Estructura Secundaria*: se relaciona con su configuración tridimensional (su estructura helicoidal básica). Esta estructura puede adquirir diversas configuraciones de acuerdo con su secuencia de bases y las condiciones en que se encuentran. Dentro de las mismas, encontramos la estructura **B-DNA**, la cual depende de que la molécula esté rodeada por abundante agua, además es la estructura más estable. Por otro lado, la estructura **A-DNA** la cual se presenta cuando la cantidad de agua es menor. Y por último, la estructura **Z-DNA** la cual se genera cuando el DNA es colocado en una solución de alta concentración salina. Las estructuras mencionadas se exhiben en la figura 1.7.

1.1.4. Replicación de DNA

Consiste en el proceso a través del cual una célula duplica su DNA antes de dividirse. Dicho proceso es muy complejo, tal es así, que un componente defectuoso puede producir síntomas patológicos graves.

La replicación es fundamental para el funcionamiento y mantenimiento de la célula y ocurre sólo una vez en cada generación celular, resultando esencial en la duplicación de los cromosomas.

Replicación semicorservativa

Básicamente la molécula de DNA se abre por el medio, separándose por bases apareadas (A=T; C=G) al nivel de los puentes de hidrógeno. A me-

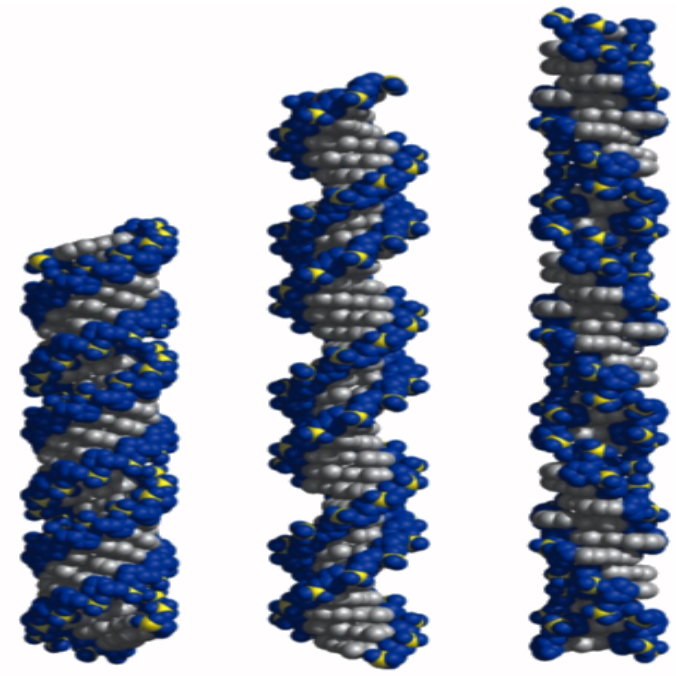


Figura 1.7: Estructuras secundarias del DNA. A-DNA, B-DNA y Z-DNA de izquierda a derecha respectivamente [9].

dida que las dos cadenas se separan, actúan como moldes o guías, es decir, cada una dirige la síntesis de una nueva cadena complementaria, utilizando las materias primas de la célula (figura 1.8). Este modelo de replicación se conoce como *replicación semiconservadora*, dado que se conserva la mitad de la molécula original en cada nueva cadena hija. Esto es, si hay una **T** presente en la cadena original, solamente puede ubicarse una **A** en el lugar correspondiente en la nueva cadena, de igual forma, una **G** sólo se apareará con una **C**, y así sucesivamente. Así, cada cadena forma una copia de su cadena complementaria original y se producen dos réplicas exactas de la molécula.

El proceso real de replicación en un poco más complejo que la replicación semiconservadora. La replicación del DNA requiere de diferentes enzimas, las cuales catalizan un paso particular del proceso.

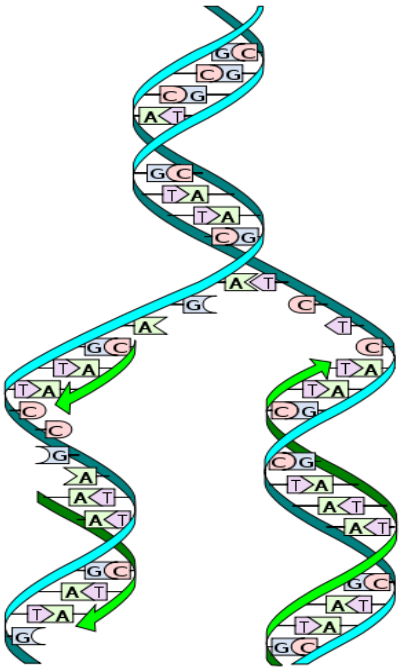


Figura 1.8: Replicación semiconservativa [10].

1.1.5. Del DNA a la Proteína

El Dogma Central de la Biología

Este dogma[9] proclamado por Francis Crick en 1956 indica que la información fluye en una única dirección, es decir, el DNA transfiere la información al RNA, el cual controla directamente la síntesis de proteínas (figura 1.9). Así, el genotipo[10] determina el fenotipo[11], dictando la composición de las proteínas. Sin embargo, estas últimas no alteran al genotipo, es decir, las proteínas no envían instrucciones de regreso.

Este dogma posee algunas excepciones, la principal de ellas es el proceso conocido como *transcripción inversa*, en el cual la información codificada por cierto virus que contienen RNA se transcribe al DNA por la acción de la enzima transcriptasa inversa.

El hecho de que la información fluya del DNA al RNA, y de éste a las

[9]Un dogma es una proposición que se asigna como cierta e innegable, no sujeta a prueba de veracidad.

[10]El genotipo es la totalidad de la información genética que posee un organismo en particular, en forma de DNA. También puede definirse como el conjunto de genes de un organismo.

[11]Se denomina fenotipo a la expresión del genotipo en función de un determinado ambiente. También puede definirse como el conjunto de rasgos de un organismo.

proteínas suministró cierta confirmación de la teoría darwinista de la evolución. Según esta teoría, la selección natural actúa sobre las variaciones heredables encontradas en el DNA[Curtis H., 2006].

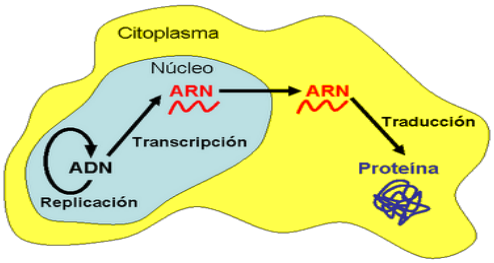


Figura 1.9: El Dogma Central [11].

Existen diferentes clases de RNA que desempeñan diferentes funciones como intermediarios entre los pasos que llevan del DNA a las proteínas. Éstos son:

- RNA mensajero($_m$**RNA):** se encuentra tanto en el núcleo como en el citoplasma de una célula. Su función es portar el código genético para las proteínas, es decir, transportan las instrucciones de codificación de las proteínas desde el DNA hasta el ribosoma. Después de unirse al ribosoma, una molécula de $_m$RNA especifica la secuencia de los aminoácidos de la proteína y proporciona un molde para unirlos.

- RNA de transferencia($_t$**RNA):** se encuentra sólo en el citoplasma. Sirve como enlace entre la secuencia codificante de nucleótidos de $_m$RNA y la secuencia de aminoácidos de una cadena polipeptídica. Cada $_t$RNA se une a un tipo particular de aminoácido y ayuda a incorporarlo a una cadena polipeptídica.

- RNA ribosómico($_r$**RNA):** se encuentra sólo en el citoplasma. Es un componente estructural y funcional del ribosoma (sitio de ensamblaje de proteínas).

- RNA interferente pequeño($_{si}$**RNA)** y microRNA($_{mi}$**RNA)**: son moléculas muy pequeñas presentes en las células eucariotas que inhiben específicamente la expresión de genes a nivel post-transcripcional, en respuesta a la presencia de RNA de doble hebra que proviene de la propia célula ($_{mi}$RNA) o del exterior de la misma (RNA pequeños de interferencia o $_{si}$RNA). Particularmente, los $_{mi}$RNA son pequeños RNA codificados en el DNA de las distintas especies que regulan la expresión

de una gran fracción de las proteínas. Para que esto ocurra, el debe encontrar nucleótidos complementarios (la unión se ve favorecida si estos nucleótidos no se ven comprometidos en la estructura secundario de la molécula) en el $_m$RNA blanco.

Transcripción

En este proceso las secuencias de DNA son copiadas a RNA mediante una enzima llamada RNA polimerasa. Dicha enzima sintetiza un RNA que mantiene la información de la secuencia del DNA. De esta manera, la transcripción del DNA también podría llamarse síntesis del RNA. En términos generales es la síntesis de una molécula de RNA a partir de un molde de DNA.

La transcripción es un proceso muy parecido a la replicación del DNA pero existe una diferencia fundamental relacionada con la longitud del molde empleado. Durante la replicación se copian todos los nucleótidos del molde de DNA pero durante la transcripción sólo pequeñas porciones de DNA se transcriben a RNA (casi siempre un único gen o a lo sumo algunos genes). Por lo cual, la transcripción es un proceso selectivo, es decir, se transcriben genes individuales sólo cuando se requieren sus productos. Esta selectividad implica la identificación de genes individuales y su transcripción en el momento y lugar adecuado.

Las moléculas de RNA son largas copias de secuencias de una cadena simple de DNA pero, a diferencia de las moléculas de DNA, se encuentran en su mayoría como moléculas de cadena única. Cada nueva molécula de RNA se copia de una de las dos cadenas de DNA (la cadena molde, la otra cadena restante de la doble hélice en general no se transcribe) según el mismo principio de apareamiento de bases que gobierna la replicación del DNA, como se puede observar en la figura 1.10.

La mayor parte de los transcriptos sufren un procedimiento posterior a la transcripción antes de dejar el núcleo e ingresar al citoplasma llamado *splicing*.

En conclusión, el RNA transcripto a partir del DNA es la copia activa de la información genética.

Traducción

En términos generales, el proceso consiste en tomar la información codificada en el DNA y transcripta en el RNA y traducirla a una cadena polipeptídica. En este proceso entran en juego además del $_m$RNA, el $_r$RNA y $_t$RNA,

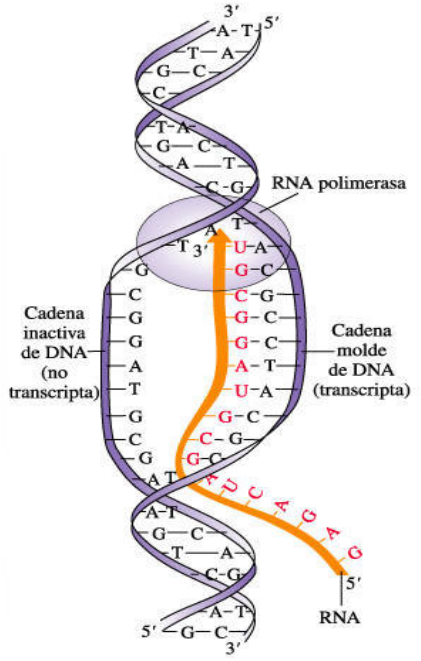


Figura 1.10: Representación esquemática de la transcripción del RNA [12].

los cuales se transcriben a partir de la cadena molde de DNA de la misma manera que el RNA.

Las moléculas de $_t$RNA son el diccionario por medio del cual se traduce el lenguaje de los ácidos nucleicos al lenguaje de las proteínas. Cada molécula de $_t$RNA tiene dos sitios de unión importantes como se observa en la Figura 1.11. Uno de ellos, conocido como el *anticodón*, se acopla al codón de la molécula de RNA. El otro, en el extremo 3' de la molécula de $_t$RNA, se acopla a un aminoácido particular. Así, el $_t$RNA permite que los aminoácidos se alineen de acuerdo con la secuencia de nucleótidos en el $_m$RNA.

Todas las moléculas de $_t$RNA presentan una estructura tridimensional (estructura secundaria) similar a la hoja de trébol con regiones de la molécula formando doble cadena como se puede observar en la figura 1.11. En el extremo 3' de la molécula se acopla al aminoácido. (siempre termina en una secuencia (5')-CCA-(3')).

La síntesis de proteínas se conoce también como **traducción**, dado que es la transferencia de información del lenguaje de los nucleótidos al de los aminoácidos. Este proceso ocurre en tres etapas: iniciación, elongación y terminación. Las mismas se dejan a criterio del lector.

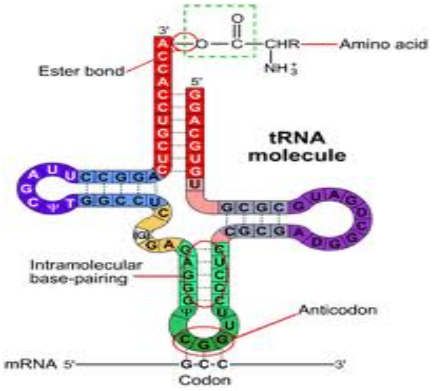


Figura 1.11: Modelo de la estructura de una molécula de $_t$RNA [13].

Excepciones al Dogma

Como se puede observar en la figura 1.12, existen dos excepciones al dogma conocidas como *transcriptasa inversa o transcripción inversa* y *replicación del RNA*.

La transcripción inversa[Curtis H., 2006] es el proceso que consiste en la transferencia de información desde el RNA a la molécula de DNA y tiene lugar en los retrovirus[12].

La replicación de RNA[Curtis H., 2006] es el proceso que consiste en sintetizar RNA a partir de RNA y tiene lugar en algunos virus con genoma RNA (ejemplo TMV).

Los mecanismos mencionados escapan a este trabajo, por lo cual se deja a criterio del lector profundizar en éstos conceptos.

Figura 1.12: Excepciones al Dogma Central [14].

[12]Un retrovirus es un virus RNA que se duplica en una célula huésped utilizando la transcriptasa inversa.

1.2. Estructura Secundaria del RNA

La estructura de una molécula de RNA se divide en tres niveles fundamentales de organización: primario, secundario y terciario. En particular, en este trabajo nos interesa el nivel secundario, por lo cual a continuación se abordará dicho nivel.

En términos generales, el plegamiento de una secuencia de RNA entre sus bases complementarias determina lo que se denomina *Estructura Secundaria de RNA*. Conocer la estructura secundaria es fundamental para comprender el funcionamiento de los distintos tipos de RNA y de la célula en general.

Más formalmente, dada una estructura primaria o secuencia de RNA de longitud n, la estructura secundaria es un conjunto S de pares (i, j) con $1 \leq i < j \leq n$ tal que para todo, $(i, j), (i', j') \in S$ se satisfacen las tres siguientes condiciones:

- $j - i > 3$
- $i = i' \Leftrightarrow j = j'$
- $i < i' \Rightarrow i < i' < j' < j \vee i < j < i' < j'$

1.2.1. Predicción de la Estructura Secundaria (*folding*)

El *folding* o predicción de la estructura secundaria corresponde a un conjunto de técnicas bioinformáticas cuyo objetivo es predecir la estructura secundaria de una secuencia de RNA o de una proteína, basándose sólo en el conocimiento de su estructura primaria (secuencia de nucleótidos o de aminoácidos, respectivamente). Se da el caso en que diferentes secuencias de RNA pueden tener la misma estructura secundaria, por lo que también interesa determinar para una estructura secundaria dada, las secuencias de RNA que conservan esa estructura.

La existencia de una asociación entre la secuencia de aminoácidos y secuencia de nucleótidos respecto de la estructura secundaria adoptada en la proteína y el RNA respectivamente es la hipótesis fundamental sobre la cual están basados todos los métodos de predicción de estructura secundaria.

En inglés, se suele denominar a la predicción de estructuras secundarias como *folding* e *inverse folding* al reconocimiento de posibles secuencias que conservan una estructura secundaria determinada. *Inverse folding* escapa a este trabajo, pero puede obtenerse mayor información consultando el proyecto **vac-o**[13].

[13] `vac-o.googecode.com`

Esencialmente existen 2 tipos de algoritmos para determinar o predecir la estructura secundaria de una secuencia de RNA.

- **Predicción por mfe (Minimal Free Energy)**: propuesto e implementado por Michael Zuker en 1981[Zuker and Stiegler, 1981], utiliza programación dinámica para encontrar la estructura secundaria que minimiza la energía libre.

- **Predicción comparativa**: utiliza diferentes métodos para comparar secuencias y estructuras con el fin de obtener una estructura por "consenso"[Gardner and Giegerich, 2004].

Si bien la "predicción comparativa" presenta un incremento en la fidelidad de los resultados obtenidos con respecto a la "predicción por mfe", este tipo de algoritmos requieren la existencia de un conjunto de secuencias relacionadas entre sí (homólogas) y no siempre es posible obtenerlas. En particular, para este trabajo interesa poder conocer la estructura secundaria de una sola secuencia, por lo que la "predicción comparativa" fue descartada.

Entre las implementaciones de la "predicción por mfe" se destacan **UNAFold**[N.R. Markham & M.Zuker., 2008] y **RNAfold**[Hofacker et al., 1994]. Ambas implementan el algoritmo propuesto por Michael Zuker con complejidad $\mathcal{O}(N^3)$ donde N es la longitud de la secuencia. En el capítulo 4 se detallarán brevemente las implementaciones mencionadas.

1.2.2. RNA Motifs

El término "Motifs"[D. K. Hendrix at el., 2006], también conocido como *elementos regulatorios o sitios de unión* representa los diversos bloques que conforman una estructura secundaria de RNA. Juegan un papel funcional en la formación de la estructura terciaria y pueden estar sujetos a restricciones evolutivas. Dentro de estos bloques se encuentran:

- *Single strand*: corresponde a un nucleótido no apareado (figura 1.13.a).

- *Helices-Dúplex*: más de la mitad de los nucleótidos están presentes en las *hélices* de doble cadena. Las regiones *dúplex* pueden formar largas interacciones, las cuales son cruciales para determinar y estabilizar el pliegue global de una molécula de RNA. Los dúplex de RNA no son uniformes, aunque su variabilidad es menor que en el DNA. Estas variaciones dependen de la secuencia y el contexto estructural de la hélice con respecto al contexto global de la estructura tridimensional. (figuras 1.13.b y 1.13.c).

- *Bulges:* se forman cuando un tracto de la doble hélice se separa por uno o más nucleótidos no apareados. (figuras 1.13.d y 1.13.e).

- *Hairpins*: corresponde al elemento más común de una estructura secundaria y varían de 2 a 14 nucleótidos. Se forma cuando las secuencias nucleótidos de una misma cadena son estructuras complementarias invertidas. Un *hairpin* consiste de una región de bases apareadas (*tallo*) y, a veces, incluye bases "no apareadas" intermedias (*bucle*). Cuando las estructuras complementarias son contiguas, el hairpin presenta un tallo pero no un bucle. Los pequeño hairpin loop contienen un alto grado de estructura, mientras que los más largos (aquellos que contienen más de 7 u 8 nucleótidos no apareados) generalmente no están bien estructurados y termodinámicamente son menos estables. (figura 1.13.f)

- *Internal Loop*: se forman cuando los dos tractos de la doble hélice se separan por uno o más nucleótidos no apareados. Los *internal loop* puede ser simétricos o asimétricos, dependiendo del número de bases que contenga cada cadena. (figuras 1.13.g y 1.13.h y 1.13.i)

- *Multi-Loop*: se forma por la intersección de dos o más dobles hélices. Estas dobles hélices se separan por secuencias de cero o más nucleótidos no apareados. (figura 1.13.j)

- *External Loop*: motif inicial de una estructura (figura 1.13.k)

Las moléculas de RNA pueden contener numerosos motifs, lo que les permite plegarse y formar estructuras muy complejas tal como se puede observar en la figura 1.14.

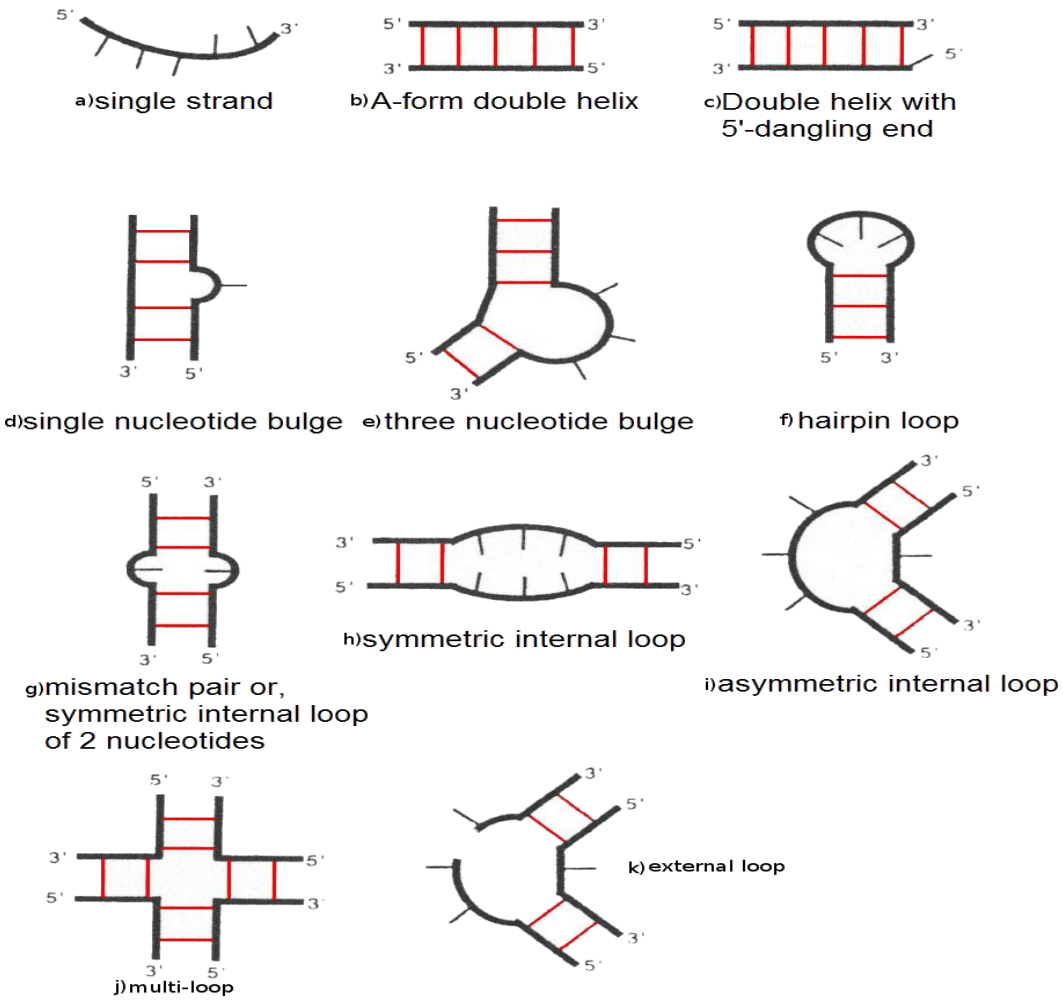


Figura 1.13: RNA Motifs [2].

