Área de XML y Web Semántica
Escuela Técnica de Ingeniería Informática

Ontología de Alus

Izaskun Mallona González

Trabajo fin de carrera
Universitat Oberta de Catalunya
2014

Consultor
Joan Anton Perez Braña

Profesor
Jordi Conesa Caralt

Ontología de Alus

Izaskun Mallona González

izaskun.mallona@gmail.com

Resumen

Aproximadamente la mitad del genoma humano consiste en elementos repetitivos, mientras que las secuencias codificantes tan solo representan el 2 %. Los elementos Alu son un linaje de retrotransposones propios de primates que pertenecen a la familia SINE (short interspersed elements). Se ha demostrado su implicación en múltiples procesos biológicos, como el posicionamiento de nucleosomas, el *splicing alternativo* o la generación de lugares de unión a factores de transcripción. No obstante, se desconoce el papel de las Alus y de otros elementos repetitivos en el modelado genómico.

El empleo de vocabularios controlados y ontologías es un campo relativamente reciente en bioinformática, aunque son de uso común algunos vocabularios controlados. Por ejemplo, Sequence Ontology (SO) ofrece una jerarquía de términos y relaciones aplicables a la anotación de datos genómicos; y Gene Ontology (GO) permite describir funciones moleculares, procesos biológicos y localizaciones celulares de genes y sus productos. No obstante, el empleo de razonadores semánticos sobre ontologías bioinformáticas no es tan usual.

En este trabajo se describe una base de conocimiento de las Alus humanas. La ontología incorpora términos SO y GO y está orientada a describir el contexto genómico del conjunto de Alus. Para cada elemento Alu se almacenan el gen y transcrito más cercanos, así como su anotación funcional de acuerdo a GO, el estado de la cromatina circundante y los factores de transcripción presentes en la Alu. Se han incorporado reglas semánticas para facilitar el almacenamiento, consulta e integración de la información. La ontología de Alus es plenamente analizable mediante razonadores como Pellet y está parcialmente transferida a una wiki semántica. Su formalización puede descargarse en `http://gattaca.imppc.org/groups/maplab/imallona/ontologies/merged.owl`.

Palabras clave
ontología, Alu

Lemas adicionales
web semántica, elementos repetitivos, anotación genómica, razonador formal

Alu ontology

Izaskun Mallona González

izaskun.mallona@gmail.com

Abstract

About half of the genome is derived from repetitive elements, whereas coding sequences represent less than the 2 %. Alu elements are a primates-specific lineage of abundant retrotransposons belonging to the short interspersed elements (SINE) family. It has been reported their implication in multiple phenomena such as nucleosome positioning, alternative splicing or the generation of novel transcription factors binding sites. Nonetheless, the role of Alus and other repeats in shaping the genome remains unclear.

The usage of ontologies is a relatively new field in bioinformatics, although some controlled vocabularies are widely used. For instance, Sequence Ontology (SO) offers a hierarchy of concepts and relationships to be used to annotate genomic data; and Gene Ontology (GO) provide a set of terms to describe molecular functions, biological processes and cellular locations of genes and gene products. However, formal reasoning over ontologies are not so widespread.

In this work we describe a knowledgebase for the human Alus. The ontology is linked to SO and GO and is devoted to address the genomic context of the Aluome. For each Alu element, the closest gene and transcript are stored, as well their functional annotation according to GO, the state of the surrounding chromatin and the transcription factors binding sites inside the Alu. Semantic rules have been used to aid efficient data storage, querying and integration. The Alu ontology has been subjected to reasoners like Pellet and it is being transferred to a Semantic MediaWiki. Its formalization is freely available at `http://gattaca.imppc.org/groups/maplab/imallona/ontologies/merged.owl`.

General Terms:
ontology, Alu

Additional Key Words and Phrases:
semantic web, repetitive elements, genomic annotation, formal reasoning

Índice general

Índice de figuras

Índice de tablas

Nomenclatura

CGI Common Gateway Interface

DNA DeoxyriboNucleic Acid

EBI European Bioinformatics Institute

EMBL European Molecular Biology Laboratory

EMBOSS European Molecular Biology Open Software Suite

EST Expressed Sequence Tag

GO Gene Ontology

GPL GNU Public License

HTML Hypertext Markup Language

JDBC Java Database Connectivity

mRNA Messenger RNA

ODBC Open Database Connectivity

ORF Open Reading Frame

PCR Polymerase Chain Reaction

RDMS Relational Database Management System

RNA RiboNucleic Acid

SINE Short INterspersed Elements

SQL Structured Query Language

SO Sequence Ontology

WWW World Wide Web

Capítulo 1

Introducción

1.1. Contexto y justificación

En el ámbito de la genética y biología molecular, el desarrollo de técnicas de secuenciación masiva a bajo coste ha producido un gran aumento de los datos a analizar, lo que ha favorecido la incorporación de técnicas punteras en el análisis de información. Estos métodos abarcan técnicas de almacenamiento, consulta, visualización e integración de datos, entre otros; y no ya actúan como apoyo en ciencia básica, sino que han adoptado una entidad propia y han establecido de nuevas disciplinas científicas, como la bioinformática o la biología computacional.

El desarrollo de nuevo software bioinformático se ha beneficiado de la disponibilidad de inferfaces de programación (APIs) de código libre, como BioPerl (Stajich *et al.*, 2002), BioPython (Cock *et al.*, 2009), BioRuby (Goto *et al.*, 2010) o BioJava (Holland *et al.*, 2008), entre otros. Estas APIs proporcionan un entorno sólido que permite desarrollar con facilidad aplicaciones robustas (Raymond, 1999); y, lo que es más importante, la difusión del código en abierto favorece la reproducibilidad científica (Barnes, 2010; Merali, 2010).

El software bioinformático, por tanto, ha de ser capaz de gestionar grandes volúmenes de datos con el fin de proporcionárselos procesados a sus usuarios, quienes no necesariamente tienen por qué ser capaces de manejar los datos crudos. Por tanto, existe una íntima relación entre los objetivos de la biología computacional y la bioinformática y los perfiles de sus usuarios. Básicamente, los objetivos de la bioinformática son tres: primero, facilitar un almacenamiento, acceso y actualización eficientes de los datos; segundo, desarrollar herramientas y recursos para analizarlos; y tercero, emplear esas herramientas para interpretar los resultados de una forma que tenga sentido biológico (Luscombe *et al.*, 2001). Tales objetivos muestran un gradiente en habilidades: el primero es propio de un perfil informático y el último requiere conocimiento biológico; en cierta manera, el objetivo intermedio supone un equilibrio entre ambos.

Por tanto, la transferencia de tecnologías informáticas al ámbito de la bioinformática es continua. Un campo en expansión es la proyección de las funcionalidades de la web semántica, o del almacenamiento de la información en *knowledge bases* en lugar de bases de datos convencionales. En este sentido, la aproximación semántica favorece la extracción de información de los datos, al tiempo que facilita la interpretación de resultados por parte de los usuarios del software; de hecho, buena parte de los *front-ends* en bioinformática se basan en servicios Web (Stein, 2002).

1.1.1. Minería de datos

El almacenamiento de datos bioinformáticos en repositorios públicos permite su análisis por parte de equipos independientes. Se puede considerar que dicho análisis forma parte de la minería de datos, que es una disciplina que emplea bases de datos a fin de detectar pautas en su contenido (Witten *et al.*, 2011). Este proceso es iterativo, y consiste en la selección automática, preparación y manipulación de datos a fin de obtener resultados no triviales y con valor añadido. Como objetivos, la minería de datos busca: primero, describir grandes cantidades de datos, permitiendo por tanto la generación de información procesable por humanos partiendo de entradas únicamente analizables mediante máquina; y segundo, modelizar, y por tanto construir modelos predictivos, empleando alguna de las variables e interacciones entre ellas (Kantardzic, 2011).

El procedimiento típico de un trabajo de minería de datos en el ámbito bioinformático comienza con una hipótesis y la recolección de datos, lo cual se realiza mayoritariamente mediante sistemas de gestión de bases de datos; el paradigma relacional de Codd (1970) es usual. Después procede el preprocesamiento, incluyendo la selección de atributos; y finalmente, la aplicación de diversas técnicas que permiten extraer la información de interés biológico, que ha de ser validada *a posteriori*. Si el control de calidad del proceso sugiere que el modelo es válido, procede su interpretación y la extracción de conclusiones. En cualquier caso, la hipótesis de partida, los datos o la estrategia de preprocesamiento pueden cambiar y por tanto puede ser necesario repetir el ciclo de minería (Kantardzic, 2011).

1.1.2. Bases de datos biológicas

Tras la adquisición de datos, el paso siguiente es su almacenamiento. En algunos casos, basta con el simple almacenamiento local en un fichero sobre el que un único usuario pueda hacer búsquedas. En otros, en cambio, el volumen de datos o la necesidad de acceso concurrente exigen un enfoque más específico. Las soluciones más sencillas se basan en archivos planos, que usualmente se integran mediante indexación (Fujibuchi *et al.*, 1998; Etzold *et al.*, 1996). No obstante, el uso de sistemas de gestión de bases de datos permite más poder analítico y proporciona controles de consistencia interna. Las bases de datos biológicas clásicamente han estado gestionadas mediante sistemas de bases de datos relacionales. Oracle Database, PostgreSQL, MariaDB o MySQL son ejemplos de estos gestores; usualmente son consultados a través de interfaces como ODBC y JDBC. La

aplicación de técnicas de minería e integración de los datos está muy relacionada con los métodos escogidos para su almacenamiento (Köhler *et al.*, 2003; Stevens *et al.*, 2001); continuamente se incluyen nuevos datos, o se modifican las relaciones entre estos, a fin de extraer más información e incrementar su potencial para la minería. En cierta manera, la explotación de la abundante información presente en las bases de datos biológicas es uno de los campos de mayor interés en bioinformática (Stein *et al.*, 2003).

1.1.3. Web semántica

La Web semántica permite almacenar y extraer metadatos semánticos y ontológicos de la Web y otro tipo de información convenientemente estructurada, lo que conduce a su extracción y análisis automáticos. Estos metadatos albergan contenido *per se* pero además permiten recoger relaciones entre datos (Berners-Lee *et al.*, 2001).

En biología, y especialmente en genética, el almacenamiento de grandes cantidades de datos estructurados de manera que se facilite su procesamiento es una necesidad. No obstante, las bases de datos suelen estar aisladas unas de otras, por lo que la integración resulta en ocasiones difícil. Por tanto, el empleo de ontologías a fin de relacionar las características semánticas es un campo prometedor, permitiendo la generación de nuevo conocimiento.

1.1.4. Almacenamiento de la información genética

A grandes rasgos, buena parte de la información heredable se almacena en los cromosomas; estos están compuestos de DNA, que es un polímero lineal en el que se suceden distintas combinaciones de los nucleótidos usualmente representados mediante las letras *A, C, T, G*.

Supongamos que partimos de una hipotética cadena de, por ejemplo, diez mil nucleótidos: en esta cadena podrían estar codificados, por ejemplo, un gen y sus secuencias reguladoras. Entre otros aspectos, el estado de la cromatina (la *maraña* finamente empaquetada y organizada de DNA que se encuentra en el interior del núcleo) circundante a este gen favorecerá o reprimirá que se exprese (esto es, que dé lugar a uno o más RNA mensajeros, o, lo que es lo mismo, transcritos; y éstos, a su vez, posiblemente a proteínas). La figura 1.1 muestra cómo se empaqueta el DNA en un cromosoma muy *condensado* (metafásico).

Por tanto, puede decirse que la información genética está almacenada en la secuencia del DNA presente en la célula; pero también que su regulación depende no sólo de la secuencia de un determinado elemento sino de su contexto (esto es, del tipo de empaquetamiento que presenta en la cromatina). De esta forma, una determinada secuencia podría influir en otra contigua, aunque no codificaran productos con una misma función biológica.

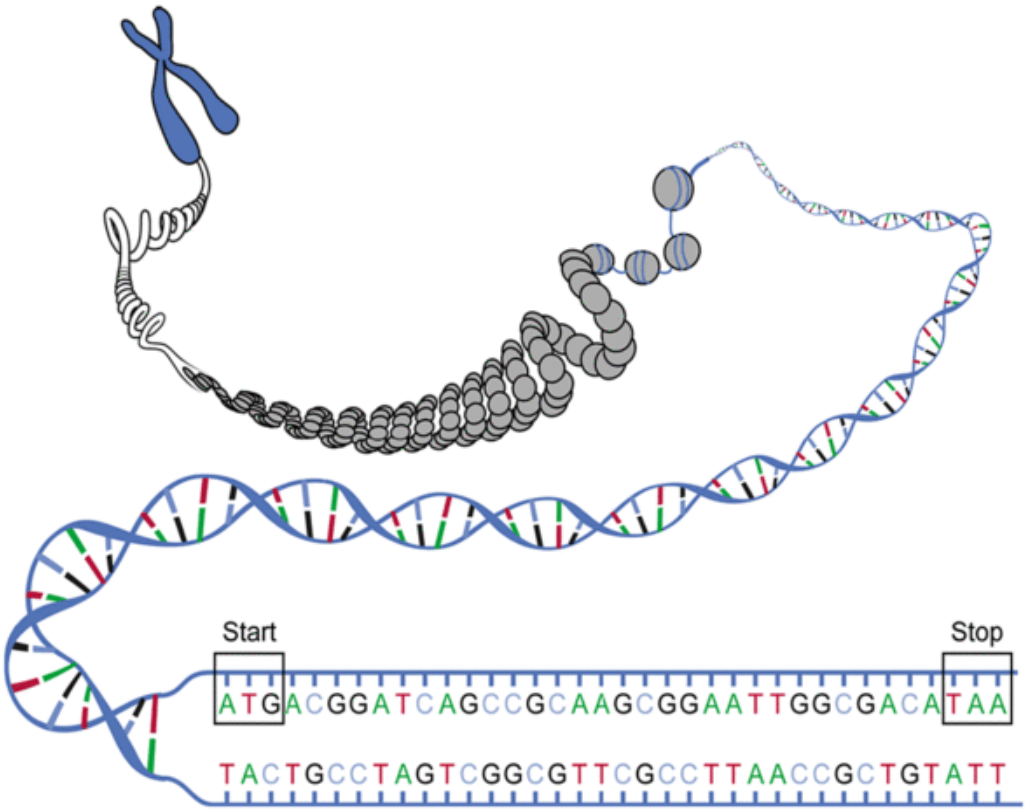


Figura 1.1: Diagrama del modo de empaquetamiento del DNA en un cromosoma metafásico. Se observa en la parte inferior cómo la parte usualmente estudiada (la secuencia) corresponde a un nivel de detalle muy fino; mientras que el cromosoma, en la parte superior, está menos aumentada. Se indica mediante *start* y *end* dónde comienza y acaba una determinada característica; un gen, por ejemplo. Fuente: `http://whatdnatest.blogspot.com.es/2012/01/structure-and-function-of-dna-molecule.html`.

Un método de almacenaje de información genética ampliamente utilizado se basa en la asignación de características a partes del genoma (en el argot, el proceso se denomina *anotación*). Por ejemplo, cada cromosoma puede representarse como una palabra de de longitud N y alfabeto A,C,T,G. Partiendo de que la longitud de la palabra es conocida, es posible definir subcadenas de ésta: por ejemplo, cabe decir que los caracteres situados entre la posición 1 y 1000 del cromosoma son el gen A; y que entre las posiciones 500 y 1500 se produce la *impresión* de una marca epigenética que indica que la zona es de alta actividad transcripcional (esto es, que promueve la manifestación, o expresión génica, de los genes que se encuentran en esa zona).

Dado que estos datos se encuentran disponibles al público mediante bases de datos relacionales mantenidas por consorcios, es posible descargar y procesar los datos y realizar inferencias basadas en características posicionales. Por ejemplo, conociendo el cromosoma y el inicio y final de una determinado elemento (un gen, por ejemplo) es posible realizar una comparación con otro elemento de localización conocida (un estado de la cromatina, por ejemplo) mediante una simple operación aritmética de medida de distancia o de solapamiento. Una explicación más detallada de tal aritmética se ofrece en la sección 3.3.1 presente en el capítulo que discute la implementación de la ontología.

1.1.5. Elementos repetitivos Alu

El 45 % del genoma humano está formado por elementos transponibles p transposones. Tradicionalmente se consideraba que este DNA era *basura* o secuencia parásita. No obstante, se ha descrito que los retrotransposones juegan un papel importante en múltiples procesos biológicos (Muotri *et al.*, 2007). Por ejemplo, presentan un rol fundamental en evolución (Cordaux and Batzer, 2009). Además, su actividad y expresión varían de acuerdo durante el desarrollo y generan diversidad genética (Xing *et al.*, 2009).

Las Alus son un tipo elemento transponible muy abundante: en el genoma humano hay más de un millón de copias y representan el 10 % de todo el ADN. A pesar de su presencia abrumadora en comparación con las regiones codificantes (menos del 2 % del genoma humano), los estudios dirigidos a explorar la complejidad de los procesos de regulación celular les han dedicado poca atención.

La longitud usual de las Alus es de unas 300 pb (pares de bases). Por esta razón se dice que son SINEs, o *short interspersed elements*). La figura 1.2, adaptada de (Batzer and Deininger, 2002), muestra su naturaleza modular: básicamente constan de dos partes, poseen un homopolímero de oligo(dA) en la cola (esto es, una repetición de `A`es) y otro, más pequeño, central; este elemento central está flanqueado por repeticiones directas que juegan un papel en el salto (retrotransposición) del elemento y su integración (Batzer and Deininger, 2002). Cabe resaltar que las Alus son elementos activos en el genoma (esto es, aún "saltan") y que, además, se transcriben *in vivo* (Paolella *et al.*, 1983).

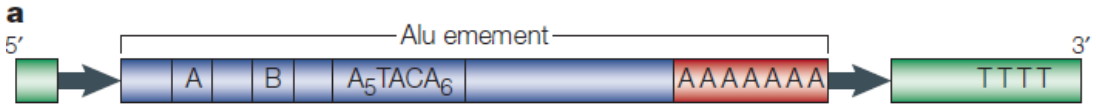


Figura 1.2: Estructura de un elemento Alu según Batzer and Deininger (2002).

1.1.6. Estado del arte

Las ontologías son una herramienta diaria en la genética y bioinformática gracias a la amplia difusión de iniciativas como, por ejemplo, *Gene Ontology* (Ashburner *et al.*, 2000). Esta ontología, de decenas de miles de términos, permite asociar a las *piezas* de información génica (genes, transcritos y proteínas) características de localización celular (ontología CC), proceso biológico (ontología BP) y función molecular (ontología MF). Existen técnicas de simplificación de ontologías [1] y de comparación (usualmente, mediante tests de sobrerrepresentación de términos) (Dennis Jr *et al.*, 2003).

Otro gran grupo de ontologías está especializado en las secuencias biológicas: esto es, características ligadas a los biopolímeros más usuales, como el DNA, que son los soportes biológicos de la información heredable. Un ejemplo paradigmático es *Sequence Ontology* (Eilbeck *et al.*, 2005).

No obstante, se han desarrollado ontologías muy variadas: por ejemplo, cabe destacar las de fenotipado (Bodenreider and Stevens, 2006), que permiten la descripción del aspecto o *fenotipo* de individuos a resultas de sus características internas o *genotipo*).

Por tanto, la integración de datos biológicos empleando ontologías de anotación funciona, secuencias o fenotipado supone un área de extremo interés. Por esta razón se plantea

[1] Los métodos de simplificación de ontologías son diversos. Difieren según la automatización del proceso (total o parcial), el algoritmo de selección de elementos (desde la selección manual a la minería de texto), el formato de partida de la ontología (formalización OWL, diagrama conceptual...), e incluso el rendimiento o reducción real del tamaño de la ontología.

En los trabajos de Kim *et al.* (2007) y Conesa and Olive (2006) se ofrece reducir la ontología empleando la especificación formal explícita de las relaciones (funcionales) entre conceptos; de este modo, se detectan los conceptos superfluos, que se eliminan, y la ontología así procesada posee una dimensión más reducida sin perder su versatilidad.

Cabe destacar que algunos editores de ontologías ya incorporan métodos para extraer módulos de ontologías ya semántica, ya sintácticamente, como se describe en Grau *et al.* (2007).

Básicamente las técnicas de extracción de módulos de ontologías se dividen en dos grandes grupos. Un grupo de publicaciones emplea un criterio taxonómico para ofrecer la detección de elementos aislados (no informativos). Puede decirse por tanto que emplean relaciones de jerarquía más que funcionales (Stuckenschmidt and Klein, 2004). El método de Grau *et al.* (2007) presenta la ventaja de estar incluido en el propio editor SWOOP.

El segundo grupo de trabajos, en cambio, sí que emplean las relaciones funcionales; en este caso, la aproximación depende de la expresividad de la ontología (por ejemplo, si se permite la disyunción de conceptos) (Seidenberg and Rector, 2006).

el desarrollo de una ontología de Alus en el presente trabajo fin de carrera.

1.2. Terminología e interés biológico

Si bien la ontología está enfocada hacia las Alus, también reúne información de otros elementos que permiten caracterizar el sistema. Pese a su naturaleza variada, casi todos ellos comparten un elemento en común: se trata de fragmentos de DNA y, por tanto, se identifican mediante su posición en el genoma, que suele ser única. En términos prácticos esto implica que deben poseer una *localización*; esto es, una terna cromosoma, nucleótido de partida y nucleótido de fin. Usualmente se denomina *coordenada* a dicha localización, que, evidentemente, depende de la especie empleada y de la versión del ensamblado genómico. En la presente ontología los datos proceden del humano (*Homo sapiens*) en su versión hg19 (Dreszer *et al.*, 2012).

En terminología genética, las localizaciones reciben el nombre de *loci*. De acuerdo a la biología molecular básica, dos *loci* solapantes (esto es, con coordenadas cuya intersección produce un intervalo no vacío) presumiblemente guardan relación. En algunos casos esta relación es causal: un *locus* codifica un gen, que se transcribe a un RNA mensajero (también denominado transcrito) y por tanto ambos solapan. En otros casos la relación, causal o no, se detecta mediante métodos estadísticos (Veyrieras *et al.*, 2008).

La figura 1.3 muestra cómo es posible acotar los distintos *loci* sobre una cadena de DNA, y las relaciones que pueden darse entre las coordenadas de los elementos que aparecen asociados físicamente (esto es, en cercanía).

