UNIVERSIDAD
REY JUAN CARLOS

INGENIERÍA INFORMÁTICA

Curso Académico 2008/2009

Proyecto Fin de Carrera

Métricas orientadas a objetos en GObject

Autor: Esteban Sánchez Muñoz
Tutor: Gregorio Robles Martínez
Cotutor: Carlos García Campos

A mis familia, presente y futura.

Agradecimientos

Resumen

Las métricas de software ayudan a dar una visión cuantitativa y cualitativa de un proyecto de software. Estas no sólo sirven para realizar estudios a posteriori, sino que con ellas podemos realizar estimaciones sobre sus costes de mantenimiento o ampliación, así como para nuevos proyectos.

Entre los esfuerzos dedicados a las estimaciones y métricas de software destaca QUALOSS[1] , un proyecto europeo que propone metodologías de alto nivel en el ámbito del software libre. El objetivo de QUALOSS es facilitar herramientas y metodologías para mejorar la productividad y calidad de la industria europea del software. A través de este esfuerzo, se intenta facilitar las decisiones estratégicas de integrar y adecuar el software libre con otros sistemas.

Dentro de las diferentes métricas existentes, las hay de ámbito general o específicas de una metodología de programación. Como generales serían número de líneas de código o puntos de función, mientras que existirían métricas específicas para programación estructurada, funcional o orientada a objetos.

Por otra parte, en cuanto a los lenguajes de programación existentes, si bien es cierto que hay algunos mejor adaptados a una metodología, en muchas ocasiones no es del todo determinante. Un ejemplo de ello es el escritorio libre GNOME, cuyo núcleo está realizado en C, en el que se ha desarrollado una librería[2] , llamada GObject, que proporciona un sistema de objetos y permite usar dicha metrodología en este lenguaje.

Sin embargo, el código desarrollado con este sistema, sigue siendo C, y no es fácil por tanto realizar métricas orientadas a objetos mirando únicamente el código fuente. La inexistencia de palabras claves que definen cuándo comienza una clase o un método de un objeto no lo hacen posible.

Todo esto cambió gracias a una herramienta llamada GObject Introspection, cuya

[1] http://www.qualoss.eu/

[2] La traducción al castellano de library es biblioteca y no librería, sin embargo hoy en día está extendido el uso de la palabra librería para referirse a las bibliotecas de programación.

finalidad es analizar el código y generar un fichero XML con toda la información de las clases y métodos existentes en una aplicación desarrollada con GObject. La finalidad de esta herramienta está alejada de la realización de métricas, pues sirve para poder realizar bindings [3]automáticamente de esos objetos en otro lenguaje de programación.

Con la información facilitada por esta herramienta, se podrían realizar métricas orientadas a objetos de una manera sencilla. Aunque no es posible realizar todo tipo de métricas, sí que se pueden conocer algunas de las más básicas y abrir una senda de futuro para obtener aun más.

En una primera parte introductoria se presentarán los conceptos y definiciones utilizadas en el proyecto. A continuación, en el Capítulo 2, se presentarán los objetivos de este proyecto, para en el siguiente capítulo entrar en los detalles de cómo ha sido desarrollado y cómo obtiene los datos. Por último, se presentará un pequeño estudio realizado sobre los datos que han podido ser obtenidos hasta la fecha, para terminar con unas conclusiones generales.

[3]Un binding es una adaptación de una librería para ser usada en un lenguaje de programación distinto de aquel en el que ha sido escrita. Se puede traducir como ligadura, aunque se prefiere el término en inglés por ser el más ampliamente usado.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

Este capítulo presenta los conceptos utilizados en el proyecto. Empezando por el escritorio GNOME, y las librerías principales que utiliza: GLib y GTK+. Una vez explicados estos conceptos más básicos, se comenta la herramienta GObject instropection, que sirve de base para el desarrollo del proyecto. A continuación, se presenta CVSAnalY, una herramienta utilizada en el grupo de investigación GSyC/LibreSoft. Por último, se introducen algunas de las métricas orientadas a objetos más importantes.

1.1. Software libre

El software libre ha pasado ya la barrera del fenómeno para convertirse en una realidad. Gracias a él se ha conseguido ahorro de costes, creación de puestos de trabajo locales o el impulso de estándares abiertos que permiten un marco de competitividad. En el ámbito de la ingeniería del software proporciona un basto abanico de posibilidades de estudio, al tener por lo general un desarrollo público.

Las técnicas de estimación de esfuerzos que se han venido utilizando en la ingeniería del software se basan, en su mayoría, en el tamaño del proyecto en líneas de código. Pero sobre todo, todas ellas, requieren de una gran cantidad de información para poder minimizar el error y ser suficientemente precisas. En este punto es donde el software libre destaca positivamente sobre el modelo de software privativo tradicional. En el modelo de software libre prácticamente todo el ciclo de vida es público, por lo que la cantidad de información disponible es enorme.

Generalmente cuando se utilizan técnicas de estimación basadas en el tamaño del proyecto en líneas de código, el problema queda trasladado a estimar el tamaño del

proyecto y poder así aplicar las ecuaciones de esfuerzo. Para llevar a cabo esta estimación de tamaño es común recurrir a la comparación del proyecto que se va a acometer con otros proyectos existentes. En el modelo de software privativo se utilizan proyectos realizados dentro de la propia empresa u organización, puesto que son los únicos de los que dispone de información suficiente. Sin embargo en software libre la base de datos de proyectos públicos es muy grande, por lo que es más fácil encontrar un proyecto que encaje bien con el que se pretende desarrollar.

Pero no sólo se trata de cantidad de información, sino que el software libre ofrece información muy precisa, porque no proviene de informes realizados a fin de mes sobre el estado del proyecto o documentación realizada a final de un proyecto, sino del desarrollo real que se lleva a cabo día a día. Se trata, por tanto, de información realmente fiable y actualizada. Algunas de estas fuentes de información son las siguientes:

- **Repositorios de código**.

  Uno de los pilares del desarrollo de software libre son los repositorios de código fuente. No se trata de meros almacenes de código, sino que son sistema de control de versiones que permiten el trabajo en grupo. Gracias estos sistema de control de versiones no sólo tenemos el código accesible sino que además tenemos mucha información relacionada, como el autor de cada cambio realizado en el código, así como el instante de tiempo en que se hizo dicho cambio. De esta manera es posible estudiar un proyecto desde muchos puntos de vista: cómo evoluciona en el tiempo, cuántos desarrolladores han sido necesarios así como el nivel de aportación de cada uno, etc. [Robles et al., 2004]

- **Sistemas de seguimiento de errores**.

  Los sistemas de seguimiento de errores son otra parte básica para el desarrollo de software libre en grupo y repleta de información útil para la comunidad investigadora. Estos sistemas registran los errores detectados por los usuarios o por los propios desarrolladores para que sean resueltos en el futuro. Estos errores pueden ser catalogados según el tipo, la importancia o la urgencia. Gracias a estos sistemas, las pruebas no quedan como algo secreto entre el equipo de desarrollo y el cliente, sino que toda la comunidad puede ser partícipe de dichas pruebas, aportando sus propias experiencias, e incluso proponiendo parches que podrán ser incluidos posteriormente en el sistema de control de versiones.

Hoy en día los sistema de seguimiento de errores se utilizan, además de para informar de un error en el programa, para proponer una nueva funcionalidad a los desarrolladores, como sistema de tareas pendientes para el equipo de desarrollo, etc. Cuantos más usos se le da, más información ofrece y más útil resulta para el estudio del software.

- **Listas de correo**.

  La comunicación es parte fundamental en cualquier desarrollo de software. Sin embargo, el software libre es desarrollado, muy a menudo, por equipos de desarrolladores cuyos miembros se encuentran geográficamente dispersos por el mundo. Esto hace imposible realizar reuniones presenciales periódicas para debatir aspectos relacionados con el proyecto. Las listas de correo se han convertido en ese *lugar* de reuniones, donde los desarrolladores pueden hablar, debatir y tomar decisiones sin importar su situación geográfica. Las listas de correo resultan incluso más interesantes si tenemos en cuenta que, generalmente, son públicas, de manera que cualquiera puede participar. Además, todos los debates, discusiones, decisiones, etc. se encuentran disponibles para todo el mundo.

Cada una de estas fuentes de información pueden ser, además, relacionadas entre sí, puesto que todas ellas ofrecen datos que suceden en tiempo real sobre un mismo proyecto de software, Así, por ejemplo, se puede ver como un error reportado al sistema de seguimiento de errores, es corregido posteriormente por un desarrollador que realiza una modificación en el sistema de control de versiones. La combinación de todas estas fuentes de información enriquece las técnicas de estimación de esfuerzos proporcionando información abundante, precisa y actualizada. Además de enriquecer las técnicas de estimación, esta información nos permite conocer el esfuerzo requerido para tareas o trabajos ya realizados, información que, en el caso concreto del software libre, es desconocida en la mayoría de los casos.

1.2. Ingeniería del software empírica

La ingeniería del software empírica es una rama de la ingeniería del software que se apoya en el estudio (normalmente de carácter estadístico) de datos del desarrollo de los proyectos para tratar de comprender el proceso de desarrollo de software. La

ingeniería del software empírica necesita, por tanto, de la mayor cantidad de datos posible para su posterior estudio.

Debido a esta necesidad de datos, parece que la ingeniería del software empírica no se adapta muy bien al modelo de software privativo, donde el secretismo está muy presente y la cantidad de datos es muy escasa, además de muy poco fiable en la mayoría de los casos. Para garantizar el éxito es necesario que el tamaño de la muestra sea considerable, si sólo disponemos de datos de unos pocos proyectos no es posible generalizar y sacar conclusiones. Sin embargo, hemos visto en la Sección 1.1 que la principal ventaja del software libre para la ingeniería del software es la gran cantidad de datos públicos disponibles así como su precisión y fiabilidad, lo que lo convierte en el escenario idóneo para la ingeniería del software empírica.

1.3. GNOME

GNOME son las siglas de GNU Network Object Model Environment. Es un entorno de escritorio libre para los sistemas operativos derivados de UNIX como Linux , BSD o Solaris . Fue creado por los mexicanos Miguel de Icaza y Federico Mena Quintero en 1997, basándose en GTK+, un toolkit existente para el desarrollo del programa de edición de imágenes GIMP.

Por aquel entonces, existía el escritorio KDE, pero se basaba en unas librerías que en ese momento no se ofrecían con una licencia libre. En este marco, surgió GNOME, que se planteó para poder ofrecer un escritorio totalmente libre.

Dentro de GNOME existen diferentes subproyectos, que forman parte de la estructura de GNOME. Entre ellos cabe destacar GTK+, Metacity, GConf, GStreamer o Nautilus. Algunos de ellos se comentarán posteriormente, mientras que otros son aplicaciones finales de usuario que no es necesario conocer para el entendimiento de este trabajo.

