Escola Tècnica Superior d'Enginyeria de Telecomunicació de Barcelona

UNIVERSITAT POLITÈCNICA DE CATALUNYA

DEPARTAMENTO DE TELEMÁTICA

PROYECTO FINAL DE CARRERA

SISTEMA DE GENERACIÓN AUTOMÁTICA DE EVENTOS CON APLICACIÓN A LA GESTIÓN DE COMUNIDADES VIRTUALES

Autor: **David Hidalgo Muñoz**
Director: **Marcel Fernández Muñoz**

Octubre de 2016

RESUMEN

El concepto de web 2.0 surgió para revolucionar el diseño y funcionalidad de las páginas web originales y el propio mundo de internet. Dicho concepto hace referencia a páginas web basadas en las conocidas comunidades virtuales o redes sociales, esas páginas en las que su principal objetivo es la interacción entre los usuarios, creando así un entorno dinámico. Las aplicaciones web orientadas al usuario han evolucionado con paso firme en los últimos años, ofreciendo cada vez más funcionalidades, a cada cual más sorprendente que la anterior.

Este proyecto tiene como objetivo crear un módulo funcional para una aplicación web de tipo red social. El módulo permitirá al usuario de la comunidad, la creación de contenidos, llamados eventos, de forma automática o casi automática dependiendo del método utilizado. Los datos para la creación se obtendrán a través de medios ajenos a la interfaz, utilizando el correo electrónico o el terminal móvil del usuario.

Para la facilidad de testeo de la aplicación e introducción de mejoras se extraerá el módulo de la web y se encapsulará dentro de una aplicación gráfica independiente denominada Generador de Eventos. El análisis de los resultados obtenidos a la hora de crear algunos eventos de prueba, determinará el comportamiento, fiabilidad y eficacia del proceso de generación automática de eventos de la aplicación.

Agradecimientos

En primer lugar, mi más sincero agradecimiento a mi tutor de proyecto Marcel Fernández por haberme brindado la oportunidad de poder realizar el presente proyecto final de carrera. También a Juan Luis Gorricho y Josep Cotrina por el tiempo que dedicaron para que los módulos de mi trabajo encajaran con el proyecto conjunto que realizaba el departamento de telemática. Especial mención a Daniel Nieto, miembro de Orange responsable de la validación del producto, por haber realizado pruebas de aplicación, incluida la generación automática de eventos, las comunicaciones vía correo electrónico y mensajes de texto y haber dado el visto bueno al funcionamiento del módulo de generación de eventos integrado en la Agenda Social.

Tengo que remarcar que gracias al apoyo de mi familia, novia, amigos y compañeros de trabajo he podido avanzar por esta odisea. Han sabido darme el apoyo que he necesitado cuando he estado falto de motivación y rebosante de estrés.

Han sido unos años muy duros, lo que en un principio pensaba que solo me iba a llevar unos meses, al final se ha convertido en un trabajo eterno. La realización de la parte técnica del proyecto, diseño e implementación de la solución, fue realizada en el tiempo marcado ya que la entrega de la aplicación venía fijada por calendario. En cambio, la realización de la memoria, la plasmación del trabajo en papel, es algo que me ha llevado mucho más tiempo del que me hubiese gustado. Los motivos han sido varios, pero sin entrar en detalle, se han generado discontinuidades en el trabajo que al retomarse te hace entrar en un círculo de rehacer cosas que ya estaban hechas e intentar mejorar el trabajo ya realizado.

Aparte de la experiencia vivida y el enriquecimiento personal me quedo con dos conceptos. El primero es el saber darse cuenta que un cierto trabajo y sobretodo un software puede intentar mejorarse eternamente pero hay que saber marcar una meta y no sobrepasarla, y lo segundo, intentar inculcarme más la frase: “No dejes para mañana lo que puedas hacer hoy”, mi madre me lo ha dicho constantemente a lo largo de mi vida y es algo que siempre me ha costado poner en práctica.

Aprovecho para incluir en la presente memoria, que pone punto final a unos estudios superiores de ingeniería de telecomunicaciones, un agradecimiento al conjunto de personas, compañeros y profesores que a lo largo de todos estos años me han ayudado a evolucionar personalmente y prepararme para el mundo laboral.

Índice general

Contenidos

Índice de figuras

Índice de tablas

Índice de fragmentos de código

Capítulo 1

Introducción

La realización de este proyecto se brinda como una oportunidad perfecta para profundizar en un concepto en constante evolución en el extendido y maduro mundo de internet: la web 2.0. Una de sus características principales es que los usuarios pasaron de ser consumidores pasivos a convertirse en participantes activos, siendo esta participación una de las claves de la evolución y de los cambios del contenido de las páginas web.

Utilizando una analogía al mundo de la televisión, los usuarios solo tenían capacidad para hacer *zapping* en la red, cambiar de canal o página. Actualmente somos productores de contenidos con la capacidad de crear canales, lugares donde somos nosotros mismos los que decidimos qué queremos que vean los demás.

El desarrollo de este trabajo se ha centrado en gran medida en otorgarles a estos usuarios una manera menos convencional de crear contenidos para la web 2.0, si entendemos como método convencional el acceder a la página web e introducir manualmente los datos en un formulario. El método desarrollado contempla la creación automática de contenidos a partir de una fuente externa de información.

1.1. Contexto histórico social

El concepto de web 2.0 fue utilizado por primera vez en la conferencia O'Reilly Media Web 2.0 ([1]) en el año 2004. El 2.0 hace referencia a una evolución de las páginas web que dieron sus primeros pasos en el mundo de la WWW[1], las consideradas 1.0. Gracias al avance tecnológico, el desarrollo de nuevas herramientas de diseño y programación, y algún que otro enfoque económico-lucrativo, aquellas páginas estáticas dieron paso a una nueva generación de páginas web con un sinfín de funcionalidades y diseños espectaculares. Esas páginas web, rebosantes de dinamismo, son consideradas como 1.5, su estructura es identificable como webs generadas dinámicamente por un CMS[2] a partir de una base de datos (BD).

La web 2.0 no marca unas pautas a seguir, ni protocolos de diseño o estructuración, simplemente recoge varios de los servicios y funcionalidades ya existentes y les da un enfoque diferente: la creación de comunidades virtuales. Algunos ejemplos de estas comunidades son las *redes sociales*, *blogs* y *wikis*. Las comunidades virtuales tienen como punto común la interacción entre los usuarios, fomentando un intercambio rápido de información. Las características más notorias son: la simplicidad de la interfaz (ahorrándole tiempo al usuario y facilitando la interactuación), el uso de estándares de lenguaje y establecimiento de una relación próxima entre el medio de comunicación y el contenido, facilitando la movilidad, publicación y consulta de contenidos (migración de aplicaciones de escritorio a entornos web). Los contenidos están creados por y para los usuarios.

Aunque no se marca ninguna tecnología a seguir, sí deben emplearse aquellas que permiten obtener las funcionalidades básicas para la creación de las comunidades virtuales. En la Tabla 1.1 se muestra la evolución comentada de las páginas web, con sus períodos dominantes y las tecnologías utilizadas ([2]).

Tabla 1.1: Evolución de las páginas web

Web	Tipo	Período	Tecnología asociada	Características
1.0	estática	1994-1997	HTML, GIF	Documentos estáticos. No actualizable.
1.5	dinámica	1997-2003	DHTML, ASP, CSS	Construcción dinámica. Utilización de bases de datos.
2.0	colaborativa	2003-hoy	Ajax, DHTML, XML, Soap	Contribución de los usuarios. Los usuarios publican información y realizan cambios en los datos.

La siguiente generación será la nombrada web 3.0, conocida como *Web Semántica* ([3]). Estas webs no son propiamente una evolución de las anteriores sino que muestran una actitud nueva, la de otorgarle a la página web un significado. Todavía es un concepto abstracto que debe pulirse, pero su principal objetivo es otorgarle al sitio web la capacidad de trabajar con un volumen mucho mayor de datos, interpretándolos e interconectándolos.

[1] World Wide Web: Red Global Mundial.
[2] Content Management System: Sistema de gestión de contenidos.

1.2. Punto de partida

Muchos de los trabajos realizados de gran envergadura como proyectos e investigaciones, suelen partir de un documento, trabajo o estudio previo. Una parte importante de la ingeniería es apoyarse en el conocimiento existente para conseguir mejores resultados. Como en muchos otros trabajos, el desarrollo de este proyecto no ha partido desde cero.

Desde un inicio el proyecto se orientó a darle continuidad al desarrollo, mejora y creación de algunas funcionalidades[3] para una aplicación web 2.0. En el momento del inicio de este proyecto varios miembros del departamento de telemática de la ETSETB[4] y compañeros proyectistas estaban trabajando en la elaboración de una Agenda Social[5] orientada a comunidades virtuales. La aplicación y portal web recibiría su nombre final de proyecto como OCEM[6], aunque a lo largo de este documento se hace referencia a él con su nombre de desarrollo: Agenda Social.

Como se ha comentado en el punto anterior, en el mundo web 2.0 son los usuarios los que crean los contenidos. Para la Agenda Social, dichos contenidos reciben el nombre de eventos. En los próximos capítulos se entrará en detalle sobre la información que contiene un evento y cómo se crea.

El presente proyecto originalmente era una pieza de la aplicación web Agenda Social. Su desarrollo desde el origen ha estado vinculado a la creación de la página web como producto completo, por lo que el trabajo realizado ha estado condicionado por la integración dentro de ésta. Aunque el desarrollo ha sido individual si ha estado condicionado al trabajo que realizaban los diferentes miembros, teniendo que adaptarse su funcionalidad y diseño según los requerimientos que eventualmente cambiaban a medida que avanzaba el desarrollo del equipo de trabajo. Uno de los principales motivos de cambios de concepto a la largo de la fase de diseño se debía a que la aplicación estaba orientada a cliente y dependía de sus necesidades y requisitos, y al estar en una fase constante de desarrollo y evolución las necesidades podían ir cambiando en pocas semanas.

Hay que mencionar que algunas de las funcionalidades de la aplicación ya habían sido iniciadas por mi tutor de proyecto y mi trabajo era darle continuidad y ampliación para ajustarse a los requisitos de desarrollo de la Agenda Social. Esto incluye una primera versión de conexión a una cuenta de correo, análisis “línea-a-línea” de un correo en texto plano y extracción de sus datos para la generación de un evento.

[3] Funcionalidad hace referencia a una o varias tareas realizadas por la aplicación.
[4] Escola Tècnica Superior d’Enginyeria de Telecomunicació de Barcelona: Escuela Técnica Superior de Ingeniería de Telecomunicación de Barcelona.
[5] Aplicación web 2.0 sobre la que se inició el desarrollo del proyecto.
[6] Orange Comunity Events Manager: Gestor de Eventos de la Comunidad Orange.

1.3. Objetivos

El objetivo principal de este proyecto final de carrera es desarrollar un módulo para la Agenda Social capaz de generar contenido para sus usuarios de forma automática o semiautomática[7]. A lo largo de esta memoria nos referiremos a este módulo como *Generador de Eventos*.

La creación de eventos se realizará a través de la interpretación de mensajes enviados mediante un terminal móvil (SMS[8]) o correos electrónicos (e-mails o simplemente *mails*) que el usuario enviará de forma manual o automática. Esta vía de creación de eventos complementará el método tradicional de creación de contenidos usando la interfaz gráfica web. Esta funcionalidad la otorga la Agenda Social y no forma parte del trabajo realizado en el presente proyecto, por lo que este trabajo se centrará exclusivamente en las funcionalidades del Generador de Eventos. El objetivo final es un bloque operativo que se obtiene una vez alcanzados varios objetivos:

- Diseño e implementación de un analizador sintáctico de contenidos basado en la utilización de expresiones regulares[9].
- Creación y estudio de patrones de búsqueda en contenidos orientados a un evento.
- Establecer conexión de entrada y salida de mensajería de texto vía correo electrónico.
- Establecer conexión de entrada y salida de mensajería de texto vía terminal móvil.
- Integración de las diferentes partes dentro de la Agenda Social.
- Análisis y evaluación de los resultados obtenidos.

Un objetivo personal marcado para la realización del módulo ha sido el poder orientar el diseño a una fácil ampliación de la aplicación o de poder ser retocado para poder ser utilizado en otro tipo de aplicaciones, darle la posibilidad de poder ser aprovechado en otro tipo de desarrollos, alejándose así del enfoque inicial de integración en la Agenda Social. Siguiendo esta línea se ha acabado extrayendo el módulo y encapsulándolo en una aplicación independiente más fácil de estudiar, analizar y ejecutar que si formara parte de un proyecto web de mucha mayor envergadura dando pie al objetivo adicional:

- Extracción del módulo y encapsulado en una aplicación independiente.

El objetivo principal alcanzado define el título de este proyecto final de carrera: la creación de un "**Sistema de generación automática de eventos con aplicación a la gestión de comunidades virtuales**".

[7] Detalle de generación automática / semiautomática en la sección 2.2.9.
[8] Sistema de mensajes de texto para teléfonos móviles.
[9] Véase expresiones regulares en la sección 2.2.5.

1.4. Conocimientos previos y adquiridos

Para la realización y seguimiento de la parte técnica del proyecto son necesarios unos conocimientos básicos sobre el lenguaje de programación Java. Se definen varias plataformas para cubrir distintos entornos de aplicación ([4]), de las cuales serán utilizadas dos para la realización del proyecto a través de sus APIs[10]. Las plataformas están detalladas en la Tabla 1.2.

Tabla 1.2: Plataformas Java utilizadas

Nombre	Descripción
J2SE	*Java 2 Platform, Standard Edition* es la plataforma base de Java. Incluye la máquina virtual de Java, el compilador, el depurador y unas cuantas herramientas más. Está orientada para el desarrollo de aplicaciones pequeñas, aplicaciones de escritorio, o clientes de aplicaciones grandes.
J2EE	*Java 2 Platform, Enterprise Edition* es la plataforma orientada al desarrollo de aplicaciones grandes, orientadas a empresa. Es un conjunto de especificaciones de cómo se deben implementar las aplicaciones para plataformas empresariales. Utilizada para aplicaciones de tipo servidor.

El Generador de Eventos ha pasado por tres fases de desarrollo. La fase inicial como aplicación realizada en Java Standard (J2SE), ya que las APIs necesarias para el diseño y creación no requieren de la funcionalidad de página web. En la segunda fase se incorporó como módulo operativo a un proyecto web, por lo que ha sido necesaria la adquisición de conocimientos básicos de aplicaciones J2EE para servidores web. Finalmente en la tercera fase se ha extraído el Generador de Eventos y se ha vinculado con una aplicación de escritorio desarrollada en JavaFX. Se enumeran a continuación, por orden de desarrollo, las áreas técnicas de conocimiento más importantes que han sido necesarias adquirir o consolidar para el desarrollo del proyecto[11]:

- Aplicaciones web 2.0, J2EE y Servlets.
- Expresiones regulares.
- Patrones de diseño orientados a la programación.
- API mail de Java y tipos MIME utilizados para extracción de información de e-mails.
- Lenguaje HTML.
- Pasarela Parlay X de Orange y API para SMS de Orange y otras APIs de terceros.
- Árboles para almacenamiento de información (Diccionarios).
- Árboles XML y obtención de datos mediante DOM.
- Consolidar conocimientos de comunicaciones TCP/IP.
- Consolidar conocimientos de desarrollo de interfaces gráficas en Java (JavaFX).
- Consolidar conocimientos de lenguaje de bases de datos relacionales MySQL.

[10] Application Program Interface: Interfaz para la programación de aplicaciones.
[11] Todos los conceptos serán introducidos en capítulos posteriores.

1.5. Estructura de la memoria

La presente memoria está dividida en diversos capítulos con los siguientes contenidos:

- *Capítulo 2. Análisis.*

  La descripción detallada del punto de partida, estudio y evolución hacia la solución final, discutiendo algunas de las soluciones propuestas y describiendo funcionalmente la más adecuada. Pretende dar respuestas a qué debe hacer. Al final de este capítulo se detallará también todo lo necesario de la aplicación web Agenda Social que influye en el desarrollo e integración del Generador de Eventos y las consideraciones a tener en cuenta para la extracción del Generador de Eventos de la Agenda Social.

- *Capítulo 3. Diseño e implementación.*

  En el capítulo 3 se aborda directamente cómo a partir de los objetivos y el análisis se crea la solución. Entra en el detalle funcional en todos los módulos y partes desarrolladas o incorporadas a la aplicación, ya sean especificaciones básicas o mejoras. Este capítulo contiene el peso técnico de la memoria, abarcando el estudio, funcionamiento y desarrollo con el lenguaje de programación Java.

- *Capítulo 4. Pruebas y resultados.*

  En el capítulo de pruebas y resultados se muestran los métodos de prueba de las diferentes partes y el conjunto global del Generador de Eventos con varios ejemplos. A partir de ellos se obtendrá una idea general de su comportamiento y la eficacia en la generación de eventos automáticamente.

- *Capítulo 5. Conclusiones.*

  Detalle de las conclusiones que podemos extraer, propuestas de fututo.

- *Bibliografía y apéndices.*

Capítulo 2

Análisis

Este capítulo entra en detalle en el *qué* de este proyecto final de carrera; qué se desea que haga y cómo será implementado. Mediante un estudio por bloques se analizarán los diferentes requerimientos y juntándolos obtendremos la solución del objetivo a cumplir.

En muchos de los puntos encontraremos propuestas de diseño y algunas soluciones que podrían satisfacernos, escogiendo las que más se adecuen a nuestras necesidades. Se detallará el análisis de la solución a largos trazos sin entrar en los pequeños detalles, reservados para el posterior capítulo.

Cabe mencionar que el contenido de este capítulo está exento en la medida de lo posible de detalle técnico o de implementación de la solución. Ha sido redactado con el propósito de ser comprendido por el mayor número de lectores sin conocimientos avanzados de programación.

Al final del capítulo se describe brevemente las partes de la Agenda Social que más influyen para la parte del diseño y las consideraciones a tener en cuenta para su extracción. También se introducirán las aplicaciones de escritorio creadas para independizar la solución del proyecto web que forman la solución final del Generador de Eventos.

2.1. Descripción funcional

Los desarrolladores de aplicaciones web orientadas a comunidades virtuales –como muchas otras aplicaciones en las que el éxito lo marca el número de usuarios- centran gran parte de sus esfuerzos en desarrollar, inventar, integrar o incluso descubrir nuevas características que atraigan a nuevos usuarios, o incluso, que mantengan a los usuarios de la comunidad.

La funcionalidad básica que ofrece un portal orientado a comunidades virtuales es que los usuarios puedan crear sus contenidos, sean productores, pero a la vez consumidores de los creados por otros usuarios.

Comúnmente, la creación de contenido para las comunidades virtuales se lleva a cabo mediante la interfaz web. Los usuarios se conectan a la aplicación mediante su *login*[12] , iniciando sesión en la web. En el menú correspondiente el usuario puede crear un evento que los demás usuarios podrán visualizar. La Agenda Social incorpora otras muchas funcionalidades, pero no serán comentadas a no ser que alguna de ellas esté relacionada con algún punto del trabajo realizado.

Este proyecto parte de la premisa de añadir a la Agenda Social la funcionalidad que otorga al usuario dos métodos alternativos para la creación de contenidos o eventos:

- Los usuarios envían o reenvían correos electrónicos automáticamente desde su cuenta de correo. Los correos se analizan y se crean eventos a partir de ellos.
- A través de mensajes de texto vía móvil que el usuario puede enviar a un número de marcación corta propiedad de la aplicación.

Al integrarse en la Agenda Social, también le ofrece la posibilidad de enviar correos y SMS informativos a los usuarios. La recepción de SMSs informativos por parte de las aplicaciones es una funcionalidad valorada positivamente por los usuarios. Esto les ofrece una gran comodidad para el seguimiento de contenidos y la recepción de notificaciones.

Al extraerse posteriormente el Generador de Eventos de la Agenda Social se gana en independencia de desarrollo y se pueden aplicar ciertas mejoras no vinculantes al proyecto web.

[12] Acceso a un sistema informático o aplicación mediante un identificador de usuario y una contraseña.

2.2. Requisitos y especificaciones

El requisito principal del Generador de Eventos es la creación de un Analizador Sintáctico de texto. Dentro del contexto del presente trabajo utilizaremos la palabra *parseador* como neologismo proveniente del inglés *parser,* para hacer referencia al analizador sintáctico. Por analogía, *parsear* a la acción de realizar un análisis sintáctico, que no es más que la extracción de información a partir de un texto con la finalidad de crear un evento.

La obtención de datos del parseador se realizará a través de mensajes de texto vía telefonía móvil o correos electrónicos. Para ello, la aplicación tiene que poder gestionar la entrada y salida de mensajes de texto y de correos electrónicos. La comunicación por SMS deberá ser integrada bajo la pasarela Parlay X[13] de la compañía de telefonía Orange, mientras que la de los correos se realizará a través de una cuenta de correo que será propiedad de la Agenda Social y gestionada por sus administradores.

Una vez obtenido un mensaje de texto, debe analizarse y obtener el máximo de información válida para crear un evento propiedad del usuario. Uno de los requisitos iniciales de diseño es que el analizador sintáctico debe realizarse mediante el uso de expresiones regulares.

Otro requisito de desarrollo es que el generador de eventos debe estar integrado completamente en la aplicación web. Esto crea una dependencia debido a que en muchos puntos del código deberá llamarse a funciones propias del proyecto web. Aparte de la integración en la Agenda Social se debe realizar un diseño que permita su fácil separación de la Agenda Social y ejecución como aplicación independiente.

Uno de los requisitos añadidos a posteriori durante la fase de desarrollo de la Agenda Social y la integración del módulo de generación de eventos fue que el parseador fuese capaz de poder interpretar varios idiomas. Inicialmente sólo se contempló como único idioma el castellano para la obtención de datos por mail y SMS. El preparar la aplicación para dar soporte a contenido multilingüe para la creación de eventos, implicó una gran remodelación del código fuente del generador.

Finalmente, el principal requisito que ha dado lugar a la aplicación llamada Generador de Eventos ha sido la extracción del módulo de generación de la Agenda Social. Envolviendo el módulo para su ejecución con una interfaz gráfica y dotándolo de una base de datos independiente a la Agenda Social.

Tener un conocimiento básico del lenguaje de programación Java es un requisito no obligado pero necesario. Los tutoriales y ayudas que se puede encontrar en internet son muy abundantes, se han utilizado los tutoriales de la página oficial de Java ([5]) y algún libro de nivel avanzado como Core Java 2 ([A]). También es un recurso indispensable la consulta y estudio de la API SE de Java ([6]) y todas aquellas APIs que recojan la información necesaria para el desarrollo del proyecto.