La minería de datos que asocia *loci* de genes, mensajeros, lugares de unión a factores de transcripción, modificaciones de histonas, etc. es un campo en constante desarrollo. Su objetivo es encontrar pautas entre los distintos elementos que permitan comprender cómo se estructura e interpreta la información genética. Las ontologías, como herramienta, permiten una aproximación integradora a este problema (Tiffin *et al.*, 2005).

1.3. Trasfondo

La ontología de Alus ha de contemplar elementos de muy distinta índole, así como las relaciones entre éstos. Básicamente, ha de reunir el vocabulario controlado de conceptos (esto es, el conjunto de elementos instanciables), la jerarquía de conceptos (también denominada taxonomía), las posibles relaciones entre éstos y los axiomas del modelo (Telnarova, 2010). Por poner un ejemplo, cabe disponer de *genes* y *transcritos* y *elementos que intervienen en la miogénesis* (conceptos), reconocer que un transcrito proviene de la transcripción de un gen (relación, jerarquía), que si un gen interviene en miogénesis el transcrito también lo hará (relación, axioma) y que de un único gen es posible obtener

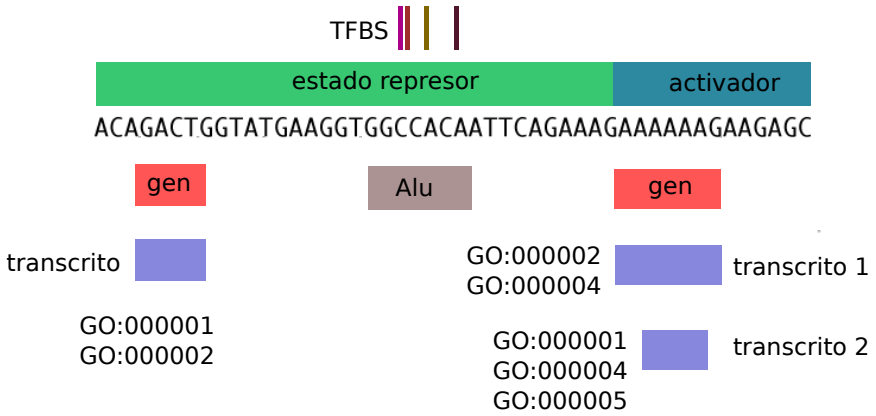


Figura 1.3: Diagrama esquemático de las relaciones entre Alus, genes y funciones. De arriba a abajo: TFBS o lugares de unión a factores de transcripción, que se han incluido para las zonas acotadas por elementos Alu; estados de la cromatina, que proporcionan un código simplificado de la actividad las regiones de la cromatina; genes y Alus, que serán asociados en la ontología de acuerdo a criterios de cercanía; transcritos, que se han modelizado como procedentes de los genes con una cardinalidad de uno o más transcritos por cada gen; y, para cada transcrito, existe un vocabulario de Gene Ontology asociado. Nótese que la longitud de cada elemento respecto de la secuencia que se proporciona como referencia no está a escala.

ninguno, uno o más transcritos (axioma).

Buena parte de los elementos de la ontología, ya sean Alus, genes, transcritos, etc. son *loci* genómicos y, por tanto, poseen por tanto un atributo de localización que es compartido por todos los seres humanos (salvo por los polimorfismos, o variantes, presentes de forma natural en las poblaciones). No obstante, estos elementos pueden sufrir ciertas modificaciones, heredables y de gran importancia en cuanto a fisiología y patología, denominados *cambios epigenéticos*. Estos cambios dependen del tipo de célula (si es nerviosa o epitelial, por ejemplo), de su estado (en proliferación o en senescencia, por ejemplo), etc. Por tanto, los elementos de la ontología que contemplan modificaciones epigenéticas no sólo deberían contener la localización, sino también un atributo *has cell line* que indicase las características del tipo celular en las que se ha descrito la mencionada modificación.

Algunos elementos *estáticos* (por ejemplo, *loci* de genes) y *dinámicos* (por ejemplo, un estado activador o *enhancer* de la cromatina) de la presente ontología son modelizables mediante Sequence Ontology (Eilbeck *et al.*, 2005). La *anotación* (o, a grandes rasgos, atribución de funciones biológicas) de estas secuencias puede realizarse según lo descrito por Gene Ontology (Ashburner *et al.*, 2000).

Tanto Sequence Ontology como Gene Ontology son proyectos de gran complejidad y repercusión en la literatura científica. Dado que se pretende realizar una ontología de

Alus simple y dirigida, se ha optados por simplificar ambas y emplear solamente un subconjunto de sus términos, manteniendo la jerarquía; en el caso de Gene Ontology tal versión reducida se denomina *GO slim* (Rhee *et al.*, 2008).

1.4. Productos obtenidos

La presente memoria posee una estructura acorde con los productos obtenidos:

- En primer lugar, se ofrece la información relacionada con el diseño conceptual de la ontología su el prototipo. Esta información se encuentra accesible en el capítulo 2. Tal ontología puede descargarse de `http://gattaca.imppc.org/groups/maplab/imallona/ontologies/alu_ontology.owl`.
- En segundo lugar, se detalla la metodología de implementación y se realiza una versión estable de la ontología ya poblada. El capítulo 3 detalla ambos aspectos. La ontología poblada está libremente disponible en `http://gattaca.imppc.org/groups/maplab/imallona/ontologies/merged.owl`.
- En tercer lugar, se realiza una exportación de la ontología a una plataforma MediaWiki. El producto aquí obtenido se describe en el capítulo 4. La vídeo de defensa del trabajo fin de carrera que acompaña a la memoria ofrece una exploración de la instalación local de la wiki; no obstante, la ontología modificada puede consultarse en `http://gattaca.imppc.org/groups/maplab/imallona/ontologies/for_rdfio.owl`.

Capítulo 2

Diseño y prototipo

2.1. Introducción

La métrica del primer prototipo de ontología de Alus indica que contiene 1788 clases, 35 propiedades de objeto y 9 propiedades de datos. De acuerdo a la planificación del proyecto, dicha ontología es un prototipo maduro a nivel lógico pero carente de una población de individuos; no obstante, se han incluido 19 individuos *test* para explorar la expresividad de la herramienta.

Puesto que el proyecto importa ontologías preexistentes, la estructura es compleja; por tanto, se ha optado por detallar algunas de las entidades de nivel superior y especializaciones de entidades, según la nomenclatura de Protégé (Gennari *et al.*, 2003), independientemente de su jerarquía. La descripción detallada de Gene Ontology y Sequence Ontology puede encontrarse en la bibliografía: véase Ashburner *et al.* (2000) y Eilbeck *et al.* (2005).

No obstante, cabe resaltar que tanto Gene Ontology como Sequence Ontology optan por vocabularios ciertamente ofuscados; por ejemplo, *GO:0003700* equivale a "transcription factor" y *SO:0000704* a "gene". Para facilitar la legibilidad de los axiomas, se han introducido manualmente sinónimos de las clases más relevantes de Sequence Ontology [2].

[2] La automatización de la introducción de sinónimos es factible dado que las ontologías más problemáticas, Sequence Ontology y Gene Ontology, ofrecen en estos datos. Por ejemplo, en su formalización OBO (véase por ejemplo `http://www.sequenceontology.org/browser/current_svn/export/term_only/obo/SO:0000110` y `http://amigo.geneontology.org/cgi-bin/amigo/browse.cgi?term=GO:0003682&format=obo`) es inmediato extraer los atributos *synonym* y, una vez definido el ámbito de sinonimia (EXACT, BROAD, NARROW o RELATED), obtener los identificadores de los términos sinónimos.

En cuanto a la abundancia de términos sinónimos, cabe destacar que la versión simplificada de levadura (un *goslim* de pequeño tamaño disponible en `http://www.geneontology.org/GO_slims/goslim_yeast.obo`), el término que posee más sinónimos es *regulation of cell cycle*, con nueve. El script que realiza tal búsqueda puede consultarse en `http://gattaca.imppc.org/groups/maplab/imallona/ontologies/synonyms.sh`.

2.2. Metodología de diseño

De acuerdo al esquema propuesto en Noy *et al.* (2001), se ha procedido a diseñar la ontología siguiendo un esquema de diseño iterativo e incremental que aprovecha, en cierta manera, el enfoque de la programación orientada a objetos. Se ha procedido a un desarrollo en las cuatro fases descritas en el artículo, a saber:

- Definición de clases de la ontología.
- Clasificación taxonómica (jerárquica).
- Definición de las propiedades de clase y de objeto.
- Instanciacion de individuos de estas clases.

Pero, debido a las características inherentes de la ontología de Alus, se han incluido dos etapas u objetivos más:

- Integración con ontologías ya establecidas, como Sequence Ontology y Gene Ontology.
- Desarrollo de reglas derivadas (*property chains*) tipo *uncle rule* (Horrocks *et al.*, 2005), especialmente para atributos funcionales.

2.3. Diseño simplificado

A continuación se proporciona una visión general de la estructura de la ontología.

(a) cellLine

- Descripción
  - Se trata del material biológico del que se extrajeron los datos epigenéticos. Por ejemplo, la línea *hESC* es de células madre embrionarias humanas; mientras que la línea *hct116* es de cáncer colorrectal humano.
- Propiedades de dato
  - *is differentiated* indica si el tipo celular ha sufrido diferenciación o no.

(b) *Biological region* (SO:0001411)

- Descripción
  - Subtipo de de *sequence feature*, corresponde a una secuencia localizada en un determinado lugar que está involucrada en algún proceso biológico.
- Propiedades de dato
  - *on strand*, indica si la característica está en la hebra (+) o (-) del ADN.

- *on chromosome*, indica en qué cromosoma se encuentra. Por convención se admiten valores del 1 al 24 (*23* y *24* son codificaciones alternativas de los sexuales) más X e Y.
- *has start point* y *has end point*, detallan inicio y fin del fragmento cromosómico.

(c) *Alu*

- Descripción
  - Se trata de un *SINE element*; esto es, secuencia repetitiva corta que se encuentra distribuida a lo largo del genoma.
- Propiedades de dato
  - *has Alu family*, categoriza a cada Alu en una determinada subfamilia, como queda descrito en Price *et al.* (2004).
  - *has transfac motif*, indica si posee en su interior una secuencia consenso de unión a factor de transcripción.
  - *has distance to nearest gene*, establece la distancia al gen más cercano.
  - *has length*, especifica la longitud de la Alu.

(d) *junction* (SO:0000699)

- Descripción
  - Se trata de un hito o frontera; esto es, una localización en el genoma que carece de longitud.

(e) *sequence alteration* (SO:0001059)

- Descripción
  - Polimorfismo, una modificación de una secuencia dada (esto es, de una instancia del tipo *region* o SO:0000001).

(f) *molecular function* (GO:003674)

- Descripción
  - Describe la actividad elemental de un producto de un gen; por ejemplo, si un gen está involucrado en una función catalítica.

(g) *cellular component* (GO:005575)

- Descripción
  - Indica la localización concreta del producto (gen, proteína, etc.) en la célula o fuera de ella; por ejemplo, si se trata de un elemento del ribosoma.

(h) *biological process* (GO:008150)

- Descripción

- Adjudica al producto un papel en algún proceso necesario para el funcionamiento de partes celulares, células, tejidos u organismos. Un proceso viene definido como un conjunto de actividades moleculares que poseen un inicio y un final; por ejemplo, la división celular.

Por tanto, *cellLine* es una clase propia de la ontología que está asociada a aquellas regiones del genoma que presentan atributos que difieren entre distintas muestras, como los estados cromatínicos. No pertenece ni a Sequence ni a Gene Ontology.

La clase Alu es una especialización de un elemento SINE (`http://www.sequenceontology.org/browser/current_svn/term/SO:0000206`).Se trata de un tipo de retrotransposón, integrable y móvil por tanto, que no deja de ser una secuencia de ADN. Un diagrama de su posición taxonómica figura en el esquema 2.1.

2.3.1. Funcionamiento

La ontología pivota sobre tres grandes grupos de elementos: las localizaciones, los genes y las Alus.

Cada localización cromosómica es un individuo con atributos *on chromosome*, *has start point*, *has end point* y *on strand*. Los genes y las Alus tienen como atributo de objeto *located in* una de estas localizaciones. La propiedad de objeto *overlaps with*, que es simétrica a *located in*, permite detallar todos los elementos que se superponen en un intervalo cromosómico dado (es decir, es posible enumerar qué genes, transcritos, marcas de histonas o Alus se encuentran en una localización concreta).

A fin de transferir las propiedades de unos objetos a otros, se ha optado por mantener una aproximación biológica conservativa. Dado que en las bases de datos biológicas los elementos mejor anotados son los genes, estos son los portadores de las anotaciones funcionales de partida.

De esta manera, los genes tienen como atributos de clase:

- *transcribed to*: el ARN mensajero al que se transcriben.
- Los términos de Gene Ontology que resumen su actividad (*molecular function*, *cellular location* y *biological process*).

Mientras que las Alus poseen como atributos, en cambio:

- *closest gene to Alu*, el gen más cercano (del cual adquirirán la anotación funcional Gene Ontology).
- *closest chromatin state to Alu*, el estado epigenético solapante.

Por tanto, la relación entre Alus y genes pone en contacto funciones biológicas entre

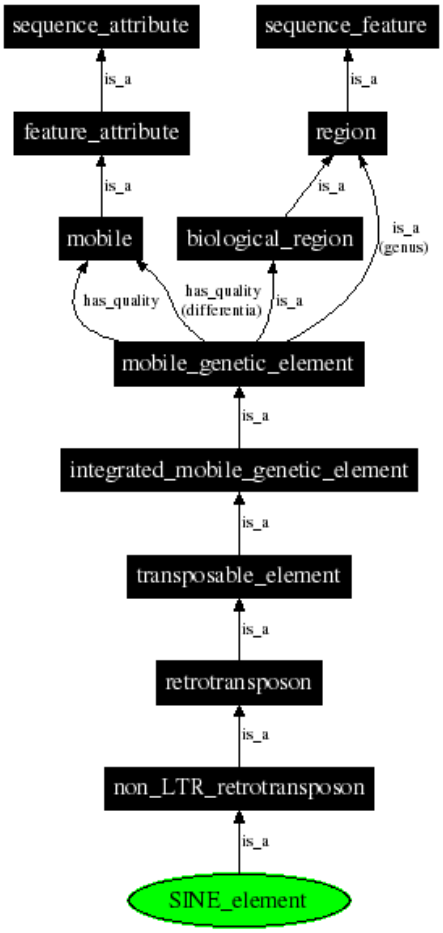


Figura 2.1: Taxonomía del elemento SINE en Sequence Ontology, según la visualización mediante Graphviz ofrecida en `http://topaz.genetics.utah.edu/img/dag_gif/current_svn/SO:0000206_sm.png` de la clase `http://www.sequenceontology.org/miso/current_release/term/SO:0000206`.

ellos y entre los transcritos asociados unívocamente al gen.

Los estados cromatínicos están definidos por su localización pero también poseen una línea celular asociada (*has cell line*). La línea celular es un elemento de la clase *cellLine*, que a su vez es categorizable según sus características (si posee características tumorales, por ejemplo). Esto permite la inclusión de estados cromatínicos procedentes de muestras distintas y la búsqueda de pautas propias; por ejemplo, de modificaciones presentes en líneas diferenciadas (atributo *is differentiated*), de tumorales de cancer de colon, etc.

La compartición de información entre objetos interrelacionados puede realizarse de varias maneras. En la aproximación propuesta en la ontología de Alus cabe destacar la funcionalidad de las cadenas de propiedades. Por poner un ejemplo, `transcribed from o involved in biological process SubPropertyOf involved in biological process` permite que los ARN mensajeros "hereden" la anotación del gen del que fueron transcritos. El razonador *Pellet* (Sirin *et al.*, 2007), que ha sido empleado a lo largo del proceso de diseño, acepta este tipo de encadenamientos.

A fin de clarificar las propiedades de objeto modelizadas, se ofrece una breve descripción de cada una a continuación.

- *closest Alu to chromatin state*: asocia la Alu más próxima a una región que posee un estado cromatínico dado.
- *closest Alu to gene*: asocia la Alu más próxima a un determinado gen.
- *closest chromatin state to Alu*: inverso de *closest Alu to chromatin state.*
- *closest gene to Alu*: inverso de *closest Alu to gene.*
- *has cell line*: atribuye una línea celular a un estado cromatínico.
- *overlaps with*: indica si una región cromosómica está colocalizada con otra. Es una propiedad encadenable consigo misma.
- *located in*: inversa de *overlaps with.*
- *member of*: primitiva de OBO.
- *transcribed from*: indica de qué gen se transcribió un determinado transcrito.
- *transcribed to*: inversa de *transcribed from.*
- *has part*
  - *biological process player.* Inversa de *part of - involved in biological process.*
  - *cellular component instance.* Inversa de *part of - located in cellular component.*
  - *chromatin state associated to feature.* Inversa de *part of - feature associated to chromatin state.*

    - *molecular function player*. Inversa de *part of - involved in molecular function*.

- *part of*
    - *involved in biological process*. Asigna a un determinado gen un proceso biológico dado (esto es, un GO:008150). Es una propiedad encadenable que asocia la función biológica de un gen a sí mismo pero también a su transcrito y a la Alu más cercana.
    - *located in cellular component*. Atribuye a un determinado gen una localización celular (según el vocabulario GO:005575). Es una propiedad encadenable que se transmite del gen a su transcrito y a la Alu más cercana.
    - *feature associated to chromatin state*. Asocia a una región biológica un estado cromatínico concreto. Como propiedad encadenable, se extiende a la Alu más cercana.
    - *involved in molecular function*. Otorga a un gen un rol en una función molecular (esto es, una instancia de GO:003674). Es una propiedad encadenable que transmite la asociación del gen a su transcrito y a la Alu más cercana.

- Primitivas de Gene Ontology
    - *owlends during*: primitiva.
    - *owlhas part*: primitiva.
    - *owloccurs in*: primitiva.
    - *owlregulates*: primitiva.
    - *owlresults in*: primitiva.
    - *owlstarts during*: primitiva.
    - *Obsolete Property*: primitiva.

2.3.2. Cardinalidad

Las restricciones biológicas del modelo se ven reflejadas en las relaciones entre individuos. Por ejemplo, un único gen puede dar lugar a decenas de transcritos, pero un transcrito dado sólo proviene de un gen. La tabla 2.1 resume la cardinalidad de las relaciones entre los individuos de la ontología más usuales.

2.4. Inclusión de ontologías consolidadas

Tanto el prototipo como la vesión estable de la ontología de Alus se encuentran integradas con Gene Ontology y Sequence Ontology. Ambos forman parte de la iniciativa OBO, *The Open Biological and Biomedical Ontologies*, un consorcio que tiene el objetivo de proporcionar ontologías para facilitar la integración de datos biológicos. OBO es accesible desde `http://www.obofoundry.org/`.

individuo	cardinalidad	individuo
gen	$1 \cdots *$	transcrito
gen	$1 \cdots *$	GO Biological Process
gen	$1 \cdots *$	GO Molecular Function
gen	$1 \cdots *$	GO Cellular Location
gen	$1 \cdots 1$	Alu
gen	$1 \cdots 1$	estado cromatínico
Alu	$1 \cdots 1$	estado cromatínico
estado cromatínico	$* \cdots 1$	línea celular

Cuadro 2.1: Resumen de las restricciones de cardinalidad del prototipo de la ontología. La notación "a $\cdots *$ b" implica que un individuo *a* está en relación con *0 o más* individuos *b*. Una representación gráfica de las interrelaciones propuestas, mostrando la cardinalidad, figura en el diagrama 2.7.
Si bien la tabla muestra de forma explícita la cardinalidad de la ontología, cabe destacar que su definición difiere de la propia del UML. Las restricciones de multiplicidad se aplican durante la generación del RDF; dado que el procesamiento se realiza iterando sobre Alus y cada una se empareja unívocamente con un gen y un estado cromatínico, no podría darse el caso de que una Alu poseyera más de un gen asociado, con distinto identificador, y que el razonador asumiera que se trata de entidades equivalentes.

2.4.1. Gene Ontology

Gene Ontology (GO) permite anotar funcionalmente genes y productos de genes. Se trata de un proyecto colaborativo que establece un marco común con descripciones consistentes; de hecho, se utiliza en multitud de bases de datos biológicas. Está estructurado en tres vocabularios: el de procesos biológicos, el de componentes celulares y el de funciones moleculares.

En el caso de la ontología de Alus, el objetivo es recoger la anotación funcional de los transcritos (en la versión definitiva de la ontología) o de los genes (en el prototipo). Un ejemplo de la jerarquía y los términos GO puede observarse en la figura 2.2.

La ontología puede descargase del repositorio OBO en `http://purl.obolibrary.org/obo/go.obo`.

2.4.2. Sequence Ontology

Sequence Ontology (SO) es una ontología dedicada a la descripción de características y atributos de secuencias biológicas. Por ejemplo, ofrece términos como *exón*, *gen* o *CpG metilada*. Tiene como principales usos y objetivos:

1. Proporcionar un vocabulario estructurado apropiado para realizar anotaciones de ácidos nucleicos.

- GO_0002719
  - GO_0002719 $\equiv$ GO_0065007 $\sqcap$ $\exists$ RO_0002212 GO_0002367
  - GO_0002719 $\sqsubseteq$ GO_0002701
  - GO_0002719 $\sqsubseteq$ $\exists$ RO_0002212 GO_0002367
  - GO_0002719 $\sqsubseteq$ GO_0002718
  - GO_0002719 $\sqsubseteq$ GO_0001818
- GO_0002720
  - GO_0002720 $\equiv$ GO_0065007 $\sqcap$ $\exists$ RO_0002213 GO_0002367
  - GO_0002720 $\sqsubseteq$ GO_0002702
  - GO_0002720 $\sqsubseteq$ GO_0001819
  - GO_0002720 $\sqsubseteq$ GO_0002718
  - GO_0002720 $\sqsubseteq$ $\exists$ RO_0002213 GO_0002367

Figura 2.2: Definición de términos Gene Ontology.

2. Favorecer la representación estructurada de información en distintas bases de datos que compartan una estructura común (Mungall *et al.*, 2007).

La figura 2.3 muestra un ejemplo de definición de Sequence Ontology.

La ontología está disponible en el repositorio `https://sourceforge.net/p/song/svn/HEAD/tree/trunk/so-xp.obo?format=raw`.