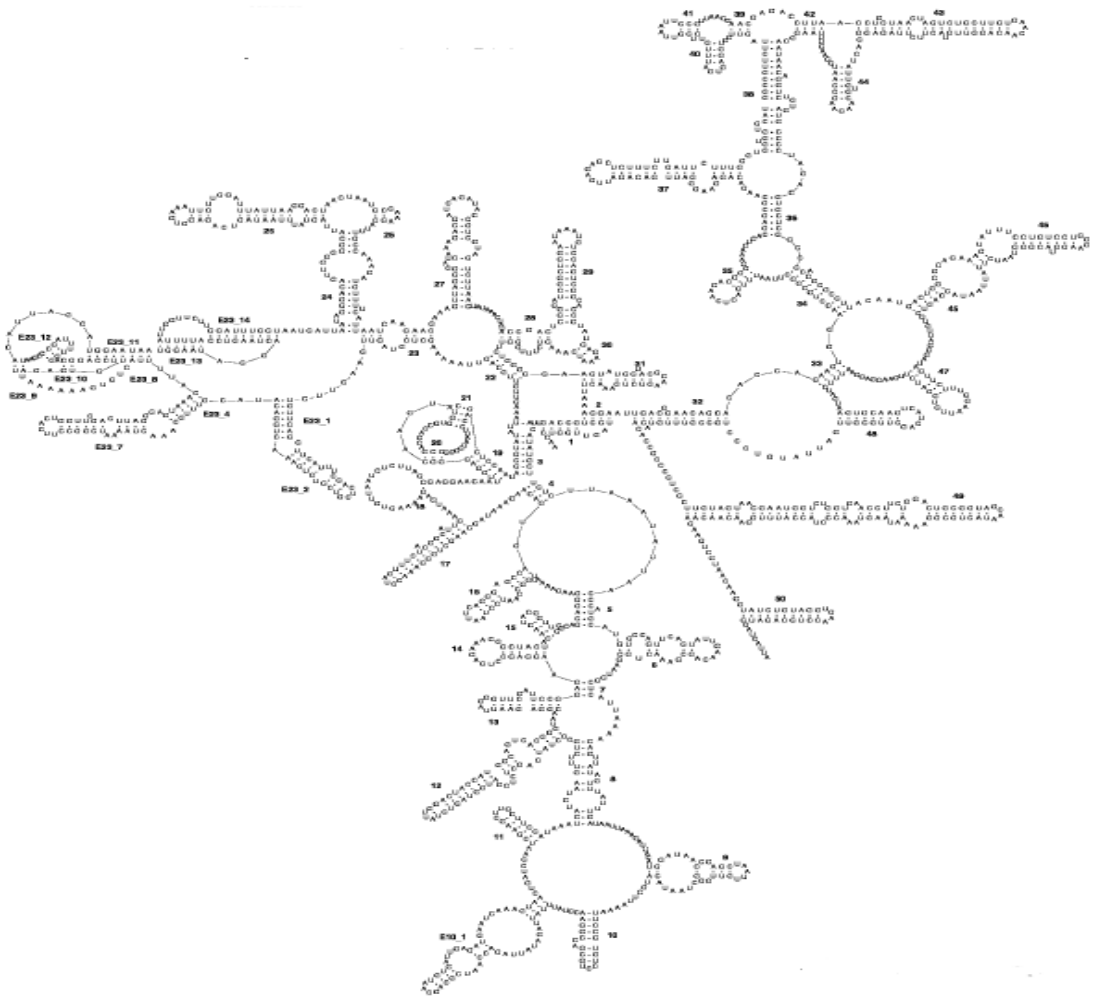


Figura 1.14: Estructura Secundaria Compleja [15].

1.3. Hibridación

El fenómeno de apareamiento de bases (bases complementarias: A-T y C-G) para formar una doble hélice se llama *hibridación*, dado que puede utilizarse para formar DNA híbrido compuesto por cadenas de diferentes orígenes tal como se exhibe en la figura 1.15.

Este proceso ocurre en solución y requiere de ciertos factores, tales como, un determinado pH, determinada temperatura, cierta concentración de sales, etcétera. Dependiendo de estas condiciones, es posible que en los híbridos existan malos apareamientos (A-C, por ejemplo).

El origen de cada una de las hebras es irrelevante, sólo importa la secuencia (que un número significativo de bases sea complementario entre las

dos).

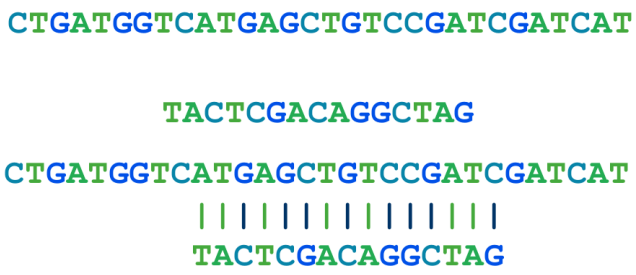


Figura 1.15: Hibridación [2].

1.4. Energía Libre y Estabilidad

El concepto de "Energía Libre" hace referencia al total de energía contenida en un sistema que está disponible para realizar trabajo, como por ejemplo una molécula de RNA con una estructura secundaria, la cual le permite al mismo realizar trabajo. Una molécula de RNA puede estar dotada de una cierta cantidad de energía libre mediante interacciones eléctricas no neutralizadas en su estructura primaria, por lo tanto, esta energía se utilizará para plegar la misma hacia una estructura secundaria más estable. De este modo, un RNA con una estructura secundaria estable, implica una molécula en la cual sus interacciones eléctricas entre nucleótidos se hallan completamente neutralizadas.

Este concepto es muy complejo e involucra nociones básicas de termodinámica, por lo cual su abordaje escapa al presente trabajo de ciencias de la computación, por lo cual puede indagarse más al respecto en [Solomon, 1998].

La termodinámica de las interacciones RNA-RNA
[U. Muckstein & H. Tafer., 2006] puede ser entendida como la suma de dos contribuciones de energía:

$$\Delta G = \Delta G_u + \Delta G_h$$

donde:

- ΔG: energía libre.
- ΔG_u: energía necesaria para exponer el sitio de unión para una interacción.
- ΔG_h: energía obtenida a partir de hibridación en el sitio de unión.

Parte II

Remo

Capítulo 2

Descripción del Problema

> Rem tene, verba sequentur (Si dominas el tema, las palabras vendrán solas).
>
> **Catón el Viejo**

Se dice que todo problema nace a raíz de una dificultad, ésta se origina a partir de una necesidad en la cual aparecen dificultades sin resolver. De ahí, la necesidad de hacer un planteamiento adecuado del problema a fin de no producir efectos secundarios del problema con la realidad que se investiga. Por tanto, el planteamiento y descripción establece la dirección del estudio para lograr ciertos objetivos.

2.1. Descripción General del Problema

Para comprender la importancia y la hipótesis del presente trabajo se deben tener en cuenta diversos hechos referidos al código genético y los microRNA.

El código genético es un código organizado en *tripletes* o *codones*, donde tres nucleótidos codifican cada aminoácido de una proteína. Si los codones constaran de una sola base, sólo habría capacidad para especificar cuatro aminoácidos ya que sólo hay cuatro bases diferentes en el DNA (C, G, A y T). Un código de dobletes podría tener 4x4 = 16 palabras de dos bases, o codones, lo cual es insuficiente para especificar sin ambigüedades veinte aminoácidos distintos. Por lo que, un código de tripletes da lugar a 4x4x4 = 64 codones, lo cual es más que suficiente para especificar los veinte aminoácidos[Tamarin, 1996]. Tres de estos 64 codones son codones de terminación, que especifican la finalización de la traducción. Así, 61 codones

llamados codones codificantes codifican los aminoácidos. Dado que hay 61 codones codificantes y sólo 20 aminoácidos distintos, el código contiene más información que la necesaria para especificar los aminoácidos, y se dice que es un *código degenerado*, lo que implica que al menos uno de los tres nucleótidos (en general el último) puede ser distinto y sin embargo codificar para el mismo aminoácido[Pierce, 2009]. Por otro lado, los codones que especifican un mismo aminoácido se consideran *codones sinónimos*. Durante muchos años se supuso que el código genético era universal, lo que significa que cada codón especifica el mismo aminoácido en todos los organismos. En la actualidad se conoce que el código genético es casi, pero no completamente universal. En cada especie, se ha seleccionado una proporción de uso de codones que guarda relación con la proporción de RNA de transferencia correspondiente de manera de optimizar la síntesis proteica. Para otros patógenos intracelulares como los virus, existe una divergencia entre el uso de codones sinónimos[Gareth M. Jenkins and Edward C. Holmes., 2003] utilizado por el virus y el huésped correspondiente. El origen de esa divergencia no esta suficientemente claro.

Por otro lado, además de la molécula clásica de RNA (una sola hebra), existen diversos RNA pequeños, dentro de los cuales se destacan el RNA interferente pequeño ($_{si}$RNA) y el microRNA ($_{mi}$RNA). Los $_{si}$RNA se dejan a criterio del lector su investigación ya que excede el presente trabajo. Un $_{mi}$RNA [YEUNG, 2005][Mahajan, 2008] es un RNA monocatenario, de una longitud de entre 21 y 25 nucleótidos, que tiene la capacidad de regular la expresión de otros genes mediante diversos procesos e inhibe la traducción del $_{m}$RNA. Los $_{mi}$RNA se encuentran codificados en el genoma y juegan un papel importante en la regulación de la expresión proteica, procesos cancerosos e infecciones virales.

Actualmente no hay evidencia respecto a si existe diferencia en cuanto a reconocimiento de $_{mi}$RNA en los $_{m}$RNA originales de un virus y los $_{m}$RNA humanizados u optimizados (entendiéndose por este último término, la cadena de $_{m}$RNA del virus en la cual el uso de codones depende de la proporción del huésped). Para tal fin, es necesario determinar la capacidad de hibridación[1] de los $_{mi}$RNA en la secuencia viral tal como existe en la naturaleza y en la secuencia humanizada, para poder establecer conclusiones al respecto. En otras palabras, determinar si el uso de codones divergente con respecto al huésped podría ser una consecuencia de la presión evolutiva generada por los $_{mi}$RNA. Si esto es así, los $_{mi}$RNA deberían tener menor capacidad de unirse

[1]Proceso por el cual se combinan dos cadenas de ácidos nucleicos antiparalelas con secuencias de bases complementarias en una única molécula de doble cadena, que toma la estructura de doble hélice, donde las bases nitrogenadas quedan ocultas en el interior.

al RNA viral que al genoma viral humanizado.

En principio esta respuesta es importante desde el punto de vista biológico, y daría una herramienta importante para desarrollos posteriores, por ejemplo actualmente se está estudiando el uso de virus modificados para combatir cánceres, el programa podría predecir qué virus sería menos afectado por los $_{mi}$RNA en células cancerosas y tenerlo en cuenta en el diseño del virus modificados.

Básicamente, el sistema a desarrollar comprenderá las siguientes característica:

- Abarcar en su totalidad los requerimientos del problema.
- Construir un sistema que puede ser extendido en otros proyectos, brindando un diseño flexible.
- Que proponga un buen uso de las prácticas de diseño para su mejor desempeño.
- Que posea documentación clara y precisa.
- Lograr un código fuente bien escrito y estructurado respetando las buenas prácticas de programación.

2.2. Objetivos

El principal objetivo de este trabajo es contrastar la teoría postulada y encomendada por el Dr. Roberto Daniel Rabinovich, la cual involucra principalmente la molécula de RNA como ya se puede inferir.

En este trabajo se estudiará si la divergencia en el uso de codones sinónimos entre virus y huésped contribuye a disminuir la interferencia de los $_{mi}$RNA en la expresión de los RNA mensajeros de origen viral. De esa manera se contribuirá a comprender mejor la relación virus-huésped y la evolución viral. Estos estudios tienen también una importancia potencial en la comprensión de la patogenia viral y en el desarrollo de herramientas terapéuticas como la terapia génica.

Para poder llevar a cabo el objetivo planteado anteriormente, será necesario elaborar una herramienta de software que realice entre otras cosas, múltiples comparaciones masivas de secuencias de $_{mi}$RNA y $_{m}$RNA. Estas comparaciones pueden llevar un tiempo de cómputo demasiado alto, puesto que el número de combinaciones que surgen del cruce entre $_{mi}$RNA y $_{m}$RNA puede ser un número lo suficientemente grande, esto se debe a que el tamaño

de las secuencias de $_m$RNA pueden ser muy grandes. Además será necesario realizar pruebas en masa para poder obtener resultados confiables.

Capítulo 3

Metodología de Trabajo

> Nada se edifica sobre la piedra,
> todo sobre la arena, pero
> nuestro deber es edificar como si
> fuera piedra la arena.
>
> **Jorge Luis Borges:**
> *Fragmentos de un evangelio apócrifo*

Los sistemas de software requieren un tiempo y esfuerzo considerable para su desarrollo. Durante este tiempo, desde que se detecta la necesidad de construir un sistema de software, se identifican diferentes etapas que en conjunto se denominan *el ciclo de vida del software*. En cada caso, en función de las características del proyecto, se configurará el ciclo de vida de forma diferente.

En particular, **Remo** se abordó empleando un enfoque metodológico que ordena rigurosamente las diversas etapas de desarrollo denominado *Modelo de Desarrollo en Cascada*. A continuación se detallan algunos aspectos esenciales de dicho modelo y algunas consideraciones.

3.1. Modelo en Cascada

El *Modelo en Cascada Clásico* es un enfoque metodológico secuencial de desarrollo en el que los pasos de desarrollo son vistos hacia abajo (como una "cascada de agua"), de tal forma que el inicio de cada etapa debe esperar a la finalización de la etapa anterior. La primera descripción formal de este método fue desarrollada por Winston Royce W. (1929–1995) en 1970, aunque Royce no utiliza el término "cascada" en su artículo[Winston Royce, 1970].

3.1.1. Etapas del Modelo

A continuación se detallan las diferentes etapas del modelo empleado.

Elicitación de Requerimientos

Corresponde al estableciendo de los requisitos de todos los elementos que formarán parte del producto. En esta fase se analizan las necesidades de los usuarios finales del software para determinar qué objetivos debe cubrir. De esta fase surge un documento llamado *"SRS"* (Documento de Especificación de Requisitos), que contiene la especificación completa de lo que debe hacer el sistema sin entrar en detalles internos.

Análisis de Requerimientos

Consiste en la investigación respecto al dominio de trabajo con el fin de comprender los requisitos de la etapa anterior. Se espera un conjunto de bibliografía y trabajos relacionados, con una breve descripción detallada de ellos (que se utilizará durante el diseño).

Diseño

El diseño del software se enfoca en cuatro atributos distintos del programa: la estructura de los datos, la arquitectura del software, el detalle procedimental y la caracterización de la interfaz. El proceso de diseño traduce los requisitos en una representación del software con la calidad requerida antes de que comience la codificación.

Como resultado de esta etapa surge el *"SDD"* (Documento de Diseño del Software), que contiene la descripción de la estructura relacional global del sistema y la especificación de lo que debe hacer cada una de sus partes, así como la manera en que se combinan unas con otras.

Es conveniente distinguir entre diseño de alto nivel o arquitectónico y diseño detallado o de bajo nivel.

Codificación

Es la fase en donde se implementa el código fuente, haciendo uso de prototipos así como de pruebas y ensayos para corregir errores.

Prueba

La prueba se centra en la lógica interna del software, y en las funciones externas, realizando pruebas que aseguren que la entrada definida produce

los resultados que realmente se requieren.

3.1.2. Ventajas y Desventajas

Luego de estudiar el modelo, se pueden detallar algunas ventajas y desventajas del mismo.

- **Ventajas**
  - La planificación es sencilla.
  - Se tiene todo organizado y no se mezclan las fases.
  - Las fases son conocidas por los desarrolladores ya que sigue los pasos intuitivos necesarios a la hora de desarrollar el software.
  - Las etapas y actividades están bien marcadas para facilitar la comprensión y claridad de los objetivos del proyecto.
- **Desventajas**
  - Iteraciones costosas.
  - El cliente debe tener paciencia. Hasta llegar a las etapas finales del proyecto, no estará disponible una versión operativa del programa.
  - Un error importante no detectado hasta que el programa este funcionando puede ser desastroso.

3.1.3. Consideraciones del Modelo

Para implementar el presente modelo en cascada se tuvieron en cuenta ciertas consideraciones con respecto al contexto en que se desarrollo.

Por un lado, las etapas mencionadas en la subsección 3.1.1 fueron llevadas a cabo por una misma persona. En este sentido, se uso el *modelo en cascada con "solapamiento"*, también conocido como *Sashimo*[McConnell, 1996], que permite comenzar una etapa sin haber terminado por completo la etapa anterior.

Por otro lado, el rol de "cliente" en el proceso de desarrollo fue llevado a cabo por miembros de **FuDePAN** que garantizaron el conocimiento del dominio del problema, no sólo permitiendo realizar la "Especificación de Requerimientos" sino que también contaban con los conocimientos técnicos de diseño y programación orientada a objetos como para supervisar el desarrollo. En este sentido, se tomaron algunos elementos de la variante al modelo de cascada "puro" que se conoce como *Staged Delivery*[McConnell, 1996] o "Implementación Incremental", que permitió realizar revisiones periódicas

con miembros de **FuDePAN** durante las diversas etapas. En la figura 3.1 se exhibe la estructura clásica del modelo, y las variantes mencionadas.

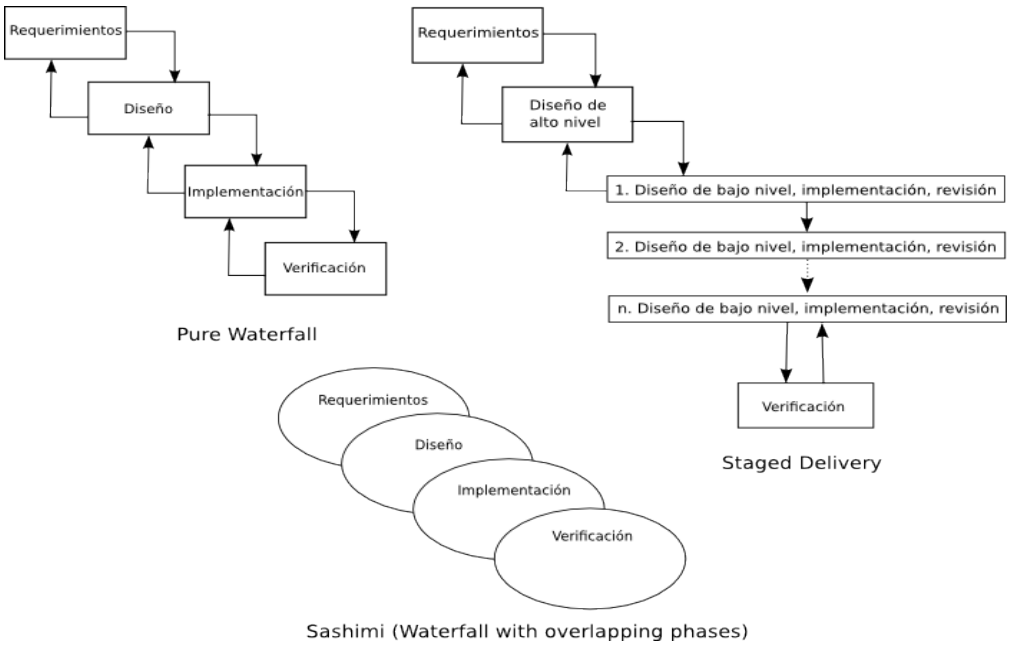


Figura 3.1: Modelos en Cascada [17].

3.2. Gestión de la Configuración

Para desarrollar este proyecto fue necesario utilizar un manejador de versiones. Para ello se manipuló un repositorio *Mercurial* (Hg) alojado en GoogleCode[1] con el fin de poder seguir la pista a todos los archivos que componen el proyecto.

3.3. Prácticas de Software

Remo se desarrolló empleando buenas prácticas de programación las cuales proporcionan múltiples ventajas al desarrollo. Éstas prácticas permiten mantener código limpio, de fácil lectura (sinónimo de fácil mantenimiento), reutilización e integración homogénea, entre otros factores. A continuación se describen algunos aspectos orientados a **Remo** :

[1] `https://code.google.com/intl/es/`

- **Elicitación y análisis:** aproximadamente el 30 por ciento del tiempo total fue dedicado a éstas etapas, dado que fue necesario investigar el dominio del problema y diversas herramientas a emplear.
- **Diseño:** se intentó obtener un diseño que respete, entre otras cosas, dos principios básicos: Simplicidad y Ocultamiento de la información. Se utilizaron Patrones de Diseño[Gamma et al., 2005] y UML [G. Booch and Jacobson, 2005].
- **Construcción de código:** aproximadamente el 30 por ciento del tiempo fue dedicado a la construcción del código. Cada vez que se implementó un nuevo componente se chequeó la integración del mismo con todo el proyecto. Cuando superaba la prueba, el código fue trasladado a la rama *default* del repositorio.
- **Revisiones:** la mayoría de las operaciones *commits* realizadas (incluyendo fuente, diagramas, documentos de responsabilidades, entre otros) fueron revisadas por al menos dos personas. No sólo se resaltaron errores, sino también cuestiones en cuando a calidad y eficiencia. Cada vez que se encontró un error o sugerencia, una nueva revisión fue creada conteniendo la solución.
- **Seguimiento de issues:** los bugs y defectos del proyecto fueron reportados como *issues*. Luego, por cada issue, se creo una nueva revisión (rama) conteniendo la solución.

3.4. Herramientas de Desarrollo Empleadas

Tanto el sistema operativo como todas las herramientas que se usaron para el desarrollo del presente proyecto son libres. **Remo** tiene la licencia libre **GPL** (General Public Licence), una copia de la misma pude ser encontrada en `http://www.gnu.org/licenses/gpl-3.0.txt`.

3.4.1. Lenguaje de Implementación

Como lenguaje de implementación se utilizó *C++*[Stroustrup, 1997].

C++ es un lenguaje de programación diseñado en el año 1979 por *Bjarne Stroustrup*[2] en los laboratorios Bell[3]. En un principio fue una extensión del

[2]Científico de la computación y catedrático de Ciencias de la Computación en la Universidad A&M de Texas.

[3]Centros de investigación científica y tecnológica ubicados en más de diez países y que pertenecen a la empresa estadounidense Lucent Technologies.

conocido lenguaje de programación C que fue denominado *"C con clases"*.

El nombre C++ fue propuesto por Rick Mascitti en el año 1983, cuando el lenguaje fue utilizado por primera vez fuera de un laboratorio científico.

En la actualidad, es un lenguaje de tipado estático, multiparadigma, versátil, potente y de propósito general. Su éxito entre los programadores profesionales le ha llevado a ocupar el primer puesto como herramienta de desarrollo de aplicaciones. C++ mantiene las ventajas de C en cuanto a riqueza de operadores y expresiones, flexibilidad, concisión y eficiencia. Además, ha eliminado algunas de las dificultades y limitaciones del C original.

Este lenguaje fue elegido porque permite el uso de técnicas orientadas a objetos y produce eficiente código assembly. Además ofrece una gran cantidad de librerías para facilitar la resolución de determinados problemas, permitiendo concentrarse en los mismos y no en implementar tipos de datos abstractos ya conocidos.

3.4.2. GNU

- **GCC (GNU Compiler Collection)**[4]**:** conjunto de compiladores creados por el proyecto GNU. GCC es software libre y se distribuye bajo licencia GPL. Estos compiladores se consideran estándar para sistemas operativos derivados de GNU.

- **GDB (The GNU Project Debbuger)**[5]: es un depurador portable que se puede utilizar en varias plataformas Unix y funciona para varios lenguajes de programación como C y C++ entre otros. GDB fue escrito por Richard Stallman[6] en 1988, es software libre y se distribuye bajo licencia GPL.

3.4.3. Sistema de Construcción

Para la compilación e inclusión de librería externas, se usó *fudepan-build*[7], un proyecto de **FuDePAN** que utiliza scons[8].

[4] `http://gcc.gnu.org/`

[5] `www.gnu.org/software/gdb/`

[6] Nacido en Manhattan, Nueva York, 16 de marzo de 1953. Fundador del movimiento por el software libre en el mundo.