Los siguientes sub-apartados describen el diseño conceptual de cada uno de los bloques que conforman el Generador de Eventos.

[13] Parlay X se detalla en el punto 3.8.1.2.

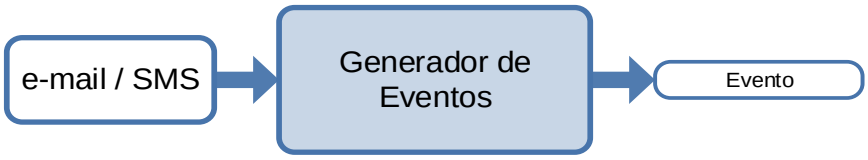
2.2.1. El Generador de Eventos

El Generador de Eventos es la figura abstracta que representa la acción de crear un evento de forma automática. Conceptualmente es el módulo principal encargado de llevar a cabo la generación de eventos a partir de los mensajes de texto y correos electrónicos que envían los usuarios a la aplicación. Ocasionalmente nos referiremos a los mensajes y correos enviados como contenidos.

El generador debe encargase de subministrar al Analizador Sintáctico un mensaje y esperar a que se pueda crear un evento válido a partir del contenido enviado. El evento deberá guardarse en la base de datos de la aplicación una vez creado. Adicionalmente se notificará al usuario que su evento ha sido creado satisfactoriamente, mediante SMS o e-mail según corresponda.

El generador ha sido diseñado implementando el concepto de única entidad procesadora. Existe un único Generador de Eventos que es invocado cada vez que llega un nuevo mensaje, una vez analizado el contenido se continúa con el siguiente.

Figura 2.1: El Generador de Eventos



2.2.2. Concepto de evento

En el contexto de la Agenda Social, los eventos son los contenidos que introducen sus usuarios. Un evento tendrá entonces un usuario creador, estableciéndose una relación bidireccional usuario - evento. Todos los usuarios registrados en la aplicación pueden crear eventos sin limitación de número. Toda información para crear un evento automáticamente que no proceda de un usuario registrado deberá ser ignorada. La información que contiene un evento se compone de diferentes campos:

Tabla 2.1: Campos de un evento

Nombre	Descripción breve
Usuario	Identificador de usuario que ha creado el evento.
Título	Título del evento.
Descripción	Contenido textual que describe el evento.
Fecha	Día relacionado con el evento.
Hora	Hora a la que tiene lugar el evento.
Dirección	Dirección donde ocurre el evento.
Ciudad	Ciudad o población donde ocurre el evento.
Código postal	Código postal de la ciudad.
País	País de la ciudad.
Categoría	Grupo en el que encaja el evento.
Privacidad	Marca el grado de privacidad del evento.

Aparte de los campos vistos en la tabla anterior, un evento contiene algunos campos más, necesarios para otras funcionalidades de la Agenda Social o para la gestión de la aplicación. También se ofrece la posibilidad de almacenar contenido multimedia: imágenes, documentos, etc. Este contenido no será tratado como un campo a buscar para la creación del evento, sino como un contenido adicional, puesto que no está en el texto del mensaje, sino incrustado en su medio de envío; como en el caso de un correo, como un dato adjunto.

Debe aclararse en este punto que para la creación de un evento no deben estar cumplimentados todos los campos. Pongamos por ejemplo que el usuario quiere registrar un evento de Categoría "Viaje", es muy probable que en este caso no tenga sentido cumplimentar la calle de destino o, dependiendo del trayecto, la hora a la que va a salir o va a llegar. De todos los campos mencionados, deberán estar cumplimentados siempre: el usuario creador, la fecha del evento y su título (y también un identificador exclusivo de evento). En caso contrario, no podrá crearse el evento por no disponer de la información mínima necesaria.

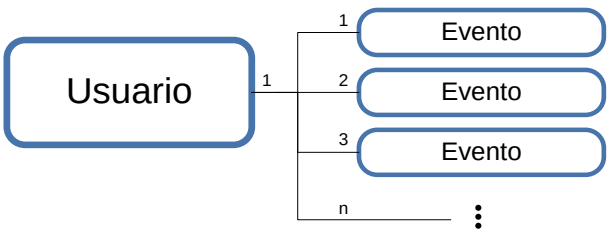
Observamos que la información que se recoge en un evento permite representar la idea de un *acontecimiento*. Aquello que el usuario plasma en su agenda web personal y que además puede compartir con los demás usuarios.

A continuación se detallan los campos por grupos:

2.2.2.1. Usuario

El usuario es el campo con más relevancia, a través de este campo se establece una relación directa entre el listado de usuarios y el listado de eventos. Como se observa en la Figura 2.2 un usuario puede crear tantos eventos como desee.

Figura 2.2: Relación usuario - evento



2.2.2.2. Categoría

En el punto 2.2.2 ha aparecido en su ejemplo la creación de un evento de categoría "Viaje". En un principio este campo no estaba acotado y permitía la introducción de cualquier descriptivo. Para facilitar la agrupación y la elección de una categoría, se ha decidido concretar y reducir el listado a un grupo fijo. De este listado es del que elegiría la categoría más apropiada el usuario o el Analizador Sintáctico. Las categorías a las que puede pertenecer un evento se muestran a continuación y su propio nombre es indicador del grupo de eventos al que representa. Si no encaja el evento en ninguna de las categorías se asignará la categoría por defecto "Otra".

Figura 2.3: Categorías de un evento

2.2.2.3. Título y descripción

Estos dos campos son los encargados de contener la mayor parte de la información de un evento. Por un lado, el título debe resumir de manera concisa sobre qué trata el evento. Por otro lado, toda información complementaria y de detalle deberá estar en su descripción.

2.2.2.4. Privacidad

La privacidad establecida para un cierto contenido indica si es accesible a otros usuarios y sus restricciones de visibilidad. La comunidad virtual define tres niveles de privacidad para un evento: público, registrado o privado. Un evento declarado como público es accesible por todos los usuarios (incluidos los no registrados en la aplicación). Solo los usuarios registrados podrán tener acceso al contenido del evento. La privacidad máxima se obtiene declarando un evento privado, en este caso es únicamente visible por el propio usuario que lo ha creado. La privacidad es un campo que tiene sentido únicamente en el escenario de generación de eventos para comunidades virtuales.

2.2.2.5. Dirección, ciudad, código postal y país

Estos campos detallan el lugar relacionado con el evento. Los dos más importantes son la ciudad y la dirección. Muchos de los acontecimientos quizá solo ocurran a nivel de ciudad o población, mientras que si tienen lugar en una dirección específica deberá indicarse en su dirección. Dentro de la dirección aglutinaremos conceptos de lugares específicos o direcciones concretas de calles.

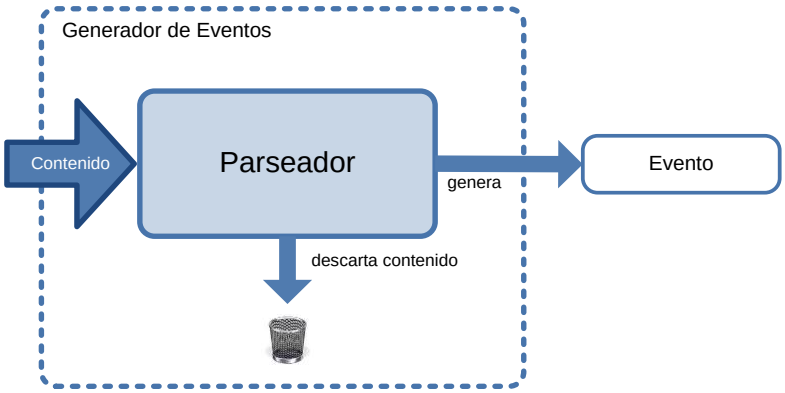
2.2.2.6. Fecha y hora

Estos campos se describen por sí solos, el único detalle a tener en cuenta es que a diferencia de los campos textuales, la fecha y la hora tienen su propio formato y las maneras de representarlas son múltiples. En este caso, los patrones de búsqueda serán más complejos.

2.2.3. El Parseador

Parseador es el nombre que recibe el Analizador Sintáctico del contenido de los mensajes recibidos. Hace todo lo posible por extraer la máxima información del mensaje para crear un evento con el mayor número de campos rellenados correctamente.

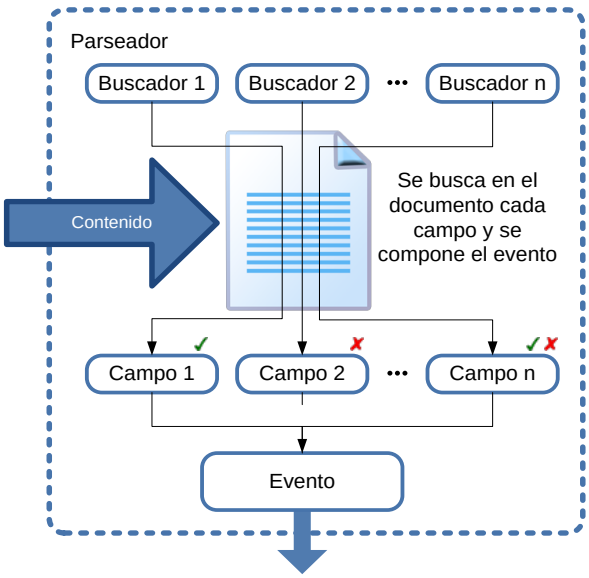
Figura 2.4: Detalle del Generador de Eventos y su Parseador



El Parseador se compone principalmente de *Buscadores* de información (ver Figura 2.5). Existe un buscador para cada campo del evento, incluido el usuario que lo ha generado. Un evento no se podrá crear si el Parseador no puede determinar quién es el propietario. La obtención del usuario está considerada como condición *sine qua non* para continuar con la obtención de los siguientes campos del evento, abortándose el proceso en caso contrario.

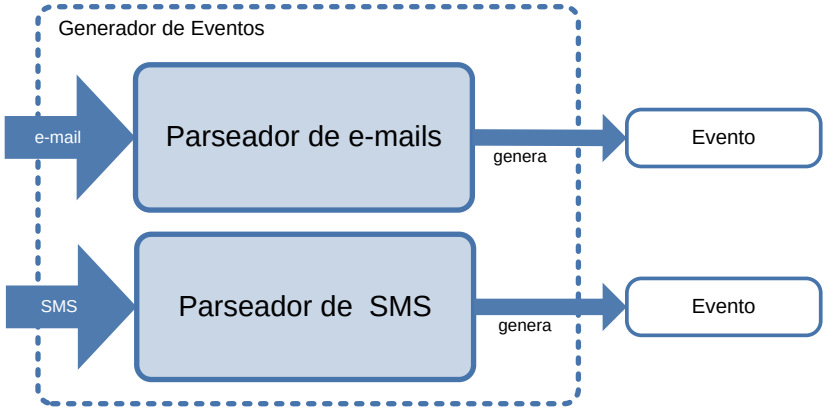
Si a un determinado evento no se le encuentra su usuario no debe considerarse una anomalía de creación de evento, es la protección básica que tiene la aplicación ante la llegada de contenido ajeno a los usuarios dados de alta en la aplicación. Todo usuario registrado tiene vinculada una cuenta de correo electrónico y un número de teléfono. Como medida de protección, solo se crearán eventos de forma automática si la información llega a través de un correo o teléfono declarado en la aplicación.

Figura 2.5: Esquema del Parseador



Aunque tienen una gran parte en común, no se busca de la misma manera en un correo electrónico que en un SMS. El correo electrónico recibido tiene asociada una cuenta de correo, mientras que el mensaje de texto un número de teléfono. En un correo se puede obtener cierta información adicional de su asunto y se pueden extraer datos de sus datos adjuntos. Esto da lugar a la idea de tener que diseñar dos Parseadores diferentes pero con una base común. El esquema de Parseador de SMS y de e-mails se muestra en la Figura 2.6.

Figura 2.6: Esquema de un doble Parseador



Se ha mencionado que los campos deben cumplimentarse *correctamente*, se introduce el problema de mayor dificultad que puede tener el diseño e implementación del Parseador: la creación incompleta o incorrecta de eventos. Consideraremos que un evento ha sido creado incompleto cuando alguno de sus campos no ha sido cumplimentado (y debería haberse hecho), e incorrecto cuando algún campo ha sido cumplimentado incorrectamente, tergiversando en este caso el evento original que hubiese querido crear el usuario manualmente.

No se debe relacionar la incorrecta creación de un evento con el malfuncionamiento del Parseador, y menos generalizarlo con un malfuncionamiento del Generador de Eventos. En este caso estaremos hablando de una limitación conocida, puesto que la no correcta cumplimentación de los campos será reducida a casos muy concretos de correos con poca estructura o sintácticamente difíciles de interpretar por reglas automáticas.

Tiene que estar presente que aunque el ojo humano sea capaz de interpretar correctamente un correo, el Generador de Eventos es un programa, y la inteligencia que tiene de análisis es reducida comparada con la capacidad de interpretación visual del ser humano, y más teniendo en cuenta que la diversidad de correos para analizar es elevadísima.

Ante la problemática presentada, se han centrado muchos de los esfuerzos de la realización del proyecto en dotar al Parseador y sus Buscadores de información de la máxima “inteligencia” posible para lograr una creación óptima de evento; potenciando en la medida de lo posible las reglas utilizadas para buscar información.

2.2.4. Buscadores y patrones de búsqueda

Los Buscadores se encargan de encontrar el campo del evento para el que han sido diseñados. Todos los Buscadores parten de la base común de utilizar una cierta lógica para hallar la información en el contenido del mensaje recibido.

Cada Buscador alberga una serie de reglas que reciben el nombre de Patrones de búsqueda o simplemente patrones basados en expresiones regulares. Estas reglas definen una estructura gramatical que debe coincidir en el texto objetivo, no con el texto completo sino con una parte de él. Los diferentes patrones se irán buscando en orden en el texto. Si no es hallado el campo se pasa al siguiente patrón. Si la estructura se encuentra en el texto, el campo es encontrado y se detiene la búsqueda dejando de aplicar los siguientes patrones.

Las expresiones regulares son el principal método de búsqueda de los Patrones y era el único método planteado en la fase inicial de diseño. Adicionalmente se tuvieron en cuenta otros métodos para cumplimentar información del evento cuando las expresiones no hubiesen encontrado el campo, por no encontrarse o no existir en el contenido. Estos métodos pueden ser búsquedas en fuentes extra de información: buscando en la base de datos de la aplicación podemos extraer algunos datos a partir de otros. Por ejemplo, se puede extraer el país de un evento si se ha encontrado su ciudad, no hace falta que el país venga implícitamente en el contenido del mensaje, si existe en la base de datos esa ciudad y tiene cumplimentado su país, podremos utilizarlo.

Obtener información a partir de la base de datos es un arma de doble filo, por un lado, cuanta más información contiene, mayor es el índice de éxito; por otro, la mala creación de un evento (con información falsa o errónea) por parte del Generador o de cualquier usuario hará que los futuros eventos automáticos también se creen erróneamente porque propagarán el error.

Los métodos que utilizarán los Buscadores quedan resumidos en:

- Patrones de búsqueda basados en expresiones regulares.
- Métodos basados en obtención de información almacenada en la base de datos.
- Cumplimentación de campos con valores por defecto.

Ante la posibilidad de ejecutar los Buscadores en paralelo o serie se decantó por la última opción. La opción de mandarlos en paralelo podría obtener algunos beneficios en tiempo de ejecución, pero la opción en serie permite que los Buscadores aprovechen los resultados de Buscadores ejecutados previamente. Por ejemplo: el país se puede obtener a partir de una ciudad, pero no al revés. Además, en caso de tratar a todos los Buscadores por igual, como se pretende, si el primer Buscador es el del usuario, si no se encuentra, se aborta el proceso y los demás buscadores no son ejecutados porque carece de sentido (el evento nunca será creado).

Contemplando esta situación tiene mucho sentido establecer un orden de búsqueda de campos y ejecutar los buscadores secuencialmente uno detrás de otro. El orden en que se ejecutarán los Buscadores es el siguiente: usuario, título, fecha, hora, dirección, ciudad, código postal, país, descripción, categoría, privacidad y finalmente el contenido multimedia.

2.2.5. Expresiones regulares

Las expresiones regulares son el método principal utilizado por los patrones para buscar los campos en un cierto texto. Una expresión regular o r*egex*[14] es una forma de describir un conjunto de cadenas de texto con características en común sin enumerar dicho conjunto. Son utilizadas sobretodo en editores de texto y aplicaciones para buscar y manipular textos. Por ejemplo: la expresión regular *c(a|o)sa* describe a las palabras o cadenas de caracteres "casa" y "cosa". De esta forma, podemos hacer referencia a dos *palabras* diferentes mediante una única expresión.

Las expresiones regulares tienen su propia sintaxis aunque ésta a menudo varía según la aplicación o lenguaje de programación que las implementa, aun así todas ellas tienen los mismos operadores, construidos a base de meta-caracteres[15].

El operador de alternación a veces denotado como una barra vertical '|' separa varias alternativas. Los operadores de cuantificación van detrás de otros caracteres e indican su frecuencia de aparición. Los más comunes suelen ser el '+', '?' ' '*'. El operador '+' indica que el carácter precedido debe aparecer al menos una vez. El operador '?' indica que el carácter precedido debe estar como mucho una vez, es decir, que esté o que no esté. El operador '*' indica que el carácter precedido puede aparecer muchas, una o ninguna vez. Los operadores de agrupación como los paréntesis '(' ')'se usan para definir un ámbito de precedencia para otros operadores. Los operadores se pueden consultar en el apéndice B.1. En las secciones 3.5 y 3.6 veremos la implementación de las expresiones regulares en el lenguaje Java y todo el conjunto de posibilidades de creación de sintaxis que ofrece.

La creación de la expresión regular debe ser un proceso meticuloso en el que cualquier fallo o alteración puede hacer que el resultado no sea el esperado, encontrando cosas no deseadas o no encontrando las que sí lo son. Cualquier pequeña variación o modificación puede hacer que el conjunto de patrones que sí funcionaban anteriormente dejen de hacerlo, no encontrando los campos del evento. Por ese motivo debe evitarse la concreción y crear reglas lo más generales posibles.

Para facilitar el mantenimiento de éstas se ha propuesto que estén todas ellas en un archivo de texto (llamado archivo de gramáticas) y sean obtenidas por los Buscadores cada vez que se desea analizar un contenido.

2.2.6. Obtención de datos externos

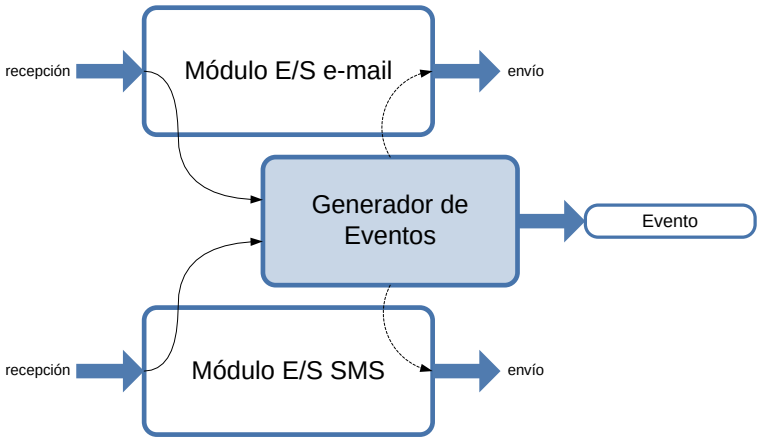
El Generador de Eventos se ha modelado como un bloque al que le llegan correos electrónicos y mensajes de texto que son transformados en eventos. Se necesitan dos entidades encargadas de:

- Recibir y enviar mensajes de correos electrónicos.
- Recibir y enviar mensajes de texto de telefonía móvil.

[14] Abreviatura del inglés "regular expression".
[15] Caracteres con un significado especial.

Como se muestra en la Figura 2.7, la atención de la entrada de nuevos mensajes pasará a ser exclusiva del módulo de generación de eventos. En cambio la salida de mensajes podrá ser utilizada también por cualquier otro bloque de la aplicación web que necesite enviar información por esos canales. Tendremos pues, dos bloques diferenciados para gestionar las comunicaciones.

Figura 2.7: Módulos de entrada y salida



2.2.7. Entrada y salida de correos electrónicos

Tanto el Generador de Eventos como la propia Agenda Social deben utilizar un servicio de correo electrónico con una cuenta de usuario para poder enviar y recibir e-mails. El servicio de correo escogido ha sido el de Gmail[16] ([7]).

Los correos reicibidos serán utilizados para generar eventos utilizando el módulo de generación. El acceso a la cuenta será mediante el protocolo IMAP[17] que permitirá crear carpetas en la cuenta de correo y acceder a los correos sin descargarlos, manipulándolos en el propio servidor de correo. Para el desarrollo del proyecto los correos se han mantenido en la cuenta, pero deberán ser eliminados tras el análisis para cumplir la política de protección de datos de los usuarios en una aplicación en producción y con usuarios reales. Los correos enviados serán de tipo notificación.

El proceso de recepción de un correo será el siguiente:

- La aplicación se conecta a su cuenta de correo y mira si tiene mensajes nuevos.
- Lee uno a uno los nuevos correos y los analiza.
- Por cada correo leído se genera un evento si se dan las condiciones necesarias.

El módulo de recepción de entrada de correos no estará continuamente mirando si hay nuevos correos; puntualmente cada cierto intervalo de tiempo abrirá sesión en su cuenta y mirará si hay algo nuevo, que en caso afirmativo se procederá a analizar.

[16] Cuenta de correo de Google, conocido también como Google Mail
[17] Internet Message Access Protocol (Protocolo de acceso a mensajes por Internet)

El principal problema que podemos encontrar es que en el intervalo de conexiones a la cuenta de correo no nos haya dado tiempo a analizar todos los correos del análisis anterior, volviendo a leer correos que todavía no han sido procesados. En este caso se produciría duplicación de eventos creados. Para evitar dicha problemática se ha pensado en una estructura de carpetas para diferenciar los correos nuevos de los anteriores.

Cuando iniciamos sesión todos los mensajes nuevos de la bandeja de entrada son movidos a la carpeta *PARSING* (parseándose) y se guarda una referencia a cada uno de estos correos. Una vez analizado un correo, si su evento ha sido analizado correctamente se mueve a la carpeta *PARSED* (parseado) en caso contrario a *UNPARSED* (no parseado). Si se crea una nueva conexión (para analizar el siguiente bloque) nos encontramos que los mensajes de la iteración anterior ya no están en la bandeja de entrada y no se volverán a leer. Aunque coexistan en la carpeta *PARSING* los correos de varias iteraciones, cada iteración tiene referenciados sus propios correos y no todos los de la carpeta.