2.5. Reglas derivadas

Las relgas derivadas (*property chains*) incluidas en la ontología son:

- *closest Alu to chromatin state* AND *feature associated to chromatin state* EQUALS *feature associated to chromatin state*
- *transcribed to* AND *involved in biological process* EQUALS *involved in biological process*
- *closest gene to Alu* AND *involved in biological process* EQUALS *involved in biological process*
- *transcribed to* AND *involved in molecular function* EQUALS *involved in molecular function*

- SO_0001193
  - SO_0001193 $\equiv$ SO_0001247 $\sqcap$ $\exists$ has_quality SO_0001192
  - SO_0001193 $\sqsubseteq$ $\exists$ has_quality SO_0001192
  - SO_0001193 $\sqsubseteq$ SO_0001247
- SO_0001194
  - SO_0001194 $\sqsubseteq$ SO_0001192
- SO_0001195
  - SO_0001195 $\equiv$ SO_0001193 $\sqcap$ $\exists$ has_quality SO_0001194
  - SO_0001195 $\sqsubseteq$ SO_0001193

Figura 2.3: Definición de términos de Sequence Ontology.

- *closest gene to Alu* AND *involved in molecular function* EQUALS *involved in molecular function*
- *transcribed to* AND *located in cellular component* EQUALS *located in cellular component*
- *closest gene to Alu* AND *located in cellular component* EQUALS *located in cellular component*
- *overlaps with* AND *overlaps with* EQUALS *overlaps with*

2.6. Ejemplo de uso

El desarrollo de la ontología se ha acoplado a la inclusión de individuos "test" que permiten evaluar sus capacidades de inferencia y aserción de cláusulas. Como ejemplos se ofrecen tres casos prácticos: véanse las figuras 2.4, 2.5 y 2.6.

- La figura 2.4 muestra la anotación de una Alu dada. Se indica que pertenece a la familia AluSc y que tiene una longitud de 200 nucleótidos. La Alu posee como gen más cercano (a 1500 pb) a `ENSG0000003`, que posee funciones (según Gene Ontology) de "transcription factor" y se encuentra localizado en "cellular component"; por tanto, el razonador atribuye a la Alu estas dos características (anotación en fondo amarillento).
- Las figuras 2.5 y 2.6 muestran la anotación de dos estados cromatínicos distintos, "repetitive/cnv 1" e "insulator 1". El primero tiene como Alu más cercana una que carece de anotación funcional (esto es, se desconoce su función), por lo que sólo

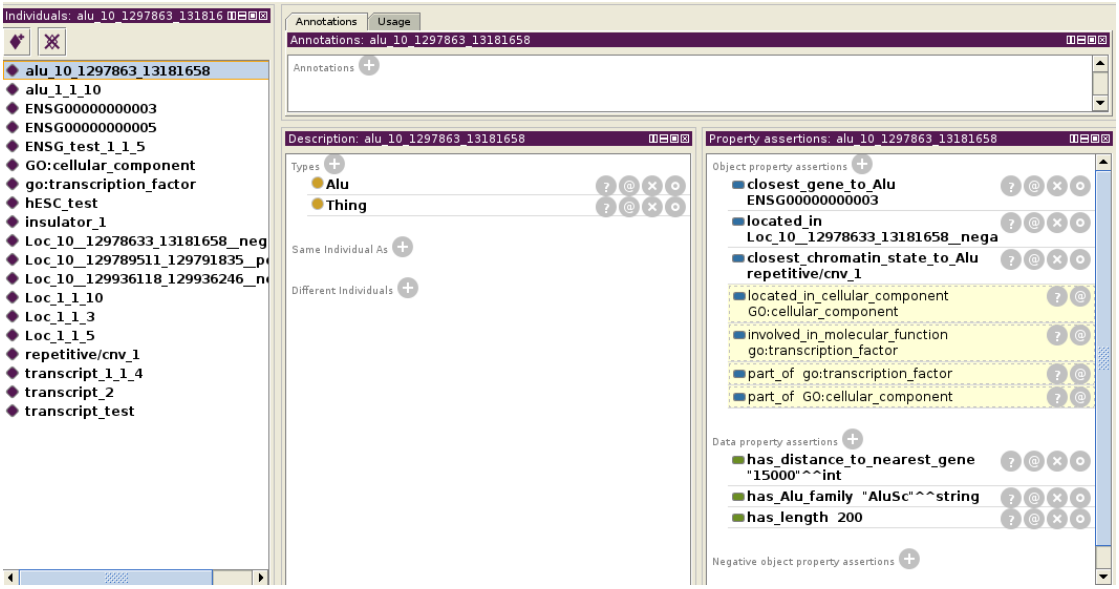


Figura 2.4: Ejemplo de anotación de una Alu. El razonador le atribuye (fondo amarillento) la anotación funcional del gen más cercano como propiedades de objeto.

ofrece la información de su localización y la de la Alu más cercana. En cambio, el caso presentado en 2.6 muestra que el estado cromatínico "insulator 1" tiene como Alu asociada la descrita en el punto inmediatamente anterior; por tanto, es capaz de heredar sus funciones biológicas, que en primera instancia provienen del gen más cercano a la Alu.

La inclusión del juego de datos de prueba permite visualizar el grado de interconexión logrado: véase la figura 2.7 para evaluar las relaciones entre los genes, transcritos, Alus y estados cromatínicos de los individuos instanciados en el prototipo.

2.6.1. Reflexión

Es de resaltar que los conceptos derivados tanto de Gene Ontology como de Sequence Ontology actúan como vocabularios controlados y no tanto como ontologías. Esto se debe a que, si bien presentan información taxonómica, no existe una gran complejidad en la descripción formal de las interrelaciones entre sus elementos. Por tanto, más allá de las especializaciones (jerarquía) y exclusiones (esto es, que ciertos elementos del vocabulario están reunidos en conjuntos disjuntos), poseen poca flexibilidad.

Puesto que los lugares de unión a factores de transcripción derivan de la aplicación de TRANSFAC (Matys *et al.*, 2003), que simplemente analiza motivos de secuencia, se ha optado por modelizar estos no como individuos asociados a una localización sino como atributos de dato de Alus concretas.

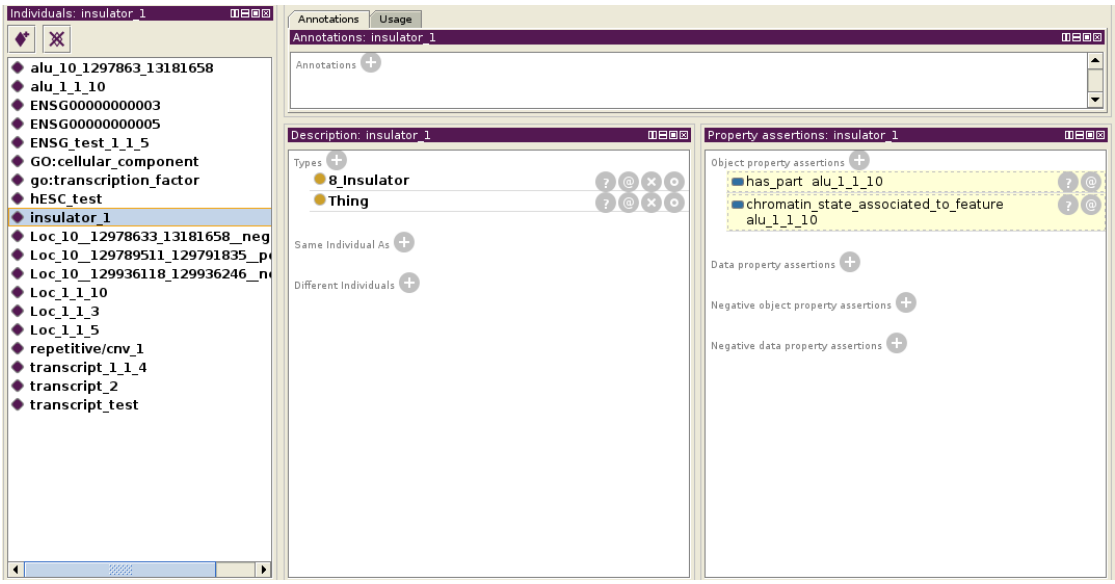


Figura 2.5: Ejemplo de anotación de un estado cromatínico. El razonador indica a qué Alu está asociada. Dado que la Alu no posee una anotación exhaustiva, no se aporta una anotación funcional de acuerdo a Gene Ontology (véase la figura 2.6 para comparar).

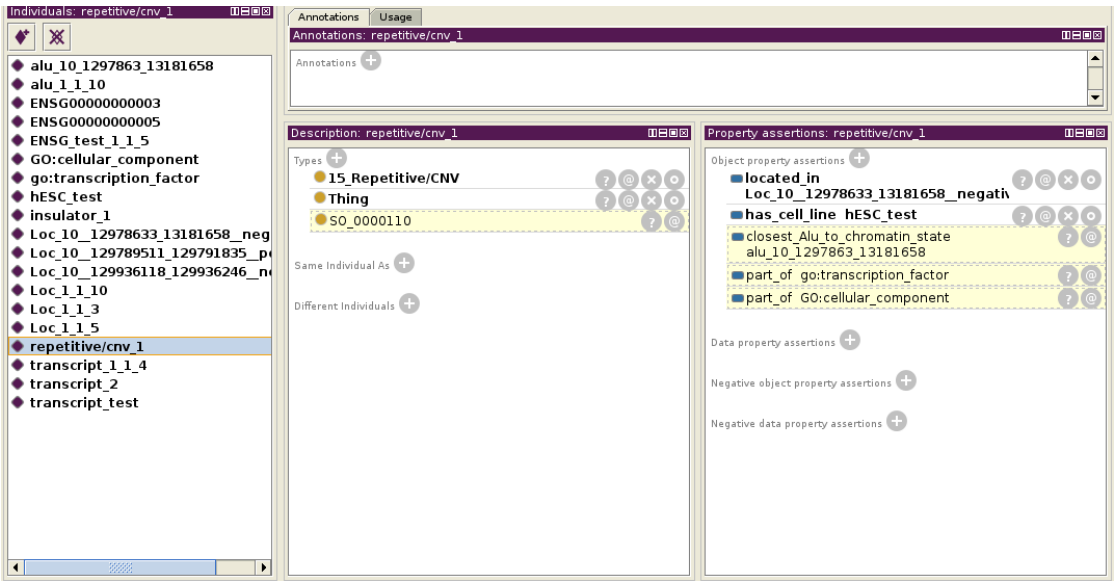


Figura 2.6: Ejemplo de anotación de un estado cromatínico II. El razonador indica a qué Alu está asociada e infiere (en fondo amarillo) la anotación funcional de la región; en este caso, se trataría de una zona codificante para un factor de transcripción.

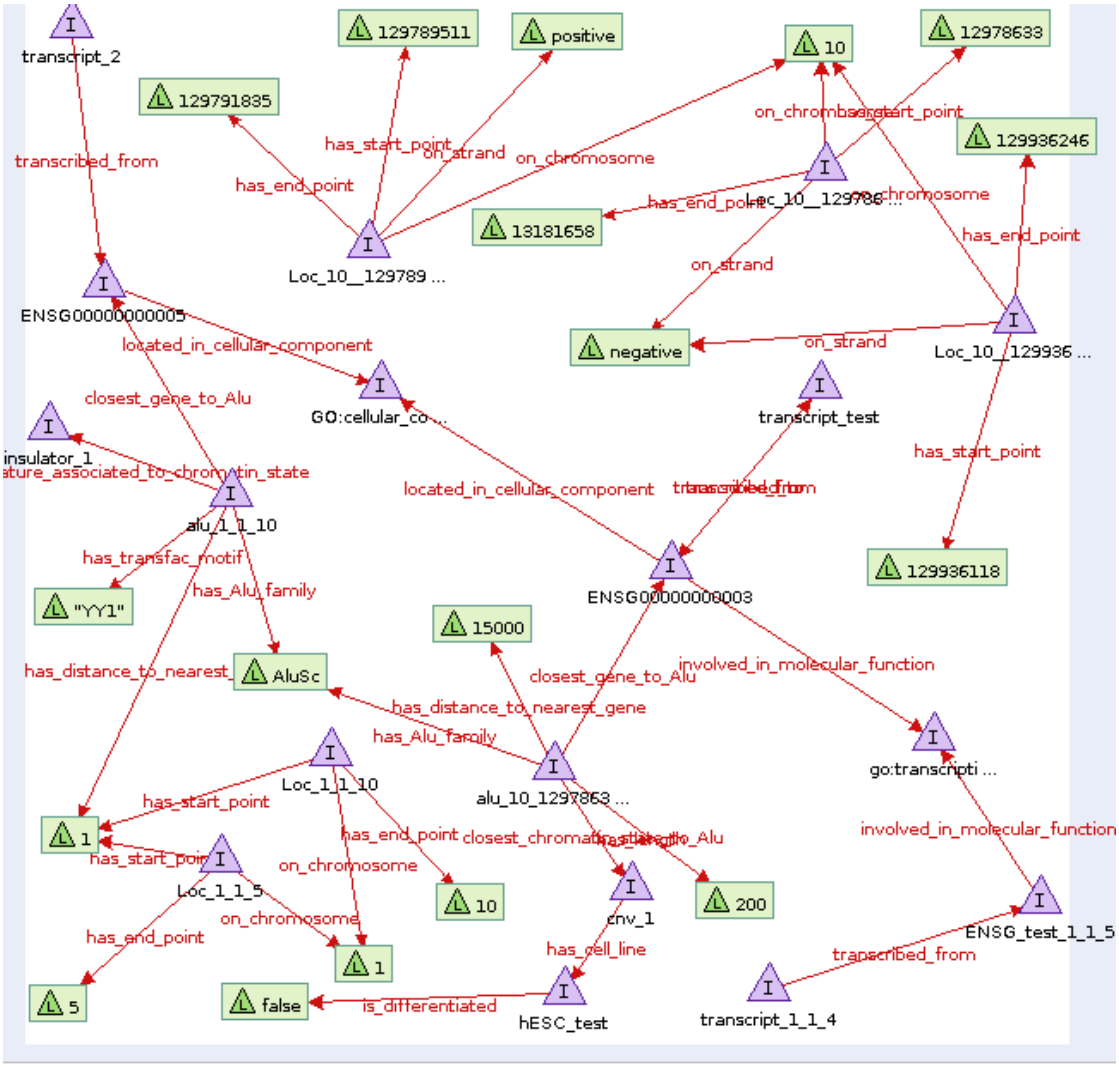


Figura 2.7: Visualización de las propiedades de objeto mediante Gravity. Los datos de origen son los empleados en la prueba de concepto del prototipo; dicho ejemplo de uso aparece detallado en la sección 2.6. La cardinalidad de las relaciones aparece descrita en la tabla 2.1.

Finalmente, no se ha implementado el atributo de dato *has homopolymer length* de la Alu contemplado en la especificación (PEC 1) dado que no se ha encontrado bibliografía que respalde su relevancia.

2.6.2. Métodos

El desarrollo del prototipo se ha efectuado mediante Protégé 4.3.0 build 304 con el plugin OWL (Knublauch *et al.*, 2004) y el razonador Pellet v2.2 (Sirin *et al.*, 2007). Como visualizador se ha utilizado RDF/Gravity v1.0.

Se ha empleado el GO Slim convencional de vertebrados proporcionado por OBO (Smith *et al.*, 2007) y disponible en `http://www.geneontology.org/GO_slims/goslim_generic.obo`, y la adaptación por (Holford *et al.*, 2010) de la versión estable 2.5.1 de Sequence Ontology presente en `http://sourceforge.net/projects/song/files/Sequence%20Ontology/`.

2.7. Disponibilidad

El prototipo de la ontología se encuentra alojado en `http://gattaca.imppc.org/groups/maplab/imallona/ontologies/alu_ontology.owl`.

Capítulo 3

Implementación

3.1. Introducción

La implementación de la ontología cuyo prototipo se describió en el capítulo 2 ha dado lugar a ligeros cambios de modelización debido a la naturaleza de los datos presentes en los repositorios bioinformáticos. En el presente capítulo se discuten estos cambios y se detalla el mecanismo de descarga e inclusión de estos datos en la ontología de Alus.

3.1.1. Particularidades de diseño del UCSC

El repositorio "human genome browser" de la Universidad de California en Santa Cruz (UCSC) ofrece múltiples bases de datos biológicas accesibles tanto mediante un explorador gráfico, denominado *human genome browser*; mediante una interfaz de exportación de datos denominada *Table Browser*; y mediante un servidor MySQL. Esta última opción será la empleada para el procesamiento de los datos en el presente trabajo. Las características del repositorio UCSC pueden consultarse en Dreszer *et al.* (2012), Karolchik *et al.* (2004) y Kent *et al.* (2002).

De forma característica las tablas del UCSC carecen de claves primarias y muestran la equivalencia entre columnas de tablas distintas mediante ficheros `.joiner`. El fundamento de tal organización puede consultarse en la documentación del proyecto (`http://genome-source.cse.ucsc.edu/gitweb/?p=kent.git;a=blob;f=src/hg/makeDb/schema/joiner.doc`), y la descripción formal del ligamiento de campos está disponible como fichero `.joiner` (`http://genome-source.cse.ucsc.edu/gitweb/?p=kent.git;a=blob;f=src/hg/makeDb/schema/all.joiner`).

La ventaja de tal aproximación es cierta flexibilidad: por ejemplo, permite que un campo contenga una lista de identificadores separados por comas, o que los identificadores equivalgan pese a poseer sufijos en algunas tablas y carecer de ellos en otras. Por otro lado, tal diseño dificulta la representación gráfica de las relaciones entre tablas, puesto que la carencia de claves primarias y foráneas *sensu stricto* impide la generación de diagramas

de entidad relación clásicos.

A fin de clarificar qué datos se extraen de qué tablas, por tanto, se procede a describir cada una de las consultas realizadas en el proyecto y su significado en la sección 1.

3.1.2. Consultas al UCSC

La consultas realizadas mediante el servidor público MySQL del UCSC pivotan sobre la selección de atributos en las bases de datos `go` (Gene Ontology) y `hg19` (la última versión del ensamblado genómico humano durante la realización el presente proyecto). A fin de establecer equivalencias entre identificadores, las distintas tablas son relacionadas usualmente mediante la relación `hg19.kgXref`.

Cabe destacar que los identificadores empleados corresponden a la base de datos Ensembl, si bien la interfaz que se emplea para la descarga de los datos es del UCSC.

(a) Obtención de transcritos. Descarga del identificador del transcrito (enst), identificador del gen (ensg), cromosoma (chrom), cadena (strand), inicio de transcripción (txStart), fin de transcripción (txEnd), inicio de la región codificante (cdsStart), fin de la zona codificante (cdsEnd), símbolo (geneSymbol), identificador de Gene Ontology (goId), término de Gene Ontology (goName), subontología de Gene Ontology (term_type).

```
SELECT eg.name AS enst,
    eg.name2 AS ensg,
    eg.chrom,
    eg.strand,
    eg.txStart,
    eg.txEnd,
    eg.cdsStart,
    eg.cdsEnd,
    xref.geneSymbol,
    go.goId,
    got.name,
    got.term_type
FROM hg19.ensGene AS eg,
    hg19.kgXref AS xref,
    go.goaPart AS go,
    hg19.knownToEnsembl AS kte,
    go.term AS got
WHERE kte.value = eg.name AND
    kte.name = xref.kgID AND
    xref.spId = go.dbObjectId AND
    go.goID = got.acc;
```

(b) Obtención de Alus. Descarga del cromosoma (genoName), inicio de la Alu (genoStart), fin de la Alu (genoEnd), tipo de repetición (repName) y familia (repFamily).

Obsérvese que se filtra por elementos cuya familia sea `Alu` y cuyo nombre comience por `Alu`, también.

```
SELECT genoName,
    genoStart,
    genoEnd,
    repName,
    repClass,
    strand,
    repFamily
FROM hg19.rmsk
WHERE repFamily = "Alu" AND
    repName LIKE "Alu%"
```

(c) Desambiguación de la longitud del gen. Se extrae como inicio y fin la máxima longitud de transcripción. Se obtiene el identificador del gen (name2), el cromosoma (chrom), el inicio (min(txStart)), el fin (max(txEnd)) y la cadena (strand).

```
SELECT name2,
    chrom,
    min(txStart),
    max(txEnd),
    strand
FROM hg19.ensGene
GROUP BY name2
```

(d) Recuperación de los estados cromatínicos para la línea celular *hESC*. Obtiene el cromosoma (chrom), inicio (chromStart), fin (chromEnd) y tipo de estado (name).

```
SELECT chrom,
    chromStart,
    chromEnd,
    name
FROM hg19.wgEncodeBroadHmmH1hescHMM;
```

3.1.3. Datos TRANSFAC

De acuerdo al diseño propuesto, cabe asociar a cada elemento Alu una lista de factores de transcripción que putativamente puedan reconocer algunas pautas en su secuencia y, por tanto, unirse físicamente a la Alu.

Para este fin, las herramientas bioinformáticas aplican distintas técnicas para predecir *in silico* dónde se producirá la unión de un factor de transcripción (esto es, un *transcription factor binding site* o TFBS). Para ello requieren una base de datos de motivos ya conocidos (por ejemplo, que la secuencia `CACGTG` es reconocida en ratón por el factor de transcripción *Arnt*, como puede observarse en la base de datos JASPAR (Sandelin *et al.*, 2004)). Un repositorio de TFBS consenso muy utilizado es TRANSFAC (Matys

et al., 2003).

Una vez reunida la información sobre motivos consenso, es preciso evaluar si uno o más factores de transcripción se encuentran en la secuencia de DNA con la que se esté trabajando. Este tipo de cómputo es complicado; en algunos casos, se emplean enfoques probabilísticos para su detección. En el diseño del presente trabajo se incluye la detección de TFBS para más de un millón de Alus humanas sin limitación de la naturaleza del factor de transcripción; puesto que el cálculo es costoso, se ha optado por emplear un juego de datos precalculado mediante la herramienta FIMO (Grant *et al.*, 2011).

La descarga de los datos se realiza sobre el repositorio del laboratorio de W.S. Noble (`http://noble.gs.washington.edu/custom-tracks/fimo.hg19.transfac-0.1.bb`); una descripción de sus características puede consultarse en `http://noble.gs.washington.edu/custom-tracks/fimo.transfac.description.html`.

3.2. Metodología

El desarrollo de la ontología se ha efectuado mediante Protégé 4.3.0 build 304 con el plugin OWL (Knublauch *et al.*, 2004) y el razonador Pellet v2.2 (Sirin *et al.*, 2007). Como visualizador se ha utilizado RDF/Gravity v1.0.