Su arquitectura se basa en diferentes capas, definidas e implementadas en varias librerías. En la base de todo se encuentra GLib, librería base para GObject, el sistema de objetos. Encima de esto se encuentra GTK+, la librería de interfaces gráfica básica de GNOME y la más importante. GDK sería una capa intermedia entre los gráficos de bajo nivel y los de alto nivel. Utilizando esta base, existen librerías auxiliares y específicas de GNOME [1]y sobre ellas, las aplicaciones finales [Jang, 2006].

[1]En la actualidad, estas librerías dejan de tener utilidad y en la versión 3 de GNOME serán obsoletas

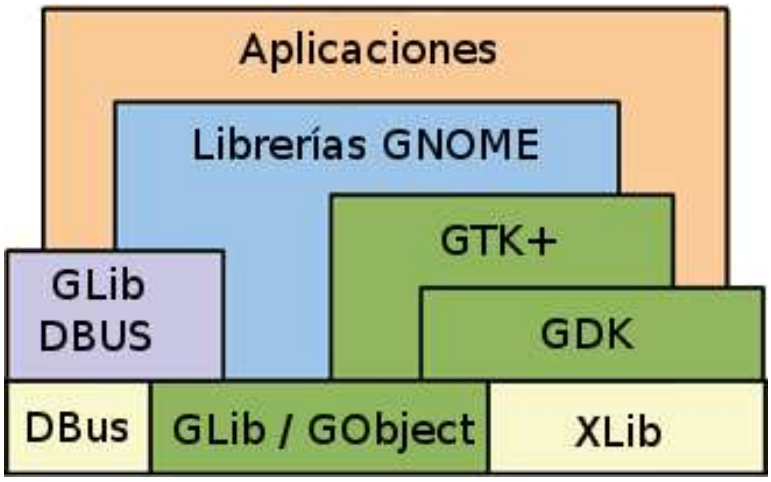

Figura 1.1: Arquitectura de GNOME.

1.3.1. GLib

GLib es una librería de utilidades de propósito general y multiplataforma. Comenzó siendo una parte de GTK+, pero para la versión 2 de la misma, se decidió separar el código específico de interfaz gráfica con el más genérico. Desde entonces GLib se puede utilizar para aplicaciones no gráficas, como librería de utilidades. Esto permite a muchas aplicaciones utilizar las variadas estructuras que implementa (colas, listas, pilas, diccionarios, etc.) como de las funciones (manejo de hilos, callbacks o temporizadores).

Esta librería proporciona una base de un sistema de tipos, genérico y dinámico. Puede manejar por defecto enteros, cadenas, números de coma flotante o referencias a objetos. Este sistema, llamado GType, permite realizar operaciones de copia, asignación o destrucción de cualquier tipo registrado.

1.3.2. GObject

Sobre la librería GLib, yace GObject el sistema de objetos, piedra angular de GTK+, y por lo tanto de GNOME. De la misma manera que GLib, este sistema estaba íntimamente ligado en un principio a GTK+, y se conocía entonces como GtkObject, la clase base de la que derivaban todos los demás objetos.

El sistema de objetos GObject permite crear un sistema de clases completo, al estilo de cualquier lenguaje de programación orientado a objetos como Java. Permite herencia simple, polimorfismo, atributos de clase o incluso privados. Sin embargo,

dado que la sintaxis es C, se requiere cierto código adicional para poder definir una clase dentro del sistema GObject y por ello la curva de aprendizaje es elevada.

En un sistema gráfico se suele usar el paradigma de programación orientada a eventos. Esta metodología permite que la programación y la ejecución estén determinados por los eventos o sucesos que ocurran en el sistema, bien por parte del usuario o autoprovocada. Mientras que en programación estructurada es el programador quien define el flujo de ejecución, en la programación orientada a eventos es el usuario quien lo determina.

La naturaleza de GObject está heredada de las necesidades de un sistema gráfico como es GTK+. Es por eso por lo que existe un mecanismo de señales fácil de implementar en una clase de GObject. Este mecanismo permite que una instancia de un objeto emita una señal cuando un evento ocurre. El resto de objetos pueden suscribirse a una señal determinada de una instancia y reaccionar a ella. De esta manera se consigue aunar la metodología orientada a objetos con la programación orientada a eventos en una misma librería.

Comparado con otros sistemas de objetos, GObject no es una ampliación del lenguaje C, sino una librería. Esto tiene ciertas ventajas, ya que no es necesario crear nuevos compiladores, y la compatibilidad binaria es fácil, debido a que C no permite sobrecarga de métodos. Por otro lado, el ya comentado sistema de señales de GObject es intrínseco en la librería, creando una tendencia en el programador a utilizarlo, consiguiendo con ello una mejor encapsulación y reutilización de los objetos.

Una gran ventaja del sistema de objetos GObject, es que dado que está realizado en C, encaja fácilmente en el sistema de objetos nativos de otros lenguajes como Python, C++, Java, Ruby o C#. Gracias a ello, es relativamente fácil realizar bindings a otros lenguajes de una librería que use GObject.

Como desventaja, la curva de aprendizaje es elevada, ya que en ocasiones es necesario conocer algunas singularidades y particularidades del sistema para poder aprovecharlo al máximo. Además, es necesario mucho código extra y repetitivo, ya que una clase está representada por unas estructuras especiales, así como el uso de unas funciones específicas.

Otra desventaja, desde el punto de vista de las métricas de software, es que dado que se trata de código C y no hay una sintaxis única, no es fácil realizar métricas orientadas a objetos simplemente analizando el código. Sin embargo, gracias a un mecanismo llamado introspección, se puede extraer metainformación sobre las clases, las

estructuras y las funciones existentes. La introspección consiste en una serie de anotaciones como comentarios en el código fuente de las aplicaciones[GNOME, 2009b]. En estos comentarios, integrados en el sistema de documentación gtk-doc, se añade metainformación a las estructuras y funciones definidas.

1.3.3. GObject introspection

GObject introspection es una serie de utilidades pensadas para centralizar toda la información proporcionada por el sistema de introspección de GObject. Con este propósito, se pueden desarrollar diferentes herramientas o utilidades con objetivos diversos. Por ejemplo, se simplifica y unifica el desarrollo de bindings a otros lenguajes para los empaquetadores y desarrolladores de GNOME. Puede servir también para que un proyecto pueda realizarse con dos lenguajes de programación, siendo uno de ellos C con GObject. Se pueden crear herramientas de documentación o de comprobación de la API de una manera automática. O se pueden realizar métricas orientadas a objetos gracias a la información disponible.

El objetivo principal de GObject introspection es la creación de bindings de una manera más sencilla y automatizada. La metainformación proporcionada permite conocer, por ejemplo, los constructores de una clase. Esto permite que las clases se implementen de manera natural en el lenguaje de destino. Hasta GObject introspection, la creación de bindings era un proceso semiautomático, y requería un gran conocimiento del sistema original, tarea que en ocasiones se veían obligados a realizar los propios desarrolladores. Gracias a este sistema, los desarrolladores únicamente tienen que añadir esta metainformación, normalizando así el código y permitiendo que la creación de bindings sea una tarea independiente y común.

El sistema GObject introspection se divide en varios elementos [GNOME, 2009a]:

- Un fichero XML llamado GIR que contiene toda la información de la introspección.
- Una aplicación hecha en Python para crear y analizar el formato GIR.
- Un escáner que analiza el código C y genera el GIR.
- Un fichero binario, llamado typelib que almacena la información en formato binario.

- Un compilador para transformar el typelib a XML y viceversa.
- Una librería en C para leer el typelib.

De todos ellos, el más interesante para la finalidad de este proyecto es el fichero GIR. En él, se almacenan todas las clases dentro un espacio de nombres. Esta información es la base para el resto de aplicaciones, pues al estar en formato XML, se pueden realizar diversas aplicaciones.

Debido a que se extrae toda la información de una librería o aplicación, un fichero GIR puede llegar a ser bastante grande en tamaño. Sin embargo, la estructura XML es muy sencilla, y vamos a mostrar algunos ejemplos de los elementos más importantes y así conocer la estructura.

- **Código XML de una función**

```
<function name="time_packet_new" c:identifier="gst_net_time_packet_new">
  <return-value transfer-ownership="full">
    <type name="TimePacket" c:type="GstNetTimePacket*"/>
  </return-value>
  <parameters>
    <parameter name="buffer" transfer-ownership="none">
      <array c:type="guint8*">
        <type name="uint8"/>
      </array>
    </parameter>
  </parameters>
</function>
```

Se puede observar que una función se compone de una serie de parámetros y devuelve un valor, especificando el tipo del mismo. Es interesante observar los atributos del elemento *function*. Se puede observar que tiene un nombre, y un identificador en C. El primero es el nombre con el que se quiere conocer al método en los bindings, mientras que el último es la función que se ejecutará realmente.

Con respecto a los parámetros se indican características como si la función se encarga de liberar el espacio asignado o el tipo real en C.

- **Código XML de una clase**

```
<class name="TimeProvider"
       c:type="GstNetTimeProvider"
```

```
     parent="Gst.Object"
     glib:type-name="GstNetTimeProvider"
     glib:get-type="gst_net_time_provider_get_type">
<constructor name="new" c:identifier="gst_net_time_provider_new">
  <return-value transfer-ownership="full">
    <type name="TimeProvider" c:type="GstNetTimeProvider*"/>
  </return-value>
  <parameters>
    <parameter name="clock" transfer-ownership="none">
      <type name="Gst.Clock" c:type="GstClock*"/>
    </parameter>
    <parameter name="address" transfer-ownership="none">
      <type name="utf8" c:type="gchar*"/>
    </parameter>
    <parameter name="port" transfer-ownership="none">
      <type name="int" c:type="gint"/>
    </parameter>
  </parameters>
</constructor>
<property name="port" writable="1">
  <type name="int" c:type="gint"/>
</property>
<property name="address" writable="1">
  <type name="utf8" c:type="gchararray"/>
</property>
<property name="clock" writable="1">
  <type name="Gst.Clock" c:type="GstClock"/>
</property>
<property name="active" writable="1">
  <type name="boolean" c:type="gboolean"/>
</property>
<field name="parent">
  <type name="Gst.Object" c:type="GstObject"/>
</field>
<field name="address">
  <type name="utf8" c:type="gchar*"/>
</field>
<field name="port">
  <type name="int" c:type="int"/>
</field>
<field name="sock">
  <type name="int" c:type="int"/>
</field>
<field name="control_sock">
  <array zero-terminated="0" c:type="int" fixed-size="2">
    <type name="int"/>
  </array>
</field>
<field name="thread">
  <type name="GLib.Thread" c:type="GThread*"/>
</field>
<field name="clock">
```

```
    <type name="Gst.Clock" c:type="GstClock*"/>
  </field>
  <field name="active">
    <type name="any" c:type="any"/>
  </field>
  <field name="priv">
    <type name="TimeProviderPrivate" c:type="GstNetTimeProviderPrivate*"/>
  </field>
  <field name="_gst_reserved">
    <array zero-terminated="0" c:type="gpointer" fixed-size="2">
      <type name="any"/>
    </array>
  </field>
</class>
```

Las clase son los elementos más completos que se pueden ver en un fichero GIR. Aquí podemos observar que la clase *TimeProvider* en realidad es un registro en C llamado *GstNetTimeProvider*. También nos indica que su clase padre es *Gst.Object*, otra clase existente en el espacio de nombres. En este caso, al hacer un binding, en el lenguaje destino se usará la sintaxis correcta para crear una clase *TimeProvider* que herede de *Gst.Object*.