2.2.7.1. El formato del correo electrónico

En la fase inicial de la aplicación y análisis de los primeros correos, se utilizaba el método semiautomático y su contenido estaba en formato "texto plano". Cada línea contenía un campo y se determinaba qué campo del evento era en función de una palabra clave seguida de un delimitador con el contenido después, como p. ej.: titulo=Título del evento. El formato texto plano está compuesto por texto sin formato, solo caracteres. No admite fuentes en negrita, cursiva, colores o imágenes incrustadas, etc. Los e-mails que vemos con colores y estructuras diferentes a un simple párrafo tienen un formato enriquecido.

Las aplicaciones encargadas de enviar mensajes y leerlos tienen que saber hablar el mismo *idioma* para poder interpretar y mostrar correctamente los correos. El formato de texto enriquecido más utilizado en los correos lo ofrece HTML[18]. HTML es un lenguaje con marcas, utilizado sobre todo en las páginas web y describe su estructura y su contenido. La especificación de si el formato del correo es HTML está en su tipo MIME[19], y es *text/html*.

MIME es un estándar que clasifica recursos y provee información a programas acerca de cómo manejarlos. Esto permite a las aplicaciones tratar los contenidos como lo que realmente son. En este caso *text/html* deber interpretarse como texto en formato HTML. El tipo MIME diferencia todos los tipos de archivos, imágenes, video, texto, etc. para que puedan ser interpretados correctamente ([8]). Los correos sin formato tienen el tipo MIME *text/plain*. Aparte del formato HTML existen otros formatos, por ejemplo Outlook[20] tiene su propio formato de texto enriquecido RTF[21]. Cuando los correos en ese formato son enviados a Internet se traducen a formato HTML.

La dificultad que añaden los formatos de texto enriquecido es que su contenido no se parece mucho a lo que sería su aspecto visual. Con solo clicar con el botón derecho en cualquier página web de un navegador y observar el código fuente de la página, se aprecia que lo que era una página web sencilla y comprensible, detrás tiene un conjunto de caracteres y símbolos difíciles de vincular directamente a lo visual. Esta complicación de interpretación del contenido se transmite en la necesidad de realizar una transformación del texto de formato

[18] HyperText Markup Language: Lenguaje de Marcado de Hipertexto.
[19] Multipurpose Internet Mail Extensions: Extensiones multipropósito de correo de internet.
[20] Outlook es el cliente de correo de Microsoft, incluido en su paquete ofimático.
[21] Rich Text Format: formato de texto enriquecido.

HTML a texto plano. Todo correo que llegue en formato HTML será sometido a un proceso de limpieza que traducirá sus etiquetas o las eliminará para dejar el contenido con una estructura sintáctica más sencilla. Simplificando el contenido, las expresiones regulares pueden realizar su función más fácilmente. El Parseador solo trabajará con expresiones para texto plano.

2.2.8. Entrada y salida de mensajería móvil

Análogamente al módulo de entrada y salida de mensajería electrónica, se implementará un módulo que se encargue de recibir y enviar SMS. Normalmente, para que una aplicación pueda gestionar comunicaciones de tipo SMS debe conectarse mediante algún protocolo (implementado en alguna API) a un módem GSM[22]. La Agenda Social, en cambio, utiliza la pasarela Parlay X de Orange: *Incomit Application Hub*.

Parlay X es un estándar de APIs Web Service para las redes de telefonía ([9]). Orange (Empresas) ofrece dicha herramienta para el desarrollo de aplicaciones SMS de terceros. La obtención o envío de información a través de la plataforma se realiza a través de Web Services. Parlay X ofrece una interacción entre servidores y aplicaciones a través de una red, que permite que dos aplicaciones se comuniquen remotamente mediante peticiones.

Estas llamadas se realizan mediante HTTP[23] a través de SOAP[24] mediante ficheros WSDL[25]. SOAP es un protocolo de los "servicios Web"[26] que permite la comunicación entre aplicaciones a través de mensajes por medio de Internet. Los ficheros WSDL están escritos mediante el metalenguaje de marcas XML[27]. Se establece así una conexión entre la Agenda Social y el servidor de Orange por el cual se pueden enviar y recibir mensajes. Esta comunicación tiene asociado un número de marcación corta para los SMS en el servidor de Orange.

Las tecnologías mencionadas anteriormente (HTTP, SOAP, WSDL y XML) son completamente transparentes a la implementación de la aplicación. Orange ofrece una API con una serie de clases y métodos encargados de establecer la conexión con su servidor, ofreciendo unos pocos métodos simples para que los desarrolladores puedan integrar el envío y recepción de SMS fácilmente en sus aplicaciones. Sí que hay que remarcar que dicha API solo permite la gestión de SMS bajo el operador Orange y su servicio de numeración corta asignada a aplicaciones de terceros.

A la hora de crear un evento, en el caso de los SMS solo tiene sentido la generación semiautomática de eventos. Esto se debe a que es el propio usuario el que redacta y envía el SMS y este envío no se hace de forma automática. Para un envío automático de SMS, el terminal móvil debería tener una aplicación que realizase esa tarea. Además, sin una tarifa plana de SMS podría suponer un coste muy elevado para el usuario.

[22] Estándar que define un sistema global para las comunicaciones móviles.
[23] Hypertext Transfer Protocol: Protocolo de Transferencia de Hipertexto.
[24] Simple Object Access Protocol: Protocolo de Acceso de Objetos Simples.
[25] Web Services Description Language: Lenguaje de descripción de los servicios Web.
[26] Del inglés *Web service.*
[27] eXtensible Markup Language: Lenguaje de marcas extensible.

2.2.9. Generación automática y semiautomática

Los términos de generación automática y semiautomática hacen referencia al tipo de creación del evento.

- Nos referimos a la creación de un evento mediante una generación semiautomática cuando es el propio usuario el que con su correo electrónico o su terminal móvil envía un mensaje para convertirlo en un evento. Dichos mensajes deben tener una cierta estructura o plantilla, marcando mediante "etiquetas" a qué campo pertenece el texto escrito a continuación. Un ejemplo sencillo sería el siguiente correo:

  Título: Quedada de compañeros del club de bolos
  Día: 21/8
  Hora: 18:00
  Lugar: Club de bolos Galaxy

- La generación automática hace referencia a cuando se crea un mensaje ajenamente a la interacción del usuario o no utiliza una plantilla. Este método se da cuando el usuario ha creado algún filtro de reenvío, reenviado él mismo algún correo, o lo ha redactado sin seguir unas reglas preestablecidas, es decir, cuando el usuario escribe un correo pero sin utilizar marcas o etiquetas que indiquen los campos.

La diferenciación entre generación automática y semiautomática no ha tenido mucho impacto en el desarrollo; el diseño del Generador de Eventos se ha centrado en la generación automática. A efectos prácticos, podemos tratar la generación semiautomática como un caso particular y fácil de tratar. En este caso, se aplicarán unos patrones de búsqueda particularizados para las etiquetas de identificación del campo. Como caso más simple, la creación de un evento de manera semiautomática no debería provocar ninguna pérdida de información o malinterpretación del contenido original del mensaje, a no ser que fuese el propio usuario el que cometiese un error a la hora de enviar la información.

La mayor complicación reside en el análisis de los correos para generar su evento asociado automáticamente. Como hemos visto hasta ahora, al ser la mayoría enviados en formato de texto enriquecido tendrán estructuras muy diferentes. El redactado de los usuarios no está preparado para ser interpretado por una aplicación (el Generador de Eventos) sino para que sea comprensible por otro ser humano. Será por lo tanto un proceso *best effort*[28], en el que a partir de varios ejemplos de prueba se añada un elveado número de patrones para encontrar el mayor número de campos. La creación de los patrones no nos dará garantías para que todos los futuros correos sean interpretados correctamente, pero sí los que conceptualmente debieran poder interpretarse y cumplan el juego de patrones implementado.

[28] Ofrecer el mejor esfuerzo.

2.2.10. Aplicación multilingüe

Desde el comienzo del trabajo, en la Agenda Social no parecía haber un idioma concreto. Los desarrolladores del equipo de trabajo que creaban la Agenda Social generaron el código –según su criterio- en varios idiomas: catalán, castellano o inglés. La parte visual de la aplicación estaba en inglés, mientras que los eventos creados en ella para pruebas eran en su totalidad en catalán y castellano. El Generador de Eventos no tuvo como requisito el tratar contenidos multilingües, su tarea era sencilla: analizar correos o SMS y crear eventos a partir de ellos. Durante la mayor parte del desarrollo, la totalidad de los correos de prueba eran en castellano, partiendo de esa base, la estructura del Generador de Eventos se diseñó para aplicar patrones exclusivamente en ese idioma.

Uno de los últimos requisitos de la Agenda Social (ya en fase final de desarrollo) determinó el inglés como idioma final para la aplicación y esto implicó la realización de una serie de adaptaciones para enfocar la generación de eventos a una comunidad de habla inglesa. Adicionalmente se incorporó un correo en inglés a la batería de correos de prueba en castellano que ya se poseían. Tener correos en varios idiomas no debería haber supuesto ningún problema, los eventos se hubiesen generado según el idioma del contenido recibido y mostrado en la página con su interfaz web en inglés. El problema reside en cómo analizar el contenido según su idioma.

Se observa que no se pueden usar las mismas reglas para buscar campos de un evento si el idioma cambia y tampoco tiene sentido utilizar etiquetas de búsqueda en idiomas no nativos al usuario. Esta parte de la aplicación forzó un re-análisis y una nueva implementación de algunas partes del Generador de Eventos. Esta serie de cambios incluían la traducción de los patrones y expresiones regulares a los idiomas a implementar y un rediseño de los bloques para trabajar con una nueva variable: el idioma.

Para poder aplicar las gramáticas y expresiones regulares para el idioma correcto hay que determinarlo primero a partir del contenido. Para determinar el idioma se ha creado un módulo *Diccionario* que mediante el almacenamiento de palabras permite hallar el idioma correspondiente de un texto.

Las palabras del diccionario se pueden guardar de distintas formas, sin profundizar en los métodos de averiguación de idiomas de documentos de las aplicaciones profesionales, se ha optado por implementar un método propio. Las opciones que se barajaron son las siguientes:

- En un fichero de texto con las palabras guardadas secuencialmente.
- En una base de datos. Las palabras se buscarían en el diccionario mediante consultas.
- Almacenando las palabras en memoria jerárquicamente en forma de árbol (conjunto de nodos conectados). El método que malgasta más recursos de memoria pero el más eficiente en tiempo para consultar las palabras.

El almacenamiento en memoria en forma de árbol ha sido el elegido, principalmente por la motivación de implementar la solución de esta manera.

2.3. Definición del flujo de ejecución e información

Llegado a este punto se tiene una idea global de todo lo que debe hacer el Generador de Eventos. El proceso de creación de un evento queda reducido a los siguientes puntos:

1) Un cierto mensaje llega a alguno de los servidores de recepción de datos, ya sea un correo electrónico o un SMS.
2) Se entrega el mensaje al Generador de Eventos para que cree un evento a partir de dicho contenido.
3) El Generador de Eventos crea un Parseador según sea el origen del contenido, SMS o e-mail.
4) El Parseador recibe el contenido y crea un evento vacío no asignado a ningún usuario.
5) El Parseador determina el idioma del contenido para configurar los Buscadores.
6) El Parseador crea tantos Buscadores como campos del evento se desean cumplimentar.
7) Los Buscadores se ejecutan en serie uno detrás de otro, cuando finaliza uno comienza el siguiente.
8) Los Buscadores emplean patrones de búsqueda basados en expresiones regulares o métodos alternativos para hallar los campos del evento en el contenido.
9) Una vez cumplimentado el evento con los campos encontrados se devuelve al Generador de Eventos.
10) El Generador de Eventos guarda el evento en base de datos si el evento es correcto. Un evento no se considerará correcto si no existe el usuario creador o ha habido algún problema no controlado en el proceso de búsqueda.
11) Se notifica al usuario mediante SMS o e-mail, según corresponda, que su evento ha sido creado correctamente.

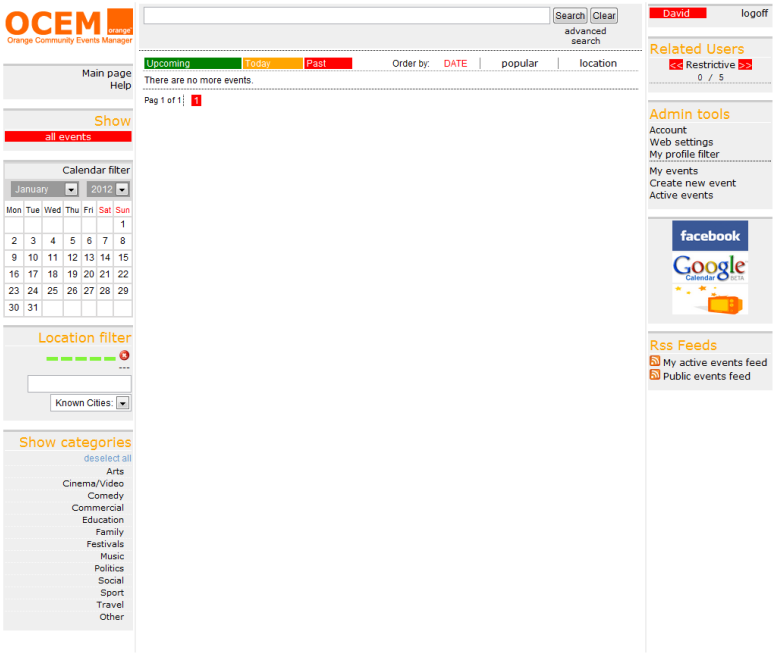
2.4. La Agenda Social

La Agenda Social es una aplicación web 2.0 orientada a comunidades virtuales desarrollada bajo la plataforma J2EE de Java. Es un prototipo creado por el departamento de telemática de la ETSETB para la empresa Orange con nombre final de producto OCEM[29]. El desarrollo de la Agenda Social no forma parte del presente proyecto, pero sí la elaboración de algunos de sus módulos y funcionalidades. Tanto su modelo de base de datos, lógica e interfaz web ya estaban implementados en el inicio del presente trabajo, y fueron evolucionando y siendo mejorados por los miembros del departamento. Como el Generador de Eventos ha estado inicialmente vinculado a la Agenda Social es necesaria una comprensión de su estructura y funcionalidad heredada, como son los campos de un evento, la utilización del modelo de aplicación y la gestión de entrada y salida de mensajería (SMS o correos electrónicos).

La comunidad de usuarios creada gira en torno al concepto de evento, un acontecimiento que el usuario registra para compartirlo con los demás miembros de la comunidad. El método tradicional de creación de un evento es a través de la interfaz web. El usuario también puede ver los eventos de los usuarios si éstos tienen la privacidad adecuada. Como funcionalidad extra, se ofrece al usuario la posibilidad de poder crear nuevos eventos mediante mensajes de texto SMS, vía móvil o correos electrónicos, enviados desde su cuenta de correo.

La aplicación web también permite a los usuarios ser notificados ante cambios en los eventos que cumplen unos filtros con ciertos parámetros definidos, conocidos como conjuntos de búsquedas de interés. Estas notificaciones serán enviadas vía SMS, e-mail o canal RSS[30] con la información asociada. Otra característica es la incorporación de un videoclub que gestiona el alquiler de películas. Mediante la creación automática del Generador de Eventos se crean eventos utilizando los e-mails de información de alquileres enviados por la aplicación.

Figura 2.8: Agenda Social – página principal



[29] Orange Comunity Events Manager.

[30] Really Simple Syndication. Fichero XML para compartir contenido en la web.

2.4.1. Entorno de desarrollo

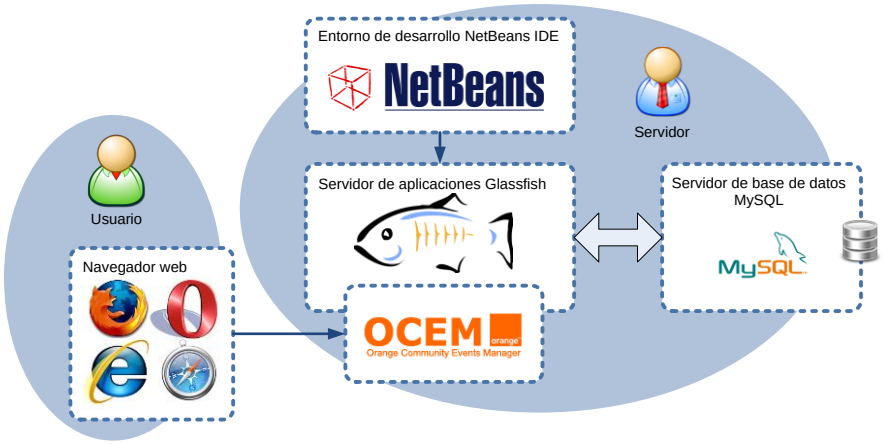
La Agenda Social ha sido desarrollada como proyecto web J2EE bajo el entorno de desarrollo NetBeans IDE[31] ([10]). Para el desarrollo de aplicaciones debe estar instalado en la máquina para trabajar el JDK[32].

El entorno de desarrollo NetBeans es libre y gratuito y facilita enormemente el desarrollo de cualquier tipo de proyecto, sobre todo los de gran envergadura, ya que incluye módulos preestablecidos para el desarrollo que simplifican enormemente la programación de algunas funcionalidades. Netbeans ofrece las herramientas de compilación, depuración y ejecución de las aplicaciones desarrolladas. Las aplicaciones de Java se ejecutan directamente en su JVM[33]. En cambio, las aplicaciones web deben arrancarse en un servidor de aplicaciones web.

Se ha utilizado para la Agenda Social el servidor Glassfish ([11]) que permite establecer una persistencia de datos entre la aplicación web y la base de datos. Netbeans se presenta como la opción más conveniente ya que integra en el propio entorno de desarrollo el servidor Glassfish. Se ofrece desde Netbeans de forma sencilla el control del servidor web: arranque, parada y carga de aplicaciones web. Los servidores web arrancan sus aplicaciones con su propio contexto, estableciéndose una ruta o *url*[34] única para cada aplicación cargada. Para ver las aplicaciones del servidor necesitaremos poner la url en un navegador web.

Los datos de la aplicación son guardados en una base de datos externa. Se ha elegido como sistema de gestión de base de datos MySQL ([12]). MySQL es una base de datos relacional, multiusuario y multihilo. Para un fácil manejo de la base de datos se ha utilizado el MySQL Workbench, que ofrece una interfaz gráfica en vez de la línea de comandos de MySQL. Una representación del entorno la podemos observar en la Figura 2.9.

Figura 2.9: Entorno de la Agenda Social



[31] Integrated Development Environment: Entorno de desarrollo integrado.
[32] Java SE Development Kit: Software de Java que permiten el desarrollo de aplicaciones.
[33] Java Virtual Machine: Máquina Virtual de Java.
[34] Uniform Resource Locator: Localizador Uniforme de Recursos.

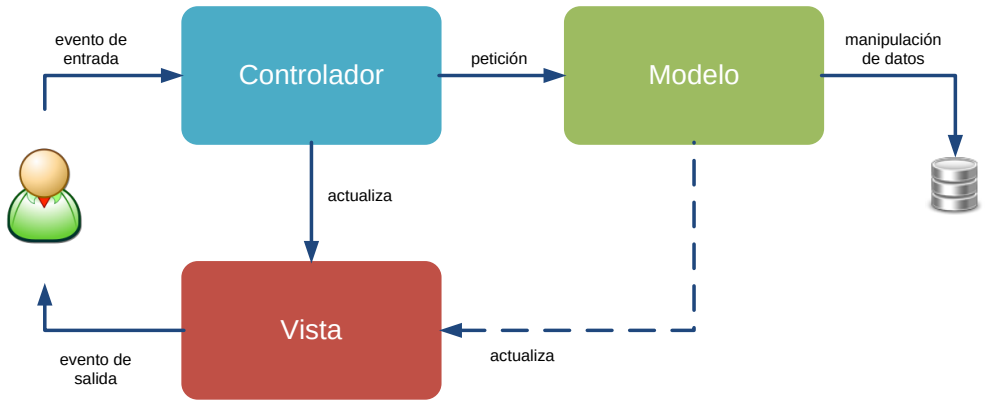
2.4.2. Patrón MVC

La aplicación web está desarrollada siguiendo uno de los patrones de arquitectura de software más utilizados actualmente: el patrón MVC[35]. Su objetivo es separar los datos de la aplicación, la interfaz gráfica y la lógica de control en tres componentes distintos y diferenciados. Este patrón se suele aplicar en todas aquellas aplicaciones o módulos que requieran la intervención de un usuario; por ejemplo, MVC es el patrón utilizado también para gestionar internamente las librerías gráficas de J2SE. Sus partes se definen como:

- Modelo: es el sistema de gestión de base de datos y la lógica de negocio[36]. Se encarga de establecer comunicación con la base de datos y establecer la persistencia de datos.
- Vista: es la parte encargada de la interfaz gráfica con la que interactuará el usuario. Se encarga de obtener datos con los que generará la vista que será mostrada al usuario después de la petición que éste realizó.
- Controlador: gestiona las entradas y las salidas de la aplicación generadas desde la vista. Aplica el tratamiento adecuado a la petición del usuario invocando peticiones al modelo.

La Figura 2.10 representa la interactuación que se establece entre las tres partes.

Figura 2.10: Patrón MVC



El flujo de operación del patrón MVC sería el siguiente:

- El usuario realiza alguna acción sobre la interfaz gráfica de la aplicación: mediante una interactuación pulsando un enlace, un botón, cambios en un campo de texto, etc. Se genera así una petición que es enviada al servidor.
- El servidor recibe la petición y procede a tratarla. El controlador es el encargado de llevar a cabo dicha tarea, gestiona el evento que ha llegado y realiza las operaciones oportunas.
- El controlador accede al modelo, realizando las modificaciones necesarias según la opción solicitada por el usuario.

[35] Model View Controller: Modelo Vista Controlador.
[36] En el contexto de la programación orientada a objetos se refiere a la funcionalidad del sistema, aquella que se ocupa de procesar los datos que envía el usuario y genera la posterior entrega del resultado.

- El controlador llama a la vista para que genere la nueva representación gráfica que le será entregada al usuario. Los datos con los que se generará la vista provienen del modelo, aunque éste no es consciente de que es la vista quien genera la consulta de datos. Según la implementación, la obtención de datos en muchos casos se hace a través del controlador.
- La nueva vista generada es enviada al usuario y la aplicación estará a la espera de nuevas peticiones.