Se ha empleado el GO Slim convencional de vertebrados proporcionado por OBO smith2007obo y disponible en `http://www.geneontology.org/GO_slims/goslim_generic.obo`, y la adaptación por (Holford *et al.*, 2010) de la versión estable 2.5.1 de Sequence Ontology presente en `http://sourceforge.net/projects/song/files/Sequence%20Ontology/`.

La máquina en la que se ejecutan los cálculos posee 8 cores, 16 GB de RAM y emplea como sistema operativo una Fedora Core de 64 bits con núcleo 2.6.35.6-45.fc14.x86_64.

La parte de la implementación queda descrita en la sección 3.3; no obstante, se ofrece la documentación del paquete de Python desarrollado para tal efecto como anexo B.

3.3. Implementación

Para la población de la ontología se ha optado por el lenguaje de scripting python. Pese a que existen módulos que procesan XML (McGrath, 2000) o específicamente RDF (rdflib (Nykanen, 2004)) y que permiten gestionar conexiones con bases de datos (como sqlAlchemy (Copeland, 2010)), se ha optado por trabajar tan sólo con plantillas y subprocesos Unix.

El enfoque seguido durante el desarrollo se ha basado en el paradigma de orientación a objetos (Larman, 2012), y mediante la metodología "extreme programming" (Beck and Andres, 2004).

Básicamente, el funcionamiento del script de construcción de la ontología es el que sigue:

1. En primer lugar se obtienen datos de los genes, estados cromatínicos, Alus y transcritos mediante consultas SQL contra el "human genome browser" de la Universidad de California en Santa Cruz (UCSC). Las características de la base de datos pueden consultarse en Dreszer *et al.* (2012), Karolchik *et al.* (2004) y Kent *et al.* (2002).

   La naturaleza de dichas consultas se describe pormenorizadamente en la sección 1.

   Dado que se trata de un gran número de datos (sólo de Alus hay más de un millón), se restringen las consultas a aquellos elementos situados en el cromosoma 22. Por ejemplo, la consulta que recupera la anotación mediante Gene Ontology, localización cromosómica e identificador de gen y transcrito produce el resultado mostrado en la figura 3.3.

   Es interesante reseñar que la interfaz MySQL para el genoma humano y de otras especies del UCSC (Karolchik *et al.*, 2004) contiene tablas para zonas repetitivas del genoma detectadas según RepeatMasker (Smit *et al.*, 1996). El UCSC por tanto alberga información sobre cuántas Alus hay en un determinado genoma, dónde están, a qué familia pertenecen, y cuáles son sus características. Al mismo tiempo, también se ofrece una información equivalente (a grandes rasgos) sobre genes, por lo que es posible descargar ambos subconjuntos de datos y relacionarlos.

   A modo de ejemplo, la figura 3.3 muestra el resultado de las consultas empleadas en la construcción de la ontología. Una vez dentro del servidor público del UCSC (mediante `$ mysql --user=genome --host=genome-mysql.cse.ucsc.edu`, por ejemplo), es posible consultar los transcritos del cromosoma 22, así cómo su gen asociado, localización, símbolo, anotación funcional y otras características. De esta forma, es posible iterar sobre estos resultados y recuperar la información necesaria para cumplir con el diseño de la ontología.

   Consultas similares se realizan para los distintos tipos de individuos, tal y como queda listado en el código en el script "ucsc_mysql_shuttle.py" presentado en la sección .

2. Una vez recogida la información, se instancian los distintos individuos modelizados. Básicamente el procedimiento a seguir es rellenar la plantilla concreta y, de ser

menester, anidar esta instanciación con la de la localización cromosómica o gen asociado. Por ejemplo, un criterio de decisión es como sigue:

a) Se recoge un nuevo transcrito mediante la consulta mostrada en 3.3; se parsea mediante `parsers.py` y se incluye en su plantilla de transcrito (siguiendo la directriz de Sequence Ontology y según quedó definido en la modelización de la ontología) mediante el código recogido en `formatters.py`.

b) Se incluye en su plantilla de localización cromosómica empleando las mismas clases en python.

c) Si no se ha procesado el gen asociado (puesto que un gen puede tener más de un transcrito), se parsea y se incluye en su plantilla de gen y de localización.

Naturalmente, en el caso de tratarse de Alus, o localizaciones cromosómicas, el algoritmo es ligeramente distinto. La figura 3.4 ofrece una captura de pantalla de la ejecución de la clase "main" del proyecto de población de la ontología.

Se ha comentado con anterioridad que la información se incluye en plantillas. A modo de ejemplo, se muestra cómo es posible sustituir los valores procedentes de las consultas SQL en individuos que cumplen la estructura del XML/RDF modelizado en el diseño de la ontología. Para una localización cromosómica dada cabe emplear el método `format_location(self, chrom, start, end, strand)` de la clase `Formatters` que posee la forma siguiente:

```
# the owl code to represent the individual
t = Template("""
<!-- http://gattaca.imppc.org/groups/maplab/imallona/ontologies/
    alu_ontology.owl#Loc.$chrom:$start-$end;$strand -->

<owl:Thing rdf:about="#Loc.$chrom:$start-$end;$strand">
   <rdf:type rdf:resource="#biological_region"/>
   <on_chromosome rdf:datatype="&xsd;string">$chrom</on_chromosome>
   <has_start_point rdf:datatype="&xsd;integer">$start</has_start_point>
   <has_end_point rdf:datatype="&xsd;integer">$end</has_end_point>
   <on_strand rdf:datatype="&xsd;string">$strand</on_strand>
</owl:Thing>
""")
```

Esta plantilla permite la sustitución iterativa de aquellos campos que comiencen por el símbolo `$`. Supongamos que deseamos generar una localización genómica en el cromosoma 1 en el rango (600, 1500) y sobre la cadena negativa. En este caso cabría emplear el código siguiente:

```
t.safe_substitute(chrom = 'chr1',
                  start = 600,
                  end = 1500,
                  strand = 'positive')
```

```
mysql>
SELECT eg.name as enst, eg.name2 as ensg, eg.chrom, eg.strand, eg.txStart, eg.txEnd, eg.cdsStart, eg.cdsEnd,
      xref.geneSymbol,
      go.goId, got.name, got.term_type
FROM hg19.ensGene as eg, hg19.kgXref as xref, go.goaPart as go, hg19.knownToEnsembl as kte, go.term as got
WHERE kte.value = eg.name AND
      kte.name = xref.kgID AND
      xref.spId = go.dbObjectId AND
      go.goID = got.acc AND
      eg.chrom = 'chr22';
LIMIT 3;
+----------------+----------------+-------+--------+----------+----------+----------+----------+------------+------------+-----------------------------------+--------------------+
| enst           | ensg           | chrom | strand | txStart  | txEnd    | cdsStart | cdsEnd   | geneSymbol | goId       | name                              | term_type          |
+----------------+----------------+-------+--------+----------+----------+----------+----------+------------+------------+-----------------------------------+--------------------+
| ENST00000332271 | ENSG00000184571 | chr22 | -      | 25115000 | 25170683 | 25115438 | 25158466 | PIWIL3     | GO:0003676 | nucleic acid binding              | molecular_function |
| ENST00000332271 | ENSG00000184571 | chr22 | -      | 25115000 | 25170683 | 25115438 | 25158466 | PIWIL3     | GO:0007275 | multicellular organismal development | biological_process |
| ENST00000400521 | ENSG00000184470 | chr22 | -      | 19863044 | 19929333 | 19864627 | 19929326 | TXNRD2     | GO:0000305 | response to oxygen radical        | biological_process |
+----------------+----------------+-------+--------+----------+----------+----------+----------+------------+------------+-----------------------------------+--------------------+
3 rows in set (0.22 sec)
```

Figura 3.1: Ejemplo de recuperación de datos de las bases de datos del “human genome browser” del UCSC. Este tipo de resultados se emplean en la implementación del constructor de la ontología recogida en 3.3.

```
Cromosoma   ================================================================
Alu                       =============
Genes       ======                                                  ======
Resultado   ======
```

Figura 3.2: Atribución del gen más cercano obviando el atributo “strand”.

3.3.1. Asignación del gen más cercano

La atribución de funciones biológicas a las Alus se ha realizado obteniendo la anotación Gene Ontology del transcrito más cercano. Del mismo modo, también se ha atribuido un estado cromatínico a cada Alu mediante el criterio de contigüidad. Este enfoque difiere entre la versión estable de la ontología y el prototipo: en el prototipo, la anotación se obtenía del gen más cercano, y no del transcrito. El cambio de modelización efectuado presenta como ventaja una mayor precisión en la asignación de funciones, dado que biológicamente existen variantes de expresión del mismo gen que poseen una actividad “in vivo” distinta (Matlin *et al.*, 2005).

Eel elemento más cercano a cada Alu ha sido adjudicado mediante el software bedtools (Quinlan and Hall, 2010), que permite trabajar con coordenadas genómicas de forma eficiente. Concretamente, se ha optado por la herramienta `bedtools closest`. Las llamadas al software quedan codificadas en la clase `Unix_subprocesses` del código documentado en el apéndice B.

El funcionamiento de la herramienta es el siguiente. Imaginemos una situación en la que tenemos una Alu en una posición genómica y queremos buscar el gen más cercano. Considerando las Alus y los genes como cajas, esto es, ternas con la forma (cromosoma, start, end), es trivial asignar la caja más cercana como figura en el esquema 3.2.

No obstante, tanto las Alus como los genes poseen direccionalidad (es decir, un principio y un final). Esto se debe a que el ADN tiene dos hebras que pueden almacenar información y que la distribución de elementos según su dirección es independiente de la hebra. Es decir: puede haber elementos contiguos en direcciones opuestas. Por tanto, la opción escogida para realizar la asignación tiene en cuenta la distancia presente entre el inicio de un gen y el inicio de una Alu, y no así las relaciones inicio-final. Un diagrama mostrando tan relación puede consultarse en la figura 3.3.

Cabe resaltar que los colores cromatínicos carecen de direccionalidad. No obstante, y por convención, se les ha otorgado un valor “+” en la ontología.

Finalmente, el solapamiento físico entre dos características genómicas queda recogido en la ontología con el valor “0”.

```
Cromosoma   ================================================================
Alu                     >>>>>>>>>>>
Genes       >>>>>                          <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
Resultado   >>>>>
```

Figura 3.3: Atribución del gen más cercano teniendo en cuenta atributo “strand”.

3.3.2. Empleo de Bedtools

Bedtools permite trabajar con ficheros BED. Estos están en texto plano, separados por tabuladores, y poseen una estructura como la descrita en la web `http://www.ensembl.org/info/website/upload/bed.html`. En su variante convencional se trata de tuplas (una por línea) con la forma *chromosome, start, end*; en algunos casos se opta por seis o más campos, en los que forzosamente el atributo strand ha de estar en la columna 6.

La obtención de tales ficheros se realiza en el código proporcionado mediante una consulta MySQL seguida de cierto parseo mediante awk. En principio dada la naturaleza secuencial de los procedimientos (comprobación de solapamiento y búsqueda de elementos más cercanos), una posibilidad de diseño podría ser obtener los resultados de la base de datos del UCSC desde dentro de python, iterar sobre cada elemento y llamar a bedtools individualmente redirigiendo las salidas estándar (`stdin` y `stdout`), sin materialización de ficheros para todos los genes (genes.bed), Alus (alus.bed), estados cromatínicos (hmm.bed), y las relaciones entre estos (summarized.bed). De hecho, existe un módulo de python denominado pybedtools que ofrece incluso una interfaz a bedtools (Dale *et al.*, 2011), lo que facilitaría tal enfoque.

No obstante, en la presente implementación se ha optado por realizar llamadas mediante subprocesos (módulo `unix subprocesses.py`) por una cuestión de sencillez de distribución e instalación del código. Bajo un entorno GNU/Linux (y posiblemente por extensión cualquier Unix), el código necesita tan sólo una instalación estándar de python $\geq$2.6 y de bedtools en cualquiera de sus versiones. Ni siquiera se utiliza un entorno de gestión XML o RDF, sino plantillas que proceden de la librería String (que es estándar).

3.3.3. Justificación de la reducción al cromosoma 22

En origen se modelizó la ontología para albergar a todas las Alus, genes, transcritos y estados cromatínicos del genoma humano en su versión “hg19”. No obstante, la abundancia de datos y la complejidad de los cálculos de los razonadores se ha optado por reducir la ontología a aquellos elementos del cromosoma 22.

```
imallona@lechuck:/imppc/labs/maplab/imallona/src/ontology
File  Edit  View  Search  Terminal  Help
[imallona@lechuck ontology]$ /usr/bin/time -v python ontology_population.py
/soft/general/python/lib/python2.7/site-packages/MySQL_python-1.2.3-py2.7-linux-x86_64.egg/_mysql.py:3: UserWarning: Module peak was already imported f
thon2.7/site-packages/PEAK-0.5a4.dev-py2.7-linux-x86_64.egg is being added to sys.path
[08 Dec 2013 11:55] Bulk UCSC MySQL query start
[08 Dec 2013 11:55] Bulk UCSC MySQL query end
[08 Dec 2013 11:55] UCSC MySQL query regarding gene locations start
[08 Dec 2013 11:55] UCSC MySQL query regarding gene locations end
[08 Dec 2013 11:55] UCSC MySQL query regarding Alus start
[08 Dec 2013 11:55] UCSC MySQL query regarding Alus end
[08 Dec 2013 11:55] UCSC MySQL query regarding HMM for hESC start
[08 Dec 2013 11:55] UCSC MySQL query regarding HMM for hESC end
[08 Dec 2013 11:55] Gene and transcript parsing start
[08 Dec 2013 11:55] Gene and transcript parsing end
[08 Dec 2013 11:55] closestBed -a /home/labs/maplab/imallona/tmp/annotation/alus.bed -b /home/labs/maplab/imallona/tmp/annotation/genes.bed -D a start
[08 Dec 2013 11:55] closestBed -a /home/labs/maplab/imallona/tmp/annotation/alus.bed -b /home/labs/maplab/imallona/tmp/annotation/genes.bed -D a end
[08 Dec 2013 11:55] closestBed -a /home/labs/maplab/imallona/tmp/annotation/alus.bed -b /home/labs/maplab/imallona/tmp/annotation/hmm.bed -D a start
[08 Dec 2013 11:55] closestBed -a /home/labs/maplab/imallona/tmp/annotation/alus.bed -b /home/labs/maplab/imallona/tmp/annotation/hmm.bed -D a end
done
        Command being timed: "python ontology_population.py"
        User time (seconds): 5.29
        System time (seconds): 0.68
        Percent of CPU this job got: 39%
        Elapsed (wall clock) time (h:mm:ss or m:ss): 0:15.09
        Average shared text size (kbytes): 0
        Average unshared data size (kbytes): 0
        Average stack size (kbytes): 0
        Average total size (kbytes): 0
        Maximum resident set size (kbytes): 436016
        Average resident set size (kbytes): 0
        Major (requiring I/O) page faults: 0
        Minor (reclaiming a frame) page faults: 62393
        Voluntary context switches: 53339
        Involuntary context switches: 836
        Swaps: 0
        File system inputs: 0
        File system outputs: 114816
        Socket messages sent: 0
        Socket messages received: 0
        Signals delivered: 0
        Page size (bytes): 4096
        Exit status: 0
[imallona@lechuck ontology]$
```

Figura 3.4: Ejecución del script que puebla la ontología para elementos del cromosoma 22.

3.3.4. Ejecución

El conjunto de scripts presentados en la sección 3.6 poseen, cuando se efectúa una llamada que afecta al cromosoma 22, un comportamiento como el que aparece representado en la figura 3.4.

La ontología del cromosoma 22 ocupa 24 MB en total.

3.4. Aplicación y perspectiva

3.4.1. Visualización en Protégé

A modo de ejemplo, se ofrecen varias capturas de pantalla de Protégé tras la aplicación del razonador sobre los individuos instanciados en la ontología de Alus del cromosoma 22. Las propiedades inferidas aparecen resaltadas en fondo amarillo, mientras que los datos proporcionados poseen un fondo blanco. La figura 3.5 muestra un individuo Alu; la 3.6, un gen; y la 3.7, un estado cromatínico.

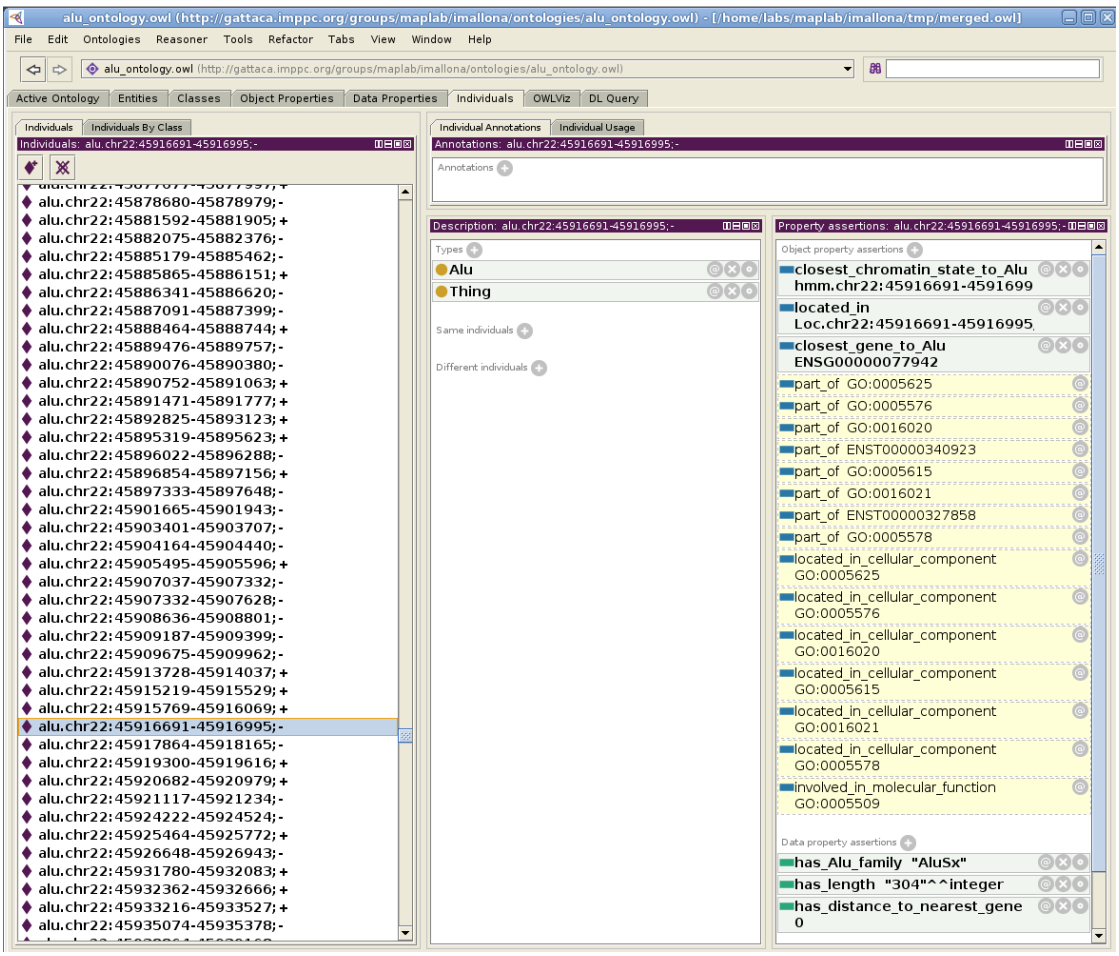


Figura 3.5: Visualización en Protégé de una Alu.

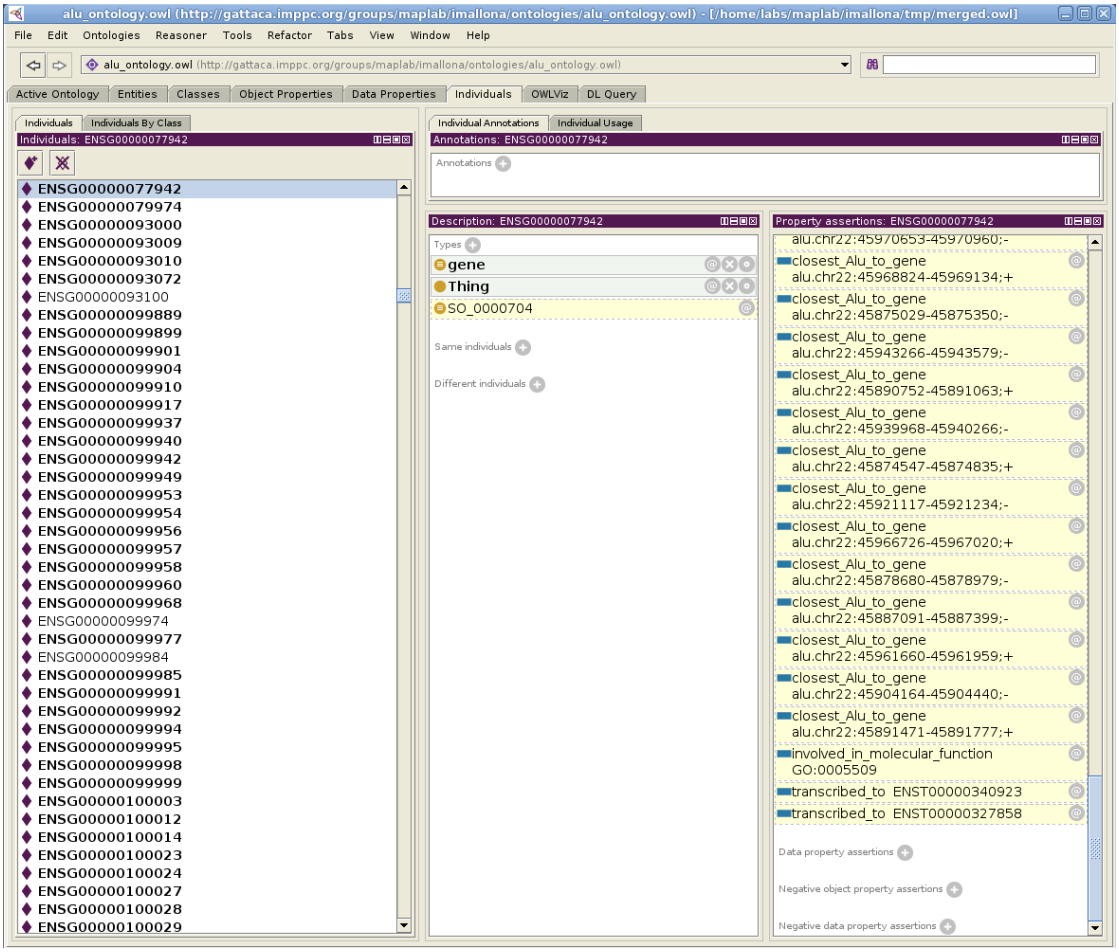


Figura 3.6: Visualización en Protégé de un gen.