[7] `https://code.google.com/p/fudepan-build/`

[8] `http://www.scons.org/`

3.4.4. LaTeX

Esta herramienta fue utilizada para llevar adelante el presente documento y todo la documentación de **Remo**. Básicamente LaTeX[Lamport, 1994] es un paquete de macros para TEX[9], originalmente escrito por Leslie Lamport para proporcionar un sistema de procesamiento de documentos más simple de uso que TEX, pero con toda su potencia.

El formato de los archivos es mucho más estable que en otros procesadores de texto, cualquier cambio es realizado localmente y no repercute en efectos colaterales, existen implementaciones para distintas plataformas y en todas el resultado es exactamente el mismo (si se tienen los mismos estilos y tipos).

3.4.5. Edición

- *Gedit*[10]*:* para la edición de texto plano.
- *Sublime*[11]*:* para la edición de texto plano. Permite simular un excelente IDE de desarrollo.

3.4.6. Gráficos

- *Gimp*[12]*:* editor de imágenes.
- *Bouml*[13]*:* editor de diagramas UML.
- *Dia*[14]*:* editor de diagramas de propósito general.
- *Diagramas online*[15]*:* editor de diagramas de secuencia.

3.4.7. Análisis Estático de Código

- *Astyle*[16]: indentador de código fuente, formateador y embellecedor para los lenguajes C, C++, C# y Java.

[9] Sistema de composición de textos de alta calidad creado por Donald E. Knuth a finales de la década de los 70, dirigido particularmente a aquellos textos que contienen una gran cantidad de expresiones matemáticas.

[10] `http://projects.gnome.org/gedit/`

[11] `www.sublimetext.com`

[12] `www.gimp.org`

[13] `http://www.bouml.fr/`

[14] `http://live.gnome.org/Dia`

[15] `www.websequencediagrams.com`

[16] `http://astyle.sourceforge.net`

- *cppcheck*[17]*:* permite detectar los tipos de errores que los compiladores normalmente no detectan. El objetivo es detectar sólo los errores reales en el código (es decir, ningún falsos positivos).
- *Cloc*[18]*:* es un programa para contar la cantidad de lineas del sistema.
- *CCCC*[19]*:* herramienta que analiza los fuentes y genera reportes en varias métricas asociadas al código.
- *GCov*[20]*:* es una herramienta GNU para realizar pruebas de cobertura sobre código fuente. Se utiliza en conjunto con ***gcc*** para determinar el número de veces que cada línea de un programa se ejecuta durante una ejecución. Esto hace que sea posible encontrar áreas del código que no se utilizan, o que no se ejecutan en los casos de pruebas planteados.

3.4.8. Análisis Dinámico de Código

- Valgrind[21]: permite el análisis dinámico de código. Puede detectar automáticamente *memory leaks* y errores de contexto.

3.4.9. Automatización de Pruebas

Se empleó gtest[22] y gmock[23] para la elaboración y ejecución de pruebas.

[17] `http://sourceforge.net/apps/mediawiki/cppcheck`
[18] `http://cloc.sourceforge.net/`
[19] `http://cccc.sourceforge.net/`
[20] http://gcc.gnu.org/onlinedocs/gcc/Gcov.html
[21] `http://valgrind.org`
[22] `http://googletest.googlecode.com`
[23] `http://googlemock.googlecode.`

Capítulo 4

Elicitación y Análisis de Requerimientos

> No hay enigmas, si un problema puede plantearse, es que puede resolverse.
>
> **Ludwig Wittgenstein**

El presente capítulo describe de forma general la elicitación de requerimientos, y de manera más detallada el análisis de los mismos.

4.1. Elicitación de Requerimientos

La IEEE[1] define los requerimientos como:

> "Una condición de la capacidad que necesita un usuario para resolver un problema o lograr un objetivo. Una condición o una capacidad que debe cumplir un sistema para satisfacer un contrato, un estándar, una especificación, o cualquier otro documento oficial establecido."

Los requerimientos de software del presente trabajo fueron provistos por miembros de **FuDePAN** y quedaron documentados en la "Especificación de Requerimientos de Software" (SRS) que se encuentra en el repositorio del proyecto. Dicho documento describe qué hará el software propuesto pero no cómo lo hará.

[1] `www.ieee.org`

4.1.1. Requerimientos Funcionales

Los requerimientos funcionales de **Remo** pueden resumirse de la siguiente manera:

- **Entrada:** dos archivos en formato FASTA (ver Apéndice C). El primero de ellos conteniendo un conjunto de secuencias de $_{mi}$RNA, y el segundo, compuesto por múltiples secuencias de $_{m}$RNA.
- **Objetivo:** satisfaciendo las restricciones impuestas, determinar la relación de divergencia y contrastar la hipótesis plasmada anteriormente (luego de un extenso análisis biológico de los resultados). En este ámbito uno de los requisitos más importantes fue calcular la estructura secundaria de las secuencias de RNA (folding/hibridación).
- **Salida:** archivos en formato CSV, que permitan poder establecer conclusiones al respecto.

Uno de los principales requerimientos funcionales que cabe destacar tiene relación con la comparación de estructuras secundarias. Para ello, por cada RNA de entrada, es necesario calcular la estructura secundaria. A partir de la misma, realizar un reconocimiento de los diversos *motif* (ver sección 1.2.2) que la componen para generar como salida un archivo comparativo. Se tubo en cuenta el tamaño de los stacks, entendiéndose como tal, las uniones entre nucleótidos presentes entre dos motif adyacentes cualesquiera.

Las estructuras pueden ser muy grandes y complejas, tales como la que se exhibió en la figura 1.13.

Desde el punto de vista del diseño, **Remo** debía cumplir con los principios de diseño conocido por el acrónico SOLID[2] [Martin, 2000]. Estos principios son los siguientes y se retomaran el la sección 5:

- **S**ingle responsibility principle (SRP)
- **O**pen/closed principle (OCP)
- **L**iskov substitution principle (LSP)
- **I**nterface segregation principle (ISP)
- **D**ependency inversion principle (DIP)
- Law of Demeter (LoD)

[2]Acrónimo nemotécnico introducido por Robert C. Martin en la década del 2000, que representa cinco principios básicos de la programación y diseño orientado a objetos

Para observar detalladamente los requerimientos funcionales puede consultarse el SRS[3] correspondiente a **Remo** .

4.2. Análisis de los Requerimientos

El objetivo básico del análisis de problemas es obtener una comprensión clara de las necesidades de los clientes, en otros término, qué es exactamente lo que se desea del software, y cuales son las limitaciones de la solución. El principio elemental empleado en el análisis es el mismo que en cualquier tarea compleja: *divide y vencerás*. Esto es la "partición" del problema en subproblemas y a continuación, tratar de entender cada subproblema y su relación con otros subproblemas en un esfuerzo por entender el problema total.

Una vez identificados los diversos requerimientos, se realizó un extenso análisis referente a las herramientas a emplear y al dominio del problema en general. A continuación se detalla dicho análisis.

4.2.1. Búsqueda de datos mínimos

Esta etapa consistió en la recolección de datos necesarios para posteriormente realizar las primeras pruebas de **Remo**. Esta fase involucró la recolección de secuencias de $_m$RNA, secuencias de $_{mi}$RNA y el software humanizador de secuencias.

$_m$RNA

La bases de datos biológicas son DNA céntricas, es decir que si un virus tiene su información como RNA, la secuencia archivada esta expresada como DNA (**T**imina en lugar de **U**racilo). Por lo cual, no se encontró $_m$RNA directamente, sino que como gen.

Los virus obtenidos para las pruebas se obtuvieron de
[Gareth M. Jenkins and Edward C. Holmes., 2003]. Dicha publicación menciona el número de acceso a *GenBank*[4] para su obtención. A continuación se exhiben algunos de los códigos que se mencionan el artículo:

[3] `r-emo.googlecode.com`

[4] Base de datos de secuencias genéticas. Colección de todas las secuencias de DNA disponibles al público. La base de datos GenBank está diseñada para proporcionar y fomentar el acceso de la comunidad científica a la mayor parte de información actualizada y completa la secuencia de DNA. Por lo tanto, *NCBI* (National Center for Biotechnology Information `http://www.ncbi.nlm.nih.gov/`) no impone restricciones sobre el uso o la distribución de los datos GenBank.

- Coxsackievirus A9 (Griggs; D00627).
- Enterovirus 71 (TW/2272/98; AF119795).
- Hepatitis A virus (HAF-203; AF268396).
- Poliovirus type 3 (23127; X04468).
- Dengue-1 virus (PDK-13; AF180818).
- Yellow fever virus (Trinidad 79A; AF094612).
- Rubella virus (Cendehill; AF188704).

A partir de los códigos de acceso, desde `http://www.ncbi.nlm.nih.gov/genbank/` se obtuvieron los DNA de cada virus en formato FASTA. Para traducir los DNA a RNA (reemplzado de U por T), se utilizó la herramienta *BioEdit*[5].

$_{mi}$RNA

Se obtuvo una base de datos[6] de $_{mi}$RNA en formato FASTA. De la misma, se seleccionaron al azar aproximadamente 50 $_{mi}$RNA para luego realizar las primeras pruebas. Los $_{mi}$RNA están formados por 22 nucleótidos aproximadamente. A continuación se exhiben algunas de las secuencias:

- AAGAUGUGGAAAAAUUGGAAUC
- CAGUGGUUUUACCCUAUGGUAG
- CAGGCAGUGACUGUUCAGACGUC
- AUCAUAGAGGAAAAUCCAUGUU
- AAGAAGAGACUGAGUCAUCGAAU
- CUCUAGAGGGAAGCGCUUUCUG
- GUUUGCACGGGUGGGCCUUGUCU
- AAAGACCGUGACUACUUUUGCA
- UGUAAACAUCCCCGACUGGAAG

[5] `http://www.mbio.ncsu.edu/BioEdit/bioedit.html`

[6] Descargada de `http://www.mirbase.org/cgi-bin/mirna\_summary.pl?org=hsa`

Otras bases de datos de $_{mi}$RNA consultadas fueron:

- `http://micrornadatabase.com/`.
- `http://microrna.org/`.
- `http://mirdb.org/miRDB/`.
- `http://www.mirbase.org/`.

4.2.2. Herramientas y Librerías

GeneDesign

GeneDesign[7] [Richardson et al., 2010] fue el software empleado para la "humanización" de las cadenas de nucleótidos. El mismo, corresponde a un software libre usado por biólogos moleculares, el gobierno y los sectores farmacéuticos, químicos, agrícolas y biotecnológicos para el diseño [Welch, et al., 2011], clonación y validación de secuencias genéticas.

GeneDesign automatiza el proceso de determinar qué pares de bases deben ser unidos entre sí en un orden determinado para un gen. Un gen codifica para una proteína específica, y el orden de los cientos o miles de pares de bases que componen ese gen determina el orden de los bloques de construcción de aminoácidos que componen esa proteína.

Este software, consta de seis módulos que pueden ser utilizados individualmente o en serie para automatizar las tareas necesarias para diseñar y manipular secuencias de DNA sintético.

Se presentaron tres opciones para el uso de este software:

- *Instalado:* corresponde a tener *geneDesign* instalado en la PC, y el producto a desarrollar lo invocará a medida que lo requiera.
- *Online:* el producto a desarrollar utiliza *geneDesign* de forma online a medida que lo necesita. (`http://genedesign.thruhere.net/gd`)
- *Offline:* correr de forma online *geneDesign* y generar con su salida dos archivos en formato FASTA. Uno de ellos, con el $_{m}$RNA original, y el otro con el humanizado. Luego el producto a desarrollar tomará los archivos mencionados como entrada.

Remo empleó la opción número 1 (*Instalado*) otorgando de esta forma mayor independencia y evitando la necesidad de conexión a internet para obtener los resultados. Para esta opción fue necesario descargar el código de *geneDesign* del repositorio git e instalar y configurar el servidor web *apache2*[8].

[7] Descarga de: `https://github.com/GeneDesign/GeneDesign`

[8] urlhttp://httpd.apache.org/

Cabe destacar que se encontró un bug en el código, básicamente el software siempre realizaba la conexión al servidor central por lo cual, aunque se ejecutara localmente con el servidor apache, no funcionaba sin conexión a internet, dado que siempre se establecía un socket con el servidor central.

Librerías a Utilizar

Además de las librería estándares que provee el jugoso lenguaje de programación C++, cabe mencionar las siguientes:

- *MiLi*[9]: es una colección de pequeñas y útiles librerías desarrolladas en el lenguaje de programación C++ por **FuDePAN**, compuestas únicamente por cabeceras (*headers*). No requiere instalación para su uso y ofrece *soluciones simples para problemas sencillos*.

  Dentro de las diversas funcionalidades podemos encontrar:

  - **binary-streams:** permite serializar diferentes tipos de datos dentro de un único objeto utilizando los operadores de stream ($\ll$ y $\gg$). Hay dos maneras de utilizar esta librería:
    1. Empaquetar datos dentro de un objeto de salida (bostream) utilizando el operador $\ll$.
    2. Extraer datos desde un objeto de entrada (bistream) utilizando el operador $\gg$.
  - **container-utils:** esta biblioteca provee un conjunto de funciones, optimizadas para cada tipo de contenedor STL:
    - find(container, element).
    - find(container, element, nothrow).
    - contains(container, element).
    - insert into(container, element).
    - copy container(from, to).

    Adicionalmente, esta biblioteca provee las siguientes clases:
    - ContainerAdapter<T>.
    - ContainerAdapterImpl<T, ContainerType>.

    Estos container adapters son una herramienta para lidiar con diferentes contenedores STL de manera homogénea, sin necesidad de conocer el tipo de contenedor que es (vector, list, map, set). Los

[9] `mili.googlecode.com`

usuarios pueden invocar al método INSERT (CONST T&) para insertar un elemento de tipo T sin la necesidad de saber el tipo del contenedor.

- **generic_exception:** ofrece una implementación de excepciones genéricas a partir de las cuales los desarrolladores pueden crear sus propias excepciones para problemas específicos de una manera muy simple.

- *Biopp*[10]: corresponde a una biblioteca C++ para Biología Molecular. La misma provee las diversas estructuras de datos y métodos para manipular las secuencias de nucleótidos con las que se trabajó.

- *Biopp-filer*[11]: refiere a una librería de persistencia para *Biopp*. Esta librería se utilizó para leer las secuencias en formato FASTA.

- *Getoptpp*[12]: es una librería que proveerá entre otras cosas, diversas funciones para el manejo de la entrada estándar.

4.2.3. Librerías a Implementar

Como resultado del análisis y pensando en la correcta modularización y futuros desarrollos, surgió la necesidad de implementar las siguientes librerías:

- **fideo (Folding Interface Dynamic Exchange Operations):** el problema que se presentó radica en que cada librería externa tanto para folding como para hibridación, poseen diferentes formas de recibir los parámetros de entrada y diferentes formas de retornar los resultados que generan. A razón de esto se decidió implementar *fideo* como una forma de unificar el acceso a estas librerías externas e integrarlas al resto del sistema. Básicamente, se encargará de proveer las funcionalidades necesarias para obtener la energía libre de una secuencia de nucleótidos. Esta librería se utilizará para la invocación externa de los programas de cálculo de estructura secundaria (folding e hibridación). Por un lado, incluye los backends UNAFold y RNAFold para folding, y por el otro, incluye los backends RNAup, RNAcofold, RNAduplex, IntaRNA y RNAHybrid para la hibridación. Fideo provee la abstracción necesaria para poder utilizar indistintamente cualquiera de los backends mencionados de forma transparente.

[10] `biopp.googlecode.com`
[11] `biopp-filer.googlecode.com`
[12] `getoptpp.googlecode.com`

- **acuoso (Abstract Codon Usage Optimization Software for Organisms):** permite la optimización (humanización) de secuencias. La idea es similar a fideo, dado que permite realizar este servicio con cualquier software de manera muy simple. Inicialmente se implementó empleando *GeneDesign*, pero de manera sencilla puede agregarse cualquier otro backend.
- **etilico (External Tools Invocation LIbrary and COmponents)**: básicamente corresponde a una librería estática compuesta por diversas funciones necesarias y de uso común para invocación de librerías, manejo de archivos temporales y diversos componentes.

4.2.4. Backends para Folding

UNAFold

Inspirado en el famoso software "mfold", el paquete de software *UNAFold* [N.R. Markham & M.Zuker., 2008] es un conjunto integrado de programas (empleando programación dinámica [Bellman, 1957]) que simulan el folding, la hibridación y la vías de fusión para una o dos secuencias de cadena simple de ácido nucleico. El paquete predice el folding para RNA de cadena simple o el DNA a través de la combinación de minimización de la energía libre, los cálculos de función de partición y muestreo estocástico. El nombre se deriva de "Folding Unificado de Ácido Nucleico". Acepta tanto archivos planos (que contengan únicamente nucleótidos) como en formato FASTA. Está disponible para su descarga en `http://www.bioinfo.rpi.edu/applications/hybrid/download.php`. A continuación se exhibe el cálculo de folding de una secuencia de prueba.

```
gringusi@gringusi:~$ echo "AAAAAAAAGGGGGGGGCCCCCCCCTTTTTTTT" > seq.fasta
gringusi@gringusi:~$ UNAfold --max=1 seq.fasta

Checking for boxplot_ng... not found
Checking for hybrid-plot-ng... found, supports Postscript
Checking for sir_graph_ng or sir_graph... not found
Checking for ps2pdfwr... found
Calculating for seq.fasta, t = 37
gringusi@gringusi:~$ l
seq.fasta seq.fasta_1.ct seq.fasta.ann seq.fasta.ct seq.fasta.det seq.fasta.dG
seq.fasta.h-num seq.fasta.plot seq.fasta.rnaml seq.fasta.run seq.fasta.ss-count
```

Dentro de los diferentes archivos generados se detacan el *.dG*, *.ct* y *.det*. El primero de ellos contiene el valor de la energía libre en la unidad kcal/mol. El segundo, define una secuencia de ácido nucleico, junto con su estructura

secundaria. Puede contener pliegues múltiples de una secuencia única o incluso múltiples plegamientos. Por último, el tercero, brinda la estructura secundaria en un formato particular, donde se pueden identificar los diversos *motifs* que la componen.

RNAFold

Básicamente lee secuencias de RNA de entrada estándar y calcula el mínimo de energía libre (MFE), la función de partición (pf) y una matriz de probabilidad en base al apareamiento. Devuelve como salida la estructura mfe en notación soporte (empleando "()"), su energía, la energía libre del conjunto termodinámico y la frecuencia de la estructura mfe. También produce archivos PostScript con parcelas de la gráfica resultante estructura secundaria y un "gráfico de puntos" de la matriz de apareamiento de bases. Para comprender mejor su comportamiento y método consultar [Hofacker et al., 1994].

Está disponible para su descarga en `http://mfold.rna.albany.edu/?q=DINAMelt/software.`

```
gringusi@gringusi:~$ RNAfold

Input string (upper or lower case); @ to quit
....,....1....,....2....,....3....,....4....,....5....,....6....,.
AAAGGCAACGGCCAU
length = 15
AAAGGCAACGGCCAU
...(((....)))..
minimum free energy =  -4.40 kcal/mol
```

El resultado obtenido es, precisamente, la energía libre y la estructura secundaria representada con paréntesis y puntos, donde los pares de paréntesis indican las bases "apareadas" o "unidas" y los puntos, las bases libres.

4.2.5. Backends para Hibridación

IntaRNA

IntaRNA [Busch et al., 2008] predice interacciones entre dos moléculas de RNA, por ejemplo, un RNA no codificante (e.g ncRNA) y un mRNA. El scoring se basa en la energía combinada de interacción que resulta de la suma de la energía libre de hibridación y la energía libre necesaria para la fabricación de los sitios de interacción accesible de ambas moléculas.

La interacción tiene que contener la semilla de una interacción, es decir, una región (casi) perfecta de complementariedad para facilitar el inicio de la interacción. Las características de esta región de semilla son definidas por el usuario.

La accesibilidad se define como la energía libre necesaria para desplegar el sitio de interacción en cada molécula. Requiere del paquete de Vienna RNA (versiones 1.8.2/1.8.5). La versión 1.8.5 puede descargarse de `http://www.bioinf.uni-freiburg.de/Software/index.html?en\#IntaRNA-download`

Existen dos variantes para realizar la predicción, la primera corresponde una aproximación completa, que es O(n2 m2) en el tiempo y O(nm) en el espacio al restringir el tamaño de los bucles internos y donde n y m son las longitudes de las secuencias de RNA que interactúan (n > m), y la segunda que es una simplificación heurística de la aproximación completa, que tiene una complejidad en tiempo de O(nm) y un espacio de complejidad O(nm), donde m = max m, L 3 y L es el tamaño de la secuencia de la ventana en el que tanto el $_m$RNA diana y la sRNA se pliegan.

RNAup

RNAup [Muckstein et al., 2005] calcula la termodinámica de las interacciones RNA-RNA, por descomposición de las uniones a través de dos etapas. La termodinámica de tales interacciones RNA-RNA puede ser entendida como la suma de dos contribuciones de energía: la energía necesaria para "abrir" el sitio de unión, y la energía obtenida de la hibridación.

En primer lugar se calcula la probabilidad de que un potencial sitio de unión siga siendo desapareado, lo que es equivalente a calcular la energía libre necesaria para romper las uniones. En segundo lugar, este cálculo se combina con la energía de interacción para obtener la energía total de unión.

Proporciona dos modos: por defecto calcula accesibilidad (en términos de energía libre para romper uniones, de longitud 4) y muestra la región de mayor accesibilidad y la energía libre necesaria para su apertura. En modo interacción, se calcula la interacción entre dos RNA. Este modo se activa si la entrada se compone de dos secuencias concatenadas por un "&" o si se emplea el flag -X[pf] o -b.

La energía libre "dG" se divide en sus componentes "dGint" (energía de interacción) y "dGu_l" (energía de apertura).

RNAup esta incluido en el paquete de Vienna RNA [Hofacker et al., 1994]. La última versión corresponde a la 2.0.7 y esta disponible para su descarga en `http://www.tbi.univie.ac.at/~ivo/RNA/`.

RNACofold

RNACofold funciona como *RNAfold*, pero permite especificar dos secuencias de RNA que formará una estructura de dímero[13].

Las secuencias de RNA se leen por entrada estándar, cada línea de entrada corresponde a una secuencia, a excepción de las líneas que comienzan con "¿", que contiene el nombre de la siguiente secuencia. Para calcular la estructura híbrida de dos moléculas, las dos secuencias deben ser concatenadas usando el carácter "&" como separador.

Al igual que RNAup, *RNAcofold* esta incluido en el paquete de Vienna RNA [Hofacker et al., 1994]. Tiene orden O(n^3)

RNADuplex

Básicamente lee dos secuencias de RNA por entrada estándar o desde un archivo y calcula las estructuras secundarias óptimas y subóptimas para su hibridación. El cálculo se simplifica al permitir sólo pares de bases intermoleculares (corresponde a un caso específico de *RNAcofold*).

La estructura calculada óptima y subóptima se retornan por salida estándar, una por línea. Cada una de ellas consiste en: la estructura de soporte en formato de punto con una "y" que separan las dos hebras, el rango de la estructura en las dos secuencias en el formato *"from,to : from,to"*, la energía de estructura dúplex en kcal/mol.

El formato es especialmente útil para el cálculo de la estructura híbrida entre una secuencia pequeña y una secuencia larga. Al igual que los dos anteriores, *RNAduplex* esta incluido en el paquete de Vienna RNA [Hofacker et al., 1994].

RNAHybrid

RNAHybrid [Rehmsmeier at el., 2004] es una herramienta para encontrar hibridaciones de mínimo de energía libre entre una secuencia larga (target) y un RNA corto (query). La hibridación se lleva a cabo de la siguiente manera: la secuencia corta se hibrida con las mejores partes de fijación del target.