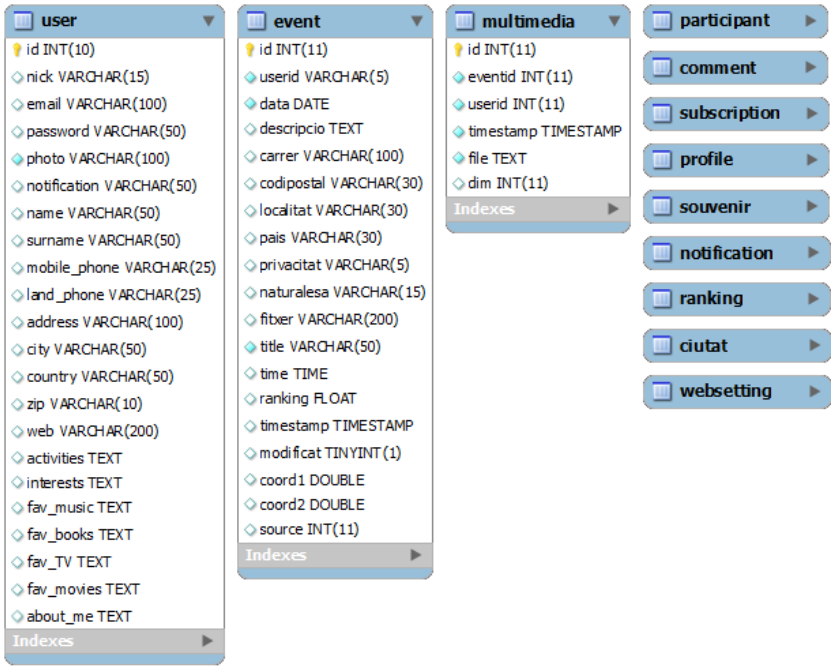
2.4.3. Base de datos

Se ha mencionado que MySQL ofrecía un sistema de base de datos relacional. En este modelo, todos los datos son almacenados manteniendo una relación ente ellos. Cada relación establece un conjunto de datos en el que el orden carece de relevancia.

El esquema de la Agenda Social está formado por un conjunto de tablas de las cuales solo unas pocas son necesarias a la hora de interaccionar con el Generador de Eventos. Sobre esas tablas se manipula la información almacenada, ya sea creando consultas de búsqueda de información o insertando nuevos eventos.

El modelo de la aplicación se ocupa de gestionar estas interacciones con la base de datos. Éste ha sido implementado siguiendo las pautas marcadas por la Java Persistence API[37] ([13]). En la Figura 2.11 se muestran los campos de las tres tablas que serán necesarias para la correcta creación de un evento mediante el Generador de Eventos. A la derecha de la figura se muestran las demás tablas que forman parte de la aplicación.

Figura 2.11: Tablas y campos de la Agenda Social



[37] API de Java para implementar la persistencia en Modelo de aplicación.

2.4.4. Tipos de generación de eventos

A continuación se detallarán cómo puede crear un usuario sus eventos en la Agenda Social. Algunos ejemplos de generaciones automáticas y semiautomáticas se encuentran detallados en el Capítulo 4.

2.4.4.1. Generación vía interfaz web

La creación vía web es el método por defecto utilizado por los usuarios. Para ello deben iniciar sesión en la aplicación web mediante su correo y contraseña. Es necesario registrarse previamente en la aplicación. Una vez iniciada la sesión pueden crearse eventos manualmente cumplimentando un formulario web.

2.4.4.2. Generación semiautomática y plantillas

La generación semiautomática se realiza mediante el envío manual de un SMS o un correo electrónico a la Agenda Social. La información enviada tiene que respetar una estructura o plantilla con los campos del evento. El uso de estas plantillas garantiza que el evento sea creado correctamente. Si se realizan modificaciones sobre la estructura el correo no será descartado puesto que no se verifica el uso de la plantilla, en ese caso los patrones de púsqueda que se aplicarán para encontrar los campos serán más generales, es decir, los utilizados para un método automático de creación del evento.

La plantilla se caracteriza por tener una etiqueta o identificador de campo y determinar un método para indicar el inicio y final del contenido. Para ofrecer más versatilidad se han definido varias etiquetas por campo y varios separadores. Esto se ha hecho así simplemente por ofrecer una facilidad a la hora de intentar escribir el correo sin tener presente que se debe escribir la etiqueta exactamente para que el evento sea creado correctamente. En el apéndice C se detalla la estructura de las plantillas según su tipología, SMS o e-mail y su idioma, describiéndose cómo deben ser utilizadas.

2.4.4.3. Generación automática

La generación automática solo es posible mediante el uso del correo electrónico del usuario. Cualquier mensaje reenviado por el usuario a la Agenda Social será analizado utilizando los patrones de búsqueda y se creará un evento. También entran dentro del ámbito de correos de generación automática aquellos que envía el usuario directamente, o que redacta él sin utilizar ninguna plantilla. Para reenviar correos automáticamente es necesaria la creación de un filtro de reenvío a la dirección de correo de la Agenda Social en la cuenta de correo del usuario. Por ejemplo, se pueden reenviar todos los correos que contengan la palabra “Viaje” en su asunto.

Una vez finalizado el proecso de creación (con generación semiautomática o automática) el usuario recibirá un correo de confirmación con los datos que han sido hallados y cumplimentados en el nuevo evento.

2.5. La aplicación Generador de Eventos

La aplicación gráfica independiente generada para ejecutar el Generador de Eventos recibe su mismo nombre y es utilizada para gestionar la ejecución del módulo de generación de eventos, controlando su inicio y parada. También añade la funcionalidad extra de permitir un mayor seguimiento de los logs[38] para su estudio y contenido de la base de datos. Hereda toda la lógica de funcionalidad de la Agenda Social, así como la interacción con el modelo de datos.

La principal motivación de la extracción de la generación de eventos de la Agenda Social ha sido el poder avanzar en el desarrollo y mejorar algunas funcionalidades del generador. Las principales ventajas obtenidas en comenzar un desarrollo alternativo han sido:

- Poder utilizar una versión más actual del JDK de java, el cual permite optimizar fragmentos de código y añade nuevas funcionalidades dentro del lenguaje Java.
- Poder crear una aplicación gráfica para seguir visualmente el comportamiento de la ejecución y el resultado de la generación de eventos.
- Crear una base de datos independiente y estructura de datos mínima y suficiente para la generación de eventos.
- Tener un mayor control sobre la evolución y mejora de las diferentes partes del generador de eventos sin tener que depender del proyecto web.
- Poder implementar un método alternativo de ejecución para pruebas no dependiente de base de datos. Es decir, permitir el análisis de correos y creación de eventos sin tener una base de datos.
- Elección del español como idioma por defecto para la creación de eventos, pero soportando igualmente el inglés.
- Profundizar conocimientos de generación de interfaces gráficas en Java.

Es importante mencionar que aunque se hacen referencias de diseño a la Agenda Social y se han comentado sus características y se comentará parte de su diseño en el siguiente capítulo, las pruebas realizadas con ella y su configuración no forma parte de la implementación final del Generador de Eventos.

2.5.1. Entorno de desarrollo

Para el desarrollo de la aplicación se han utilizado versiones portables de diferentes softwares utilizados para la Agenda Social, solo se ha mantenido el uso del servidor MySQL. Para el desarrollo del código se ha optado por una versión más reciente de Java, la versión 8, que ofrece muchas mejoras que no ofrecía la versión 6, desde la optimización de código en sintaxis a la utilización de expresiones lambda[39]. Se han utilizado versiones portables tanto del JDK como del IDE Eclipse ([14]), versión Luna con las utilidades para JavaFX e(fx)clipse ([15]), y el editor de interfaces gráficas Java Scene Builder (
[16]).

[38] Registros de ejecución del funcionamiento de una aplicación, para su visualización y análisis.
[39] Las expresiones lambda permiten reducir gran cantidad de código redundante en pocas líneas.

Aunque el servidor MySQL de base de datos se ha mantenido, se ha creado un nuevo esquema llamado “ge” para ser usado por el Generador de Eventos. Este modelo de datos es similar al de la Agenda Social pero adaptado a la nueva aplicación. En la Figura 2.12 se muestra una representación del entorno:

Figura 2.12: Entorno del Generador de Eventos

2.5.2. Base de datos

A diferencia de la Agenda Social que establecía una persistencia de datos, el Generador de Eventos realiza conexiones a la base de datos cuando necesita adquirirlos o modificarlos. Para agilizar las transacciones se ha utiliza la API BoneCP que permite la creación de un pool[40] de conexiones.

Con un modelo de datos reducido y una aplicación desarrollada en única instancia, no se observa mucha mejora en rendimiento, pero si es beneficioso la utilización de una API externa para el control y mantenimiento de pocas conexiones en vez de estar creando, conectando, adquiriendo datos y destruyendo conexiones individualmente. Con aplicaciones con muchas transacciones SQL[41] si es ventajoso trabajar con un pool de conexiones a base de datos. Las tablas y campos de los que consta el Generador de Eventos se definen en la sección 3.3.

[40] Literalmente piscina pero hace referencia a un conjunto de elementos fijo creados del cual vamos extrayendo elementos y reutilizándolos.
[41] Structured Query Language: Lenguaje de Consulta Estructurada, utilizado para comunicarse con una base de datos y realizar operaciones en ella.

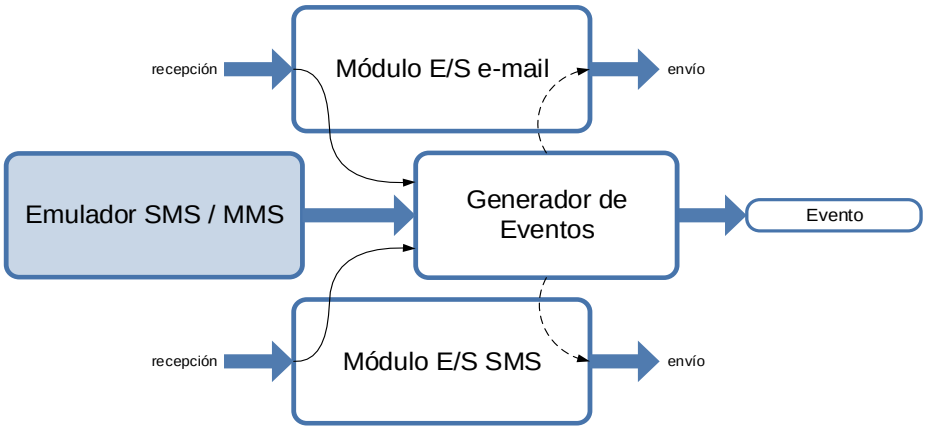
2.6. Emulador de SMS y MMS para pruebas

El Emulador se ha creado para dar soporte a la hora de realizar pruebas de generación de eventos mediante el uso del Parseador de SMS. El principal motivo de su realización es debido a que la plataforma de acceso SMS solo estuvo activa unas semanas para verificar con las pruebas la correcta llegada y envío de SMS. Una vez realizadas las pruebas de deshabilitó esta función.

Se ha implementado una aplicación independiente que emula el envío de SMS para ser analizados por el Generador de Eventos. En la interfaz gráfica deber escribirse el mensaje y especificar el número de teléfono del usuario que lo está enviando. A diferencia de la plataforma Parlay X, el emulador utiliza una conexión TCP contra un nuevo módulo de entrada y salida del Generador de Eventos.

En el Generador de Eventos se ha definido un servidor TCP para aceptar las peticiones del emulador, que hará de cliente. El estudio para la implementación está basado en comunicaciones cliente – servidor; se ha utilizado el libro Java Network Programming ([B]) para consolidar los conocimientos ya adquiridos durante la carrera. El Generador al recibir un mensaje del emulador intentará crear un evento de tipo SMS o MMS[42] utilizando el mismo método para el parseado de SMS recibidos por la plataforma Parlay X.

Figura 2.13: Módulo de Entrada de SMS / MMS



Ambos módulos (servidor y cliente) comparten una estructura de datos para entenderse mutuamente y que el servidor sepa qué le está enviando el emulador. Esta estructura de datos se compone de:

- El contenido del mensaje.
- El tipo de menaje: SMS o MMS.
- El número de teléfono del usuario creador.
- Archivos adjuntos para añadir a la creación del evento.

[42] Servicio de mensajería multimedia. Mensaje SMS con archivos multimedia adjuntos.

Capítulo 3

Diseño e implementación

Este capítulo detalla la construcción y la plasmación del análisis realizado en el capítulo precedente. La finalidad es mostrar el desarrollo, en el lenguaje de programación Java, de la solución de todos los requisitos y especificaciones que han sido definidos.

Se recogerán los aspectos más importantes del diseño por módulos y de la función de los bloques, entrando solo en detalle en aquellos aspectos de cierta relevancia, dejando todos los detalles en los comentarios presentes en el código adjunto en los anexos digitales.

Todo el código del proyecto está detallado con comentarios, por lo que para el correcto seguimiento y visión de la estructura y funcionamiento es aconsejable leer el documento teniendo el código fuente presente.

3.1. Requisitos del sistema

Para el desarrollo del proyecto se ha utilizado un ordenador personal (con al menos 1 GB de memoria RAM disponible) para la ejecución de Java y el IDE Eclipse. No se precisa de un ordenador de última generación pero si afectará a la velocidad de ejecución uno de mejores prestaciones. El espacio de disco duro recomendado es el mínimo para tener instalado el software necesario para arrancar la aplicación y poder almacenar los datos.

El software utilizado ha sido descrito en la sección 2.5.1 y las configuraciones específicas se detallan en el apéndice A.1.

Respecto al sistema operativo, el proyecto ha sido desarrollado y probado en entornos Windows de Microsoft: Windows 7 y 8.1. Todas las aplicaciones utilizadas son para máquinas con arquitectura de Sistema Operativo Windows (aunque existen sus versiones para otros sistemas operativos). Al haberse utilizado Java, un lenguaje de programación multiplataforma, no hay ningún inconveniente en utilizar las aplicaciones en un sistema Linux (u otro sistema), eso sí, descargando las aplicaciones en la página del fabricante para el correspondiente sistema operativo y realizando las configuraciones oportunas.

Para el funcionamiento del Generador de Eventos es condición indispensable el tener acceso a internet y a la cuenta de correo de Gmail mediante el protocolo IMAP. Para el funcionamiento de la recepción y envío de SMS es necesaria la conexión con el servicio Parlay X de Orange y tener dado de alta un servicio de mensajería corta en dicha empresa.

3.2. El Generador de Eventos

En el análisis del capítulo 2 se determinó que las partes más importantes que formaban el módulo eran el Parseador y el Buscador. También intervenían en el diseño los campos de un evento y las expresiones regulares o patrones de búsqueda. El conjunto de clases que conforman el Generador de Eventos se encapsulan en el paquete *eventParser.** (paquete para el parseado de eventos).

3.2.1. Estructura del paquete eventParser

Cada sub-módulo de la fase de análisis tiene asociado una clase representando ese objeto. El paquete *eventParser* define una serie de clases interconectadas conceptualmente entre ellas para definir el proceso que debe seguirse para analizar un contenido y construir paso a paso su evento. En la Tabla 3.1 se asocian los bloques y conceptos analizados con su clase correspondiente.

Tabla 3.1: Conceptos y clases del paquete eventParser

Concepto	Clase (.java)
Generador de Eventos	GeneradorEventos
Parseador	EventParser
Campos de un evento	Fields
Buscador	Finder
Conjunto de Buscadores	FinderList
Conjunto de regex de un Buscador	Grammar
Creador de regex	GrammarFactory
Lector de regex almacenadas	XMLGrammarReader

Muchas de estas clases están definidas como abstractas o de tipo interfaz, esto se debe a que, de por sí, ellas no son capaces de parsear ningún tipo de contenido. A partir de esta base se crean los parseadores específicos para cada tipo de mensaje de texto: Un Parseador para correos electrónicos en la clase `MailEventParser` y otro para SMS en la `SmsEventParser`. En la sección 3.4 se describen cada una de las particularidades de ambos parseadores.

En los siguientes sub-apartados se plasman las características más importantes de cada una de las clases del núcleo del Generador de Eventos.

3.2.1.1. La clase GeneradorEventos

La clase que inicia la creación de un evento a partir de un mensaje recibe el nombre de *GeneradorEventos*. Esta clase modela un procesador, al cual se le manda realizar la tarea de crear un evento a partir de un cierto contenido. Estructuralmente sigue el diseño propuesto por el patrón *singleton* o instancia única, por lo que únicamente existe un único procesador de eventos. Esta clase no tiene un constructor público, se crea en memoria la primera vez que es invocada, ejecutándose su constructor una única vez. El patrón de diseño de código *singleton* y muchos de los que veremos más adelante (como el *Factory* o el *Observer*) han sido estudiados utilizando el libro Java Design Pattern Essentials ([C]).

La clase `GeneradorEventos` tiene un constructor privado como se observa en el fragmento Código 3.1. Para obtener una referencia a ese objeto debe realizarse mediante su método estático `getInstance()`, una vez obtenida la referencia podremos acceder a sus métodos. Hemos de tener en cuenta que como el objeto es único, sus métodos pueden ser ejecutados en varios lugares de código de otras clases. Para mantener la integridad de los datos los métodos de acceso serán estáticos (*static)* y sincronizados (*synchronized*) evitando tener problemas de concurrencia[43].

Código 3.1: GeneradorEventos – instanciado

```
private static final GeneradorEventos instancia = new GeneradorEventos();
private GeneradorEventos() {
      dic = new Dictionaries();
}
public static GeneradorEventos getInstance() {
      return instancia;
}
```

Cuando se crea por primera y única vez crearemos un objeto `Dictionaries` que almacenará en memoria los diccionarios que utiliza la aplicación a la hora de buscar las palabras de un contenido para determinar su idioma. El proceso de creación de los diccionarios se detalla en la sección 3.7.

Para la creación de un evento se utilizan los métodos estáticos del Código 3.2, cada uno de ellos para una fuente diferente de información: un correo electrónico, un SMS o un MMS (SMS con datos adjuntos). Se ejecutará un método privado obteniendo una referencia al objeto GeneradorEventos de instancia única.

Código 3.2: GeneradorEventos – métodos de creación de un evento

```
public synchronized static void procesarSms(String message, String number) {
      GeneradorEventos.getInstance().processSms(message, number);
}
private void processSms(String message, String number) {
      ...
}
public synchronized static void procesarMail(Message message) {
      GeneradorEventos.getInstance().processMail(message);
}
private void processMail(Message message) {
      ...
}
public static void procesarMms(String content, String phone, ArrayList<File> files) {
      GeneradorEventos.getInstance().processMms(content, phone, files);
}
private void processMms(String message, String number, ArrayList<File> files) {
      ...
}
```

[43] Ejecución de código que se realiza al mismo tiempo y que pueden interactuar entre ellos, compartiendo en algunos casos los mismos recursos.

Los métodos que tienen toda la lógica de creación de un evento son: `processSms`, `processMail`, y `processMms`. Como son el principal método de creación estudiaremos con detalle el proceso de creación y guardado de un evento para el primer caso . Para los otros dos se comentarán las particularidades que tienen.

Código 3.3: GeneradorEventos – procesado de un SMS

```
private void processSms(String message, String number) {
GELogger.add(this, "PROCESS SMS");
Event event = new SmsEventParser().getEventParsed(message, number);
      if (event == null) {
            // devuelve null si no se sabe el id del usuario
GELogger.add(this, "El evento no ha podido crearse correctamente: event==null");
      } else {
            // guardamos el evento en la base de datos
            Integer idevent = EventModel.create(event);
            GELogger.add(this, "SMS recibido:\n" + message);

            if (idevent == null) {
GELogger.add(this, "Evento NO guardado en base de datos: idevent = null");
            } else {
GELogger.add(this, "Evento guardado en base de datos: idevent = " + idevent);
            }

            GELogger.add(this, "Evento creado [" + idevent + "]");
            this.printEvent(event);

            /**/// Notificación de creación
            User user = UserModel.retrieve(event.getIduser());
            if (user.getNotifemail().equals(Notification.SI)) {
                  this.sendConfirmationMail(event);
            }
            if (user.getNotifsms().equals(Notification.SI)) {
                  EnvioSMS.enviar("Event created succesfully: " + event.getTitle(),
number);
            }
      }
}
```

Código 3.4: GeneradorEventos – procesado de un e-mail

```
private void processMail(Message message) {
GELogger.add(this, "PROCESS MAIL");
       Part content = null;
      try {
            content = MailOperations.getPartToProcess(message);
      } catch (Exception e) {
GELogger.add(this, e);
      }
      try {
            if (content == null) {
                  MailOperations.moveMessageCheck("INVALID", message);
                  return;
            }
            Event event = null;
            if (!((String) content.getContent()).contains(IDMail)) {
GELogger.add(this, "Procesando Mail...");
                  event = new MailEventParser().getEventParsed(message);
            } else {
GELogger.add(this, "Mail no procesado, contiene: " + IDMail);
            }

(continúa)
```

```
            if (event == null) {
                  // devuelve null si no se sabe el id del usuario
                  MailOperations.moveMessageCheck("UNPARSED", message);
GELogger.add(this, "El evento no ha podido crearse correctamente: event==null");
            } else {
                  // Guardamos el evento en la base de datos y obtenemos u id
                  Integer idevent = EventModel.create(event);
                  if (idevent == null) {
GELogger.add(this, "Evento NO guardado en base de datos: idevent = null");
                  } else {
GELogger.add(this, "Evento guardado en base de datos: idevent = " + idevent);
      // Necesitamos que el idevent no sea nulo para crear su multimedia
      // Una vez creado el evento creamos un Finder para buscar archivos multimedia
                        MultimediaMailFinder mmf = new MultimediaMailFinder(event);
                        mmf.findFiles(message);
                  }
                  MailOperations.moveMessageCheck("PARSED", message);
                  GELogger.add(this, "Evento creado [" + idevent + "]");
                  this.printEvent(event);

                  /**/// Notificación de creación
                  User user = UserModel.retrieve(event.getIduser());
                  if (user.getNotifemail().equals(Notification.SI)) {
                        this.sendConfirmationMail(event);
                  }
                  if (user.getNotifsms().equals(Notification.SI)) {
                        EnvioSMS.enviar("Event created succesfully: " +
event.getTitle(), user.getMobilephone());
                  }
            }
      } catch (Exception e) {
            GELogger.add(this, e);
      }
}
```

La estructura de los tres métodos es muy similar y su ejecución sigue los siguientes pasos:

1) Se crea un objeto `eventParser` de tipo `Sms` o `Mail` pasando como parámetros los contenidos necesarios para la generación de un correo. En el caso de un SMS es necesario el mensaje de texto y el número de teléfono; para un correo electrónico el objeto `Message` de la API *javax.mail* que contiene toda la información del correo.

2) El Parseador nos devolverá un evento cumplimentado. Si el evento no es nulo procederemos a tratarlo. Solo se dará el caso de que el evento sea nulo si el Parseador no es capaz de encontrar el usuario creador. En el caso de estar activado el modo NO_DB_MODE el evento nunca será nulo.