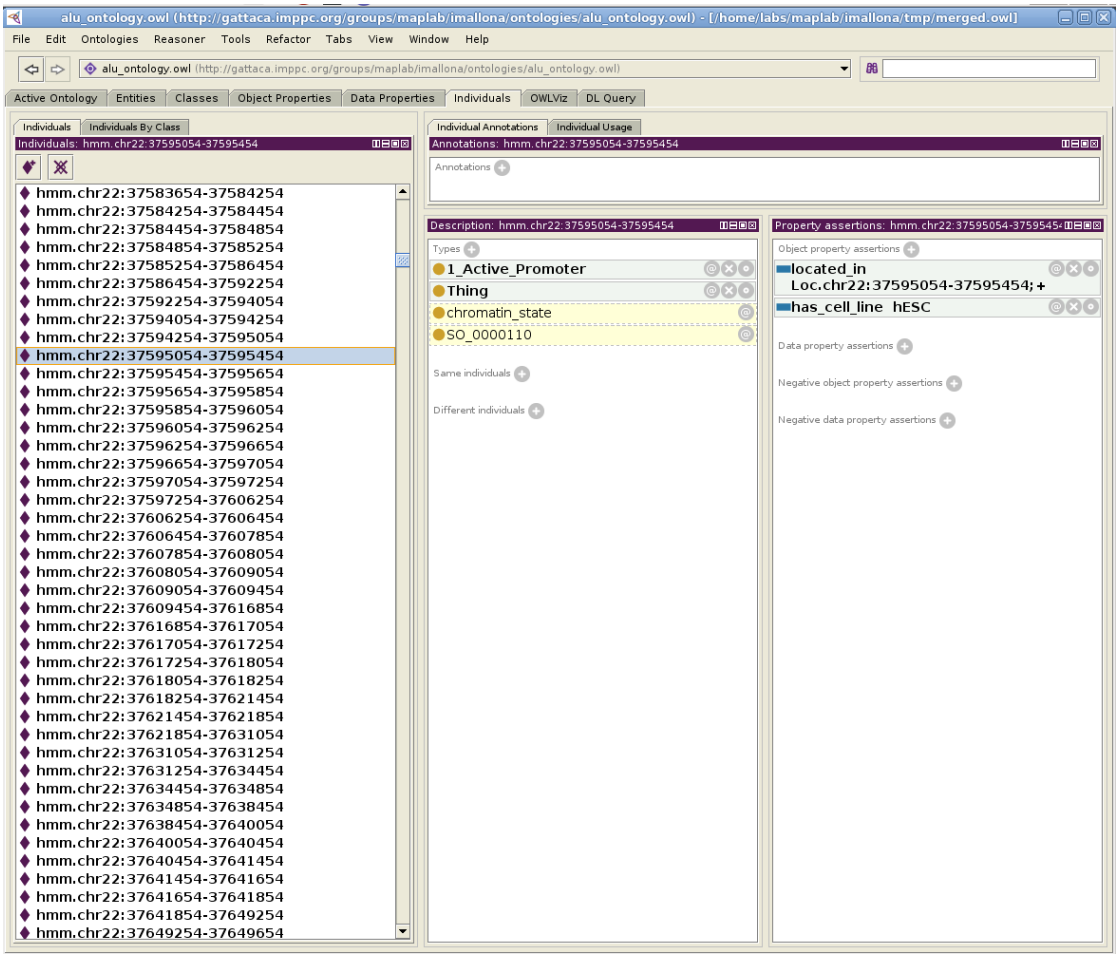


Figura 3.7: Visualización en Protegé de un color cromatínico

```
 Information about file:/home/labs/maplab/imallona/tmp/merged.owl
(<http://gattaca.imppc.org/groups/maplab/imallona/ontologies/alu_ontology.owl>)
OWL Profile = OWL 2
DL Expressivity = SRIF(D)
Axioms = 280301
Logical Axioms = 251126
GCI Axioms = 0
Individuals = 98831
Classes = 1803
Object Properties = 35
Data Properties = 9
Annotation Properties = 16
```

Figura 3.8: Información sobre la ontología restringida al cromosoma 22

3.4.2. Resumen de la ontología

La herramienta Pellet posee la opción de mostrar las características globales de la ontología mediante la orden `/pellet.sh info`. De este modo, la ontología del cromosoma 22 posee la métrica resumida en la figura 3.8.

En resumen, en la actualidad contiene 98831 individuos instanciados sobre 1803 tipos de clases. La complejidad en número de clases procede parcialmente del aprovechamiento de los vocabularios de Sequence Ontology y de Gene Ontology. La versión de la ontología que se presenta en esta entrega ofrece como característica principal la instanciación de los individuos del cromosoma 22, que precisamente conssisten en más de 90000 unidades.

3.5. Disponibilidad

La versión reducida al cromosoma 22 de la ontología se encuentra alojada en merged.owl (`http://gattaca.imppc.org/groups/maplab/imallona/ontologies/merged.owl`).

3.6. Codificación

La documentación del paquete de python que genera la ontología figura como anexo B en el presente documento. En resumen, tal paquete presenta la estructura que se describe a continuación:

- `ucsc_mysql_shuttle.py`, métodos asociados a la consulta, y subsiguiente captura de resultados, a la interfaz MySQL pública del UCSC.
- `parsers.py`, elementos de iteración y extracción de información de los resultados de `ucsc_mysql_shuttle.py`.

- formatters.py, elementos de instanciación de individuos a través de los resultados de `parsers.py`. Genera un XML compatible con pellet y Protégé.
- rdfio_compliant_formatters.py, elementos de instanciación de individuos a través de los resultados de `parsers.py`. Genera un XML adecuado para poblar la wiki semántica mediante la extensión RDFIO.
- `utils.py`, utilidades menores. En la presente versión, tan sólo una función que devuelve la fecha y la hora.
- `unix_subprocesses.py`, contiene el código preciso para realizar llamadas aplicaciones externas, ya sean estándar en Unix o específicas, como bedtools (Quinlan and Hall, 2010).
- `ontology_population.py`, clase principal que efectúa todo el proceso.

Capítulo 4

Aluwiki

4.1. Introducción

Dada la ontología modelizada en el capítulo 2 y poblada en tal y como se describe en el capítulo 3, se ha procedido al desarrollo de una interfaz de acceso amigable, autónoma y escalable. Para ello se ha empleado *Semantic MediaWiki*.

4.2. Motivación

Las wikis permiten una fácil presentación y edición de contenido Web. Típicamente, se han empleado para generar páginas de forma colaborativa, permitiendo el uso de una sintaxis muy sencilla y un mantenimiento simple basado en un lenguaje de marcado. Dado que poseen esta estructura, es trivial exportar parte de su contenido de forma automática a formatos, como el XML, que pueden ser interpretados con facilidad por máquinas. De este modo, ofrecen una solución viable tanto para los editores y lectores humanos como para el procesamiento automático (Hoehndorf *et al.*, 2006, 2009).

Además, las wikis pueden, mediante la extensión Semantic MediaWiki (Hoehndorf *et al.*, 2006), incorporar ontologías y procesarlas de una manera automática. Pero, además, dado que proporcionan el fácil acceso a los usuarios, permiten la corrección o actualización de su contenido por parte de estos; este hecho favorece la consecución de una consistencia interna, cuya falta ha sido motivo de crítica por ciertos autores (Arita, 2009).

La Aluwiki que se presenta en este capítulo busca transferir la ontología generada en el capítulo 3 a los usuarios sin necesidad de depender de software especializado, como Protégé, para su empleo; asimismo, proporciona independencia de plataforma y de recursos del sistema, lo que en una ontología de gran tamaño como lo es la de Alus es una ventaja innegable.

De esta forma, la Aluwiki permite aprovechar las capacidades de un MedaWiki ofreciendo entidades que son páginas de la wiki en lugar de instancias de la ontología; además, se

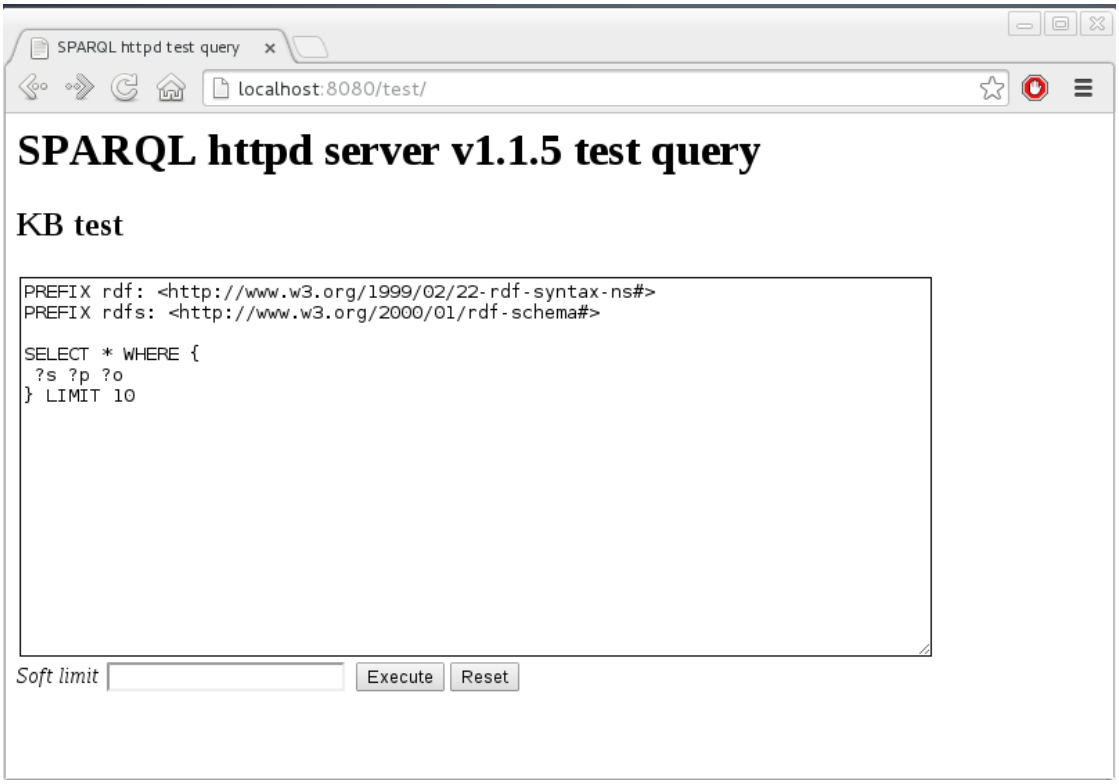


Figura 4.1: Interfaz de consulta de ejemplo del sparql endpoint 4store.

ofrecen métodos para la fácil navegación y consulta de propiedades y atributos de una forma amigable para el usuario. Como complemento, y de una manera más formal, también se ofrece el SPARQL endpoint 4store.

4.3. Métodos

4.3.1. Instalación de software

Tal y como se especificó en la planificación del proyecto, se ha compaginado la implementación de los métodos para poblar la ontología con la instalación del software preciso para facilitar la transferencia del producto a los usuarios. De esta manera, se ha instalado un Semantic Mediawiki (Krötzsch *et al.*, 2006) y el SPARQL endpoint denominado 4store (Harris *et al.*, 2009). La interfaz de consulta del SPARQL endpoint aparece en la figura 4.1 y un ejemplo de respuesta, en la figura 4.2.

4.3.2. RDFIO

Con el fin de incorporar de forma automatizada el contenido de la ontología en la wiki, se ha generado un usuario bot `Alubot` que emplea la interfaz `Special:RDFImport` de

SPARQL Httpd test query | download %285%29

file:///home/retama/Descargas/download%20%285%29

This XML file does not appear to have any style information associated with it. The document tree is shown below.


```
▼<sparql xmlns="http://www.w3.org/2005/sparql-results#">
  ▼<head>
     <variable name="s"/>
     <variable name="p"/>
     <variable name="o"/>
   </head>
  ▼<results>
    ▼<result>
      ▼<binding name="s">
         <uri>http://purl.org/obo/owlGO_0001071</uri>
       </binding>
      ▼<binding name="p">
         <uri>http://purl.org/obo/owlcreation_date</uri>
       </binding>
      ▼<binding name="o">
         <literal>2010-10-21T04:37:54Z</literal>
       </binding>
     </result>
    ▼<result>
      ▼<binding name="s">
         <uri>http://purl.org/obo/owlGO_0044281</uri>
       </binding>
      ▼<binding name="p">
         <uri>http://purl.org/obo/owlcreation_date</uri>
       </binding>
      ▼<binding name="o">
         <literal>2010-01-26T12:05:20Z</literal>
       </binding>
     </result>
```


Figura 4.2: Resultado de la consulta descrita en 4.1 del sparql endpoint 4store.

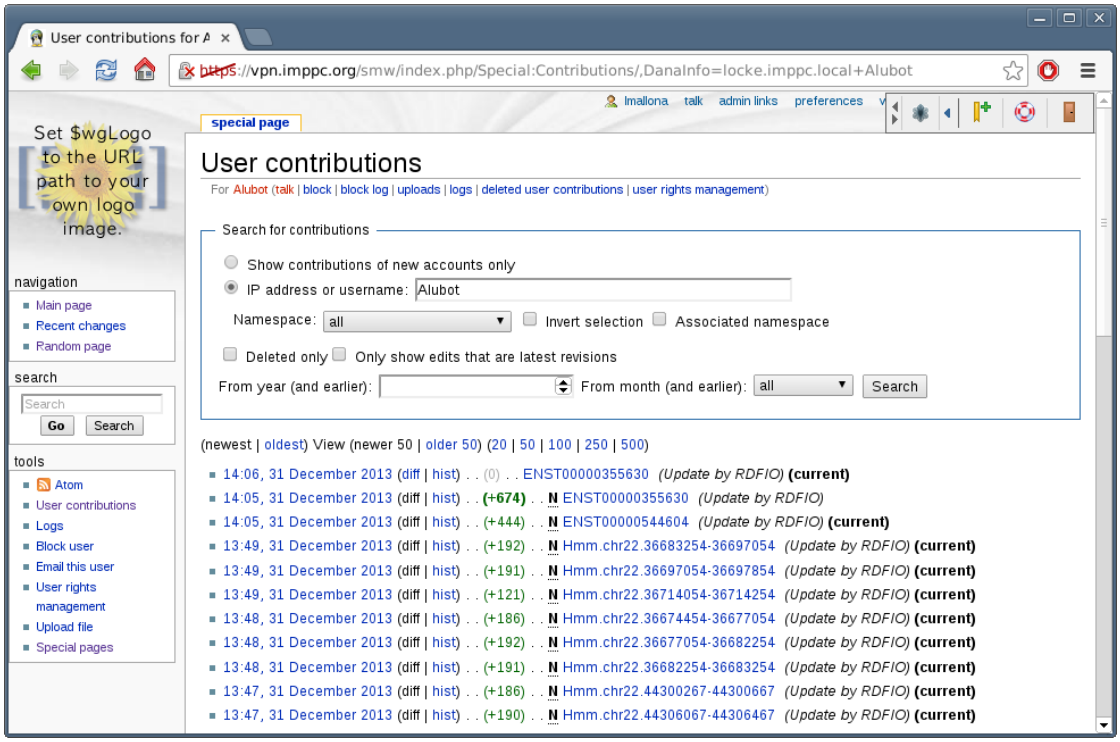


Figura 4.3: Registro de las importación de datos mediante la extensión RDFIO por parte del usuario Alubot.

la extensión `RDFIO` de Semantic MediaWiki (Marshall *et al.*, 2012). Puede consultarse parte del historial de contribuciones de tal usuario en la figura 4.3.

No obstante, la incorporación mediante RDFIO ha sido problématica, aun cuando la la ontología XML/RDF es válida según el W3C. Por esta razón, en el apéndice B se documenta una nueva clase, `Rdfio compliant formatters`, que genera un XML en el que los atributos de las propiedades de objeto y de dato se encuentran no como atributos en las etiquetas XML sino entre etiquetas de cierre y fin.

4.4. Consultas exploratorias

Los métodos `ask` de Semantic MediaWiki permiten realizar consultas sobre los datos menos farragosas que mediante SPARQL. Por ejemplo, el método `ask` ofrecido en la figura 4.4, que consulta las localizaciones y estados cromatínicos de los datos de la línea celular *hESC*, permite obtener el resultado 4.5. El poder de este tipo de sintaxis es su gran versatilidad: el usuario puede recabar datos empleando un lenguaje declarativo, y no procedimental, de acuerdo a sus necesidades; además, cabe diseñar plantillas para las

```
{{#ask:[[has_cell_line::hESC]]
|?#
|?Located in
|?Hmm
|format=broadtable
|link=all
|headers=show
|searchlabel=    further results
|class=sortable wikitable smwtable
}}
```

Figura 4.4: Código wiki para realizar la consulta ofrecida en la figura 4.5.

consultas más usuales, lo que proporciona una interfaz de acceso amigable a los datos [3].

4.4.1. Uso de plantillas

Mediante métodos `ask`

A fin de generar páginas con un contenido amigable para el usuario, se han definido plantillas que extraen los datos introducidos automáticamente por la extensión `RDFIO` y los introducen en un contexto de lenguaje natural. Por ejemplo, el transcrito `ENST00000400521` posee el wikicódigo ofrecido en la figura 4.6 ofrecerá el resultado de la figura 4.8 gracias a que emplea la plantilla 4.7. El flujo de datos para generar la página 4.8 es, por tanto, como sigue:

- Generación del OWL/RDF como queda descrito en el capítulo 3.
- Subida a la wiki mediante `RDFIO`.
- Renderización vía método `ask` que introduce los resultados en la plantilla `Template:ENST`.

Mediante plantillas anidadas

No obstante, es posible generar contenido embebido en lenguaje natural sin emplear métodos `ask`. En el caso de el artículo de prueba `Alu1`, cuya visualización se aporta como figura 4.9, la sustitución de parámetros en la plantilla es ligeramente más compleja. En este caso, el artículo posee el contenido descrito en la figura 4.10.

[3] Cabe reseñar que para obtener una única propiedad de una única página cabría emplear el método `show`, y para búsquedas más complejas, `ask`. En los ejemplos descritos en el presente capítulo se muestran tan solo consultas mediante `ask`; esto se debe a que, en el caso de la extracción de la información para páginas únicas (mediante `Template:Alubox` y `Template:Infobox`) se emplea la extensión Parserfunctions (Schindler and Vrandecic, 2009) más que en consultas *inline* convencionales.

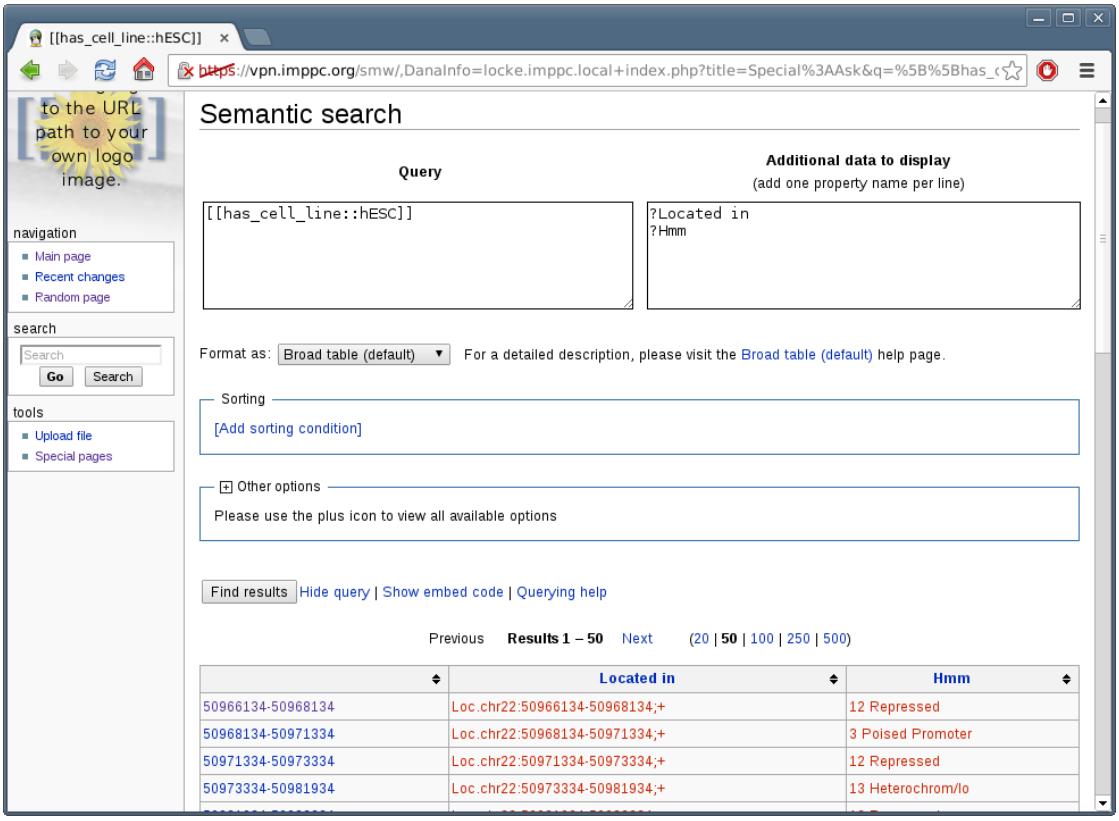


Figura 4.5: Resultado de la consulta presentada en 4.4

```
{{#ask: [[{{PAGENAME}}]]
|?Equivalent URI
|?Transcribed from
|?Biological process player
|?Involved in molecular function
|format = template
|template = ENST}}

[[Equivalent URI::http://locke.imppc.local/smw/ENST00000400521]]
[[Transcribed from::ENSG00000184470]]
[[Biological process player::GO:0007275]]
[[Involved in molecular function::GO:0003676]]

[[Category:Thing]]
[[Category:SO 0000833]]
[[Category:NamedIndividual]]

[[Category:SO:0000833]]
```

Figura 4.6: Wikicódigo del transcrito `ENST00000400521`, que está siendo visualizado mediante una plantilla con método `ask`.

```
The transcript with name '''{{PAGENAME}}''' has the equivalent URI
    {{{2}}}, is trascribed from {{{3}}}, has {{{4}}} as biological
    function and is involved in {{{5}}} as molecular function.
```

Figura 4.7: Wikicódigo de la plantilla `Template:ENST`.