Se observa también que tiene un constructor, con estructura similar a las funciones presentadas antes. Además, esta clase tiene una gran lista de campos y propiedades, indicando su tipo y visibilidad entre otra información.

- **Código XML de un registro**

```
<record name="TimeProviderClass" c:type="GstNetTimeProviderClass">
  <field name="parent_class">
    <type name="Gst.ObjectClass" c:type="GstObjectClass"/>
  </field>
</record>
```

Los registros en GObject tienen la misma finalidad que en C, que es una agrupación lógica de elementos en una misma estructura. Aquí se observa también que se proporciona un nombre para usar en el lenguaje destino y el nombre real en C.

- **Código XML de un espacio de nombres**

```
<namespace name="GstNet" version="0.10" shared-library="libgstnet-0.10.so.0">
  <alias name="ClockTimeDiff" target="int64" c:type="GstClockTimeDiff"/>
  <record>
```

```
    .....
  </record>
  <class>
    .....
  </class>
  <enumeration>
    .....
  </enumeration>
</namespace>
```

Un espacio de nombres es una agrupación lógica de elementos. Se observa que puede contener varios elementos como registros, clases, enumerados, además de callbacks, funciones o métodos. Sobre cada espacio de nombre se indica la librería en la que se encuentran los símbolos definidos.

- **Código XML de un repositorio**

```
<repository version="1.0"
          xmlns="http://www.gtk.org/introspection/core/1.0"
          xmlns:c="http://www.gtk.org/introspection/c/1.0"
          xmlns:glib="http://www.gtk.org/introspection/glib/1.0">

  <include name="GLib" version="2.0"/>
  <include name="GModule" version="2.0"/>
  <include name="GObject" version="2.0"/>
  <include name="libxml2" version="2.0"/>
  <namespace>
      .....
  </namespace>
</repository>
```

Un repositorio es a su vez un conglomerado de espacios de nombres. En él, además, se definen las librerías de las que depende.

Como se puede ver, el fichero GIR proporciona toda la información sobre las clases, los registros o incluso las constantes que se definen en el espacio de nombres de la aplicación. Gracias a ellos, se pueden realizar fácilmente bindings a otros lenguajes, al conocer de forma normalizada todo lo que compone dicha aplicación. También incluye información sobre los símbolos en C, las librerías con las que enlazan e infinidad de datos objeto de estudio.

Es fácil pensar que analizando este fichero, se pueden extraer diversas métricas orientadas a objetos. De todo ello se hablará en el Capítulo 3, al ser el objetivo de este proyecto.

1.4. CVSAnalY

CVSAnalY[2]es la herramienta más importante de Libresoft[3], que extrae información de los repositorios de código fuente para su posterior análisis. Su funcionamiento se basa en el análisis de los datos que se pueden extraer de los repositorios CVS, SVN y Git, usados comunmente por los proyectos de software libre para almacenar y llevar un control sobre el código desarrollado.

A través del análisis de los logs, o registros, de los repositorios, CVSAnalY normaliza esa información en una base de datos. A partir de esos datos, se pueden realizar estadísticas y métricas de software gracias a las extensiones disponibles.

Para el propósito de este proyecto, destacaremos que una de esas extensiones, realiza métricas dependiendo del lenguaje. Para ello, se sirve de Sloccount, un software de métricas de líneas de código, para saber en qué lenguaje se ha programado. Sloccount tiene unas heurísticas bastante fiables para conocer el lenguaje del código fuente. Otro de los objetivos de este trabajo es que Sloccount sea capaz de identificar ficheros con código GObject, con la finalidad de que se realicen las métricas desarrolladas en este proyecto. Todo esto se ve reflejado en la Sección 3.1

Con esta herramienta se han llevado a cabo diversos estudios [Robles et al., 2004] [Navarro et al., 2005], por lo que es una herramienta fiable, firme y ampliamente utilizada. Este proyecto no toca directamente CVSAnalY, pero pretende proporcionar y proponer ampliaciones de futuro.

1.5. Métricas orientadas a objetos

La metodología de desarrollo orientado a objetos proporciona muchas ventajas, como la reutilización, la descomposición del problema en partes fácilmente comprendibles y la facildad para modificaciones futuras. Sin embargo, el ciclo de vida de desarrollo es algo más complejo.

Es por ello que se necesitan unas buenas guías de desarrollo que permitan desarrollar una buena programación orientada a objetos con un código fiable y buenas prácticas. En este punto es donde las métricas tienen un fuerte valor, pues permiten medir la efectividad y fiabilidad de un desarrollo, midiendo algunos parámetros de forma

[3]http://tools.libresoft.es/cvsanaly

[3]GSyC/LibreSoft es un grupo de investigación de software libre vinculado a la Universidad Rey Juan Carlos.

estándar para poder comparar la efectividad del desarrollo entre diversos sistemas.

Cabe diferenciar dos tipos de métricas orientadas a objetos. En un primer lugar están aquellas que miden todo el sistema en su conjunto, teniendo en cuenta relaciones entre clases. Por otra parte, están aquellas que analizan una clase y dan medidas sobre la misma.

El estudio realizado por Morris [Morris, 1989] recalcaba algunas de las métricas sobre una única clase más básicas e importantes. Se pueden resumir en las siguientes:

- **Métodos ponderados**

  Se define como la suma de las complejidades ciclomáticas de todos los métodos de una clase. Se utiliza como una medida del tiempo y esfuerzo necesario para desarrollar o mantener una clase. Cuantos más métodos tenga una clase, más impacto tendrá en una clase hija, puesto que heredará todos sus métodos. Aunque aquellas clases con muchos métodos serán seguramente más específicas de una aplicación, ya que será difícil reutilizarlos.

- **Longitud de dependencias**

  Esta métrica es la distancia desde la clase a medir y su mayor ancestro en el sistema. Cuanto mayor sea esta longitud, más métodos heredará, y por tanto es posible que sea más difícil de predecir su comportamiento. Aunque también es un síntoma de que la clase tiene más potencial, al tener disponibles más métodos. En general, una mayor longitud implica más complejidad.

- **Número de clases hijas**

  Es simplemente el número de clases hijas que heredan directamente de la clase a medir. Se mira únicamente las hijas de primer nivel, sin tener en cuenta que de ellas pueden heredar a su vez a otras clases. Cuantas más subclases tenga, mayor será la reutilización del código, ya que la herencia es una forma de reutilización. Si una clase tiene excesivas clases hijas, es posible que la abstracción de dicha clase no sea adecuada. Por lo general, este valor da una indicación clara del impacto y la influencia de esa clase en todo el sistema.

Chidamber y Kemerer [Chidamber and Kemerer, 1994], por su parte, idearon algunas métricas que miden el sistema de objetos en su conjunto. Entre las más importantes se encuentran las siguientes:

- **Métodos por clase**

  Es la métrica más básica, al ser una media aritmética del número de métodos por clase. Un gran número de métodos por clase significa más dificultad en las pruebas, aunque por otro lado, significa que las subclases heredarán un gran número de métodos, aumentando la reutilización.

  Métodos por clase = Némero de métodos / Némero de clases

- **Máxima longitud de dependencias**

  La longitud del árbol de dependencias se mide como la máxima distancia desde una clase padre a una clase hija a través de la relación de herencia. Las pruebas en los árboles de dependencias más profundos son más difíciles de realizar, además de tener una mayor dificultad de comprensión del sistema. Sin embargo, es mejor que el árbol crezca a lo largo que a lo ancho, pues así hay más grado de reutilización.

- **Grado de acoplamiento de objetos**

  Esta métrica sirve para conocer cómo de cohesionado es el sistema de objetos implementado. Un alto grado de acoplamiento conlleva una dificultad de mantenimiento, dado que las relaciones e interacciones son más complejas. Por el contrario, si es un nivel bajo, los objetos pueden reutilizarse fácilmente incluso en otras aplicaciones.

  El grado de acoplamiento se calcula como:

  Grado de acoplamiento = Número de arcos / Némero de clases

  Siendo el número de arcos, el total de arcos en un grafo de uso de objetos. En dicho grafo, los nodos son las clases y existe un arco entre dos nodos cuando un objeto de una clase utiliza un objeto de otra clase.

- **Complejidad ciclomática media**

  La complejidad ciclomática media es una medida para conocer cómo de complejo es el sistema de objetos implementado. Una gran complejidad media significa más dificultad de mantenimiento, una grado menor de facilidad de aprendizaje del sistema, menor fiabilidad y dificultad en la etapa de pruebas.

  La fórmula es la siguiente:

Complejidad ciclomática media = Suma complejidad ciclomática de todos lo métodos / Número de métodos

Existe infinidad de métricas orientadas a objetos, además de las explicadas aquí. Sin embargo, algunas de ellas no pueden calcularse únicamente con los datos del código fuente y de los ficheros GIR. Por ejemplo el grado de cohesión de objetos, la efectividad de la librería en cuanto a reusabilidad o la respuesta de una clase [Ümit Karakaþ and Sultanoðlu, 199

Capítulo 2

Objetivos

2.1. Descripción del problema

Este proyecto tiene como objetivo la obtención de métricas orientadas a objetos en GNOME. Como se ha explicado anteriormente, el núcleo está desarrollado en C, pero gracias a GObject se ha conseguido realizar programación orientada a objetos en un lenguaje que no proporciona sintáxis para ello.

La idea es utilizar los ficheros GIR, generados por GObject Introspection, analizarlos y generar algunas métricas que puedan servir para estudiar desde diversos puntos de vista la implementación del sistema orientado a objetos en algunas aplicaciones que utilicen el sistema de objetos de GNOME, GObject. Sin embargo, en muchos casos, basándose únicamente en el fichero GIR, no es posible conseguir todas las métricas.

Con la información de este fichero sólo conocemos qué clases existen y qué funciones o métodos implementan. No conocemos el código, y por tanto no podemos calcular, por ejemplo, la complejidad ciclomática de un método. En este proyecto se ha desarrollado una solución para aliviantar este problema, que se explicará en el Capítulo 3.