3) Una vez obtenido el evento se procederá a su creación y guardado en base de datos.

4) Después de crear el evento tanto para el proceso de creación por correo y MMS se extraerán de los contenidos los archivos adjuntos multimedia. En el caso del procesado de correo mediante el método:

   ```
   MultimediaMailFinder mmf = new MultimediaMailFinder(event);
   mmf.findFiles(message);
   ```

   Y para los MMS:

   ```
   MmsFileExtractor mfe = new MmsFileExtractor(event);
   mfe.saveFiles(message, files);
   ```

   Estos objetos y métodos se detallaran en la parte de Buscadores de los Parseadores.

5) Una vez finalizado el proceso se comprueba que el usuario tiene activas las notificaciones por e-mail o SMS y se invocan los métodos de envío de mensaje de confirmación de creación.

El método de creación de un correo además tiene otras operaciones porque debe interactuar con la cuenta de correo. Si a un mensaje recibido no es posible extraerle el contenido para ser analizado será enviado a la carpeta “INVALID” de la cuenta de correo. Si el evento que se intenta crear acaba siendo nulo será movido a la carpeta “UNPARSED” y si es creado se moverá a la carpeta “PARSED”. Todas estas operaciones se realizan interactuando con las clases del paquete de envío y recepción de e-mails que veremos en el apartado 3.10. El método utilizado para realizar esta operación es:

```
MailOperations.moveMessageCheck("UNPARSED", message);
```

Donde el primer parámetro es el nombre del directorio de destino al que se va a mover el mensaje `message`.

Aparte de los métodos principales comentados, en el Código 3.5 observamos otros métodos y variables que también deben comentarse por su relevancia.

Código 3.5: GeneradorEventos – otros métodos y variables

```
public static final String IDMail = "--## Notification from GeneradorEventos ##--";

public enum DBMode {
      DB_MODE, NO_DB_MODE;
}
private static DBMode dbMode = DBMode.DB_MODE;

private void printEvent(Event event) {
      ...
}

private void sendConfirmationMail(Event event) {
      ...
}
```

En esta clase es donde almacenamos el modo de funcionamiento que por defecto es DB_MODE o el especial de trabajo sin base de datos NO_DB_MODE. El estado del modo es modificado por las clases que controlan la conexión a base de datos y se determina la primera vez que se intenta conectar con el servidor MySQL en el arranque de la aplicación.

El método `printEvent` muestra en el log todos los campos de un evento. El método `sendConfirmationMail` se encarga de crear un correo electrónico en formato HTML para ser enviado como confirmación de creación de un evento. Tanto la impresión del evento en el log como el correo enviado son mostrados en el capítulo de pruebas.

Es importante mencionar la cadena de caracteres `IDMail`. Este texto es incrustado de manera invisible en el contenido del correo de confirmación. En el método `processMail` podemos observar que si el contenido del correo contiene esta cadena de texto, el mensaje no es procesado. Esta es la solución implementada para evitar bucles infinitos de ejecución de análisis de correos si está configurado en la cuenta del usuario el reenvío de correos. Nos podemos encontrar el escenario en que el usuario reenvíe todos los correos con la palabra “Travel” automáticamente a la cuenta de correo de la aplicación. Tras la primera creación del evento se enviará un correo de notificación de creación del evento asociado al “Travel”. Este correo será reenviado por la cuenta de correo del usuario y el Generador de Eventos creará un nuevo evento, informando de la creación y desembocando en un bucle infinito. La incrustación de este mensaje hace detectar al Generador de Eventos que el mensaje ha sido creado por él y no debe analizarlo.

3.2.1.2. La clase Fields

La clase `Fields` define el conjunto de campos que deben ser buscados para crear un evento. Para fortalecer el concepto de campo de un evento, se almacenan claves para hacer referencia a cada campo desde cualquier otra parte del código (tipo *static*). Todas las clases que trabajan directamente con un tipo de campo hacen referencia a alguna de las claves descritas aquí.

Los métodos `getSmsFieldsList` y `getMailFieldsList` devuelven un listado de los campos para crear el conjunto de Buscadores. Como se comentó en el análisis, los Buscadores se crearán y se ejecutarán según un orden predefinido, y éste es el de insertado de campos en el objeto `ArrayList`. Este orden es el mismo para ambos métodos pero no se ha utilizado un único método por si algún campo a buscar hubiese sido exclusivo por alguno de los Parseadores, es decir, que se necesitasen dos listados diferentes e independientes.

Código 3.6: Fields – campos y método de obtención de campos

```
public static final int numSmsFields = 11;
public static final int numMailFields = 11;
public static final String title = "Title";
public static final String date = "Date";
public static final String time = "Time";
public static final String address = "Address";
public static final String city = "City";
public static final String zipcode = "ZipCode";
public static final String country = "Country";
public static final String description = "Description";
public static final String category = "Category";
public static final String user = "Owner";
public static final String privacy = "Privacy";

public static ArrayList<String> getSmsFieldsList() {
      ArrayList<String> array = new ArrayList<>(numSmsFields);
      array.add(user);
      array.add(title);
      array.add(date);
      array.add(time);
      array.add(address);
      array.add(city);
      array.add(zipcode);
      array.add(country);
      array.add(description);
      array.add(category);
      array.add(privacy);
      return array;
}

public static ArrayList<String> getMailFieldsList() {
      ArrayList<String> array = new ArrayList<>(numMailFields);
        ...
}
```

3.2.1.3. La clase EventParser

Esta clase representa un analizador sintáctico, su creador conceptualmente es el objeto GeneradorEventos mediante alguno de sus métodos de procesado. Esta clase es abstracta y está ideada para que se creen diferentes Parseadores, según sus variables de trabajo, que hereden su concepto y sus métodos definidos (globales a todas las clases que hereden de ella).

El principal objetivo de esta clase y sus herederas es la devolución de un evento creado a partir de los contenidos y parámetros suministrados. Las dos clases herederas son `MailEventParser` y `SmsEventParser` que serán estudiadas en la sección 3.4. Esta clase incluye tres métodos para almacenar el texto encontrado y formatearlo adecuadamente para cada uno de los campos de un evento.

El método `addFieldToEvent(Event event, String field, String content)` se encarga de almacenar en el evento `event`, un cierto contenido `content` para el campo de evento `field`. Para cada uno de los campos definidos en la clase Fields cumplimentaremos el campo adecuado del evento con su método `set(...)`. Mediante condiciones se evaluará el campo a almacenar y se utilizará el método de guardado para ese campo en el evento. La mayoría de campos se guardan sin realizar ninguna operación, pero la fecha y la hora tienen un algoritmo ligeramente diferente.

Por definición, el método que utiliza un Buscador devuelve una cadena de texto. Independientemente del formato de fecha u hora que contuviese el correo, el Buscador tendrá que reordenar el día, mes, fecha, hora y minutos siempre con el mismo formato para que el guardado en el objeto `Event` sea correcto. Se ha decidido que la hora siempre tiene que ser suministrada por el Buscador en alguno de estos formatos: `dd-mm-yyyy`, `dd-mm-yy`, `dd-mm` o `yyyy-mm-dd`.

Para cada uno de los formatos intentaremos crear una objeto `Date` correcto ya que es el formato que debe tener la fecha para ser almacenada en un objeto de tipo evento (ver Código 3.7). Es responsabilidad del Buscador de Fecha suministrarla en alguno de los formatos soportados por este método. En gran medida esta tarea también depende de la correcta implementación de los patrones de búsqueda.

Código 3.7: EventParser – detalle de guardado de fecha

```
int dia = 1;
int mes = 1;
int any = 2000;
Pattern p = Pattern.compile("(\\d{1,2})-(\\d{1,2})-(\\d{4})");
Matcher m = p.matcher(content);
if (m.find()) {
      dia = new Integer(m.group(1)).intValue();
      mes = new Integer(m.group(2)).intValue();
      any = new Integer(m.group(3)).intValue();
      GregorianCalendar gc = new GregorianCalendar(any, mes - 1, dia);
      event.setDate(gc.getTime());
      return;
}
```

Para la hora los formatos que deben manejarse son `hh:mm` y además puede contener la palabra `PM` para sumar 12 horas a la hora especificada.

El método `getProvisionalDescription(Event event, String content, String subject)` es utilizado para enriquecer la descripción de un evento ya sea porque no ha sido encontrado o porque pertenece a una categoría especial. Para generar descripciones auxiliares es necesario el evento `event` (para extraer alguno de sus campos o el idioma), el contenido original del mensaje `content` y el asunto del correo `subject`. Cabe destacar que al mencionar el asunto del correo, este será un método utilizado exclusivamente por el Parseador de correos.

El método `comprobarLongitutFields(Event event)` se encarga de recortar el texto a almacenar en un campo del evento. Esto es necesario porque los campos del evento en base de datos tienen una longitud máxima y si la sobrepasamos tendremos un error SQL de inserción. Los campos que son recortados son la dirección, ciudad, país, código postal y título. La descripción es almacenada con un tipo de dato que permite mayores longitudes y no tiene restricción de tamaño.

3.2.1.4. La clase Finder

Es la representación de un objeto Buscador, esta clase es también abstracta y está pensada para que extiendan de ella los Buscadores específicos para cada campo. Contiene prácticamente la totalidad de los métodos de búsqueda utilizados:

- Aplicación de expresiones regulares.
- Búsqueda de información en base de datos.
- Búsqueda de información a partir de otros datos.

Todos los buscadores que extiendan de Finder cuentan con los métodos necesarios para realizar la búsqueda del campo. No es necesaria la creación de un Buscador concreto por cada campo, eso sí, por defecto se aplicará el método de extracción de campo aplicando expresiones regulares. Los otros tipos de búsquedas son métodos adicionales cuando las expresiones regulares no han podido encontrar sus correspondientes campos o si se desea cumplimentar algunos campos utilizando lo hallado por otros Buscadores o realizando consultas a base de datos. Para aplicar estos métodos si es necesario crear un objeto heredero de `Finder` específico y modificar su lógica de obtención de campo. Por ejemplo, en un cierto mensaje puede determinarse que el campo ciudad es Barcelona. En base de datos Barcelona tendrá como país España. Si la palabra España no figura en el mensaje original es imposible que lo halle una búsqueda aplicando *regex*, en cambio, podemos buscar en la base de datos qué país tiene asociado la ciudad Barcelona si ya ha sido especificado alguna vez. Pudiendo crearse en este caso un evento más completo con información que no estaba en el contenido enviado por el usuario. Este método es un arma de doble filo, si la información de base de datos es incorrecta, ésta se propagará al nuevo evento creado.

Al ser una clase abstracta deberán instanciarse las clases que extiendan de ella, es decir, los buscadores específicos para cada campo según la tipología SMS o correo electrónico (`smsFinder` y `mailFinder`).

A la hora de aplicar correctamente los métodos descritos es necesario declarar correctamente los cuatros atributos siguientes para un Buscador:

- `String field`. Identificador de la clase Fields que hace referencia al campo a buscar.
- `String language`. Identificador del idioma del contenido original enviado por el usuario.
- `Objeto Grammar`. Objeto que contiene el listado de expresiones regulares a aplicar al patrón según su idioma, su tipología (SMS o correo) y campo.
- `Objeto Event`. Evento que ha sido creado de cero y está siendo cumplimentado. Es la referencia al objeto que los Buscadores van rellenando con la información encontrada.

El constructor de la clase creará un objeto `Finder` cumplimentando los cuatro atributos mencionados.

A continuación se muestran los métodos que contienen esta clase prestando especial atención al método `getResult` que es el núcleo de un buscador.

Código 3.8: Finder – método getResult

```
public String getResult(String content) {
try {
      int groups, number;
      Pattern pattern = null, p = null;
      Matcher matcher = null, m = null;
      String result, paternString;
      String originalContent = content;
      Vector<String> v;
GELogger.add(this, this.getField() + "Finder");
      content = StringManipulator.quitarAcentos(content);
      content = content.toUpperCase();
      Iterator<Vector<String>> it = this.grammar.getGrammarList().iterator();
      while (it.hasNext()) {
            v = it.next();
            result = v.get(1);
            paternString = v.get(0);
            pattern = Pattern.compile(paternString);
            matcher = pattern.matcher(content);

/** Suponemos que lo que buscamos solo está una sola vez, si está más veces nos
quedaremos con la primera ya que no hay posibilidad de discernir cual es el resultado
adecuado.*/
            groups = matcher.groupCount();
            if (matcher.find()) {
GELogger.add(this, "Patrón encontrado en " + this.getField() + ": " +
pattern.toString());
                  // El grupo 0 es el patrón general
/** Creamos el resultado a partir de la especificación del resultado del patrón
(Contenido en elatributo Grammar y especificado por el documento xxxGrammar.xml)*/
      p = Pattern.compile("#G(\\d+)#");
      m = p.matcher(result);
      while (m.find()) {
            number = Integer.parseInt(m.group(1));
            if (number <= groups) {
                  result = result.replace("#G" + String.valueOf(number) + "#",
      originalContent.substring(matcher.start(number), matcher.end(number)));
            } else {
                  result = result.replace("#G" + String.valueOf(number) + "#", "");
            }
            m = p.matcher(result);
      }
      // Limpieza del resultado encontrado.
        result = StringManipulator.limpiarString(result);
      result = StringManipulator.limpiarComillas(result);
GELogger.add(this, "resultado: " + result);
       return result;
      }

      }
      } catch (Exception e) {
GELogger.add(this, e);
      }
       return ""; // Valor por defecto si no se encuentra nada
}
```

Este método se encarga de realizar la búsqueda básica del campo especificado en el atributo del Buscador aplicando las expresiones regulares del objeto `Grammar`. Es el único método que se ejecutará si un Buscador específico de campo no ha sido creado específicamente, además este método es el principal utilizado por los Buscadores. Se tiene la certeza que se aplica una búsqueda básica por campo. Si ésta es la única opción de búsqueda

no es necesario crear un Buscador para realizar esta acción. Como se observa en la declaración, se devuelve un cierto `String` que contiene el campo encontrado y buscado en el `String` objetivo `content`.

Los buscadores que hereden a partir de `Finder` deberán sobrescribir ese método para alterar la lógica e implementar los métodos alternativos mencionados anteriormente. Desde el nuevo método `getResult` no se perderá la búsqueda original basada en expresiones regulares, se podrá acceder a ella (en la clase `Finder`) a través de la palabra reservada `super`, accediendo de esta manera al método anterior comentado de la superclase `Finder`.

El método `getResult` se encarga de recorrer todos los patrones almacenados y aplicarlos en el contenido para ver si son encontrados o no. Una vez encontrado uno se detiene y devuelve el resultado encontrado. Los patrones se obtienen recorriendo el objeto `Grammar` definido para ese buscador. Pueden tener varios grupos de captura y eso se especifica en la estructura de guardado de expresiones regulares con la palabra clave `#G#`, con la que se especifica el grupo a devolver si un expresión regular encuentra varios grupos de resultados.

En este proceso se utilizan antes de aplicar los patrones de expresiones regulares métodos de limpieza y adecuación del contenido a analizar para que los patrones puedan hallar fácilmente su contenido. Antes de analizar el texto guardamos una copia original del contenido y aplicamos una serie de reemplazos en él para quitar acentos y transformar todo el contenido a mayúsculas. Las expresiones regulares en el objeto `Grammar` también son almacenadas en mayúsculas por lo que de esta manera será más fácil que la expresión regular cuadre con el contenido. Una vez hallado el resultado, mediante la posición inicial y final del `String` encontrado recuperamos el formato original del texto delimitado por esas posiciones, obteniendo el resultado en el contenido original guardado como copia.

Esta clase también tiene los siguientes métodos con algunos ejemplos de código:

Código 3.9: Finder – métodos alternativos de búsqueda

```
public String getResult(String content, String patternString, String result) {
        ...
}

public static String findCategoryFromKeyWords(String content) {
      ...
}

public static String findCountryFromDB(String content) {
       List<String> list = GEModel.retrieveCountries();
      list.remove("");
      for (String pais : list) {
            String paisaux = StringManipulator.quitarAcentos(pais);
            content = StringManipulator.quitarAcentos(content);
            if (content.toUpperCase().contains(paisaux.toUpperCase())) {
                  return pais;
            }
      }
      return "";
}

public static String findCityFromDB(String content) {
      ...
}

(continúa)
```

```
public static String findCountryFromCity(String city) {
        List<String> list = GEModel.retrieveCountriesFromCity(city);
      list.remove("");
      if (list.size() > 0) {
            return list.get(0); // El primer elemento es el de más frecuencia
      }
      return "";
}

public static String findZipCodeFromCity(String city) {
      ...
}

public static String findPrivacyFromContent(String result) {
        ...
}
```

El método `getResult(String content, String patternString, String result)` es casi idéntico al método `getResult` comentado, pero en este caso solo se aplica un único patrón `patternString`.

En el método `findCategoryFromKeyWords(String content)`, a partir de un conjunto de palabras clave se buscan una a una en el contenido, la primera que se encuentra se devuelve. Este método no es utilizado por el parseado de correos, ya que sus expresiones de búsqueda son más completas. Solo es utilizado por el parseador de SMS. Se aplica la misma limpieza del contenido antes de buscar las palabras.

En los métodos `findCountryFromDB(String content)` y `findCityFromDB(String content)` obtenemos todos los países y ciudades que hay en base de datos y se buscan en el contenido. La primera que se encuentra es devuelta. Estos métodos requieren del uso del modelo de datos y la obtención de información a partir de la BD.

En los métodos `findCountryFromCity(String city)` y `findZipCodeFromCity (String city)` se aplica la misma lógica de extracción de información de BD, pero se obtienen el país y código postal solo de la ciudad especificada.

El último método `findPrivacyFromContent(String result)` devuelve la privacidad del evento. `0` para eventos públicos, `1` para eventos con visión de usuarios registrados y `2` para privados. Se utiliza la búsqueda de esas palabras (o sus iniciales) en el contenido. Se aplica la misma limpieza y adecuación del contenido antes de buscar las palabras.

Todas las consultas a base de datos devuelven un conjunto de valores no repetidos y además ordenados por frecuencia de aparición. Se observa que la mayoría de métodos de búsqueda devuelven un `String` vacío (“”) si no se encuentra nada.

3.2.1.5. La clase FinderList

`FinderList` es una interfaz (clase de tipo *interface*) que representa el listado de Buscadores que serán lanzados para buscar los campos. Se deduce que habrá tantos buscadores como campos indicados en la clase `Fields`. Según lo visto en el apartado correspondiente a la clase `Finder`, estos Buscadores pueden ser creados y específicos, o no creados y genéricos.

Por cada tipo Parseador se hará una implementación de `FinderList`. Cada una de estas implementaciones deberá especificar su método público `Hashtable<String, Finder> getFinderList()`. Este método devuelve la tabla con el listado de objetos `Finder`. Cada objeto tiene como clave el identificador correspondiente asociado al campo de la clase `Fields`. Habrá un objeto `Finder` por cada campo insertado en el `ArrayList` de identificadores declarado en la clase `Fields`.

3.2.1.6. La clase Grammar

La clase `Grammar` representa el conjunto de expresiones regulares que aplicará un Buscador para encontrar el campo objetivo. En una primera fase de diseño contenía simplemente el listado de `Strings` con las expresiones regulares para crear los patrones de búsqueda aplicados en el método `getResult` de la clase `Finder`. Posteriormente, en la fase de mejora y potenciación de la aplicación de expresiones regulares esta clase pasó a representar un conjunto de vectores. Cada vector contiene la expresión regular y un campo que indica cómo debe representarse el resultado. Los vectores están almacenados en la lista privada `ArrayList<Vector<String>> GrammarList`.

3.2.1.7. La clase GrammarFactory

Representa una fábrica de gramáticas (objetos `Grammar`), es una clase abstracta y deben crearse las clases para que extiendan de ella. Esta clase está basada en el patrón de diseño "Factoría" que pretende simplificar la creación de objetos complejos siendo accesible a su creación a través de un único método. Para obtener las gramáticas se pasa como parámetro el campo del evento a buscar.

Resulta lógico que no pueda existir una única implementación porque el almacenamiento de los patrones de búsqueda se ha separado según su tipo: *sms* o *mail*. Es por este motivo que conceptualmente es más idónea la creación de dos clases distintas que hereden en función de qué Parseador esté ejecutándose.

Se obtiene un objeto `Grammar` con el método `getGrammar(String type)`. El objeto `Grammar` contiene las gramáticas guardadas en los archivos de tipo *xxxGrammar*.xml (donde xxx debe ser Mail, Sms o cualquier otra especificación de un Parseador si se añadiera otro). La variable `type` es el identificador de la clase `Fields` que identifica el `Finder` y a los patrones dentro del archivo XML. De aquí se extrae que cada `Finder` tendrá asociado un objeto `Grammar` solo con los patrones de búsqueda de su tipo.

3.2.1.8. La clase XMLGrammarReader

XMLGrammarReader es el encargado de leer el archivo donde se almacenan las expresiones regulares. Como su nombre indica trabaja con archivos XML y se encarga de leer la información mediante la API DOM[44]. Los datos obtenidos se presentan según el formato adecuado para que `GrammarFactory` pueda crear el objeto `Grammar`.

[44] Document Object Model, Modelo de Objetos del Documento

Mediante esta clase se independiza la forma como se guardan las expresiones regulares. Si se quisiese implementar un nuevo método de almacenamiento debería modificarse únicamente esta clase o implementar otra alternativa y sería la factoría la encargada de escoger un método u otro de creación.

`XMLGrammarReader` se ocupa de crear los objetos de gramáticas `Grammar` a partir de los documentos xxxGrammar.xml (Archivos que contienen las expresiones regulares). El formato de los archivos XML tiene que seguir unas normas que están especificadas y que se pueden ver en los apéndices B.2, B.3 y en la sección 3.6.

Esta clase tiene el siguiente método constructor:

Código 3.10: XMLGrammarReader – constructor

```
public XMLGrammarReader(String file) throws ParserConfigurationException,
SAXException, IOException {
      this.fileName = file;
      getDocumentXML();
}
```

Pasamos como parámetro el archivo de gramáticas que queremos leer, en el constructor se invoca al método `getDocumentXML()` que cargará en una variable de tipo `Document` el documento leído mediante la API DOM. El documento a leer tiene que tener forzosamente una estructura de documento XML. Si dicha estructura no se cumple no se leerá el documento generando una excepción, ocasionando la no lectura de ningún patrón en el futuro.