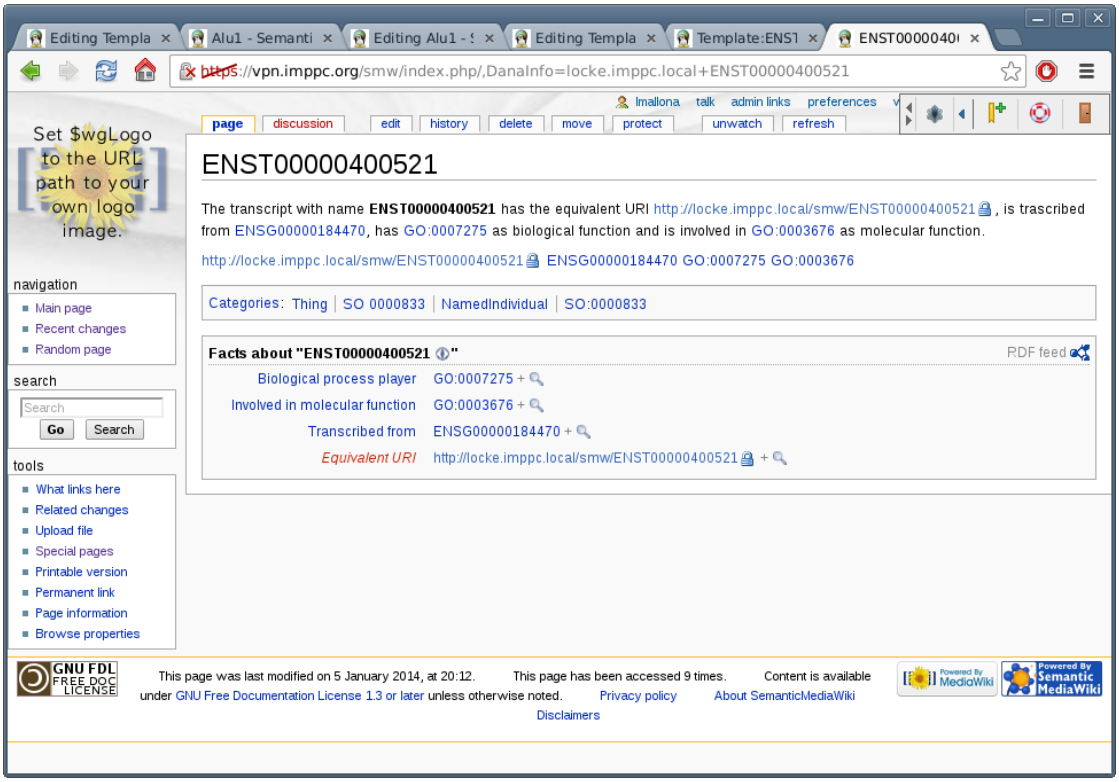


Figura 4.8: Visualización del artículo con wikicódigo 4.7.

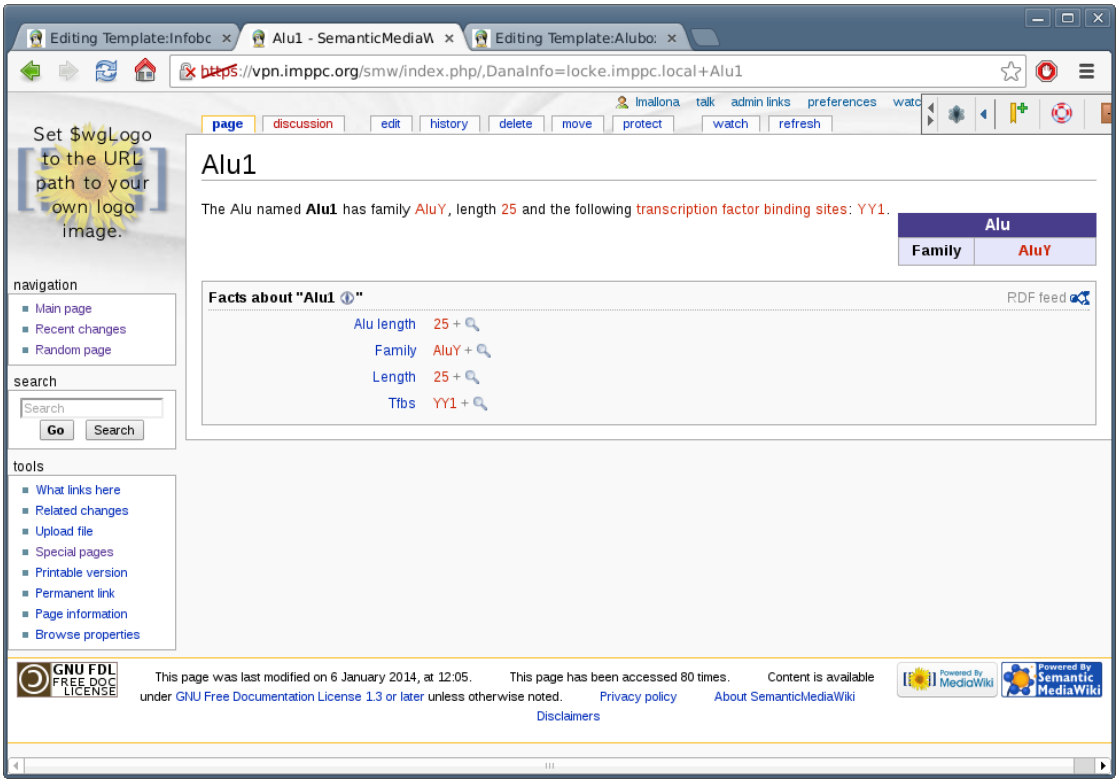


Figura 4.9: Visualización del artículo `Alu1`, cuyo wikicódigo se adjunta en la figura 4.10.

```
{{alubox
| family = [[family::AluY]]
| length = [[alu_length::25]]
| tfbs = [[tfbs::YY1]]
}}
```

Figura 4.10: Wikicódigo del artículo `Alu1`.

```
<includeonly>
{| border="0" style="background:#ffffff" align="right" class="sortable
    wikitable"
|+ align="center" style="background:DarkSlateBlue; color:white"|<big>Alu
    </big>
! width="60 px" style="background:Lavender; color:Black"   | Family
! width="100 px" style="background:Lavender; color:Black"  | [[family
    ::{{{family|}}}
|}

The Alu named '''{{PAGENAME}}''' has family [[family::{{{family|}}},
    length [[length::{{{length|}}} and the following [[transcription
    factor binding sites]]: [[tfbs::{{{tfbs|}}}.

__SHOWFACTBOX__
</includeonly>
```

Figura 4.11: Wikicódigo de la plantilla `Template:Alubox`.

El flujo de información en este caso precisa de dos plantillas, `Template:Infobox` y `Template:Alubox`. La primera, cuyo código se aporta en el apéndice A, permite la introducción de tantos campos en la plantilla como sea menester. La segunda, `Template:Alubox`, simplemente contiene el código para generar una tabla y una oración en lenguaje natural; tal wikicódigo se aporta como figura 4.11.

La incorporación de datos de la ontología empleando derivados de `Template:Infobox` aparenta una mayor flexibilidad que la variante basada en métodos `ask`. Su inconveniente, no obstante, es que la incorporación de los datos (procedentes del un XML) no podría realizarse mediante automáticamente mediante `RDFIO`. Cabría, en este caso, programar un *parser* para cada tipo de elemento instanciado que sustituyera la plantilla adecuada. Para este fin existe un *framework*, *pywikipediabot*, que, en conjunción con un iterador sobre XML (o incluso RDF, como `rdflib`), que ha sido empleado exitosamente en otros contextos (Herzig and Ell, 2010).

Capítulo 5

Conclusiones

Se ha generado una base de conocimiento de Alus que reúne información genética y epigenética e integra ontologías bien consolidadas, como Gene Ontology o Sequence Ontology.

Se han creado reglas que permiten transferir información de los elementos colindantes a cada Alu, ya se trate de información genética (anotación funcional del transcrito más cercano) o epigenética (estado cromatínico aledaño). El uso de estas reglas, proporcionadas con la ontología, y un razonador formal como Pellet (integrado, por ejemplo, como plugin en Protégé), permite integrar estos datos bioinformáticos.

Se ha explorado la adaptación de la ontología y su exportación parcial a un entorno MediaWiki para facilitar la visualización de los datos, así como su actualización.

5.1. Perspectiva

Como futuro trabajo, sería interesante ofrecer la AluWiki en un servidor Web de libre acceso, de manera que los usuarios pudiesen navegar por el conjunto de más de un millón de Alus humanas sin emplear software especializado e independientemente de las capacidades de sus máquinas.

En cuanto a la información epigenética incorporada, se han incluido estados cromatínicos tan sólo para la línea celular *human stem cells* (*hESC*); dado que existen datos públicos para otros tipos celulares (por ejemplo tumorales), y que la ontología ha sido diseñada para procesar un número ilimitado de muestras, sería productiva su descarga e inclusión.

5.2. Reflexión final

Tal y como se indicaba en el capítulo introductorio, el objetivo de la ontología de Alus es generar un medio de integración y visualización de datos que permita a distintos tipos de usuarios, incluso los no informáticos, extraer información sobre el estado de las Alus en distintos tipos de muestras. Información que no sea trivial, empleando a ser posible toda la potencia de la tecnología de la web semántica, por ejemplo aprovechando los razonadores formales.

Es un hecho que abundan los datos sobre las repeticiones Alu bajo múltiples condiciones: en líneas celulares, pacientes, individuos sanos e incluso distintas especies de primates. No obstante, aún se desconoce qué función tienen, si la tienen, o si varían coordinadamente con otros elementos genómicos (haya causalidad o no). Disponer de una herramienta de integración de datos, como la presente ontología, sobre la que realizar consultas o que realice inferencias, parece una necesidad.

Bibliografía

Arita, M. (2009). A pitfall of wiki solution for biological databases. *Briefings in bioinformatics*, 10(3):295–296. (Cited on page 41.)

Ashburner, M., Ball, C.A., Blake, J.A., Botstein, D., Butler, H., Cherry, J.M., Davis, A.P., Dolinski, K., Dwight, S.S., Eppig, J.T. *et al.* (2000). Gene Ontology: tool for the unification of biology. *Nature genetics*, 25(1):25–29. (Cited on pages 6, 8 and 11.)

Barnes, N. (2010). Publish your computer code: it is good enough. *Nature*, 467(7317):753–753. (Cited on page 1.)

Batzer, M.A. and Deininger, P.L. (2002). Alu repeats and human genomic diversity. *Nature Reviews Genetics*, 3(5):370–379. (Cited on pages 5 and 6.)

Beck, K. and Andres, C. (2004). *Extreme programming explained: embrace change.* Addison-Wesley Professional. (Cited on page 29.)

Berners-Lee, T., Hendler, J., Lassila, O. *et al.* (2001). The semantic web. *Scientific american*, 284(5):28–37. (Cited on page 3.)

Bodenreider, O. and Stevens, R. (2006). Bio-ontologies: current trends and future directions. *Briefings in bioinformatics*, 7(3):256–274. (Cited on page 6.)

Cock, P., Antao, T., Chang, J., Chapman, B., Cox, C., Dalke, A., Friedberg, I., Hamelryck, T., Kauff, F., Wilczynski, B. and de Hoon, M. (2009). Biopython: freely available Python tools for computational molecular biology and bioinformatics. *Bioinformatics*, 25(11):1422–1423. (Cited on page 1.)

Codd, E. (1970). A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387. (Cited on page 2.)

Conesa, J. and Olive, A. (2006). A method for pruning ontologies in the development of conceptual schemas of information systems. In: *Journal on Data Semantics V*, pp. 64–90. Springer. (Cited on page 6.)

Copeland, R. (2010). *Essential sqlalchemy.* O'Reilly. (Cited on page 28.)

Cordaux, R. and Batzer, M.A. (2009). The impact of retrotransposons on human genome evolution. *Nature Reviews Genetics*, 10(10):691–703. (Cited on page 5.)

Dale, R.K., Pedersen, B.S. and Quinlan, A.R. (2011). Pybedtools: a flexible Python library for manipulating genomic datasets and annotations. *Bioinformatics*, 27(24):3423–3424. (Cited on page 33.)

Dennis Jr, G., Sherman, B., Hosack, D., Yang, J., Gao, W., Lane, H. and Lempicki, R. (2003). DAVID: database for annotation, visualization, and integrated discovery. *Genome Biology*, 4(5):P3. (Cited on page 6.)

Dreszer, T.R., Karolchik, D., Zweig, A.S., Hinrichs, A.S., Raney, B.J., Kuhn, R.M., Meyer, L.R., Wong, M., Sloan, C.A., Rosenbloom, K.R. *et al.* (2012). The UCSC Genome Browser database: extensions and updates 2011. *Nucleic acids research*, 40(D1):D918–D923. (Cited on pages 7, 25 and 29.)

Eilbeck, K., Lewis, S.E., Mungall, C.J., Yandell, M., Stein, L., Durbin, R. and Ashburner, M. (2005). The Sequence Ontology: a tool for the unification of genome annotations. *Genome biology*, 6(5):R44. (Cited on pages 6, 8 and 11.)

Etzold, T., Ulyanov, A. and Argos, P. (1996). SRS: Information retrieval system for molecular biology data banks. *Methods in Enzymology*, 266:114–128. (Cited on page 2.)

Fujibuchi, W., Goto, S., Migimatsu, H., Uchiyama, I., Ogiwara, A., Akiyama, Y. and Kanehisa, M. (1998). DBGET/LinkDB: an integrated database retrieval system. In: *Pacific Symposium on Biocomputing*, volume 98, pp. 683–694. (Cited on page 2.)

Gennari, J.H., Musen, M.A., Fergerson, R.W., Grosso, W.E., Crubézy, M., Eriksson, H., Noy, N.F. and Tu, S.W. (2003). The evolution of Protégé: an environment for knowledge-based systems development. *International Journal of Human-computer studies*, 58(1):89–123. (Cited on page 11.)

Goto, N., Prins, P., Nakao, M., Bonnal, R., Aerts, J. and Katayama, T. (2010). BioRuby: Bioinformatics software for the Ruby programming language. *Bioinformatics*, 26(20):2617. (Cited on page 1.)

Grant, C.E., Bailey, T.L. and Noble, W.S. (2011). FIMO: scanning for occurrences of a given motif. *Bioinformatics*, 27(7):1017–1018. (Cited on page 28.)

Grau, B.C., Horrocks, I., Kazakov, Y. and Sattler, U. (2007). Just the right amount: extracting modules from ontologies. In: *Proceedings of the 16th international conference on World Wide Web*, pp. 717–726. ACM. (Cited on page 6.)

Harris, S., Lamb, N. and Shadbolt, N. (2009). 4store: The design and implementation of a clustered RDF store. In: *5th International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS2009)*, pp. 94–109. (Cited on page 42.)

Herzig, D.M. and Ell, B. (2010). Semantic mediawiki in operation: experiences with building a semantic portal. In: *The Semantic Web–ISWC 2010*, pp. 114–128. Springer. (Cited on page 49.)

Hoehndorf, R., Prufer, K., Backhaus, M., Herre, H., Kelso, J., Loebe, F. and Visagie, J. (2006). A proposal for a gene functions wiki. *Proceedings of OTM 2006 Workshops, Montpellier, France, Oct 29 - Nov 3, Part I, Workshop Knowledge Systems in Bioinformatics, KSinBIT of Lecture Notes in Computer Science*, 4277:669–678. (Cited on page 41.)

Hoehndorf, R., Bacher, J., Backhaus, M., Gregorio, S., Loebe, F., Prufer, K., Uciteli, A., Visagie, J., Herre, H. and Kelso, J. (2009). BOWiki: an ontology-based wiki for annotation of data and integration of knowledge in biology. *BMC Bioinformatics*, 10(Suppl 5):S5. (Cited on page 41.)

Holford, M.E., Khurana, E., Cheung, K.H. and Gerstein, M. (2010). Using semantic web rules to reason on an ontology of pseudogenes. *Bioinformatics*, 26(12):i71–i78. (Cited on pages 24 and 28.)

Holland, R.C.G., Down, T.A., Pocock, M., Prlić, A., Huen, D., James, K., Foisy, S., Dräger, A., Yates, A., Heuer, M. and Schreiber, M.J. (2008). BioJava: an open-source framework for bioinformatics. *Bioinformatics*, 24(18):2096–2097. (Cited on page 1.)

Horrocks, I., Patel-Schneider, P.F., Bechhofer, S. and Tsarkov, D. (2005). OWL rules: A proposal and prototype implementation. *Web Semantics: Science, Services and Agents on the World Wide Web*, 3(1):23–40. (Cited on page 12.)

Kantardzic, M. (2011). *Data mining: concepts, models, methods, and algorithms*. Wiley-IEEE Press. (Cited on page 2.)

Karolchik, D., Hinrichs, A.S., Furey, T.S., Roskin, K.M., Sugnet, C.W., Haussler, D. and Kent, W.J. (2004). The UCSC Table Browser data retrieval tool. *Nucleic acids research*, 32(suppl 1):D493–D496. (Cited on pages 25 and 29.)

Kent, W.J., Sugnet, C.W., Furey, T.S., Roskin, K.M., Pringle, T.H., Zahler, A.M. and Haussler, D. (2002). The human genome browser at UCSC. *Genome research*, 12(6):996–1006. (Cited on pages 25 and 29.)

Kim, J.W., Caralt, J.C. and Hilliard, J.K. (2007). Pruning bio-ontologies. In: *System Sciences, 2007. HICSS 2007. 40th Annual Hawaii International Conference on*, pp. 196c–196c. IEEE. (Cited on page 6.)

Knublauch, H., Fergerson, R.W., Noy, N.F. and Musen, M.A. (2004). The Protégé OWL plugin: An open development environment for semantic web applications. In: *The Semantic Web–ISWC 2004*, pp. 229–243. Springer. (Cited on pages 24 and 28.)

Köhler, J., Philippi, S. and Lange, M. (2003). SEMEDA: ontology based semantic integration of biological databases. *Bioinformatics*, 19(18):2420–2427. (Cited on page 3.)

Krötzsch, M., Vrandečić, D. and Völkel, M. (2006). Semantic mediawiki. In: *The Semantic Web-ISWC 2006*, pp. 935–942. Springer. (Cited on page 42.)

Larman, C. (2012). *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development, 3/e*. Pearson Education India. (Cited on page 29.)

Luscombe, N., Greenbaum, D. and Gerstein, M. (2001). What is bioinformatics? A proposed definition and overview of the field. *Methods of Information in Medicine*, 40(4):346–358. (Cited on page 1.)

Marshall, M.S., Boyce, R., Deus, H.F., Zhao, J., Willighagen, E.L., Samwald, M., Pichler, E., Hajagos, J., Prud'hommeaux, E. and Stephens, S. (2012). Emerging practices for mapping and linking life sciences data using RDF—A case series. *Web Semantics: Science, Services and Agents on the World Wide Web*, 14:2–13. (Cited on page 44.)

Matlin, A.J., Clark, F. and Smith, C.W. (2005). Understanding alternative splicing: towards a cellular code. *Nature Reviews Molecular Cell Biology*, 6(5):386–398. (Cited on page 32.)

Matys, V., Fricke, E., Geffers, R., Gößling, E., Haubrock, M., Hehl, R., Hornischer, K., Karas, D., Kel, A.E., Kel-Margoulis, O.V. *et al.* (2003). TRANSFAC®: transcriptional regulation, from patterns to profiles. *Nucleic acids research*, 31(1):374–378. (Cited on pages 21 and 27.)

McGrath, S. (2000). *XML processing with Python*. Prentice Hall. (Cited on page 28.)

Merali, Z. (2010). Computational science: Error, why scientific programming does not compute. *Nature*, 467(7317):775–777. (Cited on page 1.)

Mungall, C.J., Emmert, D.B. *et al.* (2007). A Chado case study: an ontology-based modular schema for representing genome-associated biological information. *Bioinformatics*, 23(13):i337–i346. (Cited on page 19.)

Muotri, A.R., Marchetto, M.C., Coufal, N.G. and Gage, F.H. (2007). The necessary junk: new functions for transposable elements. *Human molecular genetics*, 16(R2):R159–R167. (Cited on page 5.)

Noy, N.F., McGuinness, D.L. *et al.* (2001). Ontology development 101: A guide to creating your first ontology. (Cited on page 12.)

Nykanen, O. (2004). Metadata for learning resources: technologies and directions of the Semantic Web-a brief review. In: *Advanced Learning Technologies, 2004. Proceedings. IEEE International Conference on*, pp. 896–897. IEEE. (Cited on page 28.)

Paolella, G., Lucero, M.A., Murphy, M.H. and Baralle, F.E. (1983). The Alu family repeat promoter has a tRNA-like bipartite structure. *The EMBO journal*, 2(5):691. (Cited on page 5.)

Price, A.L., Eskin, E. and Pevzner, P.A. (2004). Whole-genome analysis of Alu repeat elements reveals complex evolutionary history. *Genome research*, 14(11):2245–2252. (Cited on page 13.)

Quinlan, A.R. and Hall, I.M. (2010). BEDTools: a flexible suite of utilities for comparing genomic features. *Bioinformatics*, 26(6):841–842. (Cited on pages 32 and 39.)

Raymond, E. (1999). The cathedral and the bazaar. *Knowledge, Technology & Policy*, 12(3):23–49. (Cited on page 1.)

Rhee, S.Y., Wood, V., Dolinski, K. and Draghici, S. (2008). Use and misuse of the gene ontology annotations. *Nature Reviews Genetics*, 9(7):509–515. (Cited on page 9.)

Sandelin, A., Alkema, W., Engström, P., Wasserman, W.W. and Lenhard, B. (2004). JASPAR: an open-access database for eukaryotic transcription factor binding profiles. *Nucleic acids research*, 32(suppl 1):D91–D94. (Cited on page 27.)

Schindler, M. and Vrandecic, D. (2009). Introducing new features to Wikipedia. *Proceedings of WebSci*, 9. (Cited on page 45.)

Seidenberg, J. and Rector, A. (2006). Web ontology segmentation: analysis, classification and use. In: *Proceedings of the 15th international conference on World Wide Web*, pp. 13–22. ACM. (Cited on page 6.)