RNAHybrid puede obtenerse desde `http://bibiserv.techfak.uni-bielefeld.de/download/tools/rnahybrid.html`. Actualmente, puede utilizarse como servicio web desde `http://alk.ibms.sinica.edu.tw/cgi-bin/RNAhybrid/RNAhybrid.cgi`. Tiene orden O(nm).

[13]Complejo macromolecular formado por dos macromoléculas, como proteínas y ácidos nucleicos, usualmente mediante enlaces no covalentes.

Capítulo 5

Diseño

> La mejor estructura no garantizará los resultados ni el rendimiento. Pero la estructura equivocada es una garantía de fracaso.
>
> **Peter Drucker**

Según la IEEE el diseño es definido como: *"El proceso de definición de la arquitectura, componentes, interfaces y otras características de un sistema o componente que resulta de este proceso"*[IEEE610.12-90]. Así, todo sistema necesita de un diseño adecuado que permita modelar adecuadamente los requerimientos con el objetivo de tener una mirada integradora del sistema. Un buen diseño ayuda a crear software con propiedades deseables, como reusabilidad, mantenibilidad, portabilidad, ente otras.

De manera similar a la elicitación de requerimientos, la descripción detallada del diseño se encuentra documentada en la "Especificación de Diseño de Software". Análogamente a los capítulos anteriores, sólo se detallarán los aspectos más relevantes del diseño, dejando como lectura adicional el mencionado documento.

5.1. Responsibility-Driven Design

Este esquema fue empleado para el análisis y descripción del diseño de **Remo**. Fue propuesto inicialmente por Rebeca Wirfs-Brock y Brian Wilkerson[Wirfs-Brock and McKean, 2002] en 1990. Básicamente, se enfoca en qué responsabilidades deben ser cubiertas por el sistema y en cuáles serán los objetos responsables de llevarlas a cabo.

Inicialmente, se describen las acciones y actividades que constituyen las "responsabilidades" del sistema. Luego, se describen dichas responsabilidades en un lenguaje comprensible, tanto para el usuario como para el desarrollador, y finalmente, se diseñan objetos de software que las implementen apropiadamente. En primera instancia, se captura la noción de cuál es el comportamiento deseado, antes de plantear la forma de obtener dicho comportamiento.

5.2. Principios de Diseño

Uno de los requerimientos de **Remo** fue que se respetara los principios del diseño orientado a objetos resumidos en el acrónimo SOLID[Martin, 2000] introducido por Robert C. Martin[1] a principios del año 2000.

En particular se puso especial atención en respetar los principios SRP, OCP y DIP debido a que repercuten directamente en que el sistema sea más simple de mantener y extender con nuevas funcionalidades. Además, el principio DIP es fundamental para lograr un software que sea verificable mediante la automatización de pruebas como se pudo comprobar más adelante, en la etapa de "verificación".

A continuación se detalla brevemente cada uno de los principios, dado que resultan de gran importancia su conocimiento.

- **Principio de Simple Responsabilidad (SRP):** establece que no debería haber nunca más de una sola razón para que una clase cambie. En el contexto de este principio, llamamos responsabilidad a una razón de cambio. Si una clase tiene más de una responsabilidad, las responsabilidades se acoplan. Esta clase de acoplamiento lleva a que el sistema no sea tolerante a las modificaciones. Básicamente, refiere a la noción de que un objeto debe tener sólo una responsabilidad.

- **Principio de Apertura-Cierre (OCP):** este principio afirma que las entidades de software (clases, módulos, funciones, etcétera) deben estar abiertas para la extensión, pero cerradas para la modificación. Esto es especialmente valioso en un entorno de producción, donde los cambios en el código fuente puede requerir revisiones de código, pruebas unitarias, y otros procedimientos para calificar para su uso en un producto.

  Este principio indica que cuando se cambian los requisitos, se extiende el comportamiento de los módulos mediante la adición de código, pero el código viejo no se cambia dado que funciona.

[1] Autor de varios libros tales como "Object-orineted Design" y "Agile Development".

- **Principio de Sustitución de Liskov (LSP):** refiere a que los objetos de un programa pueden ser reemplazados por instancias de sus subtipos, sin alterar la exactitud del programa.

  Se afirma que, en un programa, si S es un subtipo de T (T→S), entonces los objetos de tipo T puede ser reemplazado con los objetos de tipo S sin alterar ninguna de las propiedades deseables de ese programa. Más formalmente, el principio LSP es una definición particular de una relación de subtipificación, que fue introducido inicialmente por Bárbara Liskov. Se trata de una relación semántica más que sintáctica que tiene la intención de garantizar la interoperabilidad semántica de los tipos en una jerarquía, los tipos de objeto en particular.

- **Principio de Segregación de Interfaz (ISP):** refiere a la idea de que varias interfaces específicas son mejores que una interface de uso general.

  Es un principio utilizado para un desarrollo limpio. Si se cumple, ayuda a mantener un sistema desacoplado y por lo tanto más fácil de refactorizar. El ISP dice que una vez que una interfaz se ha vuelto demasiado "gorda" debe ser dividida en interfaces más pequeñas y más específicas para que los clientes sólo tengan disponibles los métodos que pertenecen a ellos. En pocas palabras, los clientes no deberían ser obligados a depender de interfaces que no usan.

- **Principio de Inversión de Dependencia (DIP):** refiere al sentido en que se debe depender de las abstracciones y no de las concreciones.

  Los módulos de alto nivel no deben depender de módulos de bajo nivel y ambos deberían depender de las abstracciones. Además establece que las abstracciones no deben depender de los detalles. Los detalles deberían depender de las abstracciones.

5.3. Arquitectura del Sistema

Según [Shaw, 1994] "la *arquitectura* es un nivel de diseño que hace foco en aspectos más allá de los algoritmos y estructuras de datos de la computación; el diseño y especificación de la estructura global del sistema es un nuevo tipo de problema".

Remo esta compuesto por 4 componentes básicos como se observar en la figura 5.1:

1. Generador de secuencias humanizadas: encargado de humanizar secuencias, es decir, cambiar los codones según el organismo especificado.

2. Generador de análisis de secuencias: tiene como tarea realizar el análisis entre las secuencias de $_m$RNA vs. $_{mi}$RNA.

3. Generador de comparaciones de estructuras: encargado de la comparación entre estructuras secundarias, para lo cual necesita de el reconocimiento de motifs.

4. PreFold: encargado de realizer el folding de secuencias.

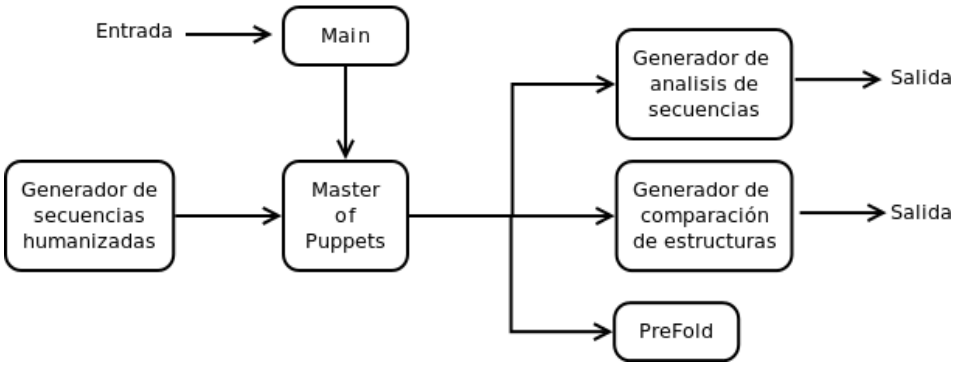


Figura 5.1: Componentes de **Remo** [2].

En la figura 5.2 se exhibe con más detalle el diseño arquitectónico de **Remo**.

5.3.1. Catálogo de componentes

A continuación se especifican los distinto componentes de **Remo**:

- **Main:** punto de entrada de **Remo**.
- **MOP:** mediador responsable de la comunicación entre los módulos del sistema.
- **CodingSectionObtainer:** permite obtener la mayor sección codificante de una secuencia.
- **TableGenerator:** interfaz para seleccionar el tipo de salida de **Remo**.
- **OldTableGenerator:** emplea un método ad-hoc usando los backend de folding.

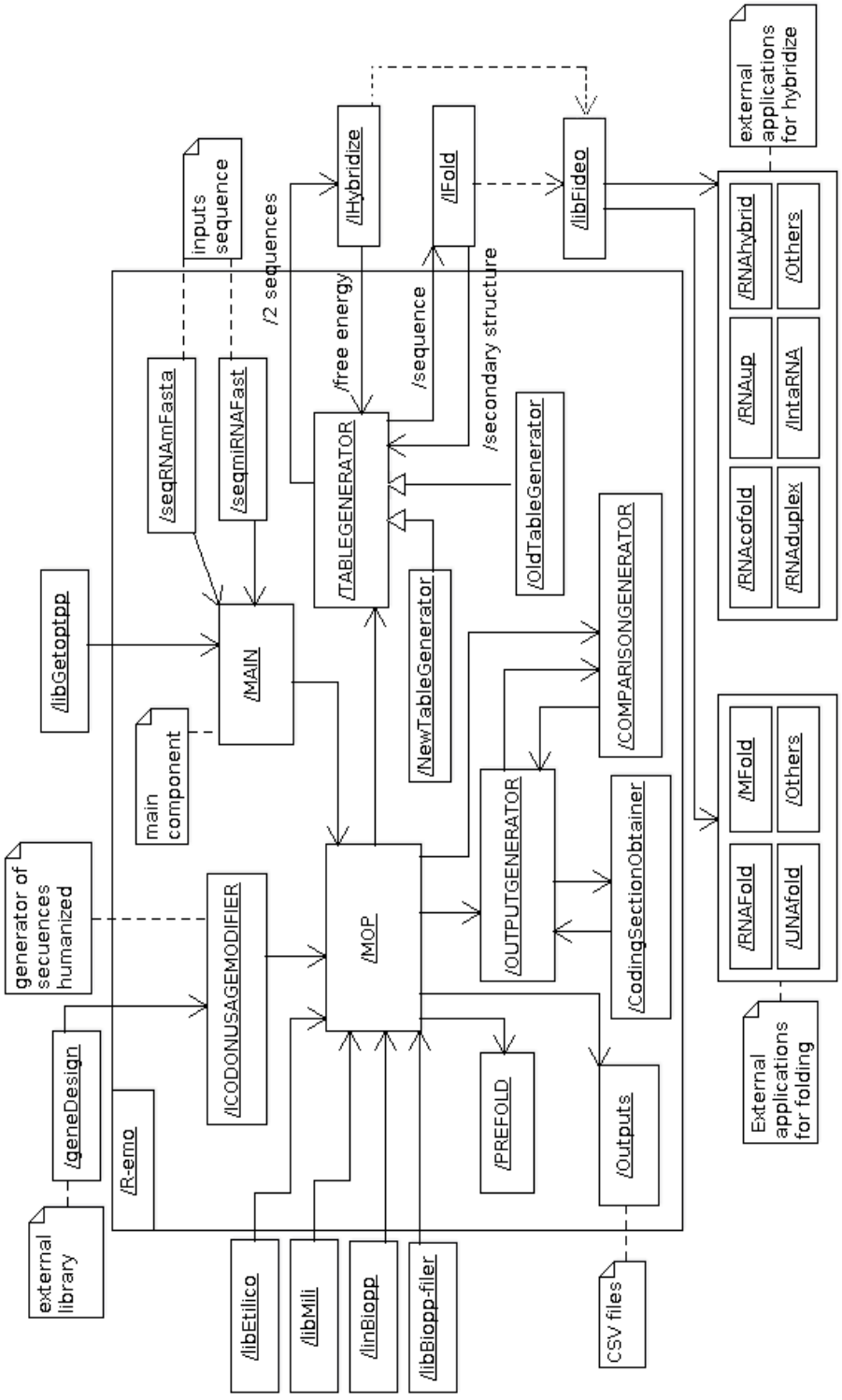


Figura 5.2: UML - Arquitectura del Sistema [2]

- **NewTableGenerator:** más formal empleando los backends de hidridacíon.
- **OutputGenerator:** permite obtener los archivos de resultado.
- **ComparisonGenerator:** permite realizar la comparación de estructuras secundarias (parser).
- **Prefold:** permite realizar el folding de las secuencias de $_m$RNA.

5.3.2. Comportamiento Dinámico

En la figura 5.3 se expone un diagrama de secuencia que permite visualizar la interacción dinámica entre los componentes de **Remo** para el análisis de secuencias de $_m$RNA y $_{mi}$RNA.

5.4. Diseño de Medio Nivel

5.4.1. Diagrama de Paquetes

En la figura 5.4 se exhiben el diagrama de paquetes correspondiente a **Remo**, donde se exhiben las distintas librerías que interactúan.

5.4.2. Diagrama de Clases

En la figura 5.5 se exhibe un diagrama de clases de **Remo**. Con el objetivo de lograr una mejor comprensión, se decidió no especificar los métodos y miembros de cada clases. Para mayor detalle puede visitar el apéndice D (figuras D.1 y D.2).

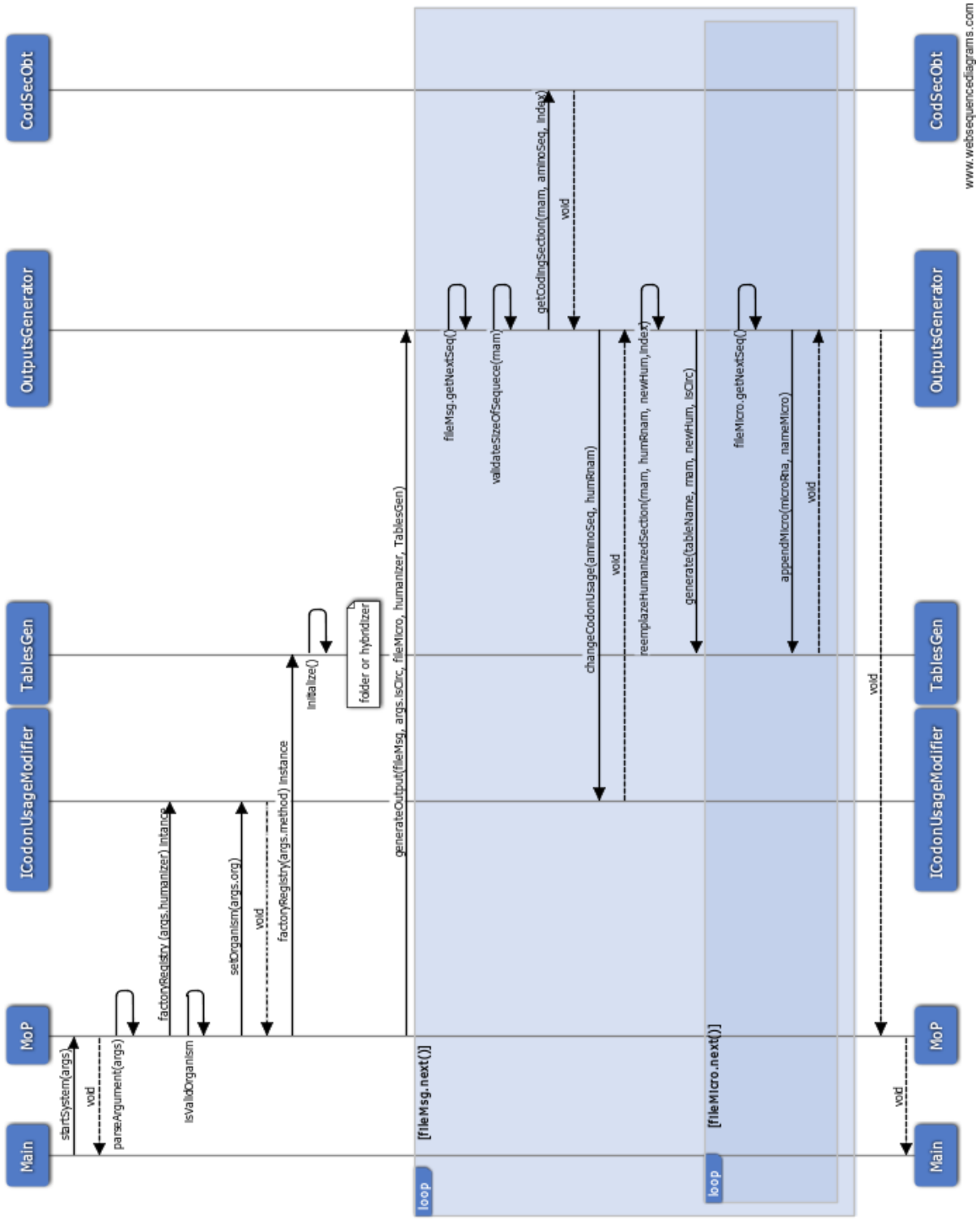


Figura 5.3: Interacción dinámica entre los componentes de **Remo** [2].

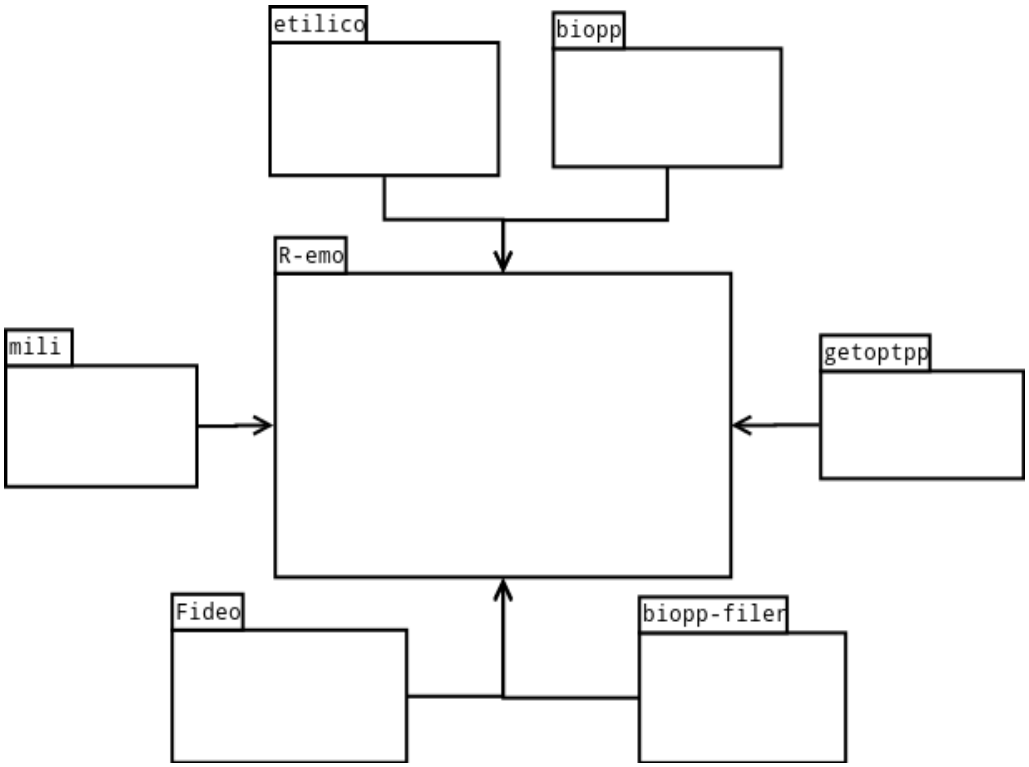


Figura 5.4: UML - Diagrama de Paquetes [2].

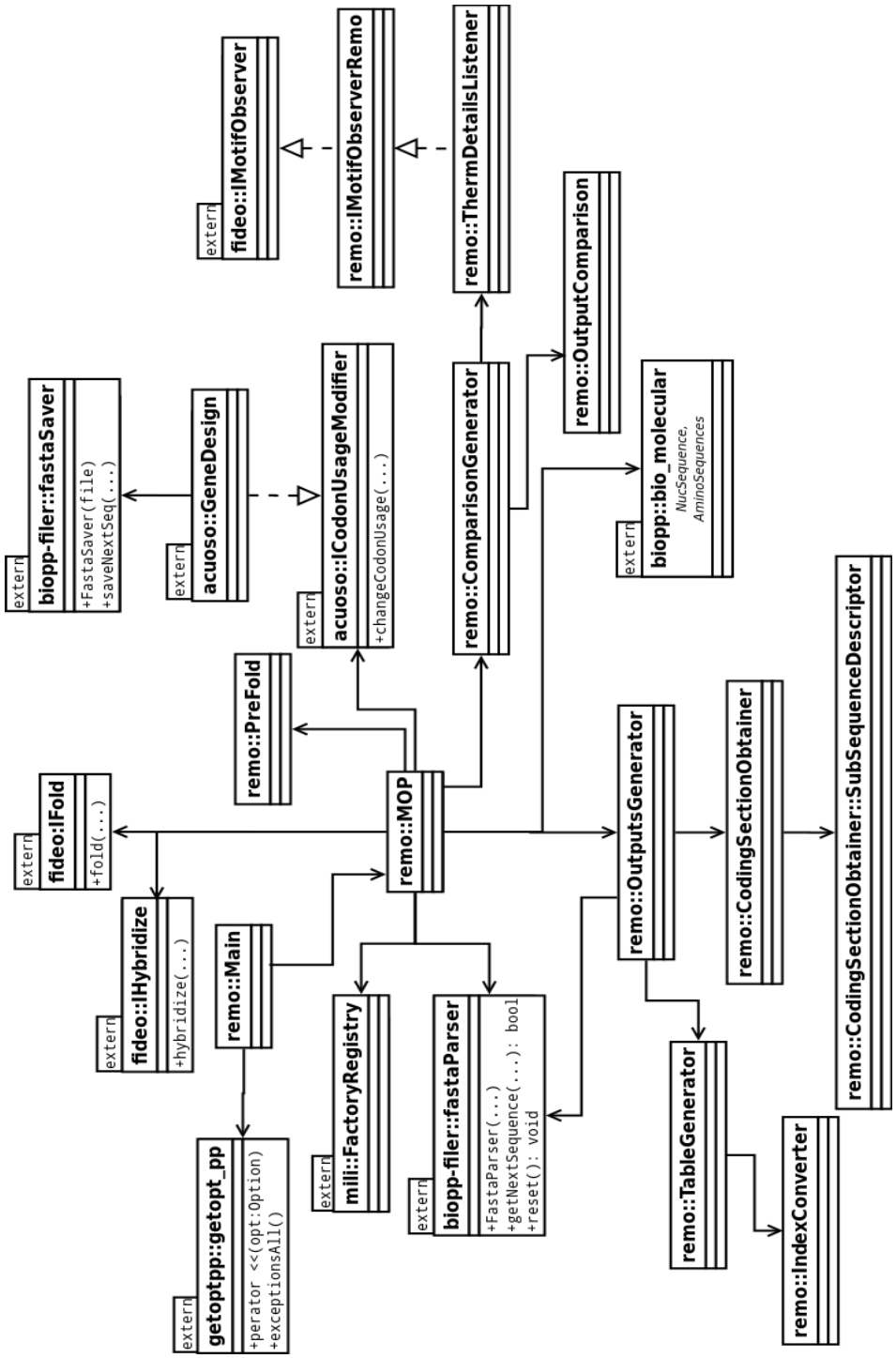


Figura 5.5: UML - Diagrama de clases de **Remo** [2].

5.5. Nuevas Librerías

5.5.1. Fideo: Folding Interface Dynamic Exchange Operations

Tal como se mencionó en la sección 4.2.3, fideo proveerá los servicios necesarios para folding e hibridación. Para tal fin, se empleó el patrón de diseño *Factory Method*(ver Apéndice E.2.2). Por otra parte, fideo proporciona un parser para identificar motifs en estructuras secundarias. Para tal fin se empleó el patrón de diseño *Observer*(ver Apéndice E.2.3). Para implementar el método *fold()* propiamente dicho se empleó el patrón de diseño *Template Method*(ver Apéndice E.2.4).