Ofrecer métricas para paradigmas orientados a objetos abre un nuevo horizonte de investigación y desarrollo. Herramientas dedicadas al estudio de la ingeniería del software como CVSAnalY, carecen de un estudios similares, y por lo tanto este proyecto subsana ese problema. Es importante reseñar también que realiza métricas orientadas a objetos en un lenguaje que no ha sido diseñado para ello, por lo que tiene una peculiaridad única.

Estas métricas y el estudio de la ingeniería del software libre se enmarcan dentro

del trabajo llevado a cabo por el grupo GSyC/LibreSoft.

2.2. GirMetrics

El sistema desarrollado que trata de resolver estos problemas, recibe el nombre de *GirMetrics*. La idea, tal y como se ha explicado, consiste en analizar el XML correspondiente al fichero GIR y a partir de la información obtenida, poder ofrecer varias métricas orientadas a objetos. Esta idea resuelve todos los problema descritos en la Sección 2.1.

El sistema analiza sistemas de objetos completos, entendiendo que todas aquellas referencias a tipos o clases externas no pertenecen al sistema de objetos y por lo tanto no forman parte de las métricas. Es importante aclarar este punto ya que las clases base de algunos sistemas pueden ser a su vez hijas de clases de otros sistemas de objetos. Se entiende que la información ofrecida por los ficheros GIR es completa y cerrada, aunque en ocasiones no sea así.

La funcionalidad de GirMetrics es sencilla, así como su uso. Basta con disponer de ficheros GIR, y opcionalmente del código fuente del sistema a analizar. Se trata de una aplicación de línea de comandos, que muestra de una forma estructurada, para que puedan ser examinados posteriormente por otras herramientas automáticas, los resultados de las métricas realizadas.

Capítulo 3

Diseño e implementación

3.1. Requisitos

Partiendo de la descripción del problema visto en la Sección 2.1, el sistema debe de cumplir los siguientes requisitos.

- Una parte del programa debe de ser capaz de analizar la estructura GIR y representar todos sus elementos en memoria.
- Una aplicación usará los elementos analizados para poder deducir o calcular métricas orientadas a objetos.
- Dado que a partir del fichero GIR no es posible acceder al código, es el usuario quien proporcionará a dicha aplicación la ruta al código fuente. La aplicación debe de poder llamar a algún analizador de complejidad ciclomática.[1]
- La aplicación debe ser sencilla y con un formato de salida sencillo, que permita un uso posterior de los datos por herramientas externas. Esta es la filosofía de trabajo de los sistemas UNIX, donde existen herramientas que hacen trabajos muy concretos y fáciles de combinar mediante tuberías.
- Por otra parte, es necesario modificar la herramienta Sloccount para poder determinar cuándo un fichero de código fuente es la implementación de un objeto con GObject, y así poder lanzar la aplicación de métricas orientadas de objetos.

[1] El grupo de investigación LibreSoft tiene diversas herramientas para ello.

3.2. Elección de la tecnología

Este proyecto no tiene grandes requisitos tecnológicos, ya que los objetivos son muy sencillos. Sin embargo, es importante elegir un lenguaje de programación que sea sencillo y tenga las suficientes herramientas para llevarlo a cabo.

Toda la tecnología empleada en el desarrollo es software libre, así como el propio entorno de desarrollo.

- Python

  Python es un lenguaje de programación interpretado, que fue desarrollado en los años 90 por Guido van Rossum. Tiene características que lo hacen especialmente interesante como la orientación a objetos, los tipos de datos que soporta de forma nativa, etc. Sin embargo al tratarse de un lenguaje interpretado no es adecuado para aquellas situaciones que requieren un alto nivel de rendimiento, afortunadamente, no es el caso.

  Toda la aplicación ha sido desarrollada íntegramente en Python, utilizando una metodología orientada a objetos.

- LibXML

  XML viene de eXtensible Markup Language o lenguaje de marcado extensible. Es un lenguaje de marcado de propósito general recomendado por la W3C[2]. Se utiliza para crear lenguajes de marcas de propósito específico, capaz de describir muchos tipos de datos. En el proyecto se utiliza para almacenar en el ordenador del desarrollador la información correspondiente a su actividad que posteriormente será enviada al grupo de investigación para su estudio. Una vez obtenidos los datos en el servidor, el código XML recibido es interpretado para extraer los datos que contiene y almacenarlos finalmente en la base de datos.

  Dado que los ficheros GIR son esencialmente ficheros XML, se necesita una librería en Python para poder analizarlo fácilmente. LibXML es una librería de manejo de XML robusta y fácil de usar. Aunque fue desarrollada inicialmente para el proyecto GNOME, se trata de una librería de propósito general, convirtiéndose casi en la librería XML por excelencia. Existen bindings para muchos lenguajes, entre ellos Python.

[2] W3C (World Wide Web Consortium) es un consorcio internacional para la creación de estándares para la web.

3.3. Herramientas externas utilizadas

Para la consecución de los objetivos del proyecto se han utilizado, además, una serie de herramientas externas relacionadas con estudios de ingeniería del software. Estas herramientas son las siguientes:

- **Cmetrics**

  Cmetrics[3] es un conjunto de herramientas, que forman parte de las utilidades de LibreSoft. De todas ellas se necesita la herramienta *mccabe*, una utilidad para calcular la complejidad ciclomática de las funciones de un fichero de código fuente C.

  Se trata de una herramienta de línea de comandos que recibe el nombre de fichero y, analizando el código, muestra diversas métricas de cada función. Entre ellas están la complejidad ciclomática y el número de sentencias return que tiene. Estos valores, principalmente la complejidad ciclomática son utilizados para realizar a su vez algunas de las métricas orientadas a objetos planteadas en la Sección 1.5.

  La salida que proporciona esta herramienta es sencilla de analizar sintácticamente para extraer los valores. A continuación se muestra un ejemplo de mccabe ejecutado sobre un fichero de código de GTK+.

```
$ mccabe gtklist.c
File           Name            Complexity No. of returns
-----------------------------------------------------
gtklist.c      gtk_list_toggle_add_mode           5          0
gtklist.c      gtk_list_unselect_all           9          0
gtklist.c      gtk_list_new             1          1
gtklist.c      gtk_list_select_all           9          0
gtklist.c      gtk_list_child_type           1          1
gtklist.c      gtk_list_class_init           1          0
gtklist.c      gtk_list_realize           1          0
gtklist.c      gtk_list_get_type           2          1
gtklist.c      gtk_list_toggle_focus_row           8          0
gtklist.c      gtk_list_init            1          0
gtklist.c      do_fake_motion           1          0
gtklist.c      list_has_grab            1          1
gtklist.c      gtk_list_unmap           4          0
gtklist.c      gtk_list_dispose           1          0
gtklist.c      gtk_list_reset_extended_selection           1          0
gtklist.c      gtk_list_end_drag_selection           4          0
```

[3] http://tools.LibreSoft.es/cmetrics

```
gtklist.c      gtk_list_start_selection           3          0
gtklist.c      gtk_list_undo_selection           9          0
gtklist.c      gtk_list_end_selection          16          0
gtklist.c      gtk_list_horizontal_timeout           1          1
gtklist.c      gtk_list_vertical_timeout           1          1
gtklist.c      ***                    257         13
```

- **Sloccount**

  Sloccount es la herramienta por excelencia para el análisis del software. Proporciona estimaciones de coste, de duración o número de desarrolladores a partir del modelo COCOMO[4] .

  A continuación mostramos un ejemplo de los resultados ofrecidos por Sloccount con el código fuente de GTK+.

```
SLOC Directory SLOC-by-Language (Sorted)
309092  top_dir         ansic=307601,python=1299,perl=171,sh=21
7516    tests           ansic=7516
131     theme-bits      ansic=131
0       stock-icons     (none)

Totals grouped by language (dominant language first):
ansic:       315248 (99.53%)
python:        1299 (0.41%)
perl:           171 (0.05%)
sh:              21 (0.01%)

Total Physical Source Lines of Code (SLOC)                = 316,739
Development Effort Estimate, Person-Years (Person-Months) = 84.48 (1,013.79)
 (Basic COCOMO model, Person-Months = 2.4 * (KSLOC**1.05))
Schedule Estimate, Years (Months)                         = 2.89 (34.69)
 (Basic COCOMO model, Months = 2.5 * (person-months**0.38))
Estimated Average Number of Developers (Effort/Schedule)  = 29.22
Total Estimated Cost to Develop                           = $ 11,412,433
```

  En este proyecto proponemos y desarrollamos una modificación para permitir identificar si un fichero es una implementación de una clase en GObject, con el objetivo de realizar las métricas orientadas a objetos aquí desarrolladas. Se trata sin duda de una gran aportación, ya que en la actualidad es una herramienta cuyo desarrollo se encuentra parado, en gran parte debido a que no existen

[4]El Modelo Constructivo de Costes (o COCOMO, por su acrónimo del inglés COnstructive COst MOdel) es un modelo matemático de base empírica utilizado para estimación de costes de software. Incluye tres submodelos, ofreciendo un nivel de detalle y aproximación cada vez mayor, a medida que avanza el proceso de desarrollo del software: básico, intermedio y detallado.

innovaciones. Al ser una herramienta con una gran utilización en los medios científicos y académicos, cuyas exigencias son altas.

3.4. Arquitectura

En vistas de una futura ampliación o reutilización de este proyecto, se ha desarrollado una librería especializada cuyo cometido es el análisis de los ficheros GIR. Dicha librería, llamada *LibGirParser*, ofrece clases que representan los elementos que se describen en un GIR y sus dependencias. Así, por ejemplo, existen clases que representan funciones, callbacks, propiedades o clases de GObject.

Por otro lado, se ha desarrollado una aplicación que, utilizando *LibGirParser*, genera las métricas orientadas a objetos que se buscan. Dicha aplicación se llama por línea de comandos recibiendo como parámetro el fichero GIR. Opcionalmente, se le pueden indicar los ficheros de código fuente, que son utilizados para realizar las métricas relativas a complejidad ciclomática. Estas métricas de complejidad ciclomática se calculan gracias a la utilidad *mccabe*.

LibGirParser tiene una orientación a objetos, gracias a que en Python el soporte de objectos es bastante sencillo y completo. El objeto principal es GirParser, que realiza las operaciones sobre el fichero GIR y el árbol XML. En un fichero GIR lo más básico que existen son los espacios de nombres, o namespaces. En orientación a objetos, un espacio de nombres se utiliza para agrupar clases con naturaleza o finalidad similares. En GObject no existen mecanismos para definir espacio de nombres, sin embargo, cada proyecto o librería se entiende como un espacio de nombre.

En el diagrama UML se puede ver como un espacio de nombres contiene clases, registros, funciones, métodos, enumerados y en general cualquier estructura representada en GObject. Ofrece dos métodos interesantes: `getInheritanceTreeMaxPath()` y `getInheritanceTreeClassMaxLength()`, utilizadas para saber el máximo nivel de profundidad del árbol de herencias de clases que ofrece el sistema analizado.