Sirin, E., Parsia, B., Grau, B.C., Kalyanpur, A. and Katz, Y. (2007). Pellet: A practical owl-dl reasoner. *Web Semantics: science, services and agents on the World Wide Web*, 5(2):51–53. (Cited on pages 16, 24 and 28.)

Smit, A.F., Hubley, R. and Green, P. (1996). RepeatMasker Open-3.0. (Cited on page 29.)

Smith, B., Ashburner, M., Rosse, C., Bard, J., Bug, W., Ceusters, W., Goldberg, L.J., Eilbeck, K., Ireland, A., Mungall, C.J. *et al.* (2007). The OBO Foundry: coordinated evolution of ontologies to support biomedical data integration. *Nature biotechnology*, 25(11):1251–1255. (Cited on page 24.)

Stajich, J., Block, D., Boulez, K., Brenner, S., Chervitz, S., Dagdigian, C., Fuellen, G., Gilbert, J., Korf, I., Lapp, H. *et al.* (2002). The Bioperl toolkit: Perl modules for the life sciences. *Genome Research*, 12(10):1611. (Cited on page 1.)

Stein, L. (2002). Creating a bioinformatics nation. *Nature*, 417(6885):119–120. (Cited on page 2.)

Stein, L. *et al.* (2003). Integrating biological databases. *Nature Reviews Genetics*, 4(5):337–345. (Cited on page 3.)

Stevens, R., Goble, C., Baker, P. and Brass, A. (2001). A classification of tasks in bioinformatics. *Bioinformatics*, 17(2):180–188. (Cited on page 3.)

Stuckenschmidt, H. and Klein, M. (2004). Structure-based partitioning of large concept hierarchies. In: *The Semantic Web–ISWC 2004*, pp. 289–303. Springer. (Cited on page 6.)

Telnarova, Z. (2010). Relational database as a source of ontology creation. In: *Computer Science and Information Technology (IMCSIT), Proceedings of the 2010 International Multiconference on*, pp. 135–139. IEEE. (Cited on page 7.)

Tiffin, N., Kelso, J.F., Powell, A.R., Pan, H., Bajic, V.B. and Hide, W.A. (2005). Integration of text-and data-mining using ontologies successfully selects disease gene candidates. *Nucleic acids research*, 33(5):1544–1552. (Cited on page 7.)

Veyrieras, J.B., Kudaravalli, S., Kim, S.Y., Dermitzakis, E.T., Gilad, Y., Stephens, M. and Pritchard, J.K. (2008). High-resolution mapping of expression-QTLs yields insight into human gene regulation. *PLoS genetics*, 4(10):e1000214. (Cited on page 7.)

Witten, I., Frank, E. and Hall, M. (2011). *Data Mining: Practical Machine Learning tools and techniques*. Morgan Kaufmann. (Cited on page 2.)

Xing, J., Zhang, Y., Han, K., Salem, A.H., Sen, S.K., Huff, C.D., Zhou, Q., Kirkness, E.F., Levy, S., Batzer, M.A. *et al.* (2009). Mobile elements create structural variation: analysis of a complete human genome. *Genome research*, 19(9):1516–1526. (Cited on page 5.)

Apéndice A

Infobox

Se detalla el código de la plantilla `Template:Infobox` discutida en el capítulo 4.

```
<includeonly>
<table class="plainlinks" width=350px style="border: 1px solid #aaa;
    background-color: #f9f9f9; color: black; margin: 0.5em 0 0.5em 1em;
    padding: 0.2em; font-size: 90%;clear: right; float: right;"><!--
--><tr><td colspan=2 align=center>{{#if:{{{header|}}}|<big>'''{{{header
    }}}'''</big>}}</td></tr>
<!--
section 0
-->
{{#if:{{{s0_value_1|}}}{{{s0_field_1|}}}|<tr>{{#if:{{{s0_field_1|}}}|<td
    align=right width={{#if:{{{field_width|}}}|{{{field_width}}}|125px}}>
'''{{{s0_field_1}}}'''</td>}}<td colspan={{#if:{{{s0_field_1|}}}|1|2
    align=center}}>{{#if:{{{s0_value_1|}}}|{{{s0_value_1}}}}}</td></tr>}}
<!--
section 1
-->
{{#if:{{{section1|}}}|<tr><td colspan=2 align=center style="background-
    color: {{#if:{{{color|}}}|{{{color}}}|&#35;ccccff}};">'''{{{section1
    }}}'''</td></tr>}}<!--
-->{{#if:{{{s1_value_1|}}}{{{s1_field_1|}}}|<tr>{{#if:{{{s1_field_1|}}}|<
    td align=right width={{#if:{{{field_width|}}}|{{{field_width}}}|125px
    }}>
'''{{{s1_field_1}}}'''</td>}}<td colspan={{#if:{{{s1_field_1|}}}|1|2
    align=center}}>{{#if:{{{s1_value_1|}}}|{{{s1_value_1}}}}}</td></tr
    >}}<!--
-->{{#if:{{{s1_value_2|}}}{{{s1_field_2|}}}|<tr>{{#if:{{{s1_field_2|}}}|<
    td align=right>'''{{{s1_field_2}}}'''</td>}}<td colspan={{#if:{{{
    s1_field_2|}}}|1|2 align=center}}>
{{#if:{{{s1_value_2|}}}|{{{s1_value_2}}}}}</td></tr>}}<!--
-->{{#if:{{{s1_value_3|}}}{{{s1_field_3|}}}|<tr>{{#if:{{{s1_field_3|}}}|<
    td align=right>'''{{{s1_field_3}}}'''</td>}}<td colspan={{#if:{{{
    s1_field_3|}}}|1|2 align=center}}>
{{#if:{{{s1_value_3|}}}|{{{s1_value_3}}}}}</td></tr>}}<!--
-->{{#if:{{{s1_value_4|}}}{{{s1_field_4|}}}|<tr>{{#if:{{{s1_field_4|}}}|<
    td align=right>'''{{{s1_field_4}}}'''</td>}}<td colspan={{#if:{{{
    s1_field_4|}}}|1|2 align=center}}>
```

```
{{#if:{{{s1_value_4|}}}|{{{s1_value_4}}}}}</td></tr>}}<!--
-->{{#if:{{{s1_value_5|}}}{{{s1_field_5|}}}|<tr>{{#if:{{{s1_field_5|}}}|<
    td align=right>'''{{{s1_field_5}}}'''</td>}}<td colspan={{#if:{{{
    s1_field_5|}}}|1|2 align=center}}>
{{#if:{{{s1_value_5|}}}|{{{s1_value_5}}}}}</td></tr>}}<!--
-->{{#if:{{{s1_value_6|}}}{{{s1_field_6|}}}|<tr>{{#if:{{{s1_field_6|}}}|<
    td align=right>'''{{{s1_field_6}}}'''</td>}}<td colspan={{#if:{{{
    s1_field_6|}}}|1|2 align=center}}>
{{#if:{{{s1_value_6|}}}|{{{s1_value_6}}}}}</td></tr>}}<!--
-->{{#if:{{{s1_value_7|}}}{{{s1_field_7|}}}|<tr>{{#if:{{{s1_field_7|}}}|<
    td align=right>'''{{{s1_field_7}}}'''</td>}}<td colspan={{#if:{{{
    s1_field_7|}}}|1|2 align=center}}>
{{#if:{{{s1_value_7|}}}|{{{s1_value_7}}}}}</td></tr>}}<!--
-->{{#if:{{{s1_value_8|}}}{{{s1_field_8|}}}|<tr>{{#if:{{{s1_field_8|}}}|<
    td align=right>'''{{{s1_field_8}}}'''</td>}}<td colspan={{#if:{{{
    s1_field_8|}}}|1|2 align=center}}>
{{#if:{{{s1_value_8|}}}|{{{s1_value_8}}}}}</td></tr>}}<!--
-->{{#if:{{{s1_value_9|}}}{{{s1_field_9|}}}|<tr>{{#if:{{{s1_field_9|}}}|<
    td align=right>'''{{{s1_field_9}}}'''</td>}}<td colspan={{#if:{{{
    s1_field_9|}}}|1|2 align=center}}>
{{#if:{{{s1_value_9|}}}|{{{s1_value_9}}}}}</td></tr>}}<!--
-->{{#if:{{{s1_value_10|}}}{{{s1_field_10|}}}|<tr>{{#if:{{{s1_field_10
    |}}}|<td align=right>'''{{{s1_field_10}}}'''</td>}}<td colspan={{#if
    :{{{s1_field_10|}}}|1|2 align=center}}>
{{#if:{{{s1_value_10|}}}|{{{s1_value_10}}}}}</td></tr>}}<!--
-->{{#if:{{{s1_value_11|}}}{{{s1_field_11|}}}|<tr>{{#if:{{{s1_field_11
    |}}}|<td align=right>'''{{{s1_field_11}}}'''</td>}}<td colspan={{#if
    :{{{s1_field_11|}}}|1|2 align=center}}>
{{#if:{{{s1_value_11|}}}|{{{s1_value_11}}}}}</td></tr>}}<!--
-->{{#if:{{{s1_value_12|}}}{{{s1_field_12|}}}|<tr>{{#if:{{{s1_field_12
    |}}}|<td align=right>'''{{{s1_field_12}}}'''</td>}}<td colspan={{#if
    :{{{s1_field_12|}}}|1|2 align=center}}>
{{#if:{{{s1_value_12|}}}|{{{s1_value_12}}}}}</td></tr>}}<!--
-->{{#if:{{{s1_value_13|}}}{{{s1_field_13|}}}|<tr>{{#if:{{{s1_field_13
    |}}}|<td align=right>'''{{{s1_field_13}}}'''</td>}}<td colspan={{#if
    :{{{s1_field_13|}}}|1|2 align=center}}>
{{#if:{{{s1_value_13|}}}|{{{s1_value_13}}}}}</td></tr>}}<!--
-->{{#if:{{{s1_value_14|}}}{{{s1_field_14|}}}|<tr>{{#if:{{{s1_field_14
    |}}}|<td align=right>'''{{{s1_field_14}}}'''</td>}}<td colspan={{#if
    :{{{s1_field_14|}}}|1|2 align=center}}>
{{#if:{{{s1_value_14|}}}|{{{s1_value_14}}}}}</td></tr>}}<!--
-->{{#if:{{{s1_value_15|}}}{{{s1_field_15|}}}|<tr>{{#if:{{{s1_field_15
    |}}}|<td align=right>'''{{{s1_field_15}}}'''</td>}}<td colspan={{#if
    :{{{s1_field_15|}}}|1|2 align=center}}>
{{#if:{{{s1_value_15|}}}|{{{s1_value_15}}}}}</td></tr>}}<!--
-->{{#if:{{{s1_value_16|}}}{{{s1_field_16|}}}|<tr>{{#if:{{{s1_field_16
    |}}}|<td align=right>'''{{{s1_field_16}}}'''</td>}}<td colspan={{#if
    :{{{s1_field_16|}}}|1|2 align=center}}>
{{#if:{{{s1_value_16|}}}|{{{s1_value_16}}}}}</td></tr>}}<!--
-->{{#if:{{{s1_value_17|}}}{{{s1_field_17|}}}|<tr>{{#if:{{{s1_field_17
    |}}}|<td align=right>'''{{{s1_field_17}}}'''</td>}}<td colspan={{#if
    :{{{s1_field_17|}}}|1|2 align=center}}>
{{#if:{{{s1_value_17|}}}|{{{s1_value_17}}}}}</td></tr>}}<!--
-->{{#if:{{{s1_value_18|}}}{{{s1_field_18|}}}|<tr>{{#if:{{{s1_field_18
```

```
    |}}}|<td align=right>'''{{{s1_field_18}}}'''</td>}}<td colspan={{#if
    :{{{s1_field_18|}}}|1|2 align=center}}>
{{#if:{{{s1_value_18|}}}|{{{s1_value_18}}}}}</td></tr>}}<!--
-->{{#if:{{{s1_value_19|}}}{{{s1_field_19|}}}|<tr>{{#if:{{{s1_field_19
    |}}}|<td align=right>'''{{{s1_field_19}}}'''</td>}}<td colspan={{#if
    :{{{s1_field_19|}}}|1|2 align=center}}>
{{#if:{{{s1_value_19|}}}|{{{s1_value_19}}}}}</td></tr>}}<!--
-->{{#if:{{{s1_value_20|}}}{{{s1_field_20|}}}|<tr>{{#if:{{{s1_field_20
    |}}}|<td align=right>'''{{{s1_field_20}}}'''</td>}}<td colspan={{#if
    :{{{s1_field_20|}}}|1|2 align=center}}>
{{#if:{{{s1_value_20|}}}|{{{s1_value_20}}}}}</td></tr>}}
</table>
</includeonly>
```

Apéndice B

Documentación del código

Alu_ontology

Generated by Doxygen 1.8.5

Fri Jan 3 2014 12:05:18

Contents

1 Todo List

Member src.formatters.Formatters.format_alu

change the length computation to be done with the location

Member src.formatters.Formatters.format_gene

Member src.formatters.Formatters.format_gene_ontology

Mind that the go must be slimmed to the go slim prsent at the ontology level using map2slim.pl available to at http://amigo.geneontology.org/cgi-bin/amigo/slimmer

Member src.rdfio_compliant_formatters.Rdfio_compliant_formatters.format_alu

change the length computation to be done with the location

Member src.rdfio_compliant_formatters.Rdfio_compliant_formatters.format_gene

Member src.rdfio_compliant_formatters.Rdfio_compliant_formatters.format_gene_ontology

Mind that the go must be slimmed to the go slim prsent at the ontology level using map2slim.pl available to at http://amigo.geneontology.org/cgi-bin/amigo/slimmer

2 Deprecated List

Member src.parsers.Parsers.parse_alus

Highly unefficient, only for testing purposes

Member src.unix_subprocesses.Unix_subprocesses.bigbed_to_bed

as the data retrieval and conversion may be done at once bigbedtobed converter tool

Member src.unix_subprocesses.Unix_subprocesses.intersect_tfbs

as summarize_intersecting_features(self, a, b) does handle this

Member src.unix_subprocesses.Unix_subprocesses.wget_getter

wget fetching tool

3 Namespace Index

3.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

4 Class Index

4.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

5 Namespace Documentation

5.1 alu_ontology Namespace Reference

5.1.1 Detailed Description

Author

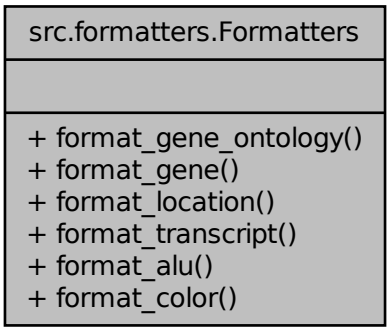
Izaskun Mallona Gets the XML/RDF formalization of an OWL ontology of human Alus

6 Class Documentation

6.1 src.formatters.Formatters Class Reference

The template-based RDF/XML formatting tool for Protégé and Pellet-compliant ontologies.

Collaboration diagram for src.formatters.Formatters:



Public Member Functions

- def format_gene_ontology

  The Gene Ontology item formatter.
- def format_gene

  The gene formatter.
- def format_location

  Formats a genomic coordinate that will be associated to a given feature, such as a transcript, gene, mark or the like.
- def format_transcript

  Formats a transcript and its annotations (GO) Takes as many bp, cc and mf annotations as provided (lists)
- def format_alu

  Formats a given Alu.
- def format_color

  Formats the hmm_hesc data retrieved with the hmm_hesc_color_getter() function Mind that the HMM do not have 'strand', although is requested.

6.1.1 Detailed Description

The template-based RDF/XML formatting tool for Protégé and Pellet-compliant ontologies.

6.1.2 Member Function Documentation

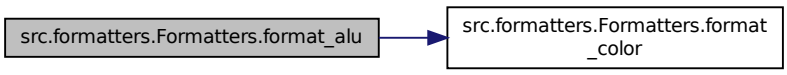
6.1.2.1 def src.formatters.Formatters.format_alu (*self, chrom, start, end, strand, distance, ensg, family, length, color*)

Formats a given Alu.

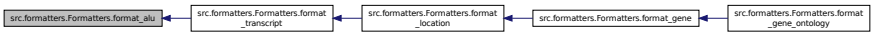
```
@param chrom string the chromosome
@param start string the start
@param end string the end
@param strand string the strand
@param distance int the distance to a given gene
@param ensg string the ensembl gene id
@param family string the alu family
@param length int the alu length
@param color string the HMM chromatin state color
@return the substituted Alu template
```

Todo change the length computation to be done with the location

Here is the call graph for this function:



Here is the caller graph for this function:



6.1.2.2 def src.formatters.Formatters.format_color (*self, chrom, start, end, name, strand, line*)

Formats the hmm_hesc data retrieved with the hmm_hesc_color_getter() function Mind that the HMM do not have 'strand', although is requested.

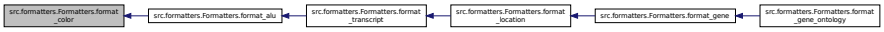
Parameters

chrom	string the chromosome
start	string the start
end	string the end
strand	string the strand
name	string the HMM chromatin state color
line	string the cell line the HMM was defined at

Returns

the substituted chromatin state template

Here is the caller graph for this function:



6.1.2.3 def src.formatters.Formatters.format_gene (*self, ensg, loc*)

The gene formatter.

Mind that this gets the cdsStart and cdsEnd of the ensGene table and thus the information passed to the different transcript variants does not contain which exons are mantained; a reimplementation of the transcript individuals including txStart and txEnd may make sense

Todo

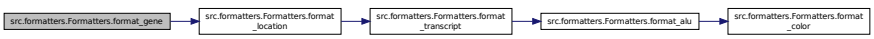
Parameters

ensg	string, the ensemble gene id
loc	string, the location

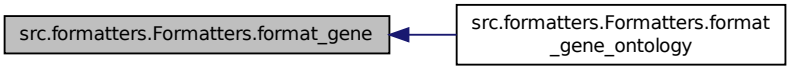
Returns

the substituted template

Here is the call graph for this function:



Here is the caller graph for this function:



6.1.2.4 def src.formatters.Formatters.format_gene_ontology (*self, go_id, go_descr*)

The Gene Ontology item formatter.

Todo Mind that the go must be slimmed to the go slim prsent at the ontology level using map2slim.pl available to at http://amigo.geneontology.org/cgi-bin/amigo/slimmer

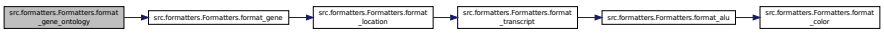
Parameters

go_id	string, the gene ontology id
go_descr	string, the gene ontology description

Returns

the substituted template

Here is the call graph for this function:



6.1.2.5 def src.formatters.Formatters.format_location (*self, chrom, start, end, strand*)

Formats a genomic coordinate that will be associated to a given feature, such as a transcript, gene, mark or the like.

Parameters

strand	must be either 'positive' or 'negative'

Returns

a tuple containing the loc accession as well the owl substituted template

Here is the call graph for this function:

Here is the caller graph for this function:

6.1.2.6 def src.formatters.Formatters.format_transcript (*self, enst, ensg, bp, cc, mf*)

Formats a transcript and its annotations (GO) Takes as many bp, cc and mf annotations as provided (lists)

Parameters

enst	string the enst
ensg	string the ensg
bp	list of strings of biological processes
cc	list of strings of cellular locations
mf	list of strings of molecular functions

Returns

the substituted template

Here is the call graph for this function:

Here is the caller graph for this function:

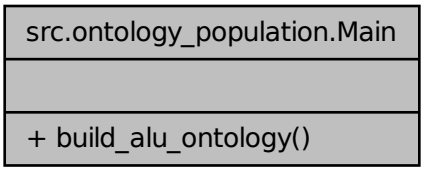
The documentation for this class was generated from the following file:

- src/formatters.py

6.2 src.ontology_population.Main Class Reference

Builds either a Protégé-compliant OWL or a RDFIO-compliant one.

Collaboration diagram for src.ontology_population.Main:



Public Member Functions

- def build_alu_ontology

  Main method of ontology making.

6.2.1 Detailed Description

Builds either a Protégé-compliant OWL or a RDFIO-compliant one.

6.2.2 Member Function Documentation

6.2.2.1 def src.ontology_population.Main.build_alu_ontology (*rdfio_compliant*)

Main method of ontology making.

Parameters

rdfio_compliant	boolean, whether the output is desired to be piped to RDFIO /SMW or to Protégé

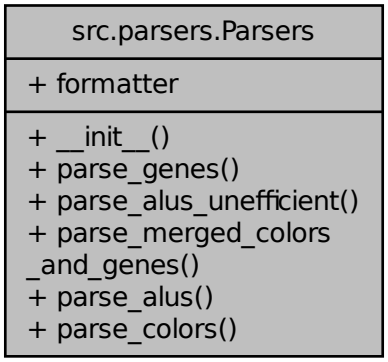
The documentation for this class was generated from the following file:

- src/ontology_population.py

6.3 src.parsers.Parsers Class Reference

Parsers and iterators for UCSC partly processed data.

Collaboration diagram for src.parsers.Parsers:



Public Member Functions

- def __init__

  Whether the formatter that will be used is for RDFIO-compliant data population or just classic, Protégé-compliant, OWL.
- def parse_genes

  Generates individuals for ENSEMBL genes and transcripts.
- def **parse_alus_unefficient**
- def parse_merged_colors_and_genes

  Iterator over bedtools output that extracts the alu as well the closest gene and color.
- def parse_alus
- def parse_colors

  Bundled with format_color for HMM colors.

Public Attributes

- **formatter**

6.3.1 Detailed Description

Parsers and iterators for UCSC partly processed data.

6.3.2 Constructor & Destructor Documentation

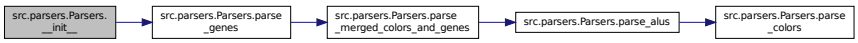
6.3.2.1 def src.parsers.Parsers.__init__ (*self, rdfio_compliant*)

Whether the formatter that will be used is for RDFIO-compliant data population or just classic, Protégé-compliant, OWL.