En la figura 5.6 se exhiben las interfaces implementadas con sus respectivas clases concretas. Se puede observar la interfaz *IMotifObserver* y *IRule* no mencionadas hasta el momento. Básicamente se busca emplear un observer, esto es, fideo proporcionará el parser de estructuras secundarias y establecerá una interfaz la cual deberá ser implementada por **Remo** para realizar en correspondiente análisis.

En este contexto, se puede apreciar la idea del principio de diseño *DIP*. Dado que, si **Remo** depende de estas interfaces y no de sus respectivas implementaciones se consigue abstraer los detalles de cada librería externa y lograr un software mas versátil.

El diagrama de clases correspondiente a fideo puede observarse en la figura 5.7. Al igual que en **Remo** , para lograr una mejor comprensión, se decidió no especificar los métodos y miembros de cada clases, puede visitar el apéndice D (figuras D.3 y D.4)para mayor detalle.

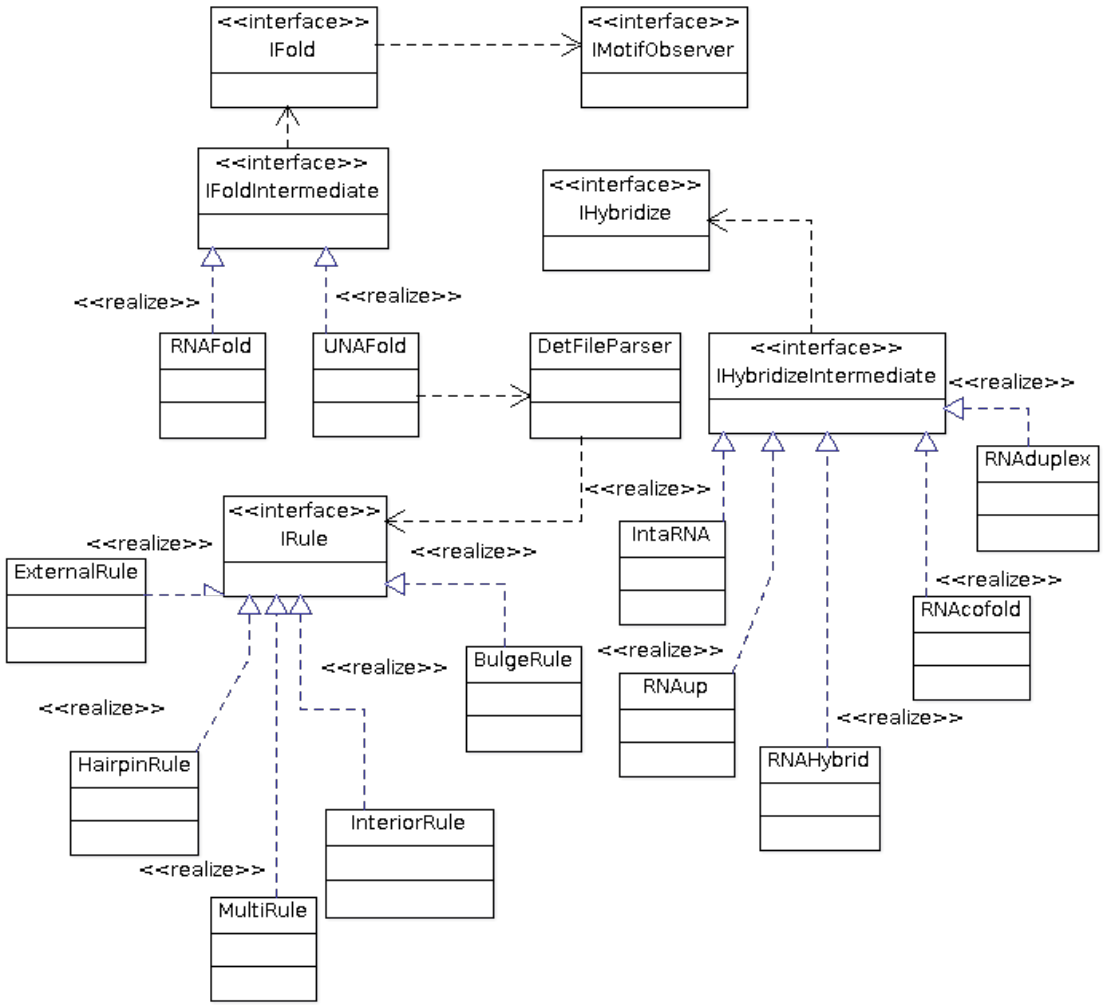


Figura 5.6: UML - Interfaces de fideo. [2]

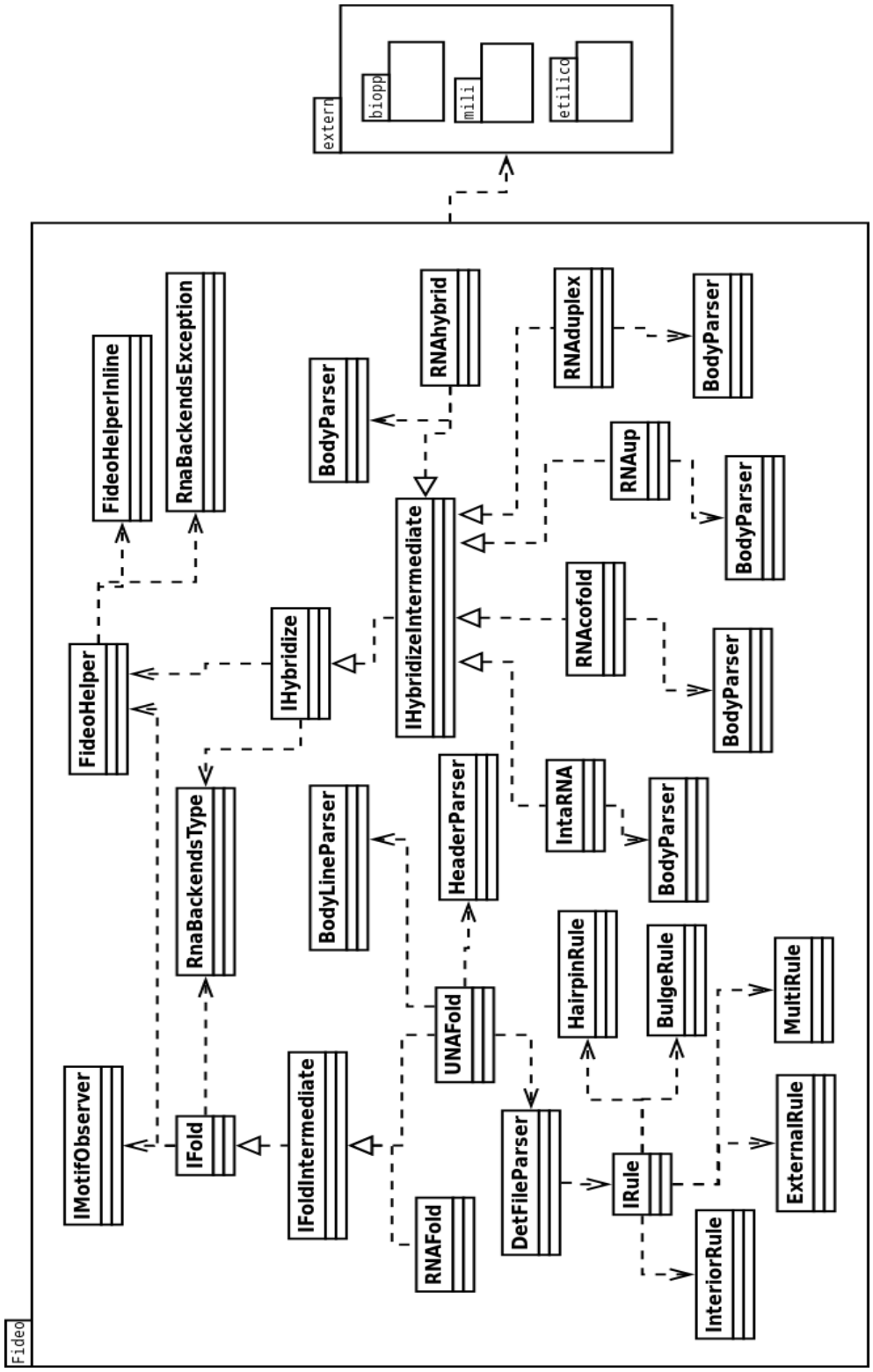


Figura 5.7: UML - Diagrama de clases de fideo [2].

5.5.2. Acuoso: Abstract Codon Usage Optimization Software for Organisms

Esta librería sigue la misma idea de fideo, se definió una interfaz *ICodonUsafeModifier* de modo que sea transparente el uso de cualquier software o método de humanización. En la figura 5.8 se observa el diagrama de clases correspondiente.

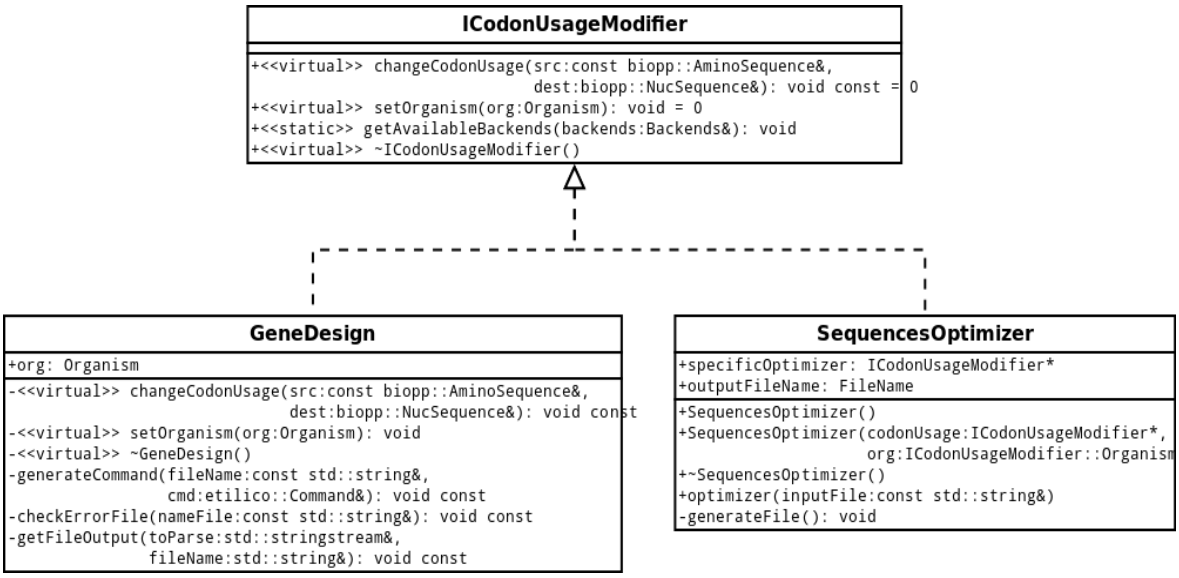


Figura 5.8: UML - Diagrama de clases de Acuoso [2].

5.5.3. Etilico: External Tools Invocation LIbrary COmponent

Etilico corresponde a una librería estática contenedora de diversos métodos necesarios cuyo diseño es muy simple. En la figura 5.9 se exhibe el diagrama las clases contenidas en etilico.

La clase *Helper* contiene diferentes funciones de gran utilidad. La clase *TmpDirectory*, representa el manejo de directorios temporales, y por último, la clase *Config* se trata de un singletón (ver Apéndice E.2.1) empleado para configurar las diferentes rutas de las herramientas empleadas.

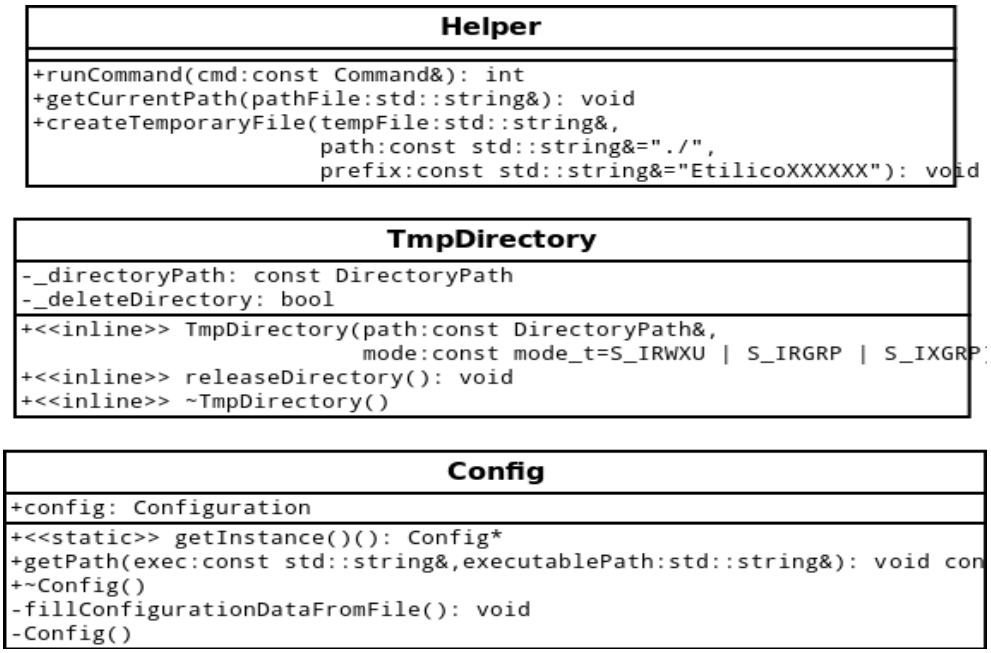


Figura 5.9: UML - Diagrama de clases de etilico [2].

Capítulo 6

Implementación

> Sea lo que sea aquello que cree o piensa que puede hacer, empiece a hacerlo. La acción tiene magia, gracia y poder.
>
> **Goethe**

El presente capítulo describe algunas características de la puesta a punto de **Remo**. Se exhibe la representación de datos bioinformáticos, el framework empleado para la distribución de trabajo y algunas porciones de pseudo-código.

6.1. Estructuras y Algoritmos Bioinformáticos

Ya presentadas las estructuras biológicas básicas en los capítulos anteriores, a continuación se exhibirá la representación de las mismas y algunos algoritmos que son de uso corriente.

6.1.1. Representación de nucleótidos, aminoácidos y proteínas

Básicamente, las cadenas de nucleótidos y de aminoácidos que se utilizaron a lo largo de **Remo** se representaron como cadenas de caracteres (aunque a nivel implementación su representación es mucho más compleja, empleando programación orientado a aspectos mediante templates). Denotamos a las mismas como expresiones regulares, las misma se observan a continuación.

- $nuc\_dna = a|t|c|g$
- $nuc\_rna = a|u|c|g$
- $aminoacido = Ala|Arg|Asn|...$

De la misma forma representamos a los genes de DNA, RNA y proteínas.

- $gen\_dna = (nuc\_dna)^+$
- $gen\_rna = (nuc\_rna)^+$
- $proteina = aminoacido(aminoacido)^+$

Traducción DNA -> RNA

Existe una forma algorítmica de traducir una secuencia de DNA a una secuencia de RNA (Código 7.1). La misma, consiste en analizar cada nucleótido de la cadena cambiando cada t por u.

```
gen_rna = traslate(gen_dna), donde:

gen_rna translate(gen_dna)
{
  forall nuc_dna in gen_dna do
  {
    if nuc_dna is t then
      replace(nuc_dna, u)
  }
}
```

Código 7.1: Traducción DNA -> RNA.

Traducción RNA -> proteínas

Podemos traducir la secuencia de RNA (compuesta de nucleótidos) a una secuencia de proteínas (compuesta de aminoácidos). Cabe recordar que cada aminoácido se corresponde por tres nucleótidos (un codón).

El algoritmo (Código 7.2) toma cada triplete y lo traduce a un aminoácido correspondiente según la tabla de correspondencia observada en la subsección 1.1.2 (figura 1.3).

```
protein = translate(gen_rna), donde:

protein translate(gen_rna)
{
  i = 0
  triplet = substr(gen_rna, i, 3)
  while (triplet != STOP && i < length(gen_rna))
  {
    aminoacid = genetic_code(triplet)
    protein += aminoacid
    i += 3
    triplet = substr(gen_rna, i, 3)
  }
}
```

Código 7.2: Traducción RNA -> proteínas.

6.2. Decisiones de Implementación

6.2.1. Implementar vs. Integrar

Ante el requerimiento de contar con algoritmos para el cálculo de la estructura secundaria de RNA, se evaluó entre implementar completamente los algoritmos o integrar/adaptar los algoritmos existentes.

En este sentido, las interfaces propuestas no descartan ninguna de las dos posibilidades, sino simplemente especifican los "servicios" que deben ofrecer al sistema las eventuales implementaciones, ya sean propias o ajenas.

Por otro lado, implementar este tipo de algoritmos requiere un profundo conocimiento del dominio (Química Molecular). Además, debido a la complejidad de los mismos, se requieren validaciones empíricas para demostrar la fidelidad de los resultados lo cual excedía el alcance de este trabajo.

Finalmente se optó por integrar al sistema los algoritmos para implementar las respectivas interfaces. A continuación se exhibe dicha relación.

Interface	**Algortimos**
fideo::IFold	UNAFold, RNAFold
fideo::IHybridize	RNAup, IntaRNA, RNAcofold, RNAduplex, RNAHybridize
acuoso::ICodonUsageModifier	GeneDesign

6.2.2. Método Ad-hoc

En primer instancia, para realizar las comparaciones necesarias de $_m$RNA y $_{mi}$RNA se empleó un método ad-hoc cumpliendo con los requerimientos definidos[1].

Básicamente la idea consistió en fijar cada secuencia de $_m$RNA y por cada secuencia de $_{mi}$RNA, ir desplazándola de a un nucleótido por vez en el mensajero desde la posición inicial. En cada paso, se realizaron los análisis correspondientes entre nucleótidos.

Esta manera de implementar las pasadas de $_{mi}$RNA sobre los $_m$RNA es ineficiente (Código 7.3). Sin embargo, se escogió adrede esta implementación debido a los siguientes motivos:

- Rápido de implementar y testear (*).
- Se conocía que el cuello de botella de performance no iba a estar ahí (que tiene $O(n^2)$ pero de un N muy chico, 22 nucleótidos aproximadamente que es el tamaño medio de los $_{mi}$RNA), sino en el folding que es $O(n^3)$, con un N dos órdenes de magnitud superior al de las $_{mi}$RNA.

(*) Este principio lo cita Donald Knuth[2], diciendo *"premature optimizations are the mother of all evils"*, lo cual significa que *optimizar antes de tener algo andando, es básicamente la madre de todos los males.*

Por otro lado, para que sea O(n), la manera es ir desplazando el $_{mi}$RNA pero en vez de recorrerlo cada vez que se avanza un nucleótido en el mensajero, solamente sacarle el nucleótido que se descarta, y agregarle el nucleótido que entra. De esta manera, el ciclo completo se hace una sola vez, y luego se opera dos veces por cada desplazamiento (una para el nucleótido que se va y otra para el que entra). Implementarlo de esta manera requería una estructura de todo el programa mucho menos imperativa/secuencial, sino que había que hacer una especie de pipeline como tienen los microprocesadores.

[1] Visitar el SRS del proyecto para más información: `www.r-emo.googlecode.com`

[2] Donald Ervin Knuth, es uno de los más reconocidos expertos en Ciencias de la Computación por su fructífera investigación dentro del análisis de algoritmos y compiladores. Es Profesor Emérito de la Universidad de Stanford.

```
try
{
    Para todo mRNA
    {
        validar secuencia
        obtener mayor seccion codificante
        humanizar mayor seccion codificante
        restablecer secuencia original

        generar archivo de salida
        Para todo miRNA
        {
            obtener nombre de miRNA
            agregar comparaciones al archivo
        }
        restablecer miRNA
    }
}
catch (...)
{
    ...
}
```

Código 7.3: Pseudo-código $_m$RNA vs. $_{mi}$RNA.

6.2.3. Formalizando

Luego de analizar los resultados intermedios que se obtuvieron con el método Ad-hoc, se concluyó que el problema es mucho más grande de lo que se pensaba inicialmente, con lo que, nuestra aproximación era demasiada poco precisa, aunque como primer aproximación sirvió.

Por tal motivo se comenzaron a emplear herramientas más avanzadas, como los son las herramientas de hibridación (subsección 4.2.5), dado que las mismas cuentan con validaciones empíricas y son confiables.

6.3. El Framework FuD

Con el objetivo de distribuir el trabajo en diversos nodos de computación para lograr menores tiempos de espera en los resultado se decidió emplear el framework **FuD** (del acrónico *FuDePAN ubiquitous Distribution*). **FuD** es

un framework para la distribución de trabajos, o un framework para la implementación de aplicaciones distribuidas[Biset, 2009]. Como tal, no depende del problema a implementar y no fuerza ningún modelo de comunicación o disposición de las unidades de procesamiento que serán utilizadas.

En términos generales, **FuD** se organiza como un esquema *Master-Worker* (un servidor y varios clientes conectados al mismo). Tanto cliente como servidor, se encuentran organizados en tres niveles bien separados (figura 6.1), cada uno de ellos con una única responsabilidad claramente definida. La comunicación entre los diferentes niveles se encuentra estrictamente limitada, es decir, por cada nivel existe un único punto de comunicación ya sea para comunicarse con la capa superior o con la inferior. Cuando un mensaje es creado, éste debe atravesar las diferentes capas comenzando desde la de nivel más alto hacia la capa inferior, y luego recorrer en sentido contrario las capas del lado. En el apéndice F se describen brevemente cada una de las capas que conforman este framework.

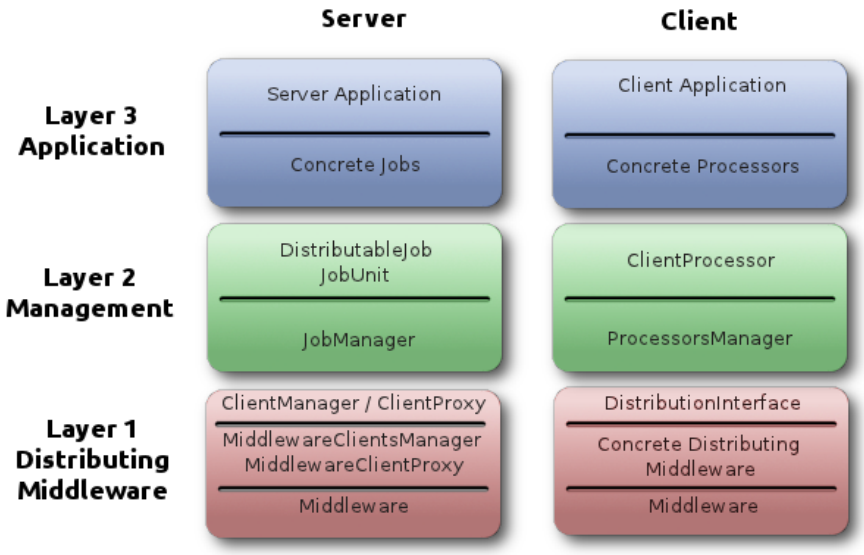


Figura 6.1: Vista abstracta de las capas de **FuD** [16].