3.5. Modificaciones en Sloccount

Sloccount está desarrollado mayoritariamente en Perl y shell script, ya que funciona únicamente en entornos UNIX. Los cambios que se necesitan para cumplir el último punto de la Sección 3.1 afectan en concreto al fichero `break_filelist`. Este

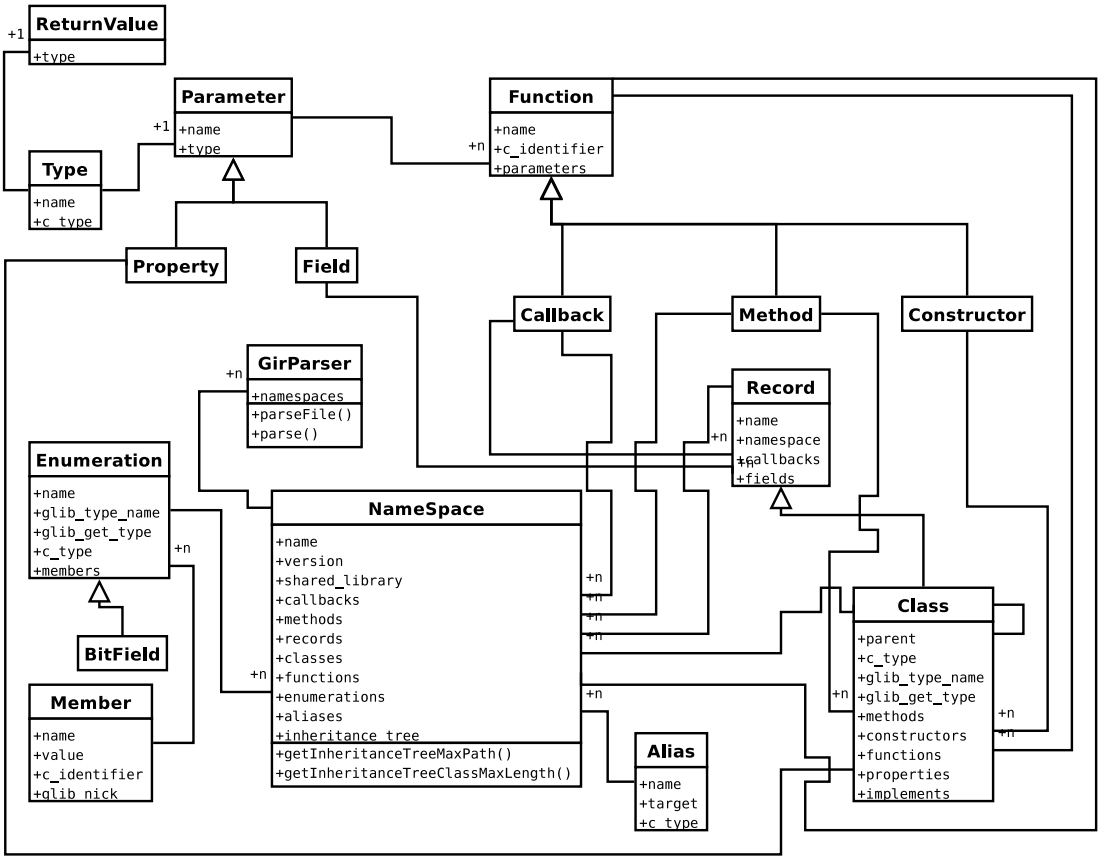

Figura 3.1: Arquitectura de *LibGirParser*.

fichero contiene las heruísticas para determinar el lenguaje que se usa en un fichero de código. Se podría pensar en un primer momento que el lenguaje viene definido por la extensión del fichero a analizar, pero esto no es así en todos los casos. Las heurísticas, por lo tanto, analizan el código en busca de patrones que puedan determinar de una manera más o menos precisa el lenguaje utilizado.

De esta manera, para determinar si un fichero utiliza GObject, hay que buscar el uso de algunas funciones específicas de este sistema de objetos. Las más importantes son las funciones que se utilizan para registrar un tipo en el sistema de tipos de GObject, en concreto el uso de las macros `G_DEFINE_TYPE`, `G_DEFINE_ABSTRACT_TYPE`, `G_DEFINE_DYNAMIC_TYPE` y derivadas o las funciones del estilo `g_type_register_`. Este sistema no detecta los ficheros que utilizan la librería GLib o objetos definidos por GObject [5], pero sí que detecta aquellos ficheros donde se declaran o implementan clases dentro de este sistema de objetos.

[5]Los falsos positivos serían excesivamente elevados puesto que GLib es una librería de propósito

Las modificaciones incluyen la creación de una función nueva, al estilo de las existentes en Sloccount llamada `is_real_gobject()`, que recibe un nombre de fichero y devuelve verdadero o falso en función de si se han encontrado las funciones de GObject. Esta función es usada dentro de la función `get_file_type()`, que es quien ejecuta las heurísticas necesarias para intentar determinar correctamente el tipo de fichero.

Por último, también se necesita modificar el fichero `compute_sloc_lang`, el fichero que realiza las métricas de líneas de código dependiendo del lenguaje a examinar. En este caso, los cambios son mínimos, puesto que únicamente hay que indicarle que el código GObject se mide igual que C, al igual que ocurre con muchos otros lenguajes como C++, PHP o Java.

Pueden existir algunas debilidades en forma de falsos negativos. Un ejemplo son los ficheros de código que implementan funciones de utilidades genéricas, que utilizan los objetos existentes, pero no implementan ninguno nuevo. Si se detectasen como ficheros GObject quizás alguna métrica sería más correcta, pero los esfuerzos para implementar mecanismos de detección de estos casos no compensarían la existencia de errores como falsos positivos. Esta característica para evitar los falsos negativos se puede observar en los resultados de las modificaciones, en la Sección 4.4.

general y su uso no implica que se esté utilizando orientación a objetos.

Capítulo 4

Resultados

Los datos proporcionados por la aplicación son diferentes métricas orientadas a objetos sobre los subsistemas de GNOME o aplicaciones que utilicen GObject y soporten la introspección mediante GObject Introspection. Estas métricas son las mismas que se presentaron en el Sección 1.5.

El estudio ha sido realizado en Agosto de 2009, siendo la última versión estable disponible de GNOME la 2.26. La versión estable de GTK+ es la 2.16, y ha sido sobre la que se han realizado las métricas. A continuación se muestra el resultado de la ejecución de la aplicación sobre los distintos ficheros GIR disponibles.

4.1. GTK+

GTK+ es el sistema de clases basado en GObject por excelencia, y por lo tanto, de donde más métricas podremos sacar. De este toolkit existen bindings a una gran cantidad de lenguajes, y por eso el fichero GIR es completo y funcional en su totalidad, siendo por lo tanto una excepcional fuente de estudio.

La ejecución de GirMetrics usando el fichero GIR de GTK+, arroja el siguiente resultado.

Se puede observar de un simple vistazo que se trata de un sistema grande, con casi 170 clases diferentes y algo más de 3000 funciones. La complejidad ciclomática media no es muy elevada, lo que parece indicar una buena implementación del código.

En cuanto a las métricas correspondientes a cada clase, sólo vamos a subrayar las más relevantes aquí, debido a la gran cantidad existente. La más importante es sin duda *GtkWidget*, una clase básica que representa un objeto genérico en una interfaz gráfica.

GTK+	Métrica
Clases	169
Métodos	3275
Métodos por clase	19,38
Máxima longitud de dependencias	8
Complejidad ciclomática media	1,59
Grado de acoplamiento	0,24

Cuadro 4.1: Métricas orientadas a objetos en GTK+.

De ella derivan, directa o indirectamente, la práctica totalidad de clases de GTK+. Se puede ver además que tiene una gran complejidad ciclomática ponderada, debido más al gran número de métodos que ofrece que a la complejidad de los mismos. Se observa también que tiene sólo 2 clases por encima en el nivel de jerarquía, mientras que de ella heredan directamente 14 clases distintas.

GtkWidget	Métrica
Métodos ponderados	150
Longitud de dependencias	2
Número de clases hijas	14

Cuadro 4.2: Métricas sobre la calse GtkWidget.

Otra clase interesante es *GtkContainer*. Esta clase es una generalización de un objeto contenedor, es decir, aquel que puede contener otros elementos en su interior. Se puede observar que es una clase relativamente básica, con sólo 3 ancestros. Teniendo 15 clases hijas, más incluso que *GtkWidget*, demuestra ser una clase con gran reutilización.

Sin embargo, es interesante destacar que se trata de una abstracción más lógica que práctica, ya que los métodos ponderados no son elevados y por lo tanto no es una clase con muchos métodos que puedan ser reutilizados. Su utilización se dirige más hacia el polimorfismo de las clases derivadas, permitiendo a éstas clases actuar como un *GtkContainer*.

GtkContainer	Métrica
Métodos ponderados	14
Longitud de dependencias	3
Número de clases hijas	15

Cuadro 4.3: Métricas sobre la calse GtkContainer.

GtkRadioToolButton es una clase también interesante. Es la que mayor longitud de dependencias muestra, con 8 clases por encima de ella. Como es lógico, no tiene ninguna clase hija, y por lo que se intuye a partir de los métodos ponderados, no define muchos métodos propios. Sin embargo, sabemos que gracias a su árbol de clases de las que hereda, dispone de una gran cantidad de métodos.

GtkRadioToolButton	Métrica
Métodos ponderados	4
Longitud de dependencias	8
Número de clases hijas	0

Cuadro 4.4: Métricas sobre la calse GtkRadioToolButton.

4.2. GDK

GDK son las siglas de **GIMP D**rawing **K**it o herramientas de dibujado de GIMP. Técnicamente es la librería encargada de abstraer a GTK+ del sistema gráfico que se encuentra por debajo. Inicialmente, GDK soportaba únicamente el servidor gráfico X, usado en los sistemas UNIX, pero actualmente funciona también tanto en Windows como en Quartz[1] .

Internamente, utiliza GObject para su desarrollo, y dado que ofrece una abstracción del sistema gráfico, también tiene soporte para bindings a través de GObject Introspection. Es por ello que existen también ficheros GIR y podemos realizar métricas de esta librería. Sigue el mismo número de versión que GTK+, y la que se corresponde con este estudio es la 2.16, última versión estable publicada hasta el momento.

Los resultados arrojados por GirMetrics son los siguiente.

GDK	Métrica
Clases	15
Métodos	371
Métodos por clase	24,73
Máxima longitud de dependencias	2
Complejidad ciclomática media	1,77
Grado de acoplamiento	0,13

Cuadro 4.5: Métricas orientadas a objetos en GDK.

[1]Quartz es el sistema gráfico presente en Mac OS X, compuesto por Quartz 2D y Quartz Compositor.