El método público para obtener el listado de patrones es `ArrayList<Vector<String>> getPatterns(String language, String tag)`. Obtenemos un `ArrayList` que contiene los patrones para crear posteriormente un objeto de Gramáticas `Grammar`. A partir de un idioma `language` y un identificador `tag` se obtiene el listado mencionado. El identificador `tag` debe ser alguno de los campos especificados en la clase `Fields`. El idioma deberá ser el adecuado para analizar el contenido (extraído del contenido recibido). Se acepta la palabra clave "default" para utilizar algún idioma por defecto si este no se especifica correctamente. Esta función utiliza el método privado:

```
ArrayList<Vector<String>> createPatterns(ArrayList<Vector<Object>>
keywordArray, ArrayList<Vector<String>> patternArray) { ... }
```

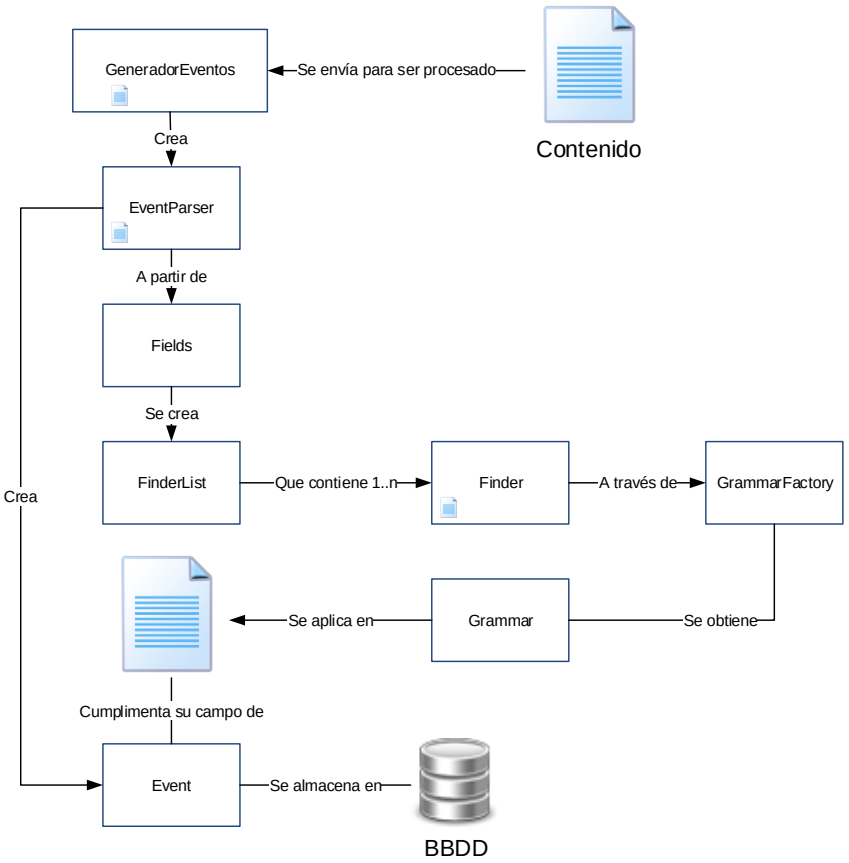
Este método crea los patrones definitivos a partir de los datos obtenidos del fichero XML. Cabe mencionar que el fichero no es un simple contenedor de patrones, contiene una ciertas funcionalidades para dotarlo de más versatilidad, pudiendo crear múltiples patrones indicando pequeñas variaciones o incluso especificando la sintaxis de lo encontrado para mostrarse de forma adecuada.

Adicionalmente, la clase contiene el método público `String getDefaultLanguage()` para averiguar el idioma por defecto del archivo de gramáticas . El idioma por defecto se ha determinado que sea el español (identificador "ESP").

3.2.2. Modelo conceptual del Generador de Eventos

En la Figura 3.1 se muestra la interacción que existe entre las diferentes clases. Esta interacción es conceptual ya que muchas de las clases son abstractas o representan interfaces. El resumen del proceso completo fue descrito en la sección 2.3.

Figura 3.1: Modelo conceptual del Generador de Eventos



3.3. El modelo y la base de datos

Antes de profundizar con los métodos utilizados para la creación de un evento detallaremos el paquete que modela los datos utilizados por el Generador de Eventos. En primer lugar tenemos el paquete *db* que contiene las clases que permiten la conexión a base de datos (BD). En segundo lugar el paquete *model,* que modela las tablas que hay en BD y las operaciones realizadas en ella.

3.3.1. Estructura del paquete db

Para establecer una conexión a base de datos se ha utilizado una API auxiliar que permite la creación de un pool de conexiones, resultando más fácil administrarlas ya que no hay que utilizar el código nativo de Java para estar creando una conexión constantemente y destruyéndola para liberar recursos. Dentro del paquete *db* tenemos el paquete *pool* con las siguientes clases y archivos de configuración:

Tabla 3.2: Clases y ficheros del paquete db

Paquete	Clase (.java)
db	DBManager
db	DBName
db	DBSql
db.pool	GEPool
db.pool	PoolObservable
db.pool	dbconfig.properties

A continuación se explican las diferentes clases y ficheros que forman el módulo de conexión y operaciones en base de datos:

3.3.1.1. Archivo de configuración de acceso

En el archivo `dbconfig.properties` tenemos la información necesaria para establecer la conexión a BD. En él especificamos el driver utilizado, la url de conexión y el usuario y password.

```
GE.driver = com.mysql.jdbc.Driver
GE.url = jdbc:mysql://localhost:3306/ge
GE.username = root
GE.password = root
GE.test = select 1
```

Para este proyecto solo se ha creado un elemento de conexión a base de datos llamado `GE`. Si hubiese habido más conexiones a bases de datos diferentes, los parámetros se habrían duplicado para las demás conexiones empezando con el nombre de cada una.

3.3.1.2. Las clases PoolObservable y GEPool

La clase `PoolObservable` es el núcleo de conexión a la BD. Está basado siguiendo la estructura definida por la API BoneCP ([17]), por lo que no ha sido diseñado para este proyecto desde cero. Los ejemplos de la API han sido adaptados para conectar a la BD del Generador de Eventos.

Los métodos más importantes de la clase es `getConnectionCP()` que invoca el método `createCP()`. Este método está definido por la estructura de la API BonceCP y delega el establecimiento de conexión a los métodos de su librería. Es en ese método donde se crea, en función del nombre de la BD pasada al constructor y el fichero de configuración la conexión, almacenándose en un objeto `BonceCP`. Solo se creará conexión la primera vez que se demande, y una vez creada se retornará una conexión suministrada y mantenida por el objeto `BoneCP connectionPool`. Antes de cerrar la aplicación debemos liberar los recursos. De esa tarea se encarga el método `closeCP()`.

Código 3.11: Métodos de obtención de conexión a base de datos y su cierre

```
protected Connection getConnectionCP() throws SQLException {
	if (!connected) {
		createCP();
	}
	if (connectionPool != null) {
		return connectionPool.getConnection();
	}
	return null;
}

protected void closeCP() {
	if (connected) {
	assert connectionPool != null;
		connectionPool.shutdown();connected = false;
		setChanged();
		notifyObservers();
	} else {
		GELogger.add(this, dbName + " no conectado");
	}
}
```

La clase `GEPool` es heredera de la clase `PoolObservable` y además implementa el patrón singleton de una manera diferente a la clase `GeneradorEventos`. En este caso se realiza mediante el uso de una clase estática `ClassHolder` que guarda la instancia a la clase, que al ser estática solo se ejecuta una única vez.

Código 3.12: Clase GEPool – instancia única

```
public class GEPool extends PoolObservable {

	private static class ClassHolder {

		private static final GEPool INSTANCE = new GEPool(DBName.GE);
	}

	public synchronized static GEPool getInstance() {
		return GEPool.ClassHolder.INSTANCE;
	}
	private GEPool(DBName dataBase) {
		super(dataBase);
	}
	...
}
```

Los métodos que utilizaremos son mayoritariamente el método `public synchronized static Connection getConnection()` que devuelve una conexión invocando el método de la clase de la que hereda (`PoolObservable`). Como hemos visto, la primera vez que se requiera una creación se instanciará tanto `GEPool` como el `PoolObservable`. A la hora de cerrar la conexión y liberar recursos utilizaremos la función `close()`.

3.3.1.3. La clase DBName

Esta clase simplemente es una enumeración que declara la base de datos del proyecto como `GE`. Si hubiese más de una base de datos se especificaría en esta clase y se añadiría su gestión en la clase `DBManager`.

3.3.1.4. La clase DBManager

Esta es la clase más importante a la hora de interactuar con el modelo de aplicación y está formada por los siguientes métodos:

Código 3.13: Clase DBManager

```
public class DBManager {

	public static Connection getConnection(DBName dataBase) throws SQLException {
		switch (dataBase) {
		case GE:
			return GEPool.getConnection();
		default:
			break;
		}
		return null;
	}

	public static void closePools() {
		GEPool.close();
	}
}
```

Los métodos `Connection getConnection(DBName dataBase)` y `closePools()` se utilizan en el arranque y parada del Generador de Eventos. Como solo manejamos una conexión a un esquema de base de datos (`GE`) solo abriremos y cerraremos conexión con dicha BD. Para obtener una conexión utilizaremos el método estático de la siguiente forma:

```
Connection con = DBManager.getConnection(DBName.GE);
```

3.3.1.5. La clase DBSql

Esta clase define un conjunto de operaciones SQL donde la más utilizada es la siguiente:

Código 3.14: Clase DBSql

```
public class DBSql {

      public static ArrayList<String> select(DBName dataBase, String select) throws
SQLException {
            Connection con = null;
            Statement stmt = null;
            ResultSet rs = null;

            ArrayList<String> array = new ArrayList<>();
            try {
                  con = DBManager.getConnection(dataBase);
                  stmt = con.createStatement();
                  rs = stmt.executeQuery(select);
                  while (rs.next()) {
                        array.add(rs.getString(1));
                  }
            }catch (Exception e) {
                  GELogger.add(DBSql.class, e);
            } finally {
                  ObjectCloser.attemptClose(rs);
                  ObjectCloser.attemptClose(stmt);
                  ObjectCloser.attemptClose(con);
            }
            return array;
      }
        ...
}
```

El método más representativo de esta clase realiza una operación de consulta *select* a tablas de la BD. Pasamos como parámetros un nombre de la base de datos para obtener su conexión mediante el `DBManager`. Acto seguido creamos un `Statement` y ejecutamos la consulta almacenada en la variable `select`. Recorremos los resultados que son almacenados en el objeto `ResultSet` y añadimos los resultados en un `ArrayList` de `Strings`. Es el método utilizado para extraer de base de datos las ciudades, países y códigos postales.

3.3.2. Estructura del paquete model

El paquete *model* alberga los objetos que representan las entidades de base de datos evento, multimedia y usuario. Cada uno de estos tres conceptos tiene asociadas dos clases, una que define su estructura y es utilizada por el Generador de Eventos y otra que define las operaciones a realizar en las tablas de BD. Adicionalmente, la clase `GEModel` tiene algunas operaciones de consulta específicas.

Las clases que definen las entidades evento, multimedia y usuario tienen definidos como atributos los mismos campos creados en la BD MySQL. Estos atributos son en Java de tipos equivalentes a los tipos de datos creados en la base de datos.

Dentro del paquete *model* tenemos las siguientes clases:

Tabla 3.3: Clases y ficheros del paquete model

Clase (.java)
Event
EventModel
GEModel
Multimedia
MultimediaModel
User
UserModel

3.3.2.1. Las clases Event, Multimedia y User

Estas clases definen el contenido de los campos a cumplimentar cuando se genera un evento o se guarda un archivo multimedia. También son los objetos que son cumplimentados cuando el Generador de Eventos lee de tablas la información que hay en BD para mostrarla en la interfaz gráfica.

- Evento.

Código 3.15: Campos de la clase Event

```
Integer idevent;
private Integer iduser;
private String title;
private String category;
private Date date;
private Date time;
private String address;
private String city;
private String zipcode;
private String country;
private String description;
private String privacy;
private Date timestamp;
private String language;
```

Todos los campos que forman un evento tienen creados los métodos de acceso `get` y `set`. Según la especificación creada para la base de datos no pueden ser nulos ni el id de usuario, el id de evento, el título y el timestamp[45]. Como veremos más adelante en el objeto `Finder` de fecha, se pondrá por defecto la fecha del día de generación si ésta no se encuentra en el proceso de análisis sintáctico del contenido para crear el evento. Además en la creación del evento se cumplimenta el timestamp con la fecha actual de modificación (creación en BD). Los métodos `set` que figuran en el objeto `Event` son utilizados mayoritariamente por el Parseador para guardar en el evento el resultado encontrado por cada buscador `Finder`.

Observamos que un evento queda determinado y es único por su id de evento. El id de usuario relaciona este evento con el usuario creador.

[45] Marca de registro temporal con la fecha de creación o modificación.

- Multimedia.

Código 3.16: Campos de la clase Mutimedia

```
protected Integer idmultimedia;
private Integer idevent;
private Integer iduser;
private String file;
private Date timestamp;
```

Para un objeto `Multimedia` tenemos tres identificadores que no pueden ser nulos. Aparte del id del propio objeto multimedia tenemos la relación con el evento propietario y el id del usuario creador del evento y dueño del fichero multimedia.

En el `String file` almacenamos el nombre del fichero creado. Para una localización única del fichero se añadirá como prefijo el id del evento al que pertenece quedando así: `Idevent_nombreOriginalDelArchivo`.

- User.

Código 3.17: Campos de la clase User

```
private Integer iduser;
private String name;
private String email;
private String mobilephone;
private String address;
private String city;
private String zipcode;
private String country;

private Notification notifemail;
private Notification notifsms;
```

Para un objeto `User` tenemos su id de usuario y la información básica para poder trabajar con él. Necesitaremos sobretodo su e-mail y número de teléfono para poder hallar una vez recibido un contenido el usuario que lo ha enviado. Además hay información de ubicación del usuario que se mantiene por herencia de la Agenda Social y es utilizada a la hora de buscar campos del evento por los Buscadores.

Las últimas variables `notifemail` y `notifsms` son del tipo `Notification`. Una enumeración que indica `SI` o `NO` para saber si el usuario tiene activadas las notificaciones según el canal que corresponda.

3.3.2.2. Las clases EventModel, MultimediaModel y UserModel

Estas clases tienen las operaciones de escritura y lectura en BD para la inserción, modificación u obtención de objetos `Event`, `Multimedia` o `User`.

- UserModel.

Código 3.18: Métodos de la clase UserModel

```
private static TreeMap<Integer, User> treeUser = new TreeMap<>();

public static void leerTabla() {
	...
}

public static Collection<User> getUsers() {
	return treeUser.values();
}

public static User retrieve(Integer iduser) {
	...
}

public static User retrieveEMail(String mail) {
	if (GeneradorEventos.getDBMode().equals(DBMode.NO_DB_MODE)) {
		return User.getVirtualUser();
	}

	try {
		Integer iduser = DBSql.selectInt(DBName.GE, "select iduser from user
where email = '" + mail + "'");
		return retrieve(iduser);
	} catch (SQLException e) {
		GELogger.add(UserModel.class, e);
		return null;
	}
}

public static User retrieveMobile(String number) {
	...
		Integer iduser = DBSql.selectInt(DBName.GE, "select iduser from user
where mobilephone = '" + number + "'");
	...
}
```

En la clase `UserModel` encontramos el método `leerTabla()`. Se rellena el `TreeMap` para el almacenado de los usuarios que hay en BD. Esta función se utiliza sobre todo para el rellenado de tablas en la interfaz gráfica. La función `retrieve(Integer iduser)` devuelve un objeto `User` cumplimentado obtenido en función del id de usuario. Es importante comentar que si el usuario no es encontrado se devolverá `null`.

Los métodos `retrieveEMail(String mail)` y `retrieveMobile(String number)` devuelven el usuario propietario del correo o número de teléfono. En estos métodos primero obtenemos el id de usuario y seguidamente mediante el método *`retrieve`*`(iduser)` obtenemos el objeto `User`.

Obsérvese que los métodos tienen el fragmento de código:

```
if (GeneradorEventos.getDBMode().equals(DBMode.NO_DB_MODE))
   return User.getVirtualUser();
```

Si está en funcionamiento el modo NO_DB_MODE en vez de devolver un usuario nulo que abortaría el proceso de creación de un evento, devolveremos un usuario virtual con información mínima. Con este usuario no se realizarán envíos de notificaciones y tendrá el id de usuario -1. Cabe destacar también que los usuarios deben crearse manualmente en la base de datos, ya que el Generador de Eventos no tiene ninguna funcionalidad para crearlos, solo para leerlos.

- EventModel y MultimediaModel.

Estas dos clases funcionan igual que la de usuario, solo que en este caso además tenemos métodos de inserción en BD. Como ejemplo se muestra en el Código 3.19 la inserción de un objeto `Event`:

Código 3.19: Inserción de un evento en base de datos

```
private static Integer obtenerIdEvent() {
	if (GeneradorEventos.getDBMode().equals(DBMode.NO_DB_MODE)) {
		return -1;
	}
	Integer id = null;
	try {
		id = DBSql.selectInt(DBName.GE, "select idevent from event order by
idevent desc limit 1");
		if (id != null) {
			if (id < 0) {
				id = 0;
			}
			return id + 1;
		}
	} catch (Exception e) {
		GELogger.add(EventModel.class, e);
	}
	return null;
}

public static synchronized Integer create(Event event) {
	if (GeneradorEventos.getDBMode().equals(DBMode.NO_DB_MODE)) {
		return -1;
	}
	Integer idevent = null;
	idevent = EventModel.obtenerIdEvent();
	event.idevent = idevent;
	if (idevent != null) { // guardamos en base de datos
		try {
			Connection con = null;
			PreparedStatement ps = null;
			try {
				con = DBManager.getConnection(DBName.GE);
				String insertTableSQL = "insert into event "
				+ "(idevent, iduser, title, category, date, time, address,
city, zipcode, country, description, privacy, timestamp, language) "
				+ "values (?,?,?,?,? , ?,?,?,?,? , ?,?,?,?)";
				ps = con.prepareStatement(insertTableSQL);
				ps.setInt(1, event.getIdevent());

				...

				ps.setString(12, event.getPrivacy());
				if (event.getTimestamp() == null) {
					ps.setNull(13, Types.TIMESTAMP);
				}else{
					ps.setTimestamp(13, new
java.sql.Timestamp(event.getTimestamp().getTime()));
				}
				ps.setString(14, event.getLanguage());
				ps.executeUpdate();
			} finally {
				ObjectCloser.attemptClose(ps);
				ObjectCloser.attemptClose(con);
			}
		} catch (Exception e) {
			GELogger.add(EventModel.class, e, "Error al guardar el evento en
base de datos");
		}
	}
	return idevent;
}
```

En el método `create(Event event)` observamos que se hace una llamada al método `EventModel.obtenerIdEvent()`. Antes de poder almacenar un evento en base de datos tenemos que tener determinado el id de evento que va a poseer. La función de obtención del evento hace una consulta y determina cual es el mayor identificador libre en la tabla de eventos. Seguidamente se lanza una operación de insertado en base de datos cumplimentando todos los campos a guardar con los del objeto `Event`. Finalmente se devuelve el id de evento creado. Si estamos en el modo NO_DB_MODE el id devuelto será -1, si no se consiguiera realizar el insertado el id devuelto sería nulo.

3.3.2.3. La clase GEModel

Esta clase se utiliza como contenedora de cuatro funciones de obtención de datos en tablas. Estas funciones son las utilizadas a la hora de extraer información de la BD cuando las expresiones regulares de los Patrones no han conseguido obtener nada. Las cuatro funciones son las siguientes:

Código 3.20: Clase GEModel

```
/* select tbl.country, count(*) from ( select country from user union all select
country from event) as tbl where tbl.country is not null and tbl.country <> '' group
by tbl.country order by count(*) desc, tbl.country; */

public static List<String> retrieveCountries() {
	List<String> list = new ArrayList<>();
	if (GeneradorEventos.getDBMode().equals(DBMode.NO_DB_MODE)) {
		return list;
	}
	try {
		list = DBSql.select(DBName.GE, "select tbl.country from "
	+ "( select country from user union all select country from event) as tbl "
	+ "where tbl.country is not null and tbl.country <> '' "
	+ "group by tbl.country order by count(*) desc, tbl.country");
	} catch (SQLException e) {
		GELogger.add(UserModel.class, e);
	}
	return list;
}

public static List<String> retrieveCountriesFromCity(String city) {
	...
}

public static List<String> retrieveZipcodeFromCity(String city) {
	...
}

public static List<String> retrieveCities() {
	...
}
```

Tenemos cuatro métodos que devuelven los resultados en listas `List<String>`. En verde tenemos un ejemplo de sql que se puede utilizar directamente en el servidor MySQL para el esquema `GE`. Hay dos métodos que devuelven listados de ciudades y países y otros dos que devuelven listados de países y códigos postales a partir de una ciudad. Todas las consultas aplican el mismo concepto: hacemos una unión del campo a buscar en base de datos de las tablas *Event* y *User* y nos quedamos con los campos ordenados por frecuencia de aparición, excluyendo duplicados, registros nulos o vacíos. Teniendo así más peso los campos que aparecen con más frecuencia en BD.

3.4. Parseador de e-mails y SMS

Los Parseadores de correos y SMS tienen un gran parecido entre ellos, mantienen la misma estructura y solo tienen ligeras diferencias en el proceso de creación de un evento. Esto se debe a que heredan de las mismas clases e interfaces que define el Generador de Eventos. Si hubiese existido otro método de entrada con contenidos ligeramente diferentes se hubiese creado una nueva concreción y sub-paquete para esa tipología de contenido.

Cada uno de estos parseadores y su conjunto de clases se organizan en un paquete anidado en *eventParser*. Estos sub-paquetes son *eventParser.mail* y *eventParser.sms* respectivamente. En la Tabla 3.4 se recogen las clases propias de cada sub-paquete.

Tabla 3.4: Clases de los paquetes eventParser.mail y eventParser.sms

eventParser	eventParser.mail	eventParser.sms
GeneradorEventos		
EventParser	MailEventParser	SmsEventParser
Fields		
Finder	MailFinder • OwnerMailFinder • DateMailFinder • TimeMailfinder • CityMailFinder • ZipCodeMailFinder • CountryMailFinder • CategoryMailFinder • PrivacyMailFinder MultimediaMailFinder	SmsFinder • OwnerSmsFinder • DateSmsFinder • CitySmsFinder • ZipCodeSmsFinder • CountrySmsFinder • CategorySmsFinder • PrivacySmsFinder MMSFileExtractor
FinderList	MailFinderList	SmsFinderList
Grammar		
GrammarFactory	MailGrammarFactory	SmsGrammarFactory
XMLGrammarReader		
	mailGrammar.xml	smsGrammar.xml

Como se vio en la Tabla 3.1, las clases que necesitan implementaciones de cada tipo tienen reflejado en la tabla anterior los nombres de sus clases respectivamente. Algunas de las clases utilizadas por los dos tipos de Parseadores tienen métodos exclusivos para cada sub-paquete.