6.3.1. Conceptos Importantes

Los siguientes conceptos son necesarios para entender el funcionamiento de una aplicación que usa FuD:

- *Cliente:* un cliente es una aplicación conectada al servidor y es el encargado de realizar las tareas de procesamiento.
- *Trabajo:* un trabajo es cualquier tarea a realizar por la aplicación (también llamado trabajo distribuible). Los mismos se pueden subdividir en unidades de trabajo.
- *Unidad de Trabajo:* representa un cómputo concreto y pertenece a un trabajo particular. No se puede dividir.
- *Manejador de clientes:* módulo encargado de manejar los clientes conectados al servidor.
- *Manejador de trabajos:* módulo de **FuD** encargado de llevar cuenta de los trabajos (de ambos tipos) que maneja el framework en un momento dado.

Como se mencionó anteriormente el cuello de botella en cuanto a complejidad se encontraba en el folding de las secuencias. Las implementaciones del algoritmo de Zuker (UNAFold y RNAFold) no son distribuibles naturalmente, es decir, no se obtiene el mismo resultado al foldear una secuencia completa que al dividirla en tramos, foldear cada uno de los tramos, y unificar los resultados. Por lo cual, el trabajo que se distribuye es el folding de secuencias (original y humanizada), es decir, cada nodo será encargado de foldear cierta secuencia. Dado a que **Remo** necesitaba ser puesto a prueba con un gran número de secuencias para que los resultados sean fiables y consistentes, el uso de **FuD** benefició notoriamente los tiempo de espera de resultados.

6.4. Métricas de Código

Para analizar el código estáticamente se usaron herramientas como CLOC[3] (Count Lines of Code) y CCCC[4] (***C*** and ***C++*** Code Counter). En esta sección se describen métricas de código generales obtenidas por ambas herramientas y se analizan los resultados obtenidos.

6.4.1. Métricas de Remo

Remo esta compuesto por 21 archivos con 3002 líneas de texto a la fecha de publicación del presente documento, dicha cantidad puede variar en el tiempo debido a posibles modificaciones, extensiones o mantenimiento.

[3]http://cloc.sourceforge.net/
[4]http://cccc.sourceforge.net/

El cuadro 6.1 resume los resultados obtenidos después de correr la herramienta CLOC sobre los archivos de **Remo**.

Files		Line Types			%
Type	**Count**	**Blank**	**Comment**	**Source**	**#Comms./Tot.**
`C++ source`	10	161	553	1108	33.2
`C++ header`	11	150	757	273	73.4
Total	21	311	1310	1381	48.6

Cuadro 6.1: Resultados de CLOC para **Remo** [2].

La herramienta CCCC proporciona un reporte completo sobre las métricas de código, incluyendo métricas de diseño Orientado a Objetos y todo tipo de información relevante en cuanto a código. Un análisis exhaustivo de los resultados de estas métricas se encuentra fuera del alcance del presente trabajo, por lo cual en el cuadro 6.2 se muestra un pequeño resumen de los mismos.

Metric	Tag	Overall	Per Module
Number of modules	NOM	31	
Lines of Code	LOC	342	11.032
McCabe's Cyclomatic Number	MVG	8	0.258
Lines of Comment	COM	787	25.387
LOC/COM	L_C	0.435	
MVG/COM	M_C	0.010	
Information Flow measure (inclusive)	IF4	0	0
Information Flow measure (visible)	IF4v	0	0
Information Flow measure (concrete)	IF4c	0	0
Lines of Code rejected by parser	REJ	49	

Cuadro 6.2: Resultados de CCCC para **Remo** [2].

Para una descripción de Número Ciclomático de McCabe ver [McCabe, 1976].

6.4.2. Métricas de Fideo

Fideo esta compuesto por 30 archivos con 3540 líneas de texto a la fecha de publicación del presente documento, dicha cantidad puede variar en el tiempo debido a posibles modificaciones, extensiones o mantenimiento.

El cuadro 6.3 resume los resultados obtenidos después de correr la herramienta CLOC sobre los archivos de fideo.

Files		Line Types			%
Type	**Count**	**Blank**	**Comment**	**Source**	**#Comms./Tot.**
`C++ source`	10	161	344	967	26.2
`C++ header`	20	243	1222	603	66.9
Total	30	404	1566	1570	49.9

Cuadro 6.3: Resultados de CLOC para fideo [2].

6.4.3. Métricas de Acuoso

Acuoso esta compuesto por 7 archivos con 663 líneas de texto a la fecha de publicación del presente documento, dicha cantidad puede variar en el tiempo debido a posibles modificaciones, extensiones o mantenimiento.

El cuadro 6.4 resume los resultados obtenidos después de correr la herramienta CLOC sobre los archivos de acuoso.

Files		Line Types			%
Type	**Count**	**Blank**	**Comment**	**Source**	**#Comms./Tot.**
`C++ source`	2	26	72	160	31
`C++ header`	5	46	247	112	68
Total	7	72	319	272	53

Cuadro 6.4: Resultados de CLOC para acuoso [2].

6.4.4. Métricas de Etilico

Etilico esta compuesto por 8 archivos con 624 líneas de texto a la fecha de publicación del presente documento, dicha cantidad puede variar en el tiempo debido a posibles modificaciones, extensiones o mantenimiento.

El cuadro 6.5 resume los resultados obtenidos después de correr la herramienta CLOC sobre los archivos de etilico.

Files		Line Types			%
Type	**Count**	**Blank**	**Comment**	**Source**	**#Comms./Tot.**
`C++ source`	2	18	67	145	31.6
`C++ header`	6	64	260	152	63.1
Total	8	82	327	297	52.4

Cuadro 6.5: Resultados de CLOC para etilico [2].

6.4.5. Métricas de Remo con Fud

Server

Remo server esta compuesto por 8 archivos con 803 líneas de texto a la fecha de publicación del presente documento, dicha cantidad puede variar en el tiempo debido a posibles modificaciones, extensiones o mantenimiento.

El cuadro 6.6 resume los resultados obtenidos después de correr la herramienta CLOC sobre los archivos correspondientes a remo server.

Files		Line Types			%
Type	**Count**	**Blank**	**Comment**	**Source**	**#Comms./Tot.**
`C++ source`	4	59	143	387	0.36
`C++ header`	4	45	129	144	0.47
Total	8	104	272	531	0.33

Cuadro 6.6: Resultados de CLOC para remo server [2].

Client

Remo client esta compuesto por 4 archivos con 285 líneas de texto a la fecha de publicación del presente documento, dicha cantidad puede variar en el tiempo debido a posibles modificaciones, extensiones o mantenimiento.

El cuadro 6.7 resume los resultados obtenidos después de correr la herramienta CLOC sobre los archivos correspondientes a un remo client.

6.4.6. Análisis de las Métricas

Un dato particularmente interesante sobre los resultados de CLOC, es la cantidad de líneas de comentarios y su porcentaje con respecto al total de líneas de código efectivas. La siguiente fórmula describe la relación comentarios/código y la misma fue usada para calcular la última columna de la tablas exhibidas anteriormente:

Files		Line Types			%
Type	**Count**	**Blank**	**Comment**	**Source**	**#Comms./Tot.**
`C++ source`	2	24	68	94	0.41
`C++ header`	2	21	84	39	0.68
Total	4	45	152	133	0.46

Cuadro 6.7: Resultados de CLOC para un remo client [2].

$$\frac{\#comment\_lines}{\#comment\_lines + \#code\_lines}$$

Este valor ronda aproximadamente el 0.50, lo que significa que hay casi la misma cantidad de líneas de comentarios que líneas de código. Este porcentaje de líneas de comentarios se debe, en mayor medida, a que por cada archivo (por más pequeño que sea) se incluye una cabecera ("header") definiendo ciertos detalles del archivo tal como se observa en el cuadro 6.9.

Para justificar más la alta cantidad de comentarios, todo componente de software, sean estos clases, estructuras, funciones, atributos, etcétera, tiene una descripción detallada a ser interpretada por *Doxygen* (el cuál incluye perfiles de funciones) para la generación de documentación automática. El ejemplo 6.9 muestra la notación utilizada para Doxygen exhibiendo además el porqué de la alta tasa de comentarios.

```
/**
* @file       CodingSectionObtainer.h
* @brief      CodingSectionObtainer provides the
*             interface that allows get the coding
*             section of sequence
*
* @author Franco Riberi
* @email  fgriberi AT gmail.com
*
* Contents:  Header file for remo providing class
*             CodingSectionObtainer.
*
* System:    remo: RNAemo - RNA research project
* Language:  C++
*
* @date       October 2012
*
* This file is part of remo.
*
* Copyright (C) 2012 - Franco Riberi, FuDePAN.
*
* Remo is free software: you can redistribute it and/or
* modify it under the terms of the GNU General Public
* License as published by the Free Software Foundation,
* either version 3 of the License, or (at your option)
* any later version.
*
* Remo is distributed in the hope that it will be useful
* but WITHOUT ANY WARRANTY; without even the implied
* warranty of MERCHANTABILITY or FITNESS FOR A
* PARTICULAR PURPOSE. See the GNU General Public
* License for more details.
*
* You should have received a copy of the GNU General
* Public License along with Remo. If not, see
* <http://www.gnu.org/licenses/>.
*/
```

Cuadro 6.8: Comentario Doxygen de encabezado de archivo en **Remo** [2].

```
/** @brief Get the coding section
*
* Method for obtaining the largest coding section of a
* given sequence.
* @param src:      original sequence
* @param dest:     larger coding section
* @param posInit: to fill with the starting position
*                 of the largest coding section
* @return void
*/
void getCodingSection(const biopp::NucSequence& src,
                      biopp::AminoSequence& dest,
                      size_t& posInit);
```

Cuadro 6.9: Comentario Doxygen de una función en **Remo** [2].

Capítulo 7

Experimentación y Testing

> No solamente el aprendizaje vulgar, sino también el científico esta sometido pues, a ensayo y error.
>
> **Saturdino de la Torre**

Las pruebas de software tienen por objetivo proporcionar información objetiva e independiente sobre la calidad del producto. Son una actividad más en el proceso de control de calidad. Dependiendo del tipo de pruebas, estas actividades podrán ser implementadas en cualquier momento del proceso de desarrollo. Las pruebas deben ser independientes, repetibles, portátiles y reutilizables.

7.1. Google C++ Testing Framework

El framework *googletest*, también conocido como *gtest*, se empleó para escribir las pruebas de todo el código implementado, tanto de **Remo** como del resto de las librerías ya mencionadas.

Básicamente soporta detección automática de pruebas, posee un amplio conjunto de aserciones, permite definir aserciones al usuario, generación de reportes XML respecto de los test, entre otras cosas.

Este framework aísla las pruebas mediante la ejecución de cada una de ellas en un objeto diferente. Cuando una prueba falla, *gtest* permite correrla de forma aislada para la depuración rápida.

Se trata de escribir test mediante el uso de aserciones, es decir, declaraciones que comprueban si una condición es verdadera, o falsa. Una afirmación del resultado puede llevar a un éxito, un fracaso no fatal o un fracaso fatal.

Si se produce un fallo fatal, se aborta la función actual, de lo contrario el programa continúa normalmente.

En el Cuadro 7.1 se exhiben las aserciones básicas y algunas aserciones binarias:

Fatal assertion	Nonfatal assertion	Verifies
ASSERT_TRUE(cond)	EXPECT_TRUE(cond)	cond == True
ASSERT_FALSE(cond)	EXPECT_FALSE(cond)	cond == False
ASSERT_EQ(exptd, actual)	EXPECT_EQ(exptd, actual)	exptd == actual
ASSERT_NE(val1, val2)	EXPECT_NE(val1, val2)	val1 ! = val2
ASSERT_LT(val1, val2)	EXPECT_LT(val1, val2)	val1 < val2
ASSERT_LE(val1, val2)	EXPECT_LE(val1, val2)	val1 <= val2
...	...	...

Cuadro 7.1: Aserciones test en *gtest* [2].

Cuando fallan, ASSERT_* produce un fallo fatal y regresa de la función actual, mientras que EXPECT_* no produce un fallo fatal, lo que permite que la función continue en funcionamiento. En cualquiera de los casos, un error de aserción significa la presencia de un error.

En el Cuadro 7.2 se puede observar un ejemplo de prueba empleando gtest.

La regla de oro es usar EXPECT_* cuando se desea que la prueba continúe revelando más errores después del error de la aserción, y utilizar ASSERT_* cuando continuar tras el fracaso no tiene sentido.

Por ejemplo, la aserción de segundo test: ASSERT_TRUE (p != NULL) en el Cuadro 7.2 tiene sentido, ya que se necesita una referencia al backend, si esto no pasara daría lugar a una violación de segmento cuando p es NULL.

Después de definir las diferentes pruebas, se ejecutan con RUN_ALL_TESTS(), que devuelve 0 si todas las pruebas tienen éxito, o 1 en caso contrario. Cuando se invoca, los RUN_ALL_TESTS() sucede lo siguiente:

1. guarda el valor de los flags de prueba.
2. gtest construye los objetos necesarios.
3. inicializa los objetos mediante setUp().
4. ejecuta las pruebas.
5. limpia las pruebas, es decir, destruye los objetos empleados mediante TearDown().

```
TEST(UnaFoldBackendTestSuite1, BasicTest)
{
    const std::string sequence = "AAGGGCT...";
    const biopp::NucSequence seq(sequence);
    biopp::SecStructure secStructure;

    IFold* p = Fold::new_class("UNAFold");
    ASSERT_TRUE(p != NULL);

    EXPECT_NO_THROW(p->fold(seq, true, secStructure));
    delete p;

    EXPECT_EQ(32, secStructure.size());
    EXPECT_TRUE(secStructure.is_circular());

    EXPECT_FALSE(HelperTest::checkDirTmp());
}
```

Cuadro 7.2: Ejemplo gtest [3].

6. restablece los flags con los valores almacenados inicialmente.

7. repite todos los pasos hasta que se hayan ejecutado todas las pruebas.

Para mayor información puede visitar `https://code.google.com/p/googletest/`.

7.2. Cobertura de Código

Se trata de una medida cuantitativa utilizada en pruebas de software que describe el grado de código fuente testeado, lo cuál funciona como una medida indirecta de calidad. Como la cantidad de tests necesarios para garantizar el correcto funcionamiento para todos los *inputs* posibles suele ser demasiado grande, se eligen casos especialmente representativos.

La cobertura de las pruebas fue realizada empleando la herramienta gcov[1].

En las tablas 7.3, 7.4 y 7.5 se muestran las cobertura de código correspondientes a fideo, acuoso y etilico respectivamente.

[1]http://gcc.gnu.org/onlinedocs/gcc/Gcov.html

	Hit	**Total**	**Coverage %**
Lines	666	751	88.7
Functions	153	218	70.2
Branches	882	1750	46.9

Cuadro 7.3: Cobertura Fideo [2].

	Hit	**Total**	**Coverage %**
Lines	108	134	80.6
Functions	26	59	44.1
Branches	141	353	39.9

Cuadro 7.4: Cobertura acuoso [2].

	Hit	**Total**	**Coverage %**
Lines	36	88	40.9
Functions	10	41	24.4
Branches	40	216	18.5

Cuadro 7.5: cobertura etilico.

7.3. Experimentación

Remo fue ejecutado en diferentes oportunidades. Las ejecuciones intermedias se realizaron con el propósito de analizar los resultados obtenidos y verificar que el trabajo realizado hasta el momento cumplía con las especificaciones requeridas. Durante estas ejecuciones se utilizó una pequeña cantidad de secuencias de ${}_{mi}$RNA (50 secuencias) y RNA$_m$ (5 secuencias pequeñas).

Para la ejecución final, se emplearon 1921 secuencias de ${}_{mi}$RNA y 39 secuencias de RNA$_m$. El tamaño promedio de estas últimas es de 9000 nucleótidos aproximadamente. Como se puede intuir, fueron necesarios muchos cálculos, por ejemplo, retomando el método ad-hoc (ver sección 6.2.2), se requieren 1921 x 9000 comparaciones, más las comparaciones necesarias entre nucleótidos en cada desplazamiento del ${}_{mi}$RNA, por cada mensajero. Además es importante recordar, que la secuencia de RNA$_m$ es analizada tanto de forma original como humanizada.

Por otro lado, el folding de los mensajeros fue una tarea sumamente costosa computacionalmente, ya que todos los algoritmos existente son de orden

exponencial. Por tal motivo, empleando FuD se distribuyó el trabajo de folding, lo cual permitió que los tiempos de espera disminuyeran notoriamente, aunque aún así bastante elevados, por ejemplo el folding de una secuencia larga (15000 nucleótidos) demoró aproximadamente 9hs.

Finalmente, dado que los archivos generados por **Remo** contienen una gran cantidad de información, demandaría mucho tiempo realizar un análisis archivo por archivo. Por tal motivo, se implementaron pequeños algoritmos con el objetivo de resumir la información de forma estadística y así poder establecer conclusiones al respecto.

Parte III

Conclusiones

Capítulo 8

Conclusiones del Trabajo

> Pones tu pie en el camino y si no cuidas tus pasos, nunca sabes a donde te pueden llevar.
>
> **John Ronald Reuel Tolkien**

8.1. Resultados Obtenidos

Determinar si la divergencia en el uso de codones sinónimos entre virus-huésped contribuye a disminuir la interferencia de los $_{mi}$RNA en la expresión de los mensajeros de origen viral puede ser abordado desde tres enfoques diferentes: análisis de estructuras secundaria, $_{mi}$RNA vs RNAm y proteínas. En el presente trabajo se abordaron los dos primeros enfoques.

Desde *el punto de vista termodinámico*, se contó la cantidad de nucleótidos de cada tipo (Adenina (A), Guanina (G), Citosina (C) y Timina (T)) en las secuencias originales y sus correspondientes humanizadas. Se encontró que la cantidad de G y C en las secuencias humanizadas es mayor que en las secuencias originales, e inversamente, la cantidad de A y T es mayor en las secuencisa originales que en las humanizadas. Con lo cual, las uniones entre G y C son más fuertes que las que existen entre A y T, lo cual tendería a que hubiera más stacks[1] en la secuencia humanizada.

Para que se produzca la unión entre nucleótidos existe una fuerza que se conoce como "fuerza de atracción". Por otro lado, en contra de esa unión

[1]Recordemos que un stack hace referencia a las uniones existentes entre nucleótidos presentes entre dos motif adyacentes cualesquiera dentro una estructura secundaria. Ver sección 1.2.2.

existen "fuerzas de torsión", por ejemplo si existe un loop[2] de 7 u 8 nucleótidos en la estructura secundaria, el mismo no esta contra de la unión, pero si fuera más grande si. Por otro lado, si el loop es muy pequeño hay fuerzas de torsión que tienden a despegar los nucleótidos, lo cual ZukerMickael Zuker es un importante investigador que posee una gran cantidad de papers respecto del tema. Un detalle insteresante, es que se estableció un contacto vía email como producto de un bug encontrado en una de sus herramientas empleadas. lo penaliza restando de las suma de atracción.

Por tanto la unión G-C que es más fuerte, soporta más la fuerza de torsión. Esto explicaría porque se encontraron más stacks en el humanizado. Sin embargo las secuencias humanizadas tienen mayor desproporción entre G-C y entre A-T, por lo cual existen nucleótidos que no van a poder aparearse.

Como resultado de la comparación de estructuras secundarias, se obtuvo que existe mayor cantidad de stacks (de todos los tamaños) en el humanizado que en el original, por lo cual se puede decir que predomina el primer efecto mencionado anteriormente.

Respecto de los estudios de $_{mi}$RNA sobre mensajeros, la comparación muestra que en general los $_{mi}$RNA tienen una mayor capacidad de hibridar con los RNA humanizados que con los originales. Este análisis es genérico para todo el genoma, donde se barre todo el gen. Por otro lado, si sólo se rescatan los valores máximos de score[3] tanto para original como humanizado, se observa el mismo resultado en la mayoría de los casos.

Desde *el punto de vista biológico*, **en los virus estudiados existe un uso de codones divergente respecto de los del huésped humano**. En principio significaría una menor velocidad de síntesis proteica y estos virus estarían en desventaja en la replicación intracelular y de no existir otros factores serían desplazados por mutantes cuyo uso de codones sea más semejante al humano.

Se intentaron varias explicaciones para este hecho. Los virus podrían utilizar esta divergencia como un método de regulación fina de la síntesis proteica o podrían encontrar menor competencia por los RNA_t (RNA de transferencia) utilizados por el RNA_m del huésped. Los resultados demostrados sugieren dos factores alternativos no contemplados en la bibliografía:

1. El RNA viral presente en la naturaleza tiene menos stacks que simulan una cadena doble del RNA que el supuesto virus humanizado. El RNA de doble cadena es característico de los virus, es reconocido por receptores de las células como el MID5 y el TLR 3 (Toll like receptor 3),

[2]Ver sección 1.2.2

[3]Score hace referencia a cantidad de uniones posibles.

este reconocimiento desencadena una serie de reacciones que inducen la síntesis de interferon, una proteína de intensa acción antiviral.

2. El RNA viral presente en la naturaleza tiene menos capacidad de ser reconocidos por $_{mi}$RNA codificados por el huésped que el potencial virus humanizado. Los $_{mi}$RNA son reguladores negativos de la síntesis proteica y por lo tanto este factor podría compensar el uso menos efectivo de los RNA$_t$.

En conjunto, estos resultados aportan a entender la importancia de un uso determinado de codones en la replicación del virus. Tienen además, un potencial uso en el desarrollo de vacunas y en la síntesis de proteínas recombinantes en células eucariotas.

Cabe destacar que en lo mencionado anteriormente se empleó un vocabulario técnico virológico/biológico propio de ambas áreas. La tarea de interpretar los resultados de este trabajo desde el punto de vista computacional puede ser muy dificultoso, dado que se empleó la computación como una herramienta, pero no por ello de menor importancia. El potencial de las ciencias de la computación dió como resultado una respuesta a un problema real, el cual fue especificado y abordado mediante el trabajo interdisciplinar.

En términos generales, se puede decir que no sólo que se logró contrastar la hipótesis planteada inicialmente, sino que surgieron dos nuevas hipótesis a analizar en futuras extensiones del presente trabajo.

8.2. Conclusión Final

Como la mayoría de la actividad humana, el trabajo científico y profesional está enmarcado por las necesidades e ideas del hombre. Dichas ideas requieren de mucho esfuerzo para poder ser llevadas a cabo con éxito, como así también surgen diversos contratiempos y dificultades que oscurecen el punto final, o quizás el principio de algo inesperado, dado que se pueden abrir nuevos caminos no conocidos hasta entonces.

Transitar el proceso que desembolsó en el presente trabajo fue continuamente desafiante, tanto desde el punto de vista humano como profesional.

En lo que respecta a lo profesional, permitió el trabajo interdisciplinar acoplando diferentes ciencias con un objetivo en común. Se formalizó un problema real de índole biológico. Además se adquirieron nuevos conceptos, referentes tanto al dominio del problema como también relacionados al desarrollo de software, tales como el acrónico SOLID, diseño orientado a responsabilidades, depuración, entre otros. Por otro lado se experimentó el desarrollo distribuido con diversas personas situadas en distintos puntos geográficos,

se aplicaron diversos conceptos teóricos estudiados durante la carrera, tales como el uso de patrones de diseño en un contexto real, ingeniería inversa, etcétera.

Desde el punto de vista personal, se adquirió gran destreza para trabajar sobre código ajeno, detección, reporte y resolución de bugs, manejo de trackers, etcétera.

Personalmente, en mi carácter de autor del presente escrito me gustaría destacar la calidez humano de las personas que trabajaron a mi lado, tanto de forma directa como indirecta. Pienso que esta conclusión no es un cierre a este trabajo, sino una puerta que se abre, quedando a pie de cañón realizar el mismo estudio a nivel proteína, lo cual aumenta aún más la complejidad dada la estructura de las mismas.