De las métricas generales, se desprende que es un sistema bastante más sencillo que GTK+, al descomponerse únicamente en 15 clases. Ofrece un mayor número de métodos por clase, seguramente debido a una mayor cantidad de operaciones disponibles en el nivel gráfico. Hay también una mayor complejidad ciclomática media que, aunque sin llegar a ser excesiva, muestra que es una librería más compleja. Por otro lado, el bajo grado de acoplamiento, da lugar a pensar que el mantenimiento del diseño de clases es algo más sencillo.

De esta librería es interesante estudiar varias clases, pero sin duda *GdkWindow* es la más importante. En los sistemas gráficos, tal y como los concebimos en la actualidad, la abstracción de ventanas es algo básico y primordial. Por ello es lógico pensar que la clase que más esfuerzo necesita es aquella que representa este concepto.

GdkWindow	Métrica
Métodos ponderados	81
Longitud de dependencias	2
Número de clases hijas	0

Cuadro 4.6: Métricas sobre la calse GdkWindow.

Con los valores, se observa un gran valor en métodos ponderados, aunque se aprecia más comparando con la siguiente clase en GDK en ese aspecto, y es que *GdkScreen* tiene un valor de 15. Es interesante saber también que no hay clases hijas en *GdkWindow*, y es que aunque existan diferencias entre diálogos, ventanas o alertas, a nivel del sistema gráfico son todo ventanas.

Con respecto al resto de clases de GDK, no hay ninguna especialmente interesante, salvo quizás *GdkDrawable*, la clase base de la que heredan algunos objetos gráficos.

GdkDrawable	Métrica
Métodos ponderados	9
Longitud de dependencias	1
Número de clases hijas	2

Cuadro 4.7: Métricas sobre la calse GdkDrawable.

4.3. Clutter

Clutter es una librería, desarrollada inicialmente por OpenedHand[2] , para crear interfaces de usuario aceleradas por hardware, permitiendo así utilizar al máximo las

posibilidades gráficas de un ordenador. Esta posibilidad proporciona una gran capacidad a los elementos gráficos proporcionando efectos visuales de gran rendimiento.

Está programado utilizando GObject, y funciona tanto en servidores X como en Windows. Proporciona bindings a varios lenguajes como Perl, Python, C#, Ruby o Vala[3] . Dispone de mecanismos de introspección y por lo tanto podemos realizar algunas métricas.

Con un sistema de 29 clases, Clutter es un sistema de clases relativamente pequeño en cuanto extensión, algo que también se desprende de la máxima longitud de dependencias, de sólo 3 clases. Aunque con algo más de 21 métodos por clase se intuye que es un sistema más complejo y práctico. Tiene una complejidad ciclomática con un valor relativamente normal, aunque es un sistema en crecimiento.

Clutter	Métrica
Clases	29
Métodos	612
Métodos por clase	21,10
Máxima longitud de dependencias	3
Complejidad ciclomática media	1,68
Grado de acoplamiento	0,24

Cuadro 4.8: Métricas orientadas a objetos en Clutter.

Antes de continuar con análisis de algunas clases de Clutter, conviene definir algunas peculiaridades que tiene este sistema.

- **Actor**: Un actor es un elemento gráfico visual en Clutter. Cualquier objeto
- **Stage**: Una stage (o escena), es un elemento visual donde se colocan los actores.
- **Behaviour**: Clutter está pensado para conseguir interfaces de usuario atractivas aprovechando la aceleración por hardware. Esto permite utilizar efectos gráficos, que en el ámbito de Clutter se llaman behaviours o comportamientos.

Se puede observar que Clutter abstrae las interfaces de usuario como si fuesen escenas de teatro: “Los actores realizan comportamientos en escenas”.

[2]OpenedHand Ltd. forma parte actualmente de Intel, tras ser adquirida a finales de 2008.

[3]Vala es un lenguaje similar a C# diseñado para facilitar la utilización de GObject. No es interpretado, si no que es traducido a C para ser posteriormente compilado.

La clase *ClutterActor* es la clase base de Clutter, una clase abstracta que representa cualquier objeto gráfico en el sistema, es decir, un actor. Este hecho se ve reflejado en el elevado número de clases hijas y sobre todo en el valor de los métodos ponderados.

ClutterActor	Métrica
Métodos ponderados	91
Longitud de dependencias	1
Número de clases hijas	6

Cuadro 4.9: Métricas sobre la calse ClutterActor.

Por otra parte está la clase *ClutterStage*, es una de las clases que más abajo se encuentra en la jerarquía, teniendo 3 ancestros. Se trata de un contenedor gráfico en donde se colocan el resto de los elementos gráficos o actores.

ClutterStage	Métrica
Métodos ponderados	29
Longitud de dependencias	3
Número de clases hijas	0

Cuadro 4.10: Métricas sobre la calse ClutterStage.

Es también interesante la clase *ClutterBehaviour*, que modela los comportamientos que puede tener un actor en el sistema Clutter. Un comportamiento es un efecto gráfico que se puede aplicar sobre un objeto en el sistema. Clutter proporciona 7 por defecto, que son exactamente los que heredan de esta superclase. Se puede observar que son las clases hijas las que cargan con el peso de las operaciones gráficas, pues los métodos ponderados en la clase *ClutterBehaviour* no es una cifra elevada.

ClutterBehaviour	Métrica
Métodos ponderados	6
Longitud de dependencias	1
Número de clases hijas	7

Cuadro 4.11: Métricas sobre la calse ClutterBehaviour.

4.4. Detección de GObject en Sloccount

Las pruebas de las modificaciones de Sloccount para detectar el código de GObject arrojan un resultado satisfactorio. Al analizar el código fuente de GTK+ con Sloccount,

éste informa que el 76,52 % de los ficheros son GObject. El resto lo completa un 23 % de ANSI C y una mínima parte de Python, Perl y Shell script.

Como se puede ver, existe un número de ficheros que no son detectados como GObject. Serían casos de falsos negativos tal y como se explicó en la Sección 3.5, aunque por regla general no supondrían ningún problema, porque suelen ser utilidades generales sin orientación a objetos y por lo tanto exentas de esta clase de métricas..

Para GDK se detecta un 50,82 % de código GObject, un 48,77 % de ANSI C y el resto Perl. Los resultados en el código de Clutter indican un 56,65 % de ficheros GObject, y el resto en ANSI C. Ambos son resultados lógicos y esperados, si tenemos en cuenta que no tiene únicamente interfaces gráficas, más proclibes a usar orientación a objetos, si no que hay una gran parte de código responsable de operaciones con las tarjetas o entornos gráficos.

En la siguiente tabla se resumen los resultados obtenidos por Sloccount con las modificaciones para detectar GObject.

	SLOC	GObject	ANSI C	Otros
GTK+	316.739	242.374 (76,52 %)	72.874 (23,01 %)	1.491 (0,47 %)
GDK	94.342	47.942 (50,82 %)	46.006 (48,77 %)	394 (0,41 %)
Clutter	57.534	32.593 (56,65 %)	24.875 (43,24 %)	66 (0,11 %)

Cuadro 4.12: Sloccount sobre sistemas GObject.

Capítulo 5

Conclusiones y trabajo futuro

Los estudios realizados hasta la fecha sobre el código de GNOME y sus librerías estaban siempre orientados hacia la ingeniería del software de COCOMO. Nunca antes se habían podido realizar métricas orientadas a objeto sobre este código y en ese sentido se ha abierto una nueva línea de investigación y desarrollo. A su vez, las modificaciones realizadas en la vastamente utilizada herramienta Sloccount suponen un logro, al realizar una ampliación en una utilidad con escasa innovación debido a su completitud.

Como ya vimos en la Sección 1.5, existe una gran variedad de métricas orientadas a objetos. Una futura amplicación de esta herramienta sería incluír más de estas métricas, destinadas sobre todo a las relaciones entre las clases. Por supuesto, dado que GNOME es un proyecto vivo y con versiones cada seis meses, una evolución de las métricas a lo largo del tiempo es también un ejercicio que puede resultar interesante.

El sistema ha sido desarrollado siguiendo el modelo de software libre, y por lo tanto todo el trabajo está disponible para su observación, estudio y futuras mejoras. Esto facilitará el estudio de esta herramienta para que posibles mejoras sean desarrolladas.

El sistema GObject Introspection no funciona actualmente con todos los módulos existentes en GNOME, ya que necesita de pequeñas modificaciones en las aplicaciones. Es una tecnología tan nueva y puntera que todavía necesita adaptarse para todos los ámbitos. A medida que más módulos de GNOME vayan soportando este sistema, se podrán realizar más métricas, y comparar así diferentes implementaciones usando GObject.

El grupo de investigación GSyC/LibreSoft será quien mayor rentabilidad le saque a este trabajo en el futuro. Este proyecto ha rellenado un espacio necesario en este

grupo, ya que nunca hasta ahora habían realizado métricas orientadas a objetos. En ese sentido, serán ellos los encargados de acoplar el desarrollo llevado a cabo en este proyecto en la herramienta CVSAnalY.

En general, el proyecto ha servido para obtener por primera vez métricas orientadas a objetos sobre la implementación de GNOME, sin duda un caso único en el mundo del software. Una nueva línea de investigación se abre con este proyecto, al proporcionar unas métricas nunca antes disponibles.

Bibliografía

[Chidamber and Kemerer, 1994] Chidamber and Kemerer (1994). A metrics suite for object oriented design. *IEEE Trans. Software Eng.*

[GNOME, 2009a] GNOME (2009a). Gobject introspection. http://live.gnome.org/GObjectIntrospection.

[GNOME, 2009b] GNOME (2009b). Gobject introspection annotations. http://live.gnome.org/GObjectIntrospection/Annotations.

[Jang, 2006] Jang, D. (2006). Gnome architecture. http://www.slideshare.net/iolo/gnome-architecture.

[Morris, 1989] Morris (1989). Metrics for object-oriented software development environments. Master's thesis, M.I.T. SloanSchool of Management.

[Navarro et al., 2005] Navarro, A., Herraiz, I., and Robles, G. (2005). Jugando con demonios: Análisis del proyecto freebsd con la herramienta cvsanaly. In *Actas del I Congreso de Tecnologías del Software Libre*, La Coruña, Spain.

[Robles et al., 2004] Robles, G., Koch, S., and González-Barahona, J. M. (2004). Remote analysis and measurement of libre software systems by means of the CVSAnalY tool. In *Proceedings of the 2nd ICSE Workshop on Remote Analysis and Measurement of Software Systems (RAMSS)*, pages 51–56, Edinburg, Scotland, UK.

[Ümit Karakaþ and Sultanoðlu, 1998] Ümit Karakaþ and Sultanoðlu, S. (1998). *Object Oriented Metrics*. Department of Computer Science & Engineering, Hacettepe University.

Apéndice A

Manual de uso de GirMetrics

GirMetrics se distribuye con dos ficheros, por un lado la librería LibGirParser y por otro lado la aplicación en sí. Este manual explica cómo utilizar la aplicación GirMetrics, ya que es la parte que se usará para calcular métricas.