También se observa que en el Parseador de SMS no existe el buscador de "Time", y tampoco el de "Title" para ambos Parseadores. Como ya se ha comentado no son necesarios porque la clase `Finder` ya implementa la búsqueda del campo aplicando expresiones regulares por defecto con su método `getResult`. Las clases `MailFinder` y `SmsFinder` extienden de `Finder` y solo se utilizan para mantener la jerarquía y poder así tener una diferenciación en nombre y herencia para el total de `Finders` creados para cada campo.

3.4.1. Semejanzas y diferencias entre los Parseadores

La estructura de clases y el contenido de los métodos de ambos paquetes son muy parecidos entre ellos; incluso los métodos de las clases del paquete principal *eventParser* que están separados según contenido se asemejan bastante. De hecho, deben realizar la misma función pero adecuándose al contenido que es suministrado.

Esta diferenciación es necesaria y además causada por la no igualdad que hay entre los dos diferentes tipos de contenidos que puede enviar el usuario. Por un lado tenemos un SMS y las características que pueden influir en el proceso de creación de un evento son las siguientes:

- Es un mensaje corto.
- El formato del mensaje está escrito en texto plano, sin formato.
- Está pensado que su contenido sigo una estructura de tipo plantilla.
- El SMS tiene asociado el número de teléfono de quién lo ha enviado.
- La determinación del usuario creador se hará a partir del número de teléfono.
- El proceso de creación está orientado a una estructura simple de contenido.
- El método de creación del evento se corresponde con una generación semiautomática.
- El lenguaje utilizado puede ser muy informal, con el llamado “lenguaje SMS” donde abundan las faltas de ortografía y la abreviación de palabras para forma otras nuevas más cortas y fáciles de interpretar por los usuarios. Se crea una dificultad a la hora de determinar el idioma en función de las palabras del mensaje.
- En el caso de un MMS tendremos ficheros multimedia adjuntos.

En cambio la creación de un correo electrónico se caracteriza por:

- La longitud es indeterminada.
- El correo puede tener varios formatos, texto plano o texto HTML.
- Su contenido puede seguir una estructura de tipo plantilla.
- Su contenido puede no seguir ninguna estructura.
- Hay total libertad a la hora de crear el mensaje por parte del usuario.
- Un correo tiene asociado un objeto *javax.mail* de tipo `Message`. Este objeto tiene asociada la estructura y datos del correo.
- El usuario de creación del evento vendrá determinado por el emisor del correo.
- Un correo puede tener archivos adjuntos.
- Los correos automáticos suelen ser generados con opciones de reenvío en filtros del cliente de correo del usuario. Se pueden ocasionar creaciones erróneas si se intenta crear un evento automáticamente a partir de una notificación de creación si se vuelve a reenviar automáticamente por el correo del usuario.
- Se puede extraer el idioma fácilmente a partir del propio contenido.

Según la planificación inicial y con el fin de dejar operativa la comunicación y una primera versión de creación de eventos a partir de SMS, se desarrolló primero el Parseador de SMS. Seguidamente se creó el Parseador de e-mails siguiendo la estructura y lógica del de SMS. En este punto del desarrollo, los dos eran bastante parecidos. Se tenían ambos Parseadores orientados exclusivamente a un método semiautomático de creación. La complejidad fue aumentando y la introducción del diccionario hizo rehacer muchas de las clases para depender de un nuevo atributo: el idioma. Seguidamente para dar soporte a correos con formato (HTML), datos adjuntos y creación automática se potenció el contenido de los métodos de búsqueda y las expresiones de carga, así como sus clases asociadas.

En la fase de enriquecimiento del proceso para mejorar la búsqueda de campos se centraron los esfuerzos en el Parseador de e-mail, aumentando la funcionalidad de los métodos de búsqueda de los Buscadores, creando nuevas expresiones regulares e implementando una nueva estructura de fichero XML con el fin de dar soporte a nuevas reglas de generación de patrones de búsqueda.

3.4.2. Clases según tipo concreto de Parseador

En este apartado veremos qué peculiaridades tienen los métodos más importantes que implementan los dos tipos de Parseadores con los aspectos en común y las diferencias, así como la función de la clase, sus métodos y los pasos que se siguen a la hora de crear un evento. Debido a la similitud en ambas clases se detalla el proceso de creación de un correo y se comentarán las diferencias con el de SMS.

3.4.2.1. Las clases MailEventParser y SmsEventParser

Estas dos clases forman el núcleo de la acción de parsear y tienen como objetivo crear el evento asociado al contenido recibido por el Generador de Eventos. Explicaremos con detalle el proceso de creación de un evento para la clase `MailEventParser` por pasos.

En la primera parte de creación (Código 3.21), extraemos el contenido almacenado en un objeto `Part` utilizando el método estático *`getPartToProcess`*`(message)` de la clase `MailOperations` definido en el paquete *mail*. Obtenemos el tipo MIME de la parte del correo y en función de si es HTML o texto plano les aplicamos unas operaciones de limpieza. Estas operaciones de limpieza intentan adecuar el contenido para que los patrones puedan encontrar los contenidos más fácilmente. Algunas de estas operaciones son eliminar espacios en blanco duplicados, reemplazar muchos espacios por el símbolo ‘:’ para ser utilizado como separador por las expresiones regulares, etc. Al final del proceso de limpieza obtenemos también el idioma del contenido a partir de las clases del paquete diccionario.

Código 3.21: MailEventParser – proceso de limpieza

```
public Event getEventParsed(Message message) {
	String language = "default";
	String result;
	String content = "";
	try {
		Part partContent = MailOperations.getPartToProcess(message);
		content = (String) partContent.getContent();
		GELogger.add(this, "Mail recibido:\n" + (String)
partContent.getContent());
		if (partContent.isMimeType("text/html")) {
			content = StringManipulator.limpiarHTML(content);
		} else {
			content = StringManipulator.limpiarPlain(content);
		}
		// Añadimos un \n para que no falle si al final del todo hay un
		// campo a buscar
		GELogger.add(this, "Contenido limpio:\n" + content);
		language =
GeneradorEventos.getInstance().getDictionaries().getLanguage(content);
	} catch (Exception e) {
		GELogger.add(this, e);
	}
```

En el siguiente paso de proceso (Código 3.22) creamos un evento `Event` en blanco. Obtenemos el listado de buscadores y los guardamos en una tabla relacionando el campo a buscar (de la clase `Fields`) y su objeto `Finder`. El listado de buscadores es proporcionado por la clase `MailFinderList`. Una vez tenemos la tabla con todos los buscadores ejecutamos para cada uno de ellos su método `getResult`. Este método estará sobrescrito si se ha creado un Finder específico para el campo a buscar, sino se utilizará el método `getResult` de la clase `Finder`. Observamos que si algún buscador devuelve nulo el proceso se aborta y se devuelve un evento nulo. Recordemos que un `Finder` debe entregar en su campo `getResult` un `String` vacío si no se ha encontrado el campo.

Código 3.22: MailEventParser – proceso de ejecución de buscadores

```
	Event event = new Event();
	Hashtable<String, Finder> ht = (new MailFinderList(event,
language)).getFinderList();
	Iterator<String> it = Fields.getMailFieldsList().iterator();


	while (it.hasNext()) {
		String field = (String) it.next();
		// Invoquem per a tots el mètode getResult(Part part)
		result = ((MailFinder) ht.get(field)).getResult(message, content);

		if (result != null) {
			if (result.length() > 0) {
				this.addFieldToEvent(event, field, result);
			}
		} else {
			return null;
		}
	}
```

En la parte final del proceso de creación (Código 3.23) se complementa la información del evento con un título y descripción por defecto si no son hallados esos campos. Finalmente se comprueban las longitudes del evento para que no den problemas al guardarse en base de datos y se retorna el evento creado.

Código 3.23: MailEventParser – proceso de finalización de creación

```
	String title = event.getTitle();
	if (title == null || title.isEmpty()) {
		this.addFieldToEvent(event, Fields.title, event.getCategory() + " Event
(Title not found)");
	}
	String description = event.getDescription();
	if (description == null || description.isEmpty()) {
		String subject = "";
		try {
			subject = message.getSubject();
			subject = StringManipulator.limpiarSubject(subject);
			this.addFieldToEvent(event, Fields.description, "AutoGenerated
Event - (Mail Subject: " + subject + ")");
		} catch (MessagingException e) {
			GELogger.add(this, e);
		}
		event.setDescription(this.getProvisionalDescription(event, content,
subject));
	}
	event.setLanguage(language);
	event = comprobarLongitutFields(event);
	return event;
}
```

Para los SMS se utilizan unos métodos de limpieza del contenido más sencillos. A la hora de determinar el lenguaje utilizaremos el método `getXMSLanguage(message)` de la clase que maneja los diccionarios. Ese método elige el idioma del mensaje en función de las etiquetas identificadoras de campo de un SMS (plantilla SMS). La parte de procesado del evento es igual solo que obteniendo el conjunto de buscadores de tipo `SmsFinder`. En la parte final se cumplimenta el título con uno por defecto si éste no ha sido rellenado.

3.4.2.2. Las clases herederas de Finder

Los objetos `Finder` específicos contienen la inteligencia de búsqueda. Se muestran tres ejemplos y para los demás se comentarán brevemente sus peculiaridades. Se han escogido los Buscadores del Parseador de correos y se comentarán las diferencias con los de SMS.

- OwnerMailFinder.

Código 3.24: OwnerMailFinder

```
public String getResult(Message message, String content) {
	try {
		String mail = "";
		User user = null;
		Address[] A = message.getFrom();
		for (int i = 0; i < A.length; i++) {
			mail = StringManipulator.limpiarMail(A[i].toString());
			user = UserModel.retrieveEMail(mail);
			if (user != null) {
		GELogger.add(this, "Propietario del e-mail: " + user.getName());
				return Integer.toString(user.getIduser());
			}
		}
	} catch (Exception e) {
		GELogger.add(this,e);
	}
	// No hemos encontrado al usuario
	GELogger.add(this, "Propietario del e-mail no encontrado");
	return null;
}
```

El Buscador de usuario es especial y su objetivo es encontrar el identificador del usuario que ha enviado el correo. A partir del objeto `Message` contenedor del correo obtenemos un `ArrayList` con las direcciones de quienes han enviado el correo (normalmente será única pero recorreremos el array completo). Cada dirección de correo será buscada en base de datos intentando obtener el usuario que la tiene asignada. Si se encuentra se devolverá el id de ese usuario transformándolo a `String` (el método `getResult` debe devolver siempre un `String`). Si no es encontrado se devolverá `null` para abortar el proceso de creación del evento.

Para el caso del SMS es similar solo que buscaremos el usuario en función del número de teléfono. Los buscadores de tipo `Sms` en vez de tener como parámetros el mensaje original de correo tienen un único contenido de texto. Para el `OwnerSmsFinder` se suministra directamente el número de teléfono como contenido al método `getResult(String number)`.

- CityMailFinder.

Código 3.25: CityMailFinder

```
public String getResult(Message message, String content) {
	String result = "";
	GELogger.add(this, this.getField() + "MailFinder");
	try {
		result = super.getResult(content);
		if (result.equals("")) {
			String address = getEvent().getAddress();
			if (address != null && !address.isEmpty()) {
				result = Finder.findCityFromDB(address + "\n");
			}
		}

		if (result.equals("")) {
			String subject = "";
			try {
				subject = message.getSubject();
			} catch (MessagingException e) {
				GELogger.add(this, e);
			}
			result = Finder.findCityFromDB(content + "\n" + subject + "\n");

		}
	} catch (Exception e) {
		GELogger.add(this, e);
	}
	return result;
}
```

La estructura de este `Finder` es típica. Primero antes de todo intentamos obtener el resultado aplicando los patrones guardados en el método `Grammar` invocando el método `getResult` pero definido en el objeto `Finder` del que hereda la clase. Si no se ha encontrado ningún resultado se aplican reglas especiales. En este caso intentamos encontrar los nombres de ciudades que hay en base de datos primero en la dirección o lugar del evento. El orden de lanzamiento de Buscadores es importante y en este caso el proceso de búsqueda de la ciudad utiliza lo encontrado por el proceso de búsqueda de la dirección. Si no se encuentra nada entonces procedemos a buscar la ciudad en el contenido del mensaje y su asunto. Para los correos también se utilizan las búsquedas en el asunto por si hay alguna palabra clave en ella que luego no figure en el correo.

A continuación describimos las peculiaridades de los demás Buscadores:

- CountryMailFinder. Igual que CityMailFinder pero además de los patrones busca países de la base datos en el contenido.
- DateMailFinder. En este caso la fecha encontrada tiene que enviarse con el formato `"dd-MM-yyyy"` por lo que después de ser hallada se reformatea. Si no se encuentra ninguna fecha se envía la fecha de adquisición del contenido.
- TimeMailFinder. Este buscador además aplica la técnica de buscado de fechas en el título del evento. El formato de búsqueda de la hora siempre es `"hh:mm"` o `"hh:mm:PM"`.
- ZipCodeFinder. Buscador básico con el añadido de buscar el código postal en base de datos en función de la ciudad encontrada.
- Privacy. Buscador que devuelve por defecto un “2” (Evento privado) si no es encontrado ningún otro campo.

3.4.2.3. Las clases MailFinderList y SmsFinderList

Estudiaremos el código de la clase MailFinderList:

Código 3.26: MailFinderList

```
public Hashtable<String, Finder> getFinderList() {
	MailGrammarFactory MGF = new MailGrammarFactory(language);
	if (language.equals("default")) {
		language = MGF.getDefaultLanguage();
	}
	ArrayList<String> arrayField = Fields.getMailFieldsList();
	Hashtable<String, Finder> ht = new Hashtable<>(Fields.numMailFields);
	Iterator<String> it = arrayField.iterator();
	while (it.hasNext()) {
		String field = it.next();
		Grammar grammar = MGF.getGrammar(field);
		MailFinder mf = new MailFinder(grammar, field, event, language);
		try {
		Class<?> c = Class.forName("eventParser.mail." + field + "MailFinder");
			Object o = c.newInstance();
			mf = (MailFinder) o;
			mf.setGrammar(grammar);
			mf.setField(field);
			mf.setEvent(event);
			mf.setLanguage(language);
		} catch (Exception e) {
		}
		ht.put(field, mf);
	}
	StringBuilder sb = new StringBuilder();
	sb.append("Mail Finder List:").append("\n");
	for(String field : ht.keySet()){
		sb.append(field + "> finder: " +
ht.get(field).getClass().getName()).append("\n");
	}
	GELogger.add(this, sb.toString());
	return ht;
}
```

Observamos que esta función devuelve una tabla con todos los `Finders` especificados en la clase `Fields`. Utilizando el iterador, el proceso intenta crear un objeto `Finder` específico de la clase `<Campo>MailFinder` si esta clase existe. Si no existiese la clase se utilizaría un

objeto `Finder` genérico. De esta manera es transparente para el código y el Generador de Eventos en sí el saber si hay clases específicas o no. Finalmente al objeto `Finder` le asignamos las variables necesarias para que pueda operar: el objeto `Grammar` obtenido de la factoría de gramáticas (estudiada a continuación), el campo que identifica el buscador, el lenguaje del contenido y la referencia al evento `Event` por si el `Finder` específico tiene que poder acceder a algún campo ya encontrado.

La estructura de la clase `SmsFinderList` es equivalente pero para el *eventParser* de SMS.

3.4.2.4. Las clases MailGrammarFactory y SmsGrammarFactory

La clase `MailGrammarFactory` es la encargada de almacenar los patrones de búsqueda almacenados en el fichero `MailGrammar.xml`. Su código es el siguiente:

Código 3.27: MailGrammarFactory

```
public class MailGrammarFactory extends GrammarFactory {

	private XMLGrammarReader reader = null;
	private String language;

	public MailGrammarFactory(String language) {
		this.language = language;
		loadDocument();
	}

	private void loadDocument() {
		GELogger.add(this, "Cargando fichero de patrones...");
		String file = "mail/MailGrammar.xml";
		try {
			reader = new XMLGrammarReader(file);
			GELogger.add(this, "Fichero de patrones " + file + " cargado");
		} catch (Exception e) {
		GELogger.add(this, e, "Error al cargar el fichero de patrones: " + file);
		}
	}

	public Grammar getGrammar(String field) {
		if (reader == null) {
			return new Grammar();
		}
		ArrayList<Vector<String>> array = reader.getPatterns(language, field);
		return new Grammar(array);
	}

	public String getDefaultLanguage() {
		return reader.getDefaultLanguage();
	}
}
```

Esta clase extiende directamente de la clase `GrammarFactory` la cual definía como abstracto el método `getGrammar(String field)`. Cuando creamos la factoría de gramáticas en el proceso de creación de los buscadores, cargamos en el objeto `XMLGrammarReader` toda la estructura de datos que almacenan los patrones. Este proceso solo se realiza una única vez. Mediante el método `getGrammar`, se le asigna el objeto `Grammar` que contiene todos los patrones de un campo en concreto al `Finder` (ya sea específico o general).

3.4.2.5. Las clases MultimediaMailFinder y MMSFileExtractor

A la hora de guardar archivos multimedia se utilizan dos métodos muy diferenciados para los correos y para los MMS, por lo que analizaremos los dos casos por separado. Hay que destacar que ambos métodos de búsqueda no se lanzan junto a los demás buscadores sino que una vez creado el evento, se procede a la búsqueda de archivos multimedia y su almacenamiento tanto físicamente en la carpeta multimedia del proyecto como en BD. El principal motivo de que este proceso se realice después es que necesitamos el identificador del evento creado a la hora de poder almacenar y relacionar el objeto `Multimedia` con el objeto `Event`.

- MultimediaMailFinder.

Dentro de esta clase tenemos varias funciones que se van llamando encadenadamente una detrás de otra y además se recorren las diferentes partes de un correo recursivamente. El método público utilizado en la clase `GeneradorEventos` es `findFiles(Message message)`. Este método invoca el método privado interno `findFile(Part content)` con el mensaje original, el objeto `Message`. Un correo electrónico tiene una estructura de almacenamiento en árbol y cada una de sus partes es un objeto `Part`. Nos podemos encontrar un objeto `MimeBodyPart` que tendrá un tipo MIME asociado o un objeto `MultiPart` que será contenedor de otros objetos `Part`.

El correo entero es recorrido recursivamente buscando si un tipo `Part` es un archivo. Sabremos que es un archivo si el contenido tiene asociado un nombre de fichero. Una vez detectado un fichero, se ejecutará el método `insertarArchivo((MimeBodyPart) content)`. Este proceso creará un objeto multimedia que será guardado en base de datos y también se procederá al almacenado en el directorio "multimedia", ubicado en el directorio padre del proyecto. Si éste no existiese a la hora de guardar el primer archivo, se crearía.

Para guardar el archivo físicamente se utiliza un método propio del objeto `MimeBodyPart` encontrado:

```
part.saveFile(new File(multimediaFolder + fileName));
```

Hay que tener en cuenta que si el id de evento o el id de usuario es nulo no se podrán almacenar los ficheros obtenidos en base de datos.

- MMSFileExtractor.

La extracción de ficheros de un MMS es más sencilla porque el Emulador TCP nos envía directamente un `ArrayList` con todos los ficheros a almacenar de tipo `File`. Se aplica la misma lógica que en el punto anterior solo que en este caso se utiliza una función de la API *common.io.file* de Google para agilizar el trabajo:

```
File newFile = new File(multimediaFolder + fileName);
Files.copy(file, newFile);
```

3.5. Las expresiones regulares

El análisis, construcción y aplicación de las expresiones regulares es el pilar que sostiene la generación de un evento. Es la parte más flexible y la que más ha variado a lo largo del desarrollo del Generador de Eventos. Las expresiones regulares en Java están basadas en dos sencillas partes. Cada una de estas partes da lugar a un objeto en Java: el objeto `Pattern` (objeto con el patrón) y el objeto `Matcher` (objeto con las coincidencias), ambos contenidos en el paquete *java.util.regex*. El objeto `Pattern` define el patrón que se está buscando y el objeto `Matcher` se encarga de buscar el patrón en un cierto texto y devolver la parte que encaja con dicho patrón.

La potencia de las expresiones regulares reside en su capacidad para describir texto en lugar de especificarlo utilizando unas ciertas reglas que debe cumplir el texto coincidente. Cogiendo como ejemplo el Código 3.28, la salida por pantalla al ejecutar el fragmento es: `examina`, `el`, `el`, `en` y `el`. La variable `regex` contiene el patrón \be\w*\b, y describe las palabras que comienzan por la letra 'e'. Nótese que en Java el carácter '\' es un carácter de escape y para escribir '\' debemos escribir "\\" a la hora de definir el texto de un `String`. En la Tabla 3.5 se analiza el patrón de ejemplo.

Código 3.28: Ejemplo de análisis de aplicación de expresiones regulares.

```
String texto = "Matcher examina el resultado de aplicar el patrón en el texto.";
String regex = "\\be\\w*\\b";
Pattern p = Pattern.compile(regex);
Matcher m = p.matcher(texto);

String resultado = null;
while (m.find()) {
      resultado = m.group();
      System.out.println("Encontrado: " + resultado);
}
if (resultado == null) {
      System.out.println("No coincidencias");
}
```

Tabla 3.5: Análisis del patrón \be\w*\b

\b	Representa la frontera de una palabra. Lo más común es un espacio, tabulación, final de línea o comienzo de línea y signos de puntuación.
e	El carácter 'e'.
\w	Carácter de tipo alfanumérico, letras de la 'a' a la 'z', de la 'A' a la 'Z' y cualquier número.
*****	Cuantificador que indica que lo anterior, en este caso \w se repite cero veces o más, es decir, que puede no estar o estar indefinidas veces.
\b	Vuelve a indicar el principio de otra palabra.

En primera instancia se crea un objeto `Pattern p` con la expresión regular. Seguidamente se aplica el patrón a un cierto texto obteniendo un objeto `Matcher` que contendrá todas las coincidencias encontradas por nuestra expresión regular. Finalmente podemos consultar los resultados obtenidos en el objeto `Matcher m`.

En el apéndice Resumen de operadoresB.1 hay documentados los operadores más importantes de las expresiones regulares en Java. Para el estudio e implementación de los patrones se han utilizado como referencia el libro Java Regular Expressions: Taming the java.util.regex Engine ([D][D][D][D]) y el tutorial de Java sobre la API utilizada ([18]).