8.3. Logros

Este proyecto de trabajo final fue presentado previamente en dos oportunidades:

- ***Relationship between divergence of using synonymous codons in host-virus and the presence of microRNA.*** Franco Riberi, Laura Tardivo, Lucia Fazzi, Guillermo Biset, Daniel Gutson, Daniel Rabinovich. En 3CAB^{2}C (3er. Congreso Argentino de Bioinformática y Biología Computacional).
- ***Estudio de la relación entre divergencia en el uso de codones del virus respecto al huésped y reconocimiento por los microRNA.*** Riberi F, Fazzi L, Tardivo L, Gutson D, Rabinovich D. En XXXII Reunión Científica Anual (SAV2012-Sociedad Argentina de Virología).

Además, **Remo** fue abalado por Secretaria de Ciencia y Técnica de la UNRC otorgando la beca TICs para la finalización de carrera.

8.4. Aportes

A FuDePAN:

- Remo : `r-emo.googlecode.com`
- Fideo : `fideo.googlecode.com`
- Acuoso : `acuoso.googlecode.com`

- Etilico : `etilico.googlecode.com`
- Otras : aportes en otras librerías tales como mili`mili.googlecode.com` y biopp`biopp.googlecode.com`.

A la comunidad científica:

El principal aporte de este trabajo fue el análisis y la formalización del problema biológico y la posterior propuesta de cómo solucionarlo computacionalmente.

Por otra parte, de alguna manera esto pretende ser un aporte al trabajo interdisciplinario como forma de aplicar los conocimientos científicos en la resolución concreta de problemas que afectan la calidad de vida de las personas.

8.5. Trabajos Futuros

- Agregar nuevos backends para folding e hibridización en *fideo*.
- Agregar nuevos backends para la humanización en *acuoso*.
- Implementar un módulo de control de marco de lectura sobre las secuencias.
- *Realizar el mismo estudia aplicado a proteínas.*

8.6. Repositorio del sistema

Se puede acceder al código fuente de este proyecto y a su documentación visitando `http://r-emo.googlecode.com`.

Bibliografía

[Bellman, 1957] Bellman, R. E. (1957). *Dynamic Programming.*

[Biset, 2009] Biset, G. (2009). *A Framework for Small Distributed Projects and a Protein Clusterer Application.*

[Busch et al., 2008] Busch et al. (2008). IntaRNA: efficient prediction of bacterial sRNA targets incorporating target site accessibility and seed regions. *Bioinformatics*, 24(24):2849–56.

[Curtis H., 2006] Curtis H., Sue Barnes N., S. A. . F. G. (2006). *Biología.* Editorial Médica Panamericana S.A.

[D. K. Hendrix at el., 2006] D. K. Hendrix at el. (2006). RNA structural motifs : building blocks of a modular biomolecule.

[G. Booch and Jacobson, 2005] G. Booch, J. R. and Jacobson, I. (2005). *Unified Modeling Language User Guide, Second Edition.* Santa Barbara: Prentice Hall Professional Technical Reference.

[Gamma et al., 2005] Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (2005). *Design Patterns Elements of Reusable Object-Oriented Software.* Addison-Wesley.

[Gardner and Giegerich, 2004] Gardner, P. P. and Giegerich, R. (2004). A comprehensive comparison of comparative rna structure prediction approaches. *BMC Bioinformatics.*

[Gareth M. Jenkins and Edward C. Holmes., 2003] Gareth M. Jenkins and Edward C. Holmes. (2003). The extent of codon usage bias in human RNA viruses and its evolutionary origin.

[Genebank, 1999] Genebank (1999). Ncbi, national center for biotechnology information. WWW page.

[Hofacker et al., 1994] Hofacker, I. L., Fontana, W., Stadler, P. F., Bonhoeffer, L. S., Tacker, M., and Schuster, P. (1994). Fast folding and comparison of rna secondary structures. *Monatshefte für Chemie*, 125(2).

[Hofacker et al., 1994] Hofacker et al. (1994). Fast Folding and Comparison of RNA Secondary Structures (The Vienna RNA Package). pages 167–188.

[Lamport, 1994] Lamport, L. (1994). *LaTeX: A Document Preparation System*. Addison-Wesley, Reading, Massachusetts.

[Mahajan, 2008] Mahajan, V. S. (2008). Virus-specific host mirnas: antiviral defenses or promoters of persistent infection?

[Martin, 2000] Martin, R. C. (2000). Design principles and design patterns. www.objectmentor.com.

[McCabe, 1976] McCabe, T. (1976). A complexity measure. *IEEE Transactions on Software Engineering*, 2(4):308–320.

[McConnell, 1996] McConnell, S. (1996). *Rapid Development: taming wild software schedules*. Microsoft Press.

[Muckstein et al., 2005] Muckstein et al. (2005). Thermodynamics of RNA-RNA Binding. pages 1–6.

[N.R. Markham & M.Zuker., 2008] N.R. Markham & M.Zuker. (2008). UNAFold: Software for Nucleic Acid Folding and Hybridization.

[Pierce, 2009] Pierce, B. A. (2009). *Genética: Un enfoque conceptual*. Editorial Médica Panamericana.

[Rehmsmeier at el., 2004] Rehmsmeier at el. (2004). Fast and effective prediction of microRNA/target duplexes. pages 1507–1517.

[Richardson et al., 2010] Richardson et al. (2010). GeneDesign 3.0 is an updated synthetic biology toolkit.

[Shaw, 1994] Shaw, G. . (1994). *An Introduction to Software Architecture*.

[Solomon, 1998] Solomon, E. P. (1998). *Biología de Ville*.

[Stroustrup, 1997] Stroustrup, B. (1997). *The C++ Programming Language, Third Edition.* Addison-Wesley, Murray Hill, New Jersey.

[Tamarin, 1996] Tamarin, R. H. (1996). *Principios de Genética*. Editorial Reverté S.A.

[U. Muckstein & H. Tafer., 2006] U. Muckstein & H. Tafer. (2006). Thermodynamics of RNA-RNA Interaction.

[Welch, et al., 2011] Welch, et al. (2011). Designing genes for successful protein expression. pages 43–66.

[Wikipedia, 2013] Wikipedia (2013). LaTeX.

[Winston Royce, 1970] Winston Royce (1970). Managing the Development of Large Software Systems.

[Wirfs-Brock and McKean, 2002] Wirfs-Brock, R. and McKean, A. (2002). *Object Design Roles, Responsibilities and Collaborations*. Addison Weasley.

[YEUNG, 2005] YEUNG, M. L. (2005). sirna, mirna and hiv: promises and challenges.

[Zuker and Stiegler, 1981] Zuker, M. and Stiegler, P. (1981). Optimal computer folding of large rna sequences using thermodynamics and auxiliary information. *Nucleic Acids Research*, 9(1).

[Zuker, Michael, 2006] Zuker, Michael (2006). Computational Methods for RNA Secondary Structure.

Fuente de Imágenes

[1] Estructura de una célula: https://www.google.com.ar/search?q=estructura+de+una+celula.
[2] Elaboración propia.
[3] Taxonomía de las especies. http://www.ceibal.edu.uy/UserFiles/P0001/ODEA/ORIGINAL/BIOPELIGRO.elp/tax.jpg
[4] Estructura de un Nucleótido. http://www.cobach-elr.com/academias/quimicas/biologia/biologia/curtis/libro/img/3-29.jpg
[5] Código genético. http://ciencias4eso.byethost22.com/Joomla/images/stories/codigo.gif
[6] Cadenas antiparalelas. http://genemol.org/biomolespa/la-molecula-de-adn/adn-01.jpg
[7] Molécula de RNA y DNA. https://upload.wikimedia.org/wikipedia/commons/thumb/0/0e/Difference_DNA_RNA-ES.svg/450px-Difference_DNA_RNA-ES.svg.png
[8] Estructura molecular de las proteínas. http://4.bp.blogspot.com/_4ww6723C4UE/TBbKBSF-6xI/AAAAAAAAAE0/12M_6XTEU88/s1600/estructuraproteinasbl9.png
[9] Estructura secundaria del DNA. http://o.quizlet.com/FQEKPJNVM3YIdzpH7yR93g.png
[10] Replicación semi conservativa. http://html.rincondelvago.com/000769161.png
[11] El Dogma Central. http://www.cbs.dtu.dk/staff/dave/transcript_transl.gif
[12] Transcripción del RNA. http://3.bp.blogspot.com/_nZqQMbHLG0s/TStZKTwDTSI/AAAAAAAAAQI/UTdKgbi6WKs/s640/Transcripcion_ADN.jpg
[13] Molécula de $_t$RNA. http://compbio.pbworks.com/f/tRNA_MBC.gif
[14] Excepciones al Dogma Central. http://pendientedemigracion.ucm.es/info/genetica/grupod/Transcripcion/DOGMA2.JPG
[15] Estructura Secundaria Compleja. http://bifi.es/~jsancho/estructuramacromoleculas/13RNA/ribosomasecundaria2.JPG
[16] [Biset, 2009]
[17] Proyecto vac-o. vac-o.googlecode.com

[18] Ejemplo Patrón Mediator. http://sourcemaking.com/design_patterns/.
[19] Modelo estructural Watson-Crick. http://www.cobach-elr.com/academias/quimicas/biologia/biologia/curtis/libro/img/14-10.jpg
[20] Clasificación de RNA. https://www.google.com.ar/search?q=Clasificacin+de+la+molcula+de+RNA.png

[21] Singleton. http://www.cristalab.com/tutoriales/patrones-de-diseno-creacionales-c99932l/
[22] Factory Registry. http://l4c.me/fullsize/tres-1-1313636385.png
[23] Observer. https://www.google.com.ar/search?q=observer+pattern+design/observer.png
[24] Diagrama de secuencias Observer. http://www.google.com.ar/imgres?imgurl=http://unpocodejava.files.wordpress.com/
[25] Template Method. https://www.google.com.ar/search?q=template+method/templatem.png

Appendices

Apéndice A

Modelo propuesto por Watson y Crick

James Watson y Francis Crick dedujeron un modelo estructural tridimensional para el DNA a partir de datos existentes a cerca del mismo. Los datos conocidos era:

1. Se sabía que la molécula de DNA era muy grande, también muy larga y delgada, y que estaba compuesta de nucleótidos que contenían las bases nitrogenadas: adenina, guanina, timina y citosina.

2. De acuerdo con la hipótesis de Levene, se suponía que estos nucleótidos estaban ensamblados en unidades repetidas de cuatro.

3. El químico L. Pauling había propuesto en 1950, que las cadenas de aminoácidos que componen las proteínas están dispuestas a menudo en forma de hélice y que se mantiene así por puentes de hidrógeno. Pauling había sugerido que la estructura del DNA podía ser similar.

4. Los físicos Maurice Wilkins y Rosalind Franklin habían aplicado la técnica de difracción de rayos X al estudio del DNA. Las fotografías obtenidas mostraban patrones que casi con certeza reflejaban los giros de una hélice gigante.

5. Datos que indicaban que, dentro del error experimental, la cantidad de adenina (A) es igual que la de timina (T) y que la de guanina (G) es igual que la de citosina (C): A=T y G=C. (Chargaff)

A partir de estos datos, algunos de ellos contradictorios, Watson y Crick intentaron construir un modelo de DNA que concordara con los hechos conocidos y explicara su papel biológico. Los dos científicos fueron capaces de

deducir que el DNA es una doble hélice, entrelazada y sumamente larga. Algunas de las conclusiones fueron:

- Las cadenas tienen dirección y corren en direcciones opuestas, es decir, la dirección desde el extremo 5' al 3' de cada cadena es opuesta y se dice que las cadenas son *antiparalelas*.

- Los nucleótidos situados en cualquiera de las cadenas de la doble hélice podían acoplarse en cualquier orden o secuencia, aunque su secuencia determina el orden de los nucleótidos de la otra cadena. Esto es necesario, porque las cadenas son complementarias. Por ejemplo: la cadena complementaria de la secuencia **(5')- T T C A G T A C A T T G C C A-(3')** tiene que tener la cadena de nucleótidos **(3')- A A G T C A T G T A A C G G T-(5')**.

- No solamente las purinas no podrían aparearse con purinas, ni las pirimidinas con pirimidinas, sino que, a causa de las estructuras particulares de las bases, la adenina sólo podía aparearse con la timina, formando dos puentes de hidrógeno (A=T) y la guanina solamente con la citosina, formando tres puentes de hidrógeno (G=C). Las bases apareadas eran complementarias.

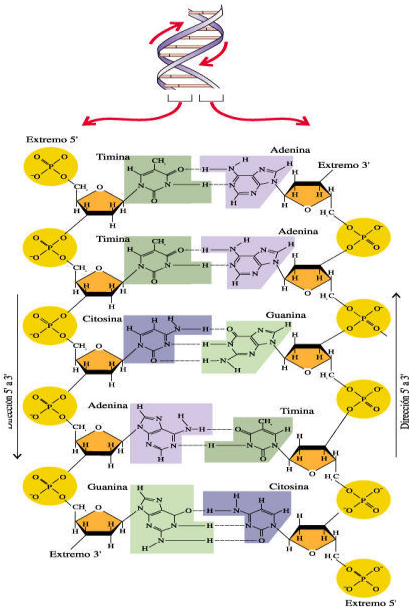


Figura A.1: Modelo estructural Watson-Crick [19].

Apéndice B

Clasificación de RNA

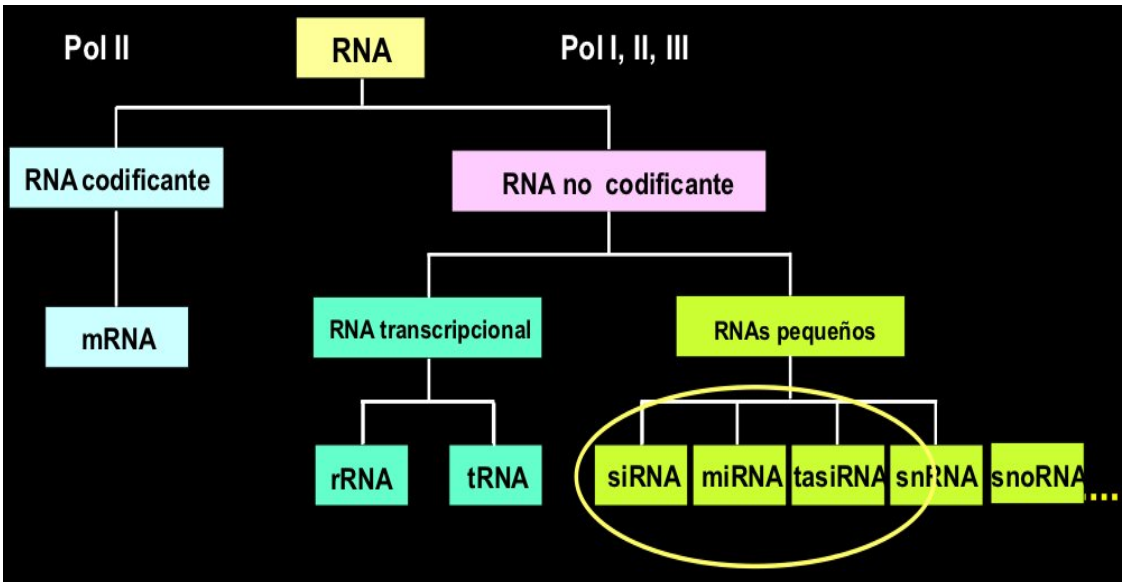


Figura B.1: Clasificación de la molécula de RNA [20].

Apéndice C

Formato FASTA

FASTA es un formato de archivo informático basado en texto, utilizado para representar secuencias de ácidos nucleicos, o de péptido, y en el que los pares de bases o los aminoácidos se representan usando códigos de una única letra. El formato también permite incluir nombres de secuencias y comentarios que preceden a las secuencias en sí.

Una secuencia bajo formato FASTA comienza con una descripción en una única línea (línea de cabecera), seguida por líneas de datos de secuencia. La línea de descripción se distingue de los datos de secuencia por un símbolo ">". La palabra siguiente a este símbolo es el identificador de la secuencia, y el resto de la línea es la descripción (ambos son opcionales). No debería existir espacio entre el ">" y la primera letra del identificador. La secuencia termina si aparece otra línea comenzando con el símbolo ">", lo cual indica el comienzo de otra secuencia.

A continuación se exhibe un ejemplo de una secuencia en tal formato:

```
>gi|5524211|gb|AAD44166.1| cytochrome b [Elephas maximus maximus]
LCLYTHIGRNIYYGSYLYSETWNTGIMLLLITMATAFMGYVLPWGQMSFWGATVITNLFSAIPYIGTNLV
EWIWGGFSVDKATLNRFFAFHFILPFTMVALAGVHLTFLHETGSNNPLGLTSDSDKIPFHPYYTIKDFLG
LLILILLLLLLALLSPDMLGDPDNHMPADPLNTPLHIKPEWYFLFAYAILRSVPNKLGGVLALFLSIVIL
GLMPFLHTSKHRSMMLRPLSQALFWTLTMDLLTLTWIGSQPVEYPYTIIGQMASILYFSIILAFLPIAGX
IENY
```

C.1. Representación de secuencias

Cada línea de una secuencia debería tener algo menos de 80 caracteres. Las secuencia pueden corresponder a secuencias de proteínas o de ácidos nucleicos, y pueden contener huecos (o gaps) o caracteres de alineamiento.

- Los codones que codifican a un mismo aminoácido son:

Aminoácido	Nucleótidos
Ala (A)	GCU, GCC, GCA, GCG
Arg (R)	CGU, CGC, CGA, CGG, AGA, AGG
Asn (N)	AAU, AAC
Asp (D)	GAU, GAC
Cys (C)	UGU, UGC
Gin (Q)	CAA, CAG
Glu (E)	GAA, GAG
Gly (G)	GGU, GGC, GGA, GGG
His (H)	CAU, CAC
IIe (I)	AUU, AUC, AUA
Leu (L)	UUA, UUG, CUU, CUC, CUA, CUG
Val (V)	GUU, GUC, GUA, GUG
Tyr (Y)	UAU, UAC
Trp (W)	UGG
Thr (T)	ACU, ACC, ACA, ACG
Ser (S)	UCU, UCC, UCA, UCG, AGU, AGC
Sec (U)	UGA
Pro (P)	CCU, CCC, CCA, CCG
Phe (F)	UUU, UUC
Met (M)	AUG
Lys (K)	AAA, AAG
Parada (*)	UAG, UGA, UAA
Comienzo	AUG

C.2. Extensión de archivos

No hay una extensión de archivo estándar para un archivo de texto conteniendo secuencias formateadas bajo FASTA. Los archivos de este formato tienen a menudo extensiones como *.fa, .mpfa, .fna, .fsa, .fas o .fasta.*

Apéndice D

Diseño en detalles

D.1. Diagrama de Clases de Remo

D.2. Diagrama de Clases de Fideo

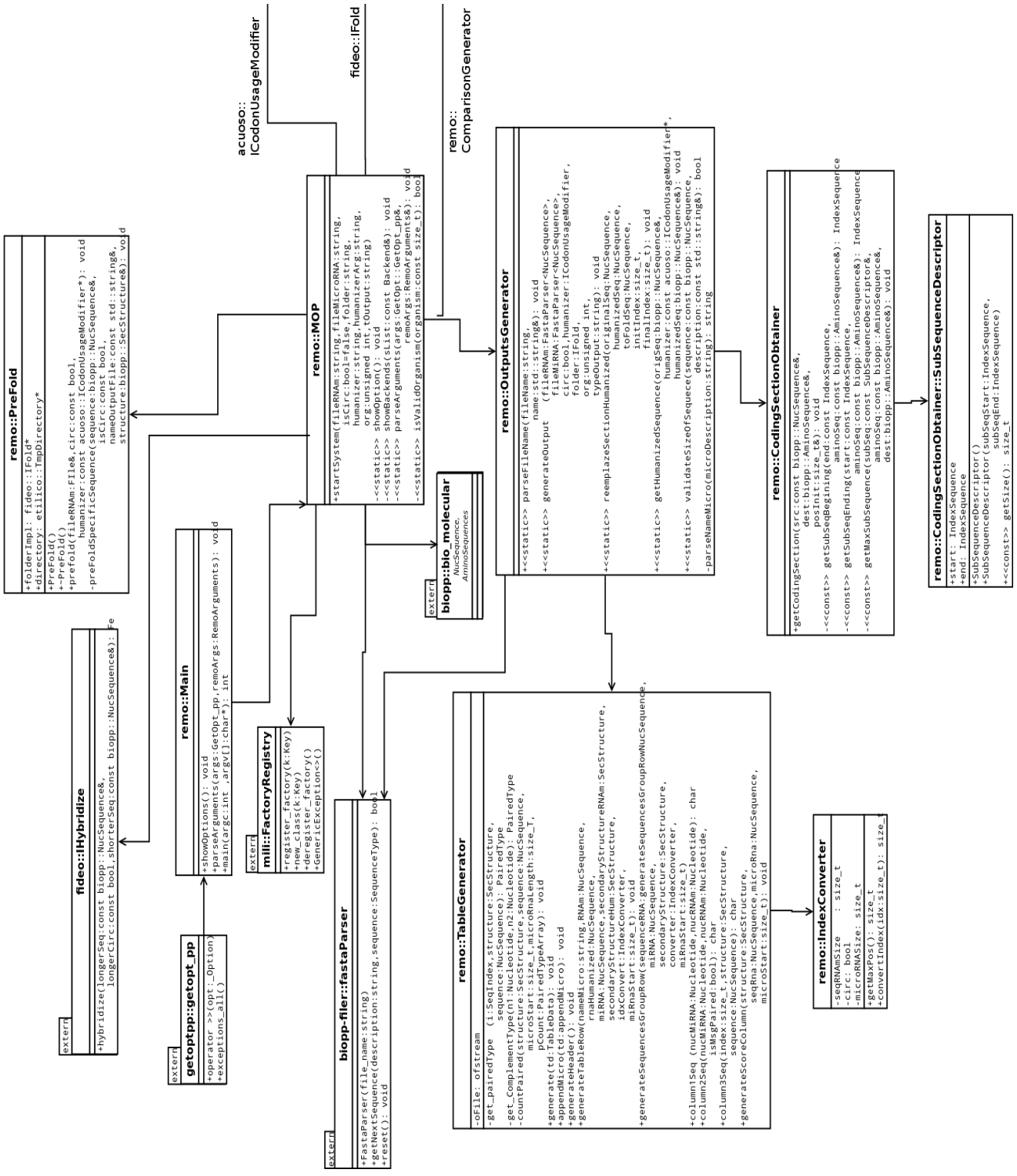


Figura D.1: UML - Diagrama de clases de **Remo**, parte 1 de 2 [2].