A.1. Instalación

- **Obtener el paquete de fuentes**

  El CD-ROM adjunto a este proyecto incluye todo el código fuente del proyecto. En el directorio dist se encuentra el paquete correspondiente a la versión 0.1 de GirMetrics. El primer paso será, por tanto, copiar el fichero girmetrics-0.1.tar.gz al directorio donde queramos utilizar el proyecto para posteriormente descomprimirlo.

- **Instalar dependencias**

  Las dependencias de GirMetrics son escasas. Basta con tener un intérprete de Python, la librería LibXML para dicho lenguaje y las utilidades cmetrics de Libresoft..

  - Dependencias de GirMetrics:
    - Python
    - Python-LibXML
    - cmetrics

El nombre de los paquetes a instalar de las dependencias de Python dependerá del sistema operativo y de la distribución concreta. Se recomienda consultar la documentación del sistema operativo para saber como instalar estas dependencias.

Las herramientas cmetrics vienen incluídas en el CD-ROM adjunto. Es necesaria su compilación e instalación manual. Para ello, hay que descomprimir el fichero cmetrics-svn.tar.gz e iniciar la instalación con las siguientes órdenes.

```
$ tar xzf extras/cmetrics-svn.tar.gz -C /tmp
$ cd /tmp/cmetrics/
$ ./configure
$ make
# make install
```

- **Obtención de ficheros GIR**

  Los ficheros GIR se generan con de la aplicación *g-ir-generate* a partir del código fuente del sistema a analizar. Sin embargo, el proceso es bastante complejo y se necesitan varias librerías y procesos.

  Algunos de estos ficheros, sin embargo, se pueden encontrar empaquetados por la distrubución. En las derivadas de Debian, el paquete gobject-introspection-repository proporciona una gran variedad de ficheros GIR, incluyendo los ejemplos descritos.

  En el CD-ROM adjunto se incluyen algunos ficheros GIR, así como el código fuente de los ejemplos descritos en la memoria.

A.2. Utilización

En el directorio donde se ha descomprimido GirMetrics existe el fichero *girmetrics.py*, de la aplicación. Para poder usarlo, primero hay que otorgarle permisos de ejecución

```
$ chmod u+x girmetrics.py
```

Con esto, para ejecutar GirMetrics basta con ejecutarlo de la siguiente forma.

```
$ ./girmetrics.py /path/to/File.gir
```

La ejecución proporcionando únicamente el fichero GIR no proporciona todas las métricas, ya que necesita los ficheros de código fuente también. Aquellas métricas que no sea posible calcular tendrán un valor de cero.

```
$ ./girmetrics.py /path/to/File.gir /path/to/sourcecode/file.c
```

Se pueden proporcionar varios ficheros de código, y se recomienda utilizar los comodines proporcionados por la shell para ello.

```
$ ./girmetrics.py /path/to/File.gir /path/to/sourcecode/*.c
```

El sistema imprime por pantalla el resultado de los cálculos. En un primer lugar muestra las métricas por cada espacio de nombres existente. Posteriormente, muestra las métricas de todas las clases existentes.

A.3. Preguntas frecuentes

¿Qué significan las siglas en la cabecera?

Para ahorrar espacio y evitar malinterpretaciones con los espacios en blanco, se han utilizado siglas en la cabecera de los resultados. El significado de las mismas es el siguiente.

- **MPC**: *Methods Per Class*. Métodos por clase.
- **MIL**: *Maximum Inheritance Length*. Máxima longitud de dependencias.
- **AMC**: *Average Method Complexity*. Complejidad ciclomática media.
- **DOC**: *Degree Of Coupling*. Grado de acoplamiento.

Con respecto a las métricas de cada clase, las siglas correspondientes son.

- **WMC**: *Weighted Methods per Class*. Métodos ponderados.
- **DIT**: *Depth of Inheritance Tree*. Longitud de dependencias.
- **NOC**: *Number Of Childrens*. Número de clases hijas

GirMetrics muestra el error “No module named”

Python es un lenguaje interpretado, donde las dependencias se resuelven en el momento de la ejecución. Es posible que falte alguna de las dependencias exigidas por GirMetrics. Asegúrese de que en el mismo directorio donde se encuentra girmetrics.py están los ficheros Command.py y libgirparser.py.

Apéndice B

Licencia Reconocimiento-CompartirIgual 2.5 España

Licencia

LA OBRA (SEGÚN SE DEFINE MÁS ADELANTE) SE PROPORCIONA BAJO LOS TÉRMINOS DE ESTA LICENCIA PÚBLICA DE CREATIVE COMMONS ("CCPL" O "LICENCIA"). LA OBRA SE ENCUENTRA PROTEGIDA POR LA LEY ESPAÑOLA DE PROPIEDAD INTELECTUAL Y/O CUALESQUIERA OTRAS NORMAS RESULTEN DE APLICACIÓN. QUEDA PROHIBIDO CUALQUIER USO DE LA OBRA DIFERENTE A LO AUTORIZADO BAJO ESTA LICENCIA O LO DISPUESTO EN LAS LEYES DE PROPIEDAD INTELECTUAL.

MEDIANTE EL EJERCICIO DE CUALQUIER DERECHO SOBRE LA OBRA, USTED ACEPTA Y CONSIENTE LAS LIMITACIONES Y OBLIGACIONES DE ESTA LICENCIA. EL LICENCIADOR LE CEDE LOS DERECHOS CONTENIDOS EN ESTA LICENCIA, SIEMPRE QUE USTED ACEPTE LOS PRESENTES TÉRMINOS Y CONDICIONES.

1. Definiciones

1. La "obra" es la creación literaria, artística o científica ofrecida bajo los términos de esta licencia.

2. El "autor" es la persona o la entidad que creó la obra.

3. Se considerará “obra conjunta” aquella susceptible de ser incluida en alguna de las siguientes categorías:

   a) “Obra en colaboración”, entendiendo por tal aquella que sea resultado unitario de la colaboración de varios autores.

   b) “Obra colectiva”, entendiendo por tal la creada por la iniciativa y bajo la coordinación de una persona natural o jurídica que la edite y divulgue bajo su nombre y que esté constituida por la reunión de aportaciones de diferentes autores cuya contribución personal se funde en una creación única y autónoma, para la cual haya sido concebida sin que sea posible atribuir separadamente a cualquiera de ellos un derecho sobre el conjunto de la obra realizada.

   c) “Obra compuesta e independiente”, entendiendo por tal la obra nueva que incorpore una obra preexistente sin la colaboración del autor de esta última.

4. Se considerarán “obras derivadas” aquellas que se encuentren basadas en una obra o en una obra y otras preexistentes, tales como: las traducciones y adaptaciones; las revisiones, actualizaciones y anotaciones; los compendios, resúmenes y extractos; los arreglos musicales y, en general, cualesquiera transformaciones de una obra literaria, artística o científica, salvo que la obra resultante tenga el carácter de obra conjunta en cuyo caso no será considerada como una obra derivada a los efectos de esta licencia. Para evitar la duda, si la obra consiste en una composición musical o grabación de sonidos, la sincronización temporal de la obra con una imagen en movimiento (“synching”) será considerada como una obra derivada a los efectos de esta licencia.

5. Tendrán la consideración de “obras audiovisuales” las creaciones expresadas mediante una serie de imágenes asociadas, con o sin sonorización incorporada, así como las composiciones musicales, que estén destinadas esencialmente a ser mostradas a través de aparatos de proyección o por cualquier otro medio de comunicación pública de la imagen y del sonido, con independencia de la naturaleza de los soportes materiales de dichas obras.

6. El “licenciador” es la persona o la entidad que ofrece la obra bajo los términos de esta licencia y le cede los derechos de explotación de la misma conforme a lo dispuesto en ella.

7. “Usted” es la persona o la entidad que ejercita los derechos cedidos mediante esta licencia y que no ha violado previamente los términos de la misma con respecto a la obra, o que ha recibido el permiso expreso del licenciador de ejercitar los derechos cedidos mediante esta licencia a pesar de una violación anterior.

8. La “transformación” de una obra comprende su traducción, adaptación y cualquier otra modificación en su forma de la que se derive una obra diferente. Cuando se trate de una base de datos según se define más adelante, se considerará también transformación la reordenación de la misma. La creación resultante de la transformación de una obra tendrá la consideración de obra derivada.

9. Se entiende por “reproducción” la fijación de la obra en un medio que permita su comunicación y la obtención de copias de toda o parte de ella.

10. Se entiende por “distribución” la puesta a disposición del público del original o copias de la obra mediante su venta, alquiler, préstamo o de cualquier otra forma.

11. Se entenderá por “comunicación pública” todo acto por el cual una pluralidad de personas pueda tener acceso a la obra sin previa distribución de ejemplares a cada una de ellas. No se considerará pública la comunicación cuando se celebre dentro de un ámbito estrictamente doméstico que no esté integrado o conectado a una red de difusión de cualquier tipo. A efectos de esta licencia se considerará comunicación pública la puesta a disposición del público de la obra por procedimientos alámbricos o inalámbricos, incluida la puesta a disposición del público de la obra de tal forma que cualquier persona pueda acceder a ella desde el lugar y en el momento que elija.

12. La “explotación” de la obra comprende su reproducción, distribución, comunicación pública y transformación.

13. Tendrán la consideración de “bases de datos” las colecciones de obras ajenas, de datos o de otros elementos independientes como las antologías y las bases de datos propiamente dichas que por la selección o disposición de sus contenidos constituyan creaciones intelectuales, sin perjuicio, en su caso, de los derechos que pudieran subsistir sobre dichos contenidos.

14. Los “elementos de la licencia” son las características principales de la licencia según la selección efectuada por el licenciador e indicadas en el título de esta

licencia: Reconocimiento de autoría (Reconocimiento), Compartir de manera igual (CompartirIgual).

2. Límites y uso legítimo de los derechos. Nada en esta licencia pretende reducir o restringir cualesquiera límites legales de los derechos exclusivos del titular de los derechos de propiedad intelectual de acuerdo con la Ley de Propiedad Intelectual o cualesquiera otras leyes aplicables, ya sean derivados de usos legítimos, tales como el derecho de copia privada o el derecho a cita, u otras limitaciones como la derivada de la primera venta de ejemplares.

3. Concesión de licencia. Conforme a los términos y a las condiciones de esta licencia, el licenciador concede (durante toda la vigencia de los derechos de propiedad intelectual) una licencia de ámbito mundial, sin derecho de remuneración, no exclusiva e indefinida que incluye la cesión de los siguientes derechos:

1. Derecho de reproducción, distribución y comunicación pública sobre la obra.

2. Derecho a incorporarla en una o más obras conjuntas o bases de datos y para su reproducción en tanto que incorporada a dichas obras conjuntas o bases de datos.