Parameters

rdfio_compliant	boolean whether RDFIO compliance is required

Here is the call graph for this function:

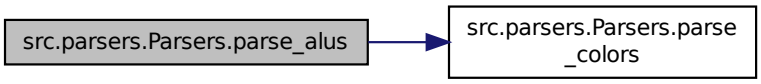


6.3.3 Member Function Documentation

6.3.3.1 def src.parsers.Parsers.parse_alus (*self, indiv_fh, closest_gene, closest_color*)

Deprecated Highly unefficient, only for testing purposes

Here is the call graph for this function:



Here is the caller graph for this function:

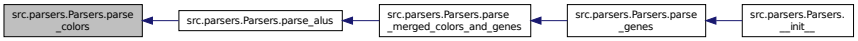
6.3.3.2 def src.parsers.Parsers.parse_colors (*self, indiv_fh, colors, cell_line*)

Bundled with format_color for HMM colors.

Parameters

indiv_fh	the output fh
colors	the data estructure containing the closest colors
cell_line	string the line the chromatin color was defined at

Here is the caller graph for this function:



6.3.3.3 def src.parsers.Parsers.parse_genes (*self, indiv_fh, data, genes*)

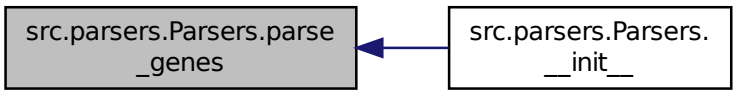
Generates individuals for ENSEMBL genes and transcripts.

Parameters

indiv_fh	a filehandle to write to
data	the data structure the iteration will be done at
genes	the genes dict

Here is the call graph for this function:

Here is the caller graph for this function:



6.3.3.4 def src.parsers.Parsers.parse_merged_colors_and_genes (*self, indiv_fh, merged*)

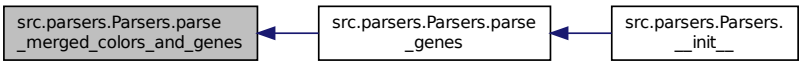
Iterator over bedtools output that extracts the alu as well the closest gene and color.

Parameters

indiv_fh	the filehandle to write to the unix-based merged bedtools outputs

Here is the call graph for this function:

Here is the caller graph for this function:



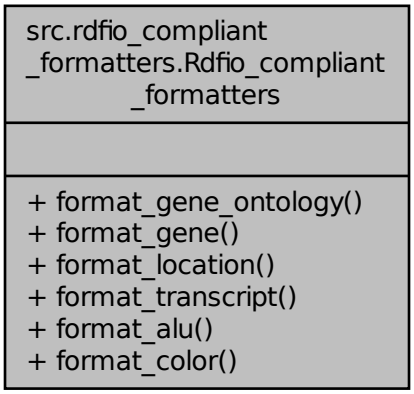
The documentation for this class was generated from the following file:

- src/parsers.py

6.4 src.rdfio_compliant_formatters.Rdfio_compliant_formatters Class Reference

The RDFIO/SMW complaint ontology formatter.

Collaboration diagram for src.rdfio_compliant_formatters.Rdfio_compliant_formatters:



Public Member Functions

- def format_gene_ontology
- def format_gene

  The gene formatter.
- def format_location

Formats a genomic coordinate that will be associated to a given feature, such as a transcript, gene, mark or the like.

- def format_transcript

  Formats a transcript and its annotations (GO) Takes as many bp, cc and mf annotations as provided (lists)

- def format_alu

  Formats a given Alu.

- def format_color

  Formats the hmm_hesc data retrieved with the hmm_hesc_color_getter() function Mind that the HMM do not have 'strand', although is requested.

6.4.1 Detailed Description

The RDFIO/SMW complaint ontology formatter.

6.4.2 Member Function Documentation

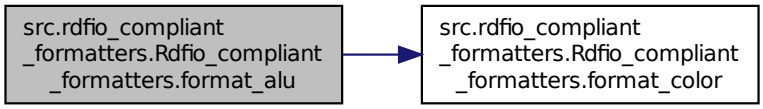
6.4.2.1 def src.rdfio_compliant_formatters.Rdfio_compliant_formatters.format_alu (*self, chrom, start, end, strand, distance, ensg, family, length, color*)

Formats a given Alu.

```
@param chrom string the chromosome
@param start string the start
@param end string the end
@param strand string the strand
@param distance int the distance to a given gene
@param ensg string the ensembl gene id
@param family string the alu family
@param length int the alu length
@param color string the HMM chromatin state color
@return the substituted Alu template
```

Todo change the length computation to be done with the location

Here is the call graph for this function:



Here is the caller graph for this function:

6.4.2.2 def src.rdfio_compliant_formatters.Rdfio_compliant_formatters.format_color (*self, chrom, start, end, name, strand, line*)

Formats the hmm_hesc data retrieved with the hmm_hesc_color_getter() function Mind that the HMM do not have 'strand', although is requested.

Parameters

chrom	string the chromosome
start	string the start
end	string the end
strand	string the strand
name	string the HMM chromatin state color
line	string the cell line the HMM was defined at

Returns

the substituted chromatin state template

Here is the caller graph for this function:

6.4.2.3 def src.rdfio_compliant_formatters.Rdfio_compliant_formatters.format_gene (*self, ensg, loc*)

The gene formatter.

Mind that this gets the cdsStart and cdsEnd of the ensGene table and thus the information passed to the different transcript variants does not contain which exons are mantained; a reimplementation of the transcript individuals including txStart and txEnd may make sense

Todo

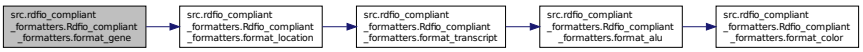
Parameters

ensg	string, the ensemble gene id
loc	string, the location

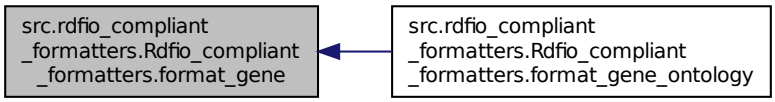
Returns

the substituted template

Here is the call graph for this function:



Here is the caller graph for this function:



6.4.2.4 def src.rdfio_compliant_formatters.Rdfio_compliant_formatters.format_gene_ontology (*self, go_id, go_descr*)

Todo Mind that the go must be slimmed to the go slim prsent at the ontology level using map2slim.pl available to at `http://amigo.geneontology.org/cgi-bin/amigo/slimmer`

Parameters

go_id	string, the gene ontology id
go_descr	string, the gene ontology description

Returns

the substituted template

Here is the call graph for this function:

6.4.2.5 def src.rdfio_compliant_formatters.Rdfio_compliant_formatters.format_location (*self, chrom, start, end, strand*)

Formats a genomic coordinate that will be associated to a given feature, such as a transcript, gene, mark or the like.

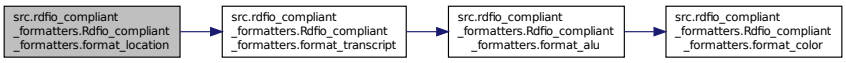
Parameters

strand	must be either 'positive' or 'negative'

Returns

a tuple containing the loc accession as well the owl substituted template

Here is the call graph for this function:



Here is the caller graph for this function:

6.4.2.6 def src.rdfio_compliant_formatters.Rdfio_compliant_formatters.format_transcript (*self, enst, ensg, bp, cc, mf*)

Formats a transcript and its annotations (GO) Takes as many bp, cc and mf annotations as provided (lists)

Parameters

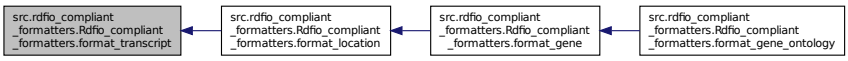
enst	string the enst
ensg	string the ensg
bp	list of strings of biological processes
cc	list of strings of cellular locations
mf	list of strings of molecular functions

Returns

the substituted template

Here is the call graph for this function:

Here is the caller graph for this function:



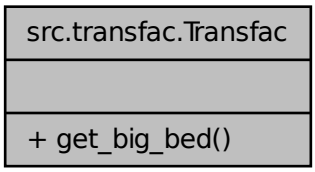
The documentation for this class was generated from the following file:

- src/rdfio_compliant_formatters.py

6.5 src.transfac.Transfac Class Reference

A FIMO-based transcription factor binding sites getter Still on development.

Collaboration diagram for src.transfac.Transfac:



Public Member Functions

- def get_big_bed

  A method to fetch the Noble's precomputed data with transfac, fimo on hg19.

6.5.1 Detailed Description

A FIMO-based transcription factor binding sites getter Still on development.

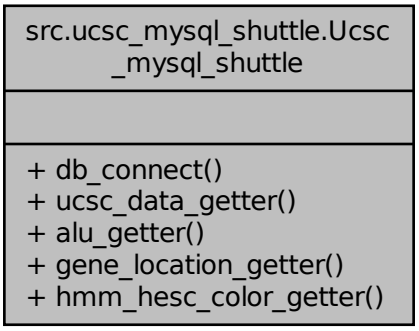
The documentation for this class was generated from the following file:

- src/transfac.py

6.6 src.ucsc_mysql_shuttle.Ucsc_mysql_shuttle Class Reference

Connects to the UCSC MySQL public server to fetch the data for the Alu ontology population.

Collaboration diagram for src.ucsc_mysql_shuttle.Ucsc_mysql_shuttle:



Public Member Functions

- def db_connect
- def ucsc_data_getter

  Fetches the genomic data from the UCSC public MySQL Server.
- def alu_getter

  Fetches from the UCSC public MySQL Server some data from hg19 Alus.
- def gene_location_getter

  Fetches the lowest txStart and longest txEnd of the transcripts of a given gene to assign a genomic coordinate to it Queries the UCSC public MySQL server.
- def hmm_hesc_color_getter

  Queries the UCSC data for Schema for Broad ChromHMM - Chromatin State Segmentation by HMM from ENCODE/Broad.

6.6.1 Detailed Description

Connects to the UCSC MySQL public server to fetch the data for the Alu ontology population.

6.6.2 Member Function Documentation

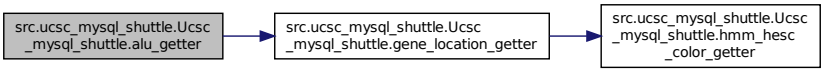
6.6.2.1 def src.ucsc_mysql_shuttle.Ucsc_mysql_shuttle.alu_getter (*self, db*)

Fetches from the UCSC public MySQL Server some data from hg19 Alus.

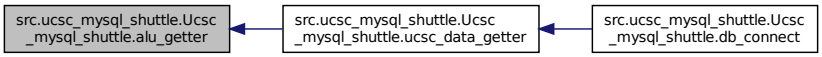
Parameters

db,the	database connection

Here is the call graph for this function:



Here is the caller graph for this function:



6.6.2.2 def src.ucsc_mysql_shuttle.Ucsc_mysql_shuttle.db_connect (*self*)

Returns

a connection to the database

Here is the call graph for this function:

6.6.2.3 def src.ucsc_mysql_shuttle.Ucsc_mysql_shuttle.gene_location_getter (*self, db*)

Fetches the lowest txStart and longest txEnd of the transcripts of a given gene to assign a genomic coordinate to it Queries the UCSC public MySQL server.

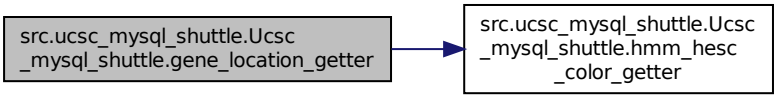
Parameters

db	the database connection

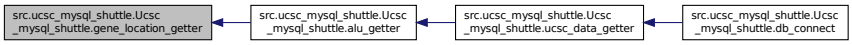
Returns

a dictionary of locations hashed by ensg identifier

Here is the call graph for this function:



Here is the caller graph for this function:



6.6.2.4 def src.ucsc_mysql_shuttle.Ucsc_mysql_shuttle.hmm_hesc_color_getter (*self, db*)

Queries the UCSC data for Schema for Broad ChromHMM - Chromatin State Segmentation by HMM from ENCODE/Broad.

Parameters

db	the database connection

Here is the caller graph for this function:

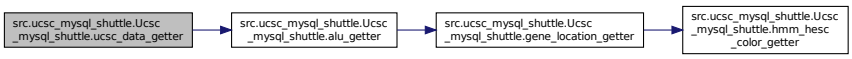
6.6.2.5 def src.ucsc_mysql_shuttle.Ucsc_mysql_shuttle.ucsc_data_getter (*self, db*)

Fetches the genomic data from the UCSC public MySQL Server.

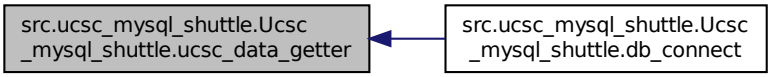
Parameters

db	the database connection

Here is the call graph for this function:



Here is the caller graph for this function:



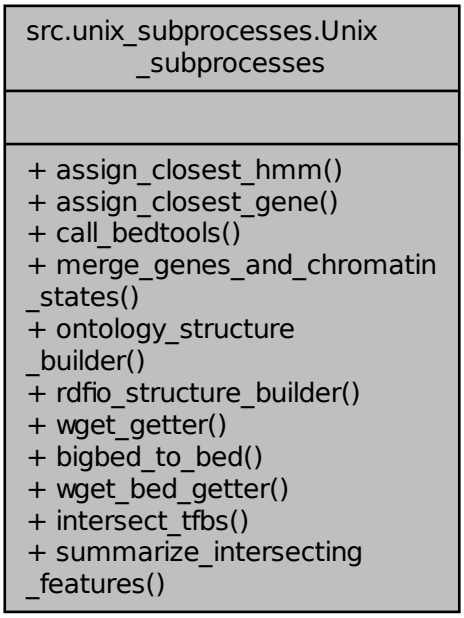
The documentation for this class was generated from the following file:

- src/ucsc_mysql_shuttle.py

6.7 src.unix_subprocesses.Unix_subprocesses Class Reference

Utilities related with Unix built-ins, such merging, awking and the like In most of the cases, bedtools are used.

Collaboration diagram for src.unix_subprocesses.Unix_subprocesses:



Public Member Functions

- def assign_closest_hmm

  Closest chromatin color assign by bedtools.
- def assign_closest_gene

  Closest gene assign by bedtools.
- def call_bedtools

  Calls a bedtools closestBed via a shell subprocess.
- def merge_genes_and_chromatin_states

  Takes to lists of closestBed outputs and merges them according to their first three columns.
- def ontology_structure_builder

  Concatenates the already designed elements of the ontology to the individuals passed as parameter (that are computed by this very same script)
- def rdfio_structure_builder

  Concatenates the already designed elements of the ontology to the individuals passed as parameter (that are computed by this very same script) in a manner compliant with the RDFIO tool for Semantic MediaWiki integration.
- def wget_getter

  wget tool
- def bigbed_to_bed
- def wget_bed_getter

  Retrieves silently a bigbedfile from a given url and transforms it using bigBedToBed into a bedfile Designed from FIMO-genome-wide-precomputed TRANSFAC motifs.
- def intersect_tfbs

  A bedtools intersect tool for TFBS detection inside Alus.
- def summarize_intersecting_features

intersectBed -a hg_19_track_repeatMasker_table_rmsk_repFamily_matches_Alu.bed -b transfac.bed -loj -wb | groupBy -g 1,2,3,4 -c 10 -o collapse > colapsed

6.7.1 Detailed Description

Utilities related with Unix built-ins, such merging, awking and the like In most of the cases, bedtools are used.

6.7.2 Member Function Documentation

6.7.2.1 def src.unix_subprocesses.Unix_subprocesses.assign_closest_gene (*self, alus, genes, tmp*)

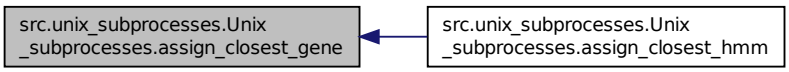
Closest gene assign by bedtools.

Parameters

alus	object containing the alus (tuple of tuples)
genes	the dict containing the genes
tmp	the temp folder that will be created and/or emptied afterwards

Here is the call graph for this function:

Here is the caller graph for this function:



6.7.2.2 def src.unix_subprocesses.Unix_subprocesses.assign_closest_hmm (*self, alus, hmm, tmp*)

Closest chromatin color assign by bedtools.

Parameters

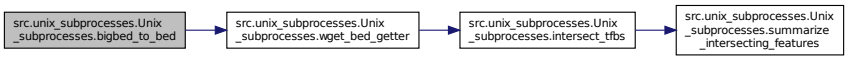
alus	object containing the alus (tuple of tuples)
genes	the dict containing the genes
tmp	the temp folder that will be created and/or emptied afterwards

Here is the call graph for this function:

6.7.2.3 def src.unix_subprocesses.Unix_subprocesses.bigbed_to_bed (*self, bb, bed*)

Deprecated as the data retrieval and conversion may be done at once bigbedtobed converter tool

Here is the call graph for this function:



Here is the caller graph for this function:

6.7.2.4 def src.unix_subprocesses.Unix_subprocesses.call_bedtools (*self, a, b, tmp*)

Calls a bedtools closestBed via a shell subprocess.

Parameters

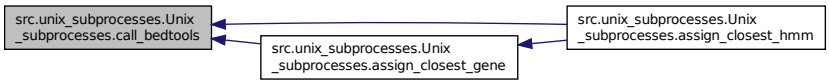
a	the bedfile a
b	the bedfie b
tmp	the tmp folder the processing will be done at

Returns

a list of bed-like strings

Here is the call graph for this function:

Here is the caller graph for this function:

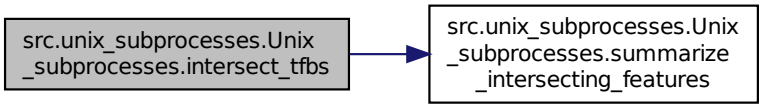


6.7.2.5 def src.unix_subprocesses.Unix_subprocesses.intersect_tfbs (*self, bed_file, fimo_file*)

A bedtools intersect tool for TFBS detection inside Alus.

Deprecated as summarize_intersecting_features(self, a, b) does handle this

Here is the call graph for this function:



Here is the caller graph for this function:

6.7.2.6 def src.unix_subprocesses.Unix_subprocesses.merge_genes_and_chromatin_states (*self, closest_gene, closest_color, temp*)

Takes to lists of closestBed outputs and merges them according to their first three columns.

Requires the two files to be sorted, as calls the unix 'join' as subprocess.

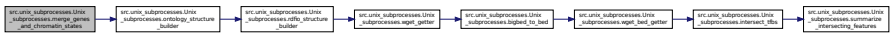
Returns

the joined list (an item per alu)

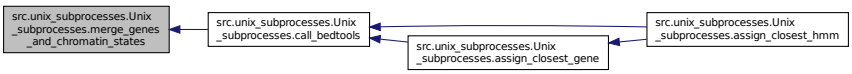
Parameters

temp	the temp folder the processing will be done at
closest_gene	the nearest gene
closest_color	the narest chromatin state

Here is the call graph for this function:



Here is the caller graph for this function:



6.7.2.7 def src.unix_subprocesses.Unix_subprocesses.ontology_structure_builder (*self, data_path, individuals_owl, out_owl*)

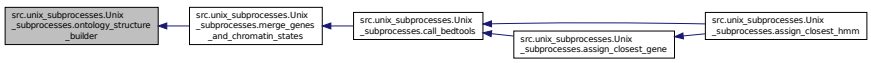
Concatenates the already designed elements of the ontology to the individuals passed as parameter (that are computed by this very same script)

Parameters

individuals_owl	the fn of the individuals file
data_path,the	folder which the other owl files are located at
out_owl,the	file that will receive the fully populated ontology

Here is the call graph for this function:

Here is the caller graph for this function:



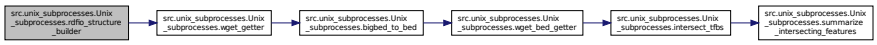
6.7.2.8 def src.unix_subprocesses.Unix_subprocesses.rdfio_structure_builder (*self, data_path, individuals_owl, out_owl*)

Concatenates the already designed elements of the ontology to the individuals passed as parameter (that are computed by this very same script) in a manner compliant with the RDFIO tool for Semantic MediaWiki integration.

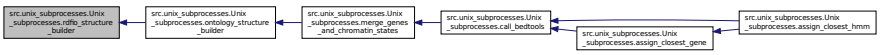
Parameters

individuals_owl	the fn of the individuals file
data_path,the	folder which the other owl files are located at
out_owl,the	file that will receive the fully populated ontology

Here is the call graph for this function:



Here is the caller graph for this function:



6.7.2.9 def src.unix_subprocesses.Unix_subprocesses.summarize_intersecting_features (*self, a, b*)

intersectBed -a hg_19_track_repeatMasker_table_rmsk_repFamily_matches_Alu.bed -b transfac.bed -loj -wb | groupBy -g 1,2,3,4 -c 10 -o collapse > colapsed

Parameters

a	bed file, i.e. the hg19 aluome
b	the transfac bed obtained from wget_bed_getter

Returns

the summarized bedfile

Here is the caller graph for this function:

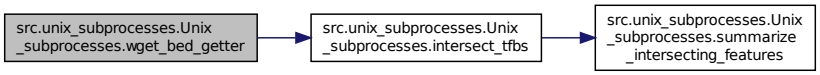
6.7.2.10 def src.unix_subprocesses.Unix_subprocesses.wget_bed_getter (*self, url, dest_bb, dest_bed*)

Retrieves silently a bigbedfile from a given url and transforms it using bigBedToBed into a bedfile Designed from FIMO-genome-wide-precomputed TRANSFAC motifs.

Parameters

dest_bb,the	destination bigbed
dest_bed,the	destination bedfile

Here is the call graph for this function:



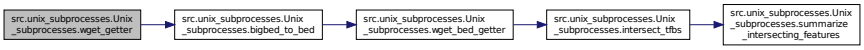
Here is the caller graph for this function:

6.7.2.11 def src.unix_subprocesses.Unix_subprocesses.wget_getter (*self, url, destination*)

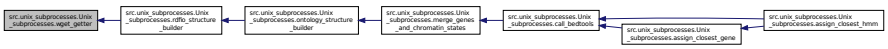
wget tool

Deprecated wget fetching tool

Here is the call graph for this function:



Here is the caller graph for this function:



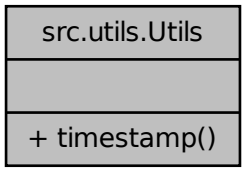
The documentation for this class was generated from the following file:

- src/unix_subprocesses.py

6.8 src.utils.Utils Class Reference

Simple utilities.

Collaboration diagram for src.utils.Utils:



Public Member Functions

- def timestamp

  Returns a formatted timestamp.

6.8.1 Detailed Description

Simple utilities.

The documentation for this class was generated from the following file:

- src/utils.py

Index