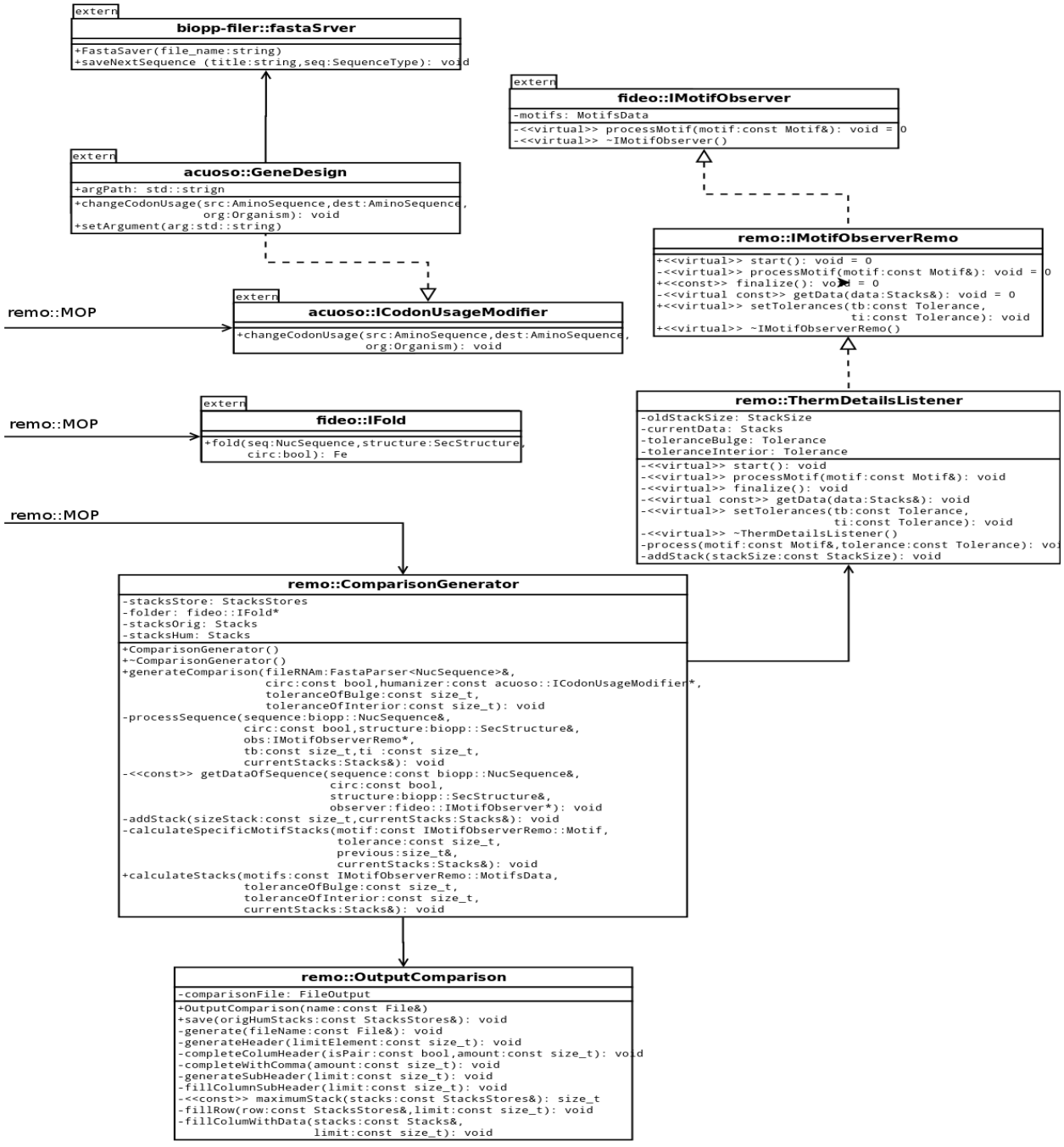


Figura D.2: UML - Diagrama de clases de **Remo**, parte 2 de 2 [2].

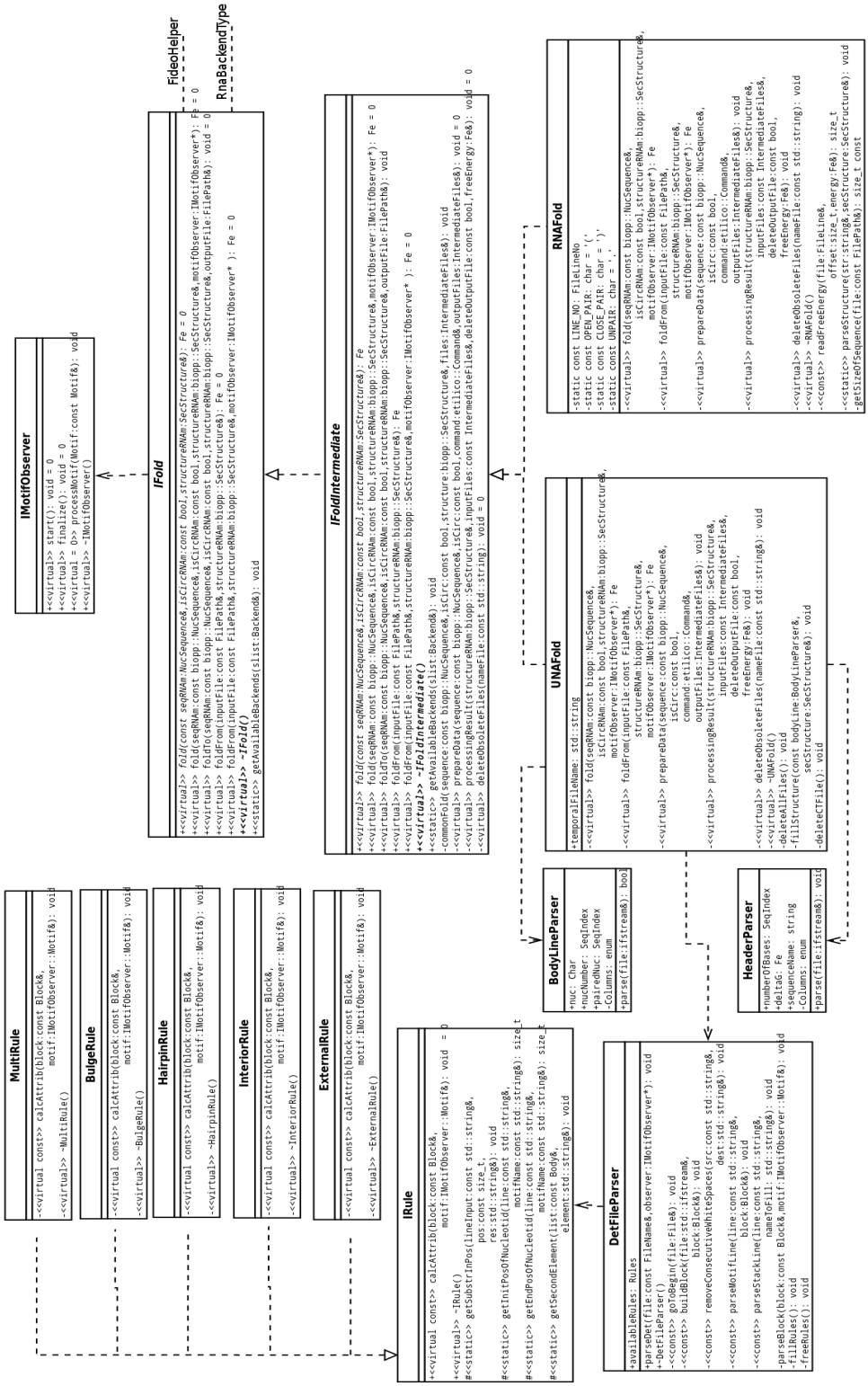


Figura D.3: UML - Diagrama de clases de fideo, parte 1 de 2 [2].

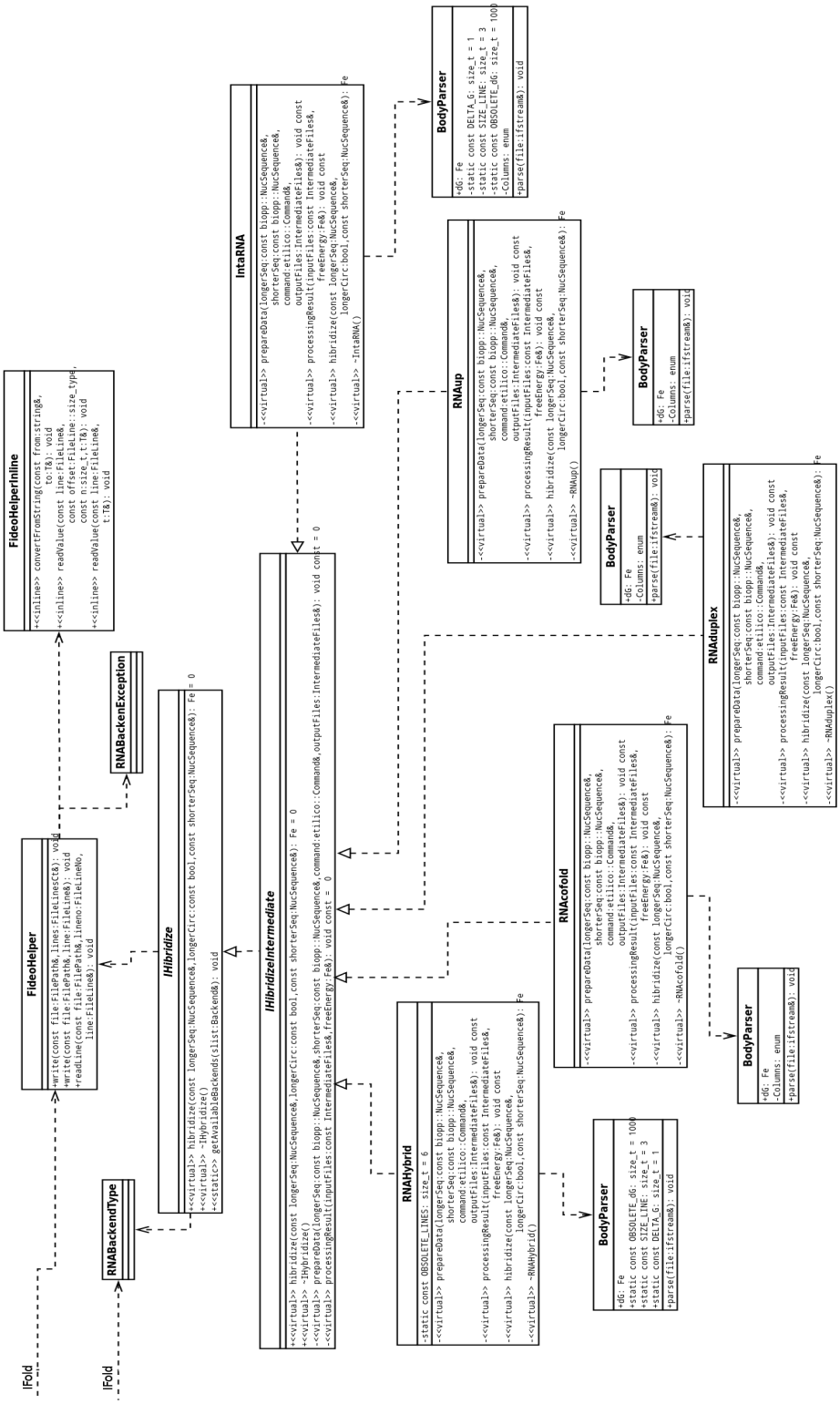


Figura D.4: UML - Diagrama de clases de fideo, parte 2 de 2 [2].

Apéndice E

Patrones de Diseño

E.1. ¿Qué son?

En principio, se puede pensar a un patrón como una manera especialmente inteligente e intuitiva de resolver una clase de problema en particular. Según Christopher Alexander:

"... un patrón describe un problema que sucede una y otra vez en nuestro entorno, y luego describe el núcleo de la solución a ese problema, de tal forma que puede utilizar esa solución un millón de veces más, sin siquiera hacerlo dos veces de la misma manera."

Los patrones de diseño [Gamma et al., 2005] existen independientemente de cualquier implementación particular y pueden implementarse de diversas maneras. Básicamente se clasifican según tres propósitos:

1. *Creacional:* tiene que ver del Cómo se puede crear un objeto. Habitualmente incluye aislar los detalles de la creación del objeto, de forma que su código no dependa de los tipos de objeto que hay y por lo tanto, no es necesario cambiarlo cuando se añade un nuevo tipo de objeto. Ejemplos: *Singleton*, *Abstract Factory*, *Factory Method*, entre otros.

2. *Estructural:* cuando afecta a la manera en que los objetos se conectan con otros objetos para asegurar que los cambios del sistema no requieren cambiar esas conexiones. Ejemplos: *Proxy*, Ejemplos: *Adapter*, entre otros.

3. *Comportamiento:* se emplean cuando los objetos manejan tipos particulares de acciones dentro de un programa. Éstos encapsulan procesos que quiere que se ejecuten, como interpretar un lenguaje, completar una

petición, moverse a través de una secuencia implementar un algoritmo. Ejemplos: *iterator*, *State*, *Observer*, *Strategy*, entre otros.

E.2. Patrones Empleados

E.2.1. Singleton

Corresponde al patrón de diseño más simple de todos, que se aplica cuando sólo se requiere una única instancia de una clase.

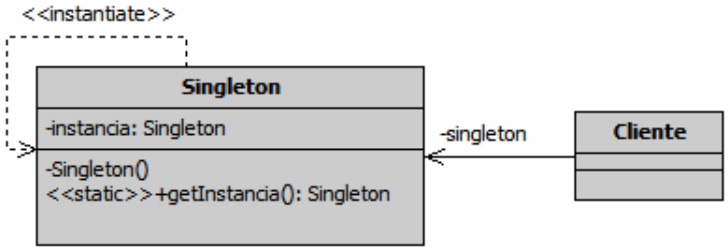


Figura E.1: Patrón Singleton [21].

E.2.2. Factory Method

Consiste en definir una interfaz para crear objetos de tipo genérico permitiendo a las subclases decidir que tipo de objetos concretos crear. Es similar al *Abstract Factory* pero sin el énfasis en las familias. Es un patrón muy útil dado que permite escribir aplicaciones que son más flexibles respecto de los tipos a utilizar difiriendo la creación de las instancias en el sistema a subclases que pueden ser extendidas a medida que se extiende el sistema.

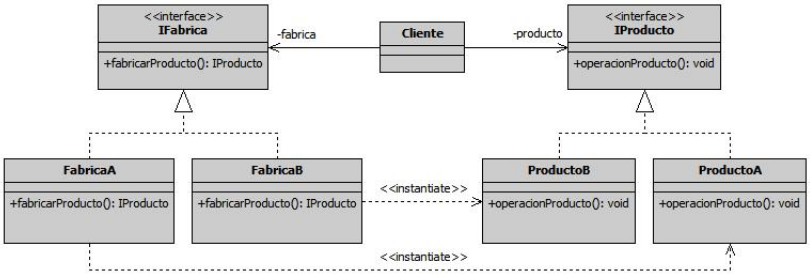


Figura E.2: Patrón Factory Method [22].

```
struct Speaker
{
    virtual void saySomething() = 0;
    virtual ~Speaker(){}
};

class Hello: public Speaker
{
    virtual void saySomething();
};

void Hello::saySomething()
{
    std::cout << "Hello" << std::endl;
}

REGISTER_FACTORIZABLE_CLASS(Speaker, Hello, std::string, "Hello");

class Goodbye: public Speaker
{
    virtual void saySomething();
};

void Goodbye::saySomething()
{
    std::cout << "Goodbye" << std::endl;
}

REGISTER_FACTORIZABLE_CLASS(Speaker, Goodbye, std::string, "Goodbye");

int main()
{
Speaker* speaker = mili::FactoryRegistry<Speaker, std::string>::new_class("Hello
speaker->saySomething();
delete speaker;
return 0;
}

Output: Hello
```

E.2.3. Observer

Define una dependencia del tipo $1- > N$ entre objetos, de tal forma que cuando el objeto cambia de estado, todos sus objetos dependientes son notificados automáticamente. Los objetos observadores se añadan a una lista de objetos, y el objeto observado notificará el cambio a todos los objetos de esta lista cuando se produzca el cambio. Básicamente se aplica cuando se necesita consistencia entre clases relacionadas, pero con independencia, es decir, con un bajo acoplamiento.

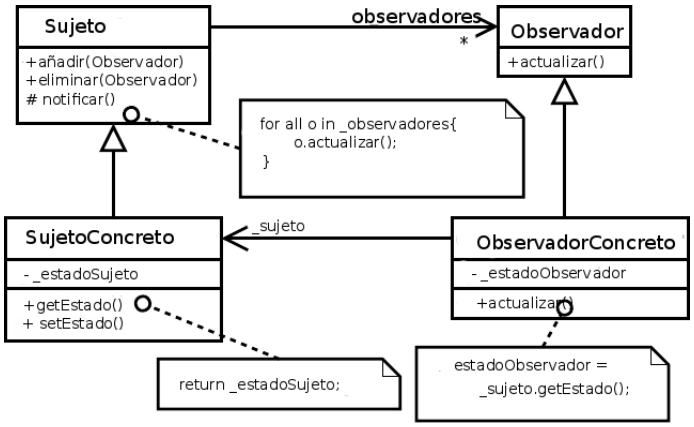


Figura E.3: Patrón Observer [23].

Un funcionamiento básico de este patrón se exhibe en la figura E.4.

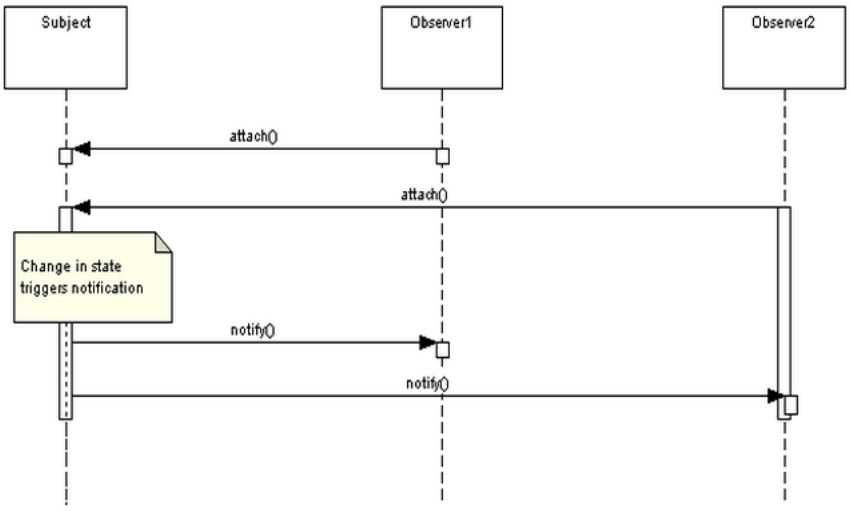


Figura E.4: Diagrama de Secuencias Observer [24].

E.2.4. Template Method

Corresponde a un patrón de diseño de comportamiento. Este tipo de patrón ayuda a resolver problemas de interacción entre clases y objetos. Surge de la necesidad de extender determinados comportamientos dentro de un mismo algoritmo por parte de diferentes entidades. Es decir, diferentes entidades tienen un comportamiento similar pero que difiere en determinados aspectos puntuales en función de la entidad concreta. Una posible solución consiste en copiar el algoritmo en cada de las diferentes entidades cambiando la parte concreta en la que difieren, aunque tiene una consecuencia negativa ya que se genera código duplicado.

La solución que propone el mencionado patrón es abstraer todo el comportamiento que comparten las entidades en una clase (abstracta) de la que, posteriormente, extenderán dichas entidades. Esta superclase definirá un método que contendrá el esqueleto de ese algoritmo común (método plantilla) y delegará determinada responsabilidad en las clases hijas, mediante uno o varios métodos abstractos que deberán implementar. La estructura del patrón se exhibe en la figura E.2.4.

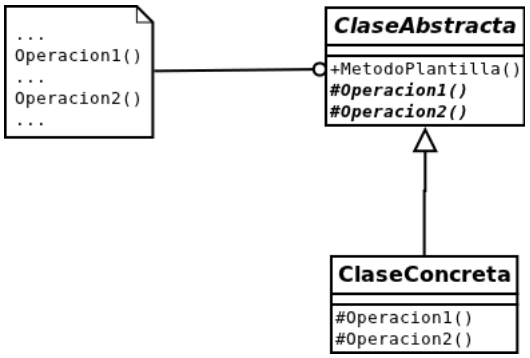


Figura E.5: Template Method [25].

Es un patrón que se adecuada cuando contamos con un algoritmo con varios pasos que no cambian, de modo que dichos pasos invariantes serían implementados en una superclase, dejando la implementación de los pasos que cambian para las subclases. A continuación se muestra un ejemplo del patrón en pseudo-código:

```
class Base
{
  public:
    void execute()
    {
```

```
        a();
        ph1();
        c();
        ph2();
        e();
    }
  private:
    void a() { ... }
    void c() { ... }
    void e() { ... }
    virtual void ph1() = 0;
    virtual void ph2() = 0;
};

class One: public Base
{
    /*virtual*/void ph1() { ... }
    /*virtual*/void ph2() { ... }
};

class Two: public Base
{
    /*virtual*/void ph1() { ... }
    /*virtual*/void ph2() { ... }
};
```

Código D.1: Pseudo-código patrón Template Method.

E.2.5. Mediator

El patrón de diseño *Mediator* está considerado como un patrón de comportamiento. Básicamente encapsula la comunicación entre muchas clases mediante un objeto mediador.

En POO seguramente se tendrán que crear un montón de clases que interactúan entre sí. Si no se aplican ciertos principios, cada objeto dependerá de muchos otros. Con el fin de evitar marcos acoplados, surge el patrón mediador para facilitar la interacción entre los objetos de una manera en la que cada objeto no es conscientes de la existencia de otros objetos.

En la figura E.6 se exhibe un ejemplo de uso presente patrón.

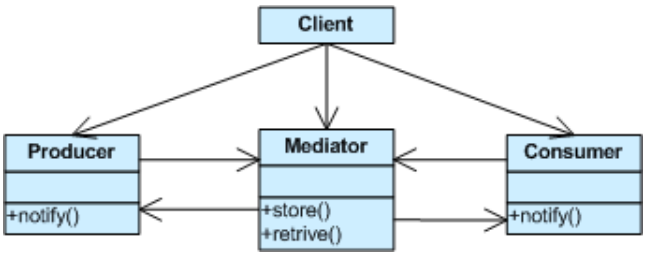


Figura E.6: Ejemplo patrón Mediator [18].

Apéndice F

Detalles de FuD

FuD fue implementado empleando usando un esquema *Divide & Conquer* donde el procesamiento es llevado a cabo por los nodos procesadores. "Divide" cuando se realiza la división de un trabajo en una unidad de trabajo. "Conquer" al incorporar los resultados de una unidad de trabajo.

F.1. Diseño

A continuación se describe brevemente cada una de las capas que constituyen el framework.

F.1.1. Application Layer (L3)

Básicamente proporciona los componentes que contienen todos los aspectos del dominio del problema a resolver. Estos aspectos incluyen todas las definiciones de los datos usados y su manipulación correspondiente, como así también todos los algoritmos relevantes para la solución al problema en general.

Es necesario que del lado del servidor se implemente la aplicación principal, la cual hará uso de una simple interfaz en la abstracción de un trabajo distribuible permitiendo así codificar la estrategia de distribución de trabajos. Del lado cliente, solo se necesita implementar los métodos encargados de realizar las computaciones indicadas por una unidad de trabajo.

F.1.2. Job Management Layer (L2)

Esta capa permite manejar los trabajos que se desean distribuir como así también generar las unidades de trabajo que serán entregadas a los clientes

para su procesamiento. Estas unidades de trabajo llegan a su cliente correspondiente gracias a la capa más baja, encargada de la distribución. Una vez finalizado el procesamiento, se informa que todo ha terminado y otorga los resultados a la capa superior.

F.1.3. Distributing Middleware Layer (L1)

En esta capa existe el único vínculo real entre clientes y servidor. La responsabilidad principal es manejar los clientes conectados al servidor y llevar a cabo los procedimientos de comunicación entre ambos.

Las implementaciones concretas de este nivel son variables y están determinadas por el middleware a utilizar, por ejemplo Boost.Asio[1], MPI[2] o BOINC[3].

Actualmente, *FuD* cuenta con dos capas más, conocidas como *FuD-RecAbs* y *FuD-CombEng*. Estas capas no se describen dado que sólo se empleará **FuD** original, para mayor información consultar `fud.googlecode.com`.

[1] `http://www.boost.org/doc/libs/1\_4\_00/doc/html/boost\_asio.html`
[2] `http://www.mcs.anl.gov/research/projects/mpi/`
[3] `http://boinc.berkeley.edu/`