3. Derecho para efectuar cualquier transformación sobre la obra y crear y reproducir obras derivadas.

4. Derecho de distribución y comunicación pública de copias o grabaciones de la obra, como incorporada a obras conjuntas o bases de datos.

5. Derecho de distribución y comunicación pública de copias o grabaciones de la obra, por medio de una obra derivada.

6. Para evitar la duda, sin perjuicio de la preceptiva autorización del licenciador, y especialmente cuando la obra se trate de una obra audiovisual, el licenciador renuncia al derecho exclusivo a percibir, tanto individualmente como mediante una entidad de gestión de derechos, o varias, (por ejemplo: SGAE, Dama, VEGAP), los derechos de explotación de la obra, así como los derivados de obras derivadas, conjuntas o bases de datos, si dicha explotación pretende principalmente o se encuentra dirigida hacia la obtención de un beneficio mercantil o la remuneración monetaria privada.

Los anteriores derechos se pueden ejercitar en todos los medios y formatos, tangibles o intangibles, conocidos o por conocer. Los derechos mencionados incluyen el derecho a efectuar las modificaciones que sean precisas técnicamente para el ejercicio de los derechos en otros medios y formatos. Todos los derechos no cedidos expresamente por el licenciador quedan reservados.

4. Restricciones. La cesión de derechos que supone esta licencia se encuentra sujeta y limitada a las restricciones siguientes:

1. Usted puede reproducir, distribuir o comunicar públicamente la obra solamente bajo los términos de esta licencia y debe incluir una copia de la misma, o su Identificador Uniforme de Recurso (URI), con cada copia o grabación de la obra que usted reproduzca, distribuya o comunique públicamente. Usted no puede ofrecer o imponer ningún término sobre la obra que altere o restrinja los términos de esta licencia o el ejercicio de sus derechos por parte de los cesionarios de la misma. Usted no puede sublicenciar la obra. Usted debe mantener intactos todos los avisos que se refieran a esta licencia y a la ausencia de garantías. Usted no puede reproducir, distribuir o comunicar públicamente la obra con medidas tecnológicas que controlen el acceso o uso de la obra de una manera contraria a los términos de esta licencia. Lo anterior se aplica a una obra en tanto que incorporada a una obra conjunta o base de datos, pero no implica que éstas, al margen de la obra objeto de esta licencia, tengan que estar sujetas a los términos de la misma. Si usted crea una obra conjunta o base de datos, previa comunicación del licenciador, usted deberá quitar de la obra conjunta o base de datos cualquier crédito requerido en el apartado 4c, según lo que se le requiera y en la medida de lo posible. Si usted crea una obra derivada, previa comunicación del licenciador, usted deberá quitar de la obra derivada cualquier crédito requerido en el apartado 4c, según lo que se le requiera y en la medida de lo posible.

2. Usted puede reproducir, distribuir o comunicar públicamente una obra derivada solamente bajo los términos de esta licencia, o de una versión posterior de esta licencia con sus mismos elementos principales, o de una licencia iCommons de Creative Commons que contenga los mismos elementos principales que esta licencia (ejemplo: Reconocimiento-CompartirIgual 2.5 Japón). Usted debe incluir una copia de la esta licencia o de la mencionada anteriormente, o bien su Identificador Uniforme de Recurso (URI), con cada copia o grabación de la obra

que usted reproduzca, distribuya o comunique públicamente. Usted no puede ofrecer o imponer ningún término respecto de las obras derivadas o sus transformaciones que alteren o restrinjan los términos de esta licencia o el ejercicio de sus derechos por parte de los cesionarios de la misma. Usted debe mantener intactos todos los avisos que se refieran a esta licencia y a la ausencia de garantías. Usted no puede reproducir, distribuir o comunicar públicamente la obra derivada con medidas tecnológicas que controlen el acceso o uso de la obra de una manera contraria a los términos de esta licencia. Lo anterior se aplica a una obra derivada en tanto que incorporada a una obra conjunta o base de datos, pero no implica que éstas, al margen de la obra objeto de esta licencia, tengan que estar sujetas a los términos de esta licencia.

3. Si usted reproduce, distribuye o comunica públicamente la obra o cualquier obra derivada, conjunta o base datos que la incorpore, usted debe mantener intactos todos los avisos sobre la propiedad intelectual de la obra y reconocer al autor original, de manera razonable conforme al medio o a los medios que usted esté utilizando, indicando el nombre (o el seudónimo, en su caso) del autor original si es facilitado, y/o reconocer a aquellas partes (por ejemplo: institución, publicación, revista) que el autor original y/o el licenciador designen para ser reconocidos en el aviso legal, las condiciones de uso, o de cualquier otra manera razonable; el título de la obra si es facilitado; de manera razonable, el Identificador Uniforme de Recurso (URI), si existe, que el licenciador especifica para ser vinculado a la obra, a menos que tal URI no se refiera al aviso sobre propiedad intelectual o a la información sobre la licencia de la obra; y en el caso de una obra derivada, un aviso que identifique el uso de la obra en la obra derivada (e.g., "traducción castellana de la obra de Autor Original,.º "guión basado en obra original de Autor Original"). Tal aviso se puede desarrollar de cualquier manera razonable; con tal de que, sin embargo, en el caso de una obra derivada, conjunta o base datos, aparezca como mínimo este aviso allá donde aparezcan los avisos correspondientes a otros autores y de forma comparable a los mismos.

4. En el caso de la inclusión de la obra en alguna base de datos o recopilación, el propietario o el gestor de la base de datos deberá renunciar a cualquier derecho relacionado con esta inclusión y concerniente a los usos de la obra una vez extraída de las bases de datos, ya sea de manera individual o conjuntamente con

otros materiales.

5. Exoneración de responsabilidad.

A MENOS QUE SE ACUERDE MUTUAMENTE ENTRE LAS PARTES, EL LICENCIADOR OFRECE LA OBRA TAL CUAL (ON AN .ᴬS-IS"BASIS) Y NO CONFIERE NINGUNA GARANTÍA DE CUALQUIER TIPO RESPECTO DE LA OBRA O DE LA PRESENCIA O AUSENCIA DE ERRORES QUE PUEDAN O NO SER DESCUBIERTOS. ALGUNAS JURISDICCIONES NO PERMITEN LA EXCLUSIÓN DE TALES GARANTÍAS, POR LO QUE TAL EXCLUSIÓN PUEDE NO SER DE APLICACIÓN A USTED.

6. Limitación de responsabilidad.

SALVO QUE LO DISPONGA EXPRESA E IMPERATIVAMENTE LA LEY APLICABLE, EN NINGÚN CASO EL LICENCIADOR SERÁ RESPONSABLE ANTE USTED POR CUALQUIER TEORÍA LEGAL DE CUALESQUIERA DAÑOS RESULTANTES, GENERALES O ESPECIALES (INCLUIDO EL DAÑO EMERGENTE Y EL LUCRO CESANTE), FORTUITOS O CAUSALES, DIRECTOS O INDIRECTOS, PRODUCIDOS EN CONEXIÓN CON ESTA LICENCIA O EL USO DE LA OBRA, INCLUSO SI EL LICENCIADOR HUBIERA SIDO INFORMADO DE LA POSIBILIDAD DE TALES DAÑOS.

7. Finalización de la licencia.

1. Esta licencia y la cesión de los derechos que contiene terminarán automáticamente en caso de cualquier incumplimiento de los términos de la misma. Las personas o entidades que hayan recibido obras derivadas, conjuntas o bases de datos de usted bajo esta licencia, sin embargo, no verán sus licencias finalizadas, siempre que tales personas o entidades se mantengan en el cumplimiento íntegro de esta licencia. Las secciones 1, 2, 5, 6, 7 y 8 permanecerán vigentes pese a cualquier finalización de esta licencia.

2. Conforme a las condiciones y términos anteriores, la cesión de derechos de esta licencia es perpetua (durante toda la vigencia de los derechos de propiedad intelectual aplicables a la obra). A pesar de lo anterior, el licenciador se reserva el derecho a divulgar o publicar la obra en condiciones distintas a las presentes, o de retirar la obra en cualquier momento. No obstante, ello no supondrá dar por concluida esta licencia (o cualquier otra licencia que haya sido concedida,

o sea necesario ser concedida, bajo los términos de esta licencia), que continuará vigente y con efectos completos a no ser que haya finalizado conforme a lo establecido anteriormente.

8. Miscelánea.

1. Cada vez que usted explote de alguna forma la obra, o una obra conjunta o una base datos que la incorpore, el licenciador original ofrece a los terceros y sucesivos licenciatarios la cesión de derechos sobre la obra en las mismas condiciones y términos que la licencia concedida a usted.

2. Cada vez que usted explote de alguna forma una obra derivada, el licenciador original ofrece a los terceros y sucesivos licenciatarios la cesión de derechos sobre la obra original en las mismas condiciones y términos que la licencia concedida a usted.

3. Si alguna disposición de esta licencia resulta inválida o inaplicable según la Ley vigente, ello no afectará la validez o aplicabilidad del resto de los términos de esta licencia y, sin ninguna acción adicional por cualquiera las partes de este acuerdo, tal disposición se entenderá reformada en lo estrictamente necesario para hacer que tal disposición sea válida y ejecutiva.

4. No se entenderá que existe renuncia respecto de algún término o disposición de esta licencia, ni que se consiente violación alguna de la misma, a menos que tal renuncia o consentimiento figure por escrito y lleve la firma de la parte que renuncie o consienta.

5. Esta licencia constituye el acuerdo pleno entre las partes con respecto a la obra objeto de la licencia. No caben interpretaciones, acuerdos o términos con respecto a la obra que no se encuentren expresamente especificados en la presente licencia. El licenciador no estará obligado por ninguna disposición complementaria que pueda aparecer en cualquier comunicación de usted. Esta licencia no se puede modificar sin el mutuo acuerdo por escrito entre el licenciador y usted.

Creative Commons no es parte de esta licencia, y no ofrece ninguna garantía en relación con la obra. Creative Commons no será responsable frente a usted o a cualquier parte, por cualquier teoría legal de cualesquiera daños resultantes, incluyendo, pero no

limitado, daños generales o especiales (incluido el daño emergente y el lucro cesante), fortuitos o causales, en conexión con esta licencia. A pesar de las dos (2) oraciones anteriores, si Creative Commons se ha identificado expresamente como el licenciador, tendrá todos los derechos y obligaciones del licenciador.

Salvo para el propósito limitado de indicar al público que la obra está licenciada bajo la CCPL, ninguna parte utilizará la marca registrada “Creative Commons” o cualquier marca registrada o insignia relacionada con “Creative Commons” sin su consentimiento por escrito. Cualquier uso permitido se hará de conformidad con las pautas vigentes en cada momento sobre el uso de la marca registrada por “Creative Commons”, en tanto que sean publicadas su sitio web (website) o sean proporcionadas a petición previa.

Puede contactar con Creative Commons en: `http://creativecommons.org/`.