3.5.1. Los objetos Pattern y Matcher

El objeto `Pattern` se encarga de realizar una compilación de la expresión regular. No tiene método constructor, por lo que el error más frecuente es crear un objeto `Pattern` mediante él, en vez de utilizar su método estático `compile(regex)`. El método que utilizaremos sobre nuestro objeto es `matcher(texto)`, el cual retorna un objeto `Matcher` que buscará nuestra expresión regular en el `texto` pasado como parámetro. Sobre nuestro objeto tipo `Matcher` aplicaremos las búsquedas de coincidencias sobre el texto según el método adecuado. Para buscar las coincidencias del patrón invocaremos al método `find()`, `lookingAt()` o `matches()` que devuelven `cierto` en caso de encontrar la expresión regular o `falso` en caso contrario. Extraeremos su valor mediante el método `group()`.

- `find`: este método devuelve `cierto` si ha habido alguna coincidencia de la expresión compilada en el texto donde ha sido aplicada. Cada vez que se ejecuta `find`, y no ha sido reseteado, continúa con la siguiente coincidencia. Se utiliza para recorrer el conjunto de coincidencias y es el método principal de búsqueda utilizado.
- `lookingAt`: devuelve `cierto` si el inicio del texto coincide con el patrón, `falso` en caso contrario. Útil para averiguar si un cierto texto comienza de una determinada forma.
- `matches`: devuelve `cierto` si el texto coincide íntegramente con el patrón compilado.
- `group`: devuelve el texto que ha sido encontrado por el patrón y referenciado por `find`. Cada vez que se ejecuta `find`, `group` adquiere el texto de la siguiente coincidencia. Puede pasarse como parámetro el valor del subgrupo de la expresión regular a encontrar, siendo por defecto si no se especifica ninguno el patrón entero.

A parte de estos métodos existen otros que son necesarios ser mencionados, el método `start()` y `end()` devuelven el índice del inicio y el final de la coincidencia en el texto analizado. El método `replaceAll(reemplazo)` se puede utilizar para remplazar todas las coincidencias del patrón por un cierto `String`.

La mayoría de funciones comentadas en este apartado son utilizadas por los Buscadores.

3.5.2. Creación de patrones

Existen varias técnicas para la creación de una expresión regular, para patrones desde cero podemos partir de lo que se quiere encontrar e ir generalizando todo lo posible mediante símbolos y expresiones que engloban los resultados deseados y descartan los demás. A partir de un patrón podemos realizar ligeras modificaciones para poder crear uno nuevo. También podemos concatenar varios patrones pequeños para crear uno mayor. Hay que decir que no existe una única expresión regular para especificar unas ciertas reglas, el lenguaje de expresiones es lo suficientemente amplio como para poder permitir expresar el patrón de varias maneras diferentes, todas ellas igual de válidas.

3.5.2.1. Ejemplo de patrón de fecha

Es igual de válido expresar 0|1|2|3|4|5|6|7|8|9, como [0123456789], como [0-9], como \d, siendo la más elegante ésta última.

A continuación vamos a crear un patrón que encuentre fechas que cumplen el siguiente criterio: día/mes/año, como 23/12/2020

- Nos centraremos primero en hallar los números. Para hallar un número utilizaremos el patrón \d, por lo que dos números seguidos se obtendrán mediante la expresión regular \d\d. Esto es aplicable al día y el mes y generalizando, para el año será \d\d\d\d. Finalmente especificaremos el delimitador '/' obteniendo el patrón \d\d/\d\d/\d\d\d\d.
- Una vez obtenida una primera aproximación el patrón debe pulirse. Lo primero que salta a la vista es que estamos obligando que haya dos dígitos obligatorios para el día y el mes. Este patrón no nos haría encontrar fechas como 2/1/2020. Vamos a introducir los caracteres especiales '{' y '}' que denotan cuantificación quedando el patrón: \d{1,2}/\d{1,2}/\d{4}.

  \d{1,2} especifica que el símbolo \d debe aparecer como mínimo una vez y además no más de dos veces por lo que engloba los patrones \d y \d\d.

  \d{4} especifica que \d debe aparecer exactamente cuatro veces.

- Podríamos mejorar el patrón añadiendo más delimitadores de fechas: \d{1,2}[-\\ /_]\d{1,2}[-\\ /_]\d{4}). Se permite así hallar fechas en las que sus dígitos están separadas por '-', '\', ' ' (un espacio en blanco), '/' o '_'.

El último patrón fue el creado en la primera fase de desarrollo para encontrar fechas. Con él se probó el Generador de Eventos extremo a extremo y ha sido dejado para la aplicación de reglas de análisis en el archivo de gramáticas de SMS.

3.5.2.2. Patrón de fecha extendido

Analizando detenidamente el patrón creado observamos que es capaz de encontrar las fechas con varios formatos, pero en muchos casos no nos basta con encontrar lo deseado sino que además el patrón debe descartar lo no deseado.

El patrón actual daría por buena una fecha como 88/5/9063. Claramente para una mente humana la sucesión de números no tiene un valor de fecha, el año y el mes si que encajan, pero no en el ámbito de fechas actuales dentro del Generador de Eventos, es cierto que puede llegar a existir mayo del año 9063, pero no tiene sentido la creación de eventos para tal fecha. Respecto al día, vemos claramente como \d{1,2} hace encontrar números del 0 al 9 y del 00 al 99, cuando se sabe que el día solo puede ir del 1 al 31; Lo mismo se aplica al mes que va del 1 al 12; incluso podríamos limitar el año a la centena actual del 2000 al 2099.

El objetivo del siguiente patrón mejorado es descartar todos los resultados no deseables, para ello vamos a especificar qué rango debe tener cada dígito:

- Para el día especificaremos que el primer dígito sea 1, 2 o 3 [123] y que el segundo sea cualquier dígito, obteniendo [123]\d.

El patrón [123]\d nos permite encontrar los días del 10 al 39, como hemos especificado cada uno de los dos dígitos no podemos utilizar el cuantificador "{}" por lo que tendremos que añadir que encuentre también los días del 1 al 9.

La parte [1-9]|[123]\d encuentra los días del 1 al 39. El operador '|' aplica una OR de las condiciones de izquierda y derecha. Especificaremos ahora además que la trentena debe ser 30 o 31 obteniendo [1-9]|[12]\d\|3[01].

Si queremos que soporte los días que empiecen por cero como 09 añadiremos que el 0 inicial sea opcional: 0?[1-9]|[12]\d|3[01]

- Aplicando el mismo procedimiento al mes su patrón quedaría: 0?[1-9]|1[012]
- Para el año simplemente fijaremos los dos primeros dígitos 20\d\d. La combinación total da lugar al patrón día/mes/año:

  (0? [1-9]|[12]\d|3[01])[-\\ /_](0?[1-9]|1[012])[-\\ /_](20\d\d)

  Nótese que para concatenar los diferentes sub-patrones hemos utilizado '(' ')', de esta forma delimitamos el alcance de la OR '|'

Este patrón puede utilizarse para determinar si un cierto texto es una fecha o no. En caso de que lo sea, el patrón nos dará el texto como encontrado. Si lo que se desea es buscar una fecha en un texto tendremos que poner limitadores delante y detrás de nuestro patrón. Añadiremos \D delante y detrás para forzar que el carácter inmediatamente anterior o posterior no sean dígitos y de esta manera no interfieran en el patrón:

\D(0? [1-9]|[12]\d|3[01])[-\\ /_](0?[1-9]|1[012])[-\\ /_](20\d\d)\D

3.5.2.3. Los grupos de los patrones

Los paréntesis permiten obtener grupos de resultados. Podremos obtener el texto que cuadra con el patrón haciendo referencia a su grupo. Los grupos van ordenados por orden de aparición.

El grupo 0 hace referencia a toda la expresión regular:

\D(0? [1-9]|[12]\d|3[01])[-\\ /_](0?[1-9]|1[012])[-\\ /_](20\d\d)\D

El grupo 1 hace referencia a (0? [1-9]|[12]\d|3[01]), el grupo 2 a (0?[1-9]|1[012]) y el grupo 3 a (20\d\d). Si aplicamos el patrón al texto "fui el 22/12/2012 a comprar". Si solo nos quedamos con el grupo 3 obtendríamos el resultado 2012. Los grupos siempre se cuentan de izquierda a derecha empezando desde el paréntesis de apertura.

Si queremos que un grupo no sea captable, es decir, no forme parte del cómputo de grupos lo indicaremos mediante los símbolos "?:", por ejemplo (?:\d) sería un subgrupo con \d pero no captable o *anónimo*; aun así seguiría existiendo el grupo 0 que da lugar a aplicar todo el patrón.

El anterior patrón es el escogido para el análisis de los correos electrónicos. Aun así podemos encontrar patrones mucho más elaborados y robustos como el siguiente, que detecta fechas de años bisiestos:

```
^(?:(?:(?:0?[13578]|1[02])(\/|-|\.)31)\1|(?:(?:0?[1,3-9]|1[0-2])(\/|-|\.)(?:29|30)\2))
(?:(?:1[6-9]|[2-9]\d)?\d{2})$|
^(?:0?2(\/|-|\.)29\3(?:(?:(?:1[6-9]|[2-9]\d)?(?:0[48]|[2468][048]|[13579][26])|
(?:(?:16|[2468][048]|[3579][26])00))))$|
^(?:(?:0?[1-9])|(?:1[0-2]))(\/|-|\.)(?:0?[1-9]|1\d|2[0-8]) \4(?:(?:1[6-9]|[2-9]\d)?\d{2})$
```

Hemos visto la potencia de descripción de texto de las expresiones regulares, a parte de su uso como buscador de coincidencias en un texto también pueden ser utilizadas como un método de búsqueda y reemplazo, validación de datos y separación de cadenas de texto. Aunque es cierto que las expresiones regulares son una herramienta muy potente no se debe abusar de ellas, en muchas ocasiones aunque estemos pensando en la solución mediante expresiones regulares pueden existir métodos más sencillos para hallar la solución del problema.

3.5.3. Aplicación de los patrones

Observando el ejemplo de fecha del punto anterior, nos damos cuenta que las expresiones regulares más elaboradas son más difíciles de interpretar y, en caso de malfuncionamiento, de depurar y corregir, necesitando volver a analizar el patrón paso a paso. Como la aplicación podía sufrir algunos cambios durante el desarrollo, y sabiendo que muchos patrones tenían que reajustarse a medida que aparecían mas correos y SMS de prueba, era más interesante aplicar un patrón troceado en varias porciones. En el ejemplo de la fecha, su expresión regular se podría haber separado en 3 patrones diferentes y ser analizados uno detrás del otro. En caso de haber una coincidencia de alguno de ellos se daría como patrón encontrado y en caso de fallar los tres, como no encontrado.

La estructura del Generador de Eventos está basada en la aplicación de múltiples patrones de forma que sea más fácil su mantenimiento, aún así, el número total de patrones creados es elevado y nos podemos encontrar con muchos patrones para buscar el campo de un evento. Tiene mucho sentido el poder separar la aplicación de los patrones al texto enviado por el usuario, poniendo en primer lugar los que busquen situaciones más concretas y dejando para el final los patrones más generales que son los que pueden hacer que haya coincidencias no deseadas a la hora de rellenar el campo de un evento.

Hay una delgada línea entre encontrar algo no deseado y no encontrarlo. Bastará con eliminar de la lista de patrones a aplicar los más generales en ese caso, dejando solo los concretos. Optimizando la lista de patrones en uno solo (con múltiples operadores OR) nos daría un posible detrimento de la velocidad; y decimos posible porque si hay una elevada probabilidad de encontrar los patrones más concretos, se ahorra el tiempo de compilar todos los siguientes patrones y aplicarlos al texto. Pero a su vez, si ninguno de los patrones generados casa en el texto, habremos compilado y aplicado muchos patrones en vez de solo uno; eso sí, perderíamos el orden de ejecución y la facilidad de mantenimiento.

Se puede observar dicha optimización y generación de patrones más robustos en la fase de creación de gramáticas para análisis de correos electrónicos, quedándose los más sencillos de la primera fase de desarrollo par los SMS.

3.5.4. Almacenamiento de patrones

Hay muchos patrones y su manipulación mediante el objeto `String` es tediosa ya que tiene interpretación propia de las contra-barras. Como también es un carácter especial para la definición de expresiones regulares, hace que el patrón declarado mediante un `String` que buscase una contra-barra fuese \\\\.

Todos los patrones de búsqueda diseñados para cada campo del evento se guardan en un fichero de texto. La principal ventaja que tiene es que se pueden modificarse sin tener que compilar de nuevo la clase afectada. Tener las expresiones en un fichero de texto hace su mantenimiento más cómodo y más si pensamos que si guardamos el patrón en un Objeto `String` debemos de doblar todos los '\'.

El método más recomendado para almacenar las expresiones regulares es el de utilizar `FileChannels`, `ByteBuffers` o cualquier método de lectura de ficheros a nivel de byte. Es desaconsejable totalmente el uso de archivos .`properties`. En este tipo de archivos se deben guardar los patrones tal como guardaríamos en un objeto `String`, además se debe tener en cuenta como trata la lectura de dichos archivos caracteres como '\n', '\t', '\\' o el delimitador de salto de línea '\' (de los fichero `properties`). Aún desaconsejándose en cierta medida los archivos escritos en XML por la aparición de ciertos caracteres especiales se ha utilizado dicho método por la estructuración que se le ha dado al fichero de gramáticas.

De esta forma se ha decido guardar los patrones en un fichero XML para aprovechar y ampliar conocimientos de lectura de dichos ficheros, salvando los imprevistos comentados de los caracteres especiales y aprovechándonos de la estructura jerárquica de los datos guardados en un fichero XML. Como en la creación de los patrones no ha supuesto ningún grave inconveniente el guardado de los patrones en dicho formato se ha mantenido, en caso contrario se habría aplicado algún método también recomendado como es el guardado en base de datos.

3.5.5. Adecuación entre contenido y patrones

Para que los patrones de creación se ajusten al mayor número de contenidos diferentes se han aplicado una serie de reglas para trabajar antes de realizar las búsquedas.

A todo contenido se les han aplicado reglas de limpieza de símbolos raros, acentos y todo lo relacionado con etiquetas HTML si formaban parte de un correo con ese formato. También se transforma el contenido a mayúsculas antes de realizar la búsqueda en él. Así mismo, los patrones aunque estén escritos en minúsculas o mayúsculas también son pasados a mayúsculas para que la similitud entre el contenido y el patrón sea máxima. Además se les aplica la misma operación de limpieza de quitar acentos que en el contenido a analizar.

3.5.6. Eficiencia de los patrones

Se han de aplicar algunas técnicas para elaborar patrones que sean más eficientes en tiempo de ejecución, optimizando así los programas que hacen uso de ellos. A continuación se describen algunos métodos que tendremos en cuenta en la creación de los patrones:

- Usar grupos no captables siempre que sea posible. Capturar los grupos implica que la máquina virtual les siga el rastro. Es muy útil si queremos extraer los grupos a posteriori. Es el método a aplicar para crear unidades lógicas. Por ejemplo, el patrón coche (?:rojo|azul|negro|blanco) agrupa varios colores en una unidad lógica que no es necesario capturarse.
- Verificar los candidatos. Si buscamos `Strings` específicas, se puede ahorrar mucho tiempo si podemos descartar el texto a analizar *a priori*. Por ejemplo, si buscamos un correo electrónico en un cierto texto, no tiene sentido realizar la búsqueda de la expresión regular de la dirección si en ese texto no aparece el símbolo '@'. Podemos buscar la arroba y acto seguido si es encontrada buscar el correo mediante el patrón.
- Poner siempre delante la alternativa más probable. En el ejemplo del patrón coche (?:rojo|azul|negro|blanco), si sabemos que el color más común es el negro, deberíamos reordenar el grupo así (?:negro|rojo|azul|blanco). Se consigue reducir el procesado ya que el candidato se encuentra antes.
- Especificar todo lo posible. Cuanto más específica es la expresión más rápida es, porque tiene que comprobar menos combinaciones de símbolos. Si se sabe con certeza que el patrón debe contener una cierta palabra, debe usarse esta especificación. Por ejemplo, si queremos buscar tres dígitos seguidos usaremos el patrón \d\d\d, pero si sabemos a ciencia cierta que el primero siempre va a ser '9' lo especificaremos en el patrón quedando 9\d\d.
- Especificar la posición de la coincidencia. Si se sabe con certeza que el texto candidato aparece al principio de línea lo especificaremos. Por ejemplo, si queremos encontrar frases que empiezan por "Hola" el patrón será ^Hola. La potencia de la especificación reside en poder parar la búsqueda si no se cumple la condición inicial. En este caso solo se analizarían los dos primeros caracteres del patrón y se seguiría analizando si éstos coincidiesen con el patrón.
- Especificar el tamaño de la coincidencia. Siempre se debe declarar el número de coincidencias y reducir el uso de los operadores de cantidad. \w\w\w\w+ es mucho más eficiente que \w+ si sabemos a ciencia cierta que van a haber como mínimo tres caracteres.
- Limitar el alcance de las alternativas. Cuantas menos alternativas se tengan que evaluar mejor. Es mucho más eficiente el patrón Buenas (?:Tardes|Noches) que Buenas Tardes|Buenas Noches ya que en el primer caso no se toma ninguna decisión hasta que se ha encontrado la palabra "Buenas". El segundo patrón en cambio es buscado dos veces, una para "Buenas Tardes" y otra para "Buenas Noches".

3.6. Las expresiones regulares de los Buscadores

Se ha definido hasta este punto cómo se generan o representan las expresiones regulares y cómo van a ser almacenadas en la aplicación. En el presente punto vamos a mencionar los detalles de las expresiones regulares más características que emplearán los buscadores; no entrando en detalle en todas ya que muchas pueden derivarse de las anteriores realizando algún pequeño cambio y son en esencia muy parecidas.

3.6.1. Patrones de generación automática y generación semiautomática.

La mayor diferencia que existe entre el archivo de gramáticas de correos y de SMS es que el primero tiene un formato evolucionado a partir del segundo. El formato de expresiones regulares del SMS es muy simple porque las *regex* están orientadas a etiquetas mientras que las *regex* de correo soportan también un mayor número de etiquetas y de búsquedas alternativas. La mayor diferencia reside sobretodo en la estructura del fichero XML que las contiene. Hay que destacar que la clase `XMLGrammarReader` es capaz de leer tanto el formato antiguo contenido en el fichero `SmsGrammar.xml` como el nuevo más versátil del fichero `MmsGrammar.xml`.

Para los correos, el conjunto de patrones aplicados para la búsqueda de campos en la generación automática es el mismo que el aplicado en la generación semiautomática. De cara al Generador de Eventos ambas generaciones son iguales, es más, el proceso es único. Los patrones están implementados de manera que se le otorga más prioridad a las estructuras sintácticas de la generación semiautomática, por lo que si el correo ha sido generado por el usuario mediante una plantilla ésta será interpretada correctamente ya que no se dará lugar a la aplicación de otros patrones más generalizados (para correos automáticos sin estructura fija o plantilla).

La totalidad de expresiones regulares utilizadas se puede ver en los apéndices B.2 y B.3. En los siguientes puntos comentaremos la estructura, cómo están guardados y la explicación de una selección de expresiones regulares.

3.6.2. Patrones para SMS

Observamos la estructura en árbol del fichero de gramáticas de SMS en el Código 3.29. En el nodo raíz *Grammar* tenemos el identificador del conjunto de patrones que se utilizará por defecto si ningún diccionario es cargado, tiene que coincidir con los nombres de las ramas “ENG” o “ESP”. Para que los patrones sean cargados correctamente, en función del idioma del contenido a analizar, tenemos que tener creada una rama de patrones por cada diccionario definido en la aplicación.

Seguidamente comienza un conjunto de grupos. Cada grupo tiene que tener el mismo nombre que los campos a buscar definidos en la clase `Fields`. Por cada campo pondremos todos los patrones uno de detrás de otro entre las etiquetas `<Pattern>` y `</Pattern>`. Nótese a simple vista que en formato XML hay caracteres especiales y que por ejemplo, se debe almacenar el carácter ‘&’ como “&”.

Código 3.29: Extracto de gramáticas de SMS

```
<Grammar defaultLanguage="ESP">
    <ENG>
    <Title>
        <Pattern>S[ ]?[:][ ]?([\w [\p{Punct}&amp;&amp;[^\.:]]]*)[\.]</Pattern>
        <Pattern>SUB\w*[ ]?[:][ ]?([\w [\p{Punct}&amp;&amp;[^\.:]]]*)[\.]</Pattern>
        <Pattern>TIT\w*[ ]?[:][ ]?([\w [\p{Punct}&amp;&amp;[^\.:]]]*)[\.]</Pattern>
    </Title>
    <Date>
        <Pattern>D[ ]?[:][ ]?(\d{1,2}[-\\ /_]\d{1,2}[-\\ /_]\d{4})[ ]?[\.]?</Pattern>
        <Pattern>D[ ]?[:][ ]?(\d{1,2}[-\\ /_]\d{1,2}[-\\ /_]\d{2})[ ]?[\.]?</Pattern>
        <Pattern>D[ ]?[:][ ]?(\d{1,2}[-\\ /_]\d{1,2})[ ]?[\.]?</Pattern>
        <Pattern>DAT\w*[ ]?[:][ ]?(\d{1,2}[-\\ /_]\d{1,2}[-\\ /_]\d{4})[ ]?[\.]?</Pattern>
        <Pattern>DAT\w*[ ]?[:][ ]?(\d{1,2}[-\\ /_]\d{1,2}[-\\ /_]\d{2})[ ]?[\.]?</Pattern>
        <Pattern>DAT\w*[ ]?[:][ ]?(\d{1,2}[-\\ /_]\d{1,2})[ ]?[\.]?</Pattern>
        <Pattern>(\d{1,2}[-\\/]\d{1,2}[-\\/]\d{4})</Pattern>
        <Pattern>(\d{1,2}[-\\/]\d{1,2}[-\\/]\d{2})</Pattern>
        <Pattern>(\d{1,2}[-\\/]\d{1,2})</Pattern>
    </Date>
      ...
    </ENG>

    <ESP>
    <Title>
        <Pattern>T[ ]?[:][ ]?([\w [\p{Punct}&amp;&amp;[^\.:]]]*)[\.]</Pattern>
        <Pattern>TEM\w*[ ]?[:][ ]?([\w [\p{Punct}&amp;&amp;[^\.:]]]*)[\.]</Pattern>
        <Pattern>TIT\w*[ ]?[:][ ]?([\w [\p{Punct}&amp;&amp;[^\.:]]]*)[\.]</Pattern>
    </Title>
    <Date>
        <Pattern>F[ ]?[:][ ]?(\d{1,2}[-\\ /_]\d{1,2}[-\\ /_]\d{4})[ ]?[\.]?</Pattern>
        <Pattern>F[ ]?[:][ ]?(\d{1,2}[-\\ /_]\d{1,2}[-\\ /_]\d{2})[ ]?[\.]?</Pattern>
        <Pattern>F[ ]?[:][ ]?(\d{1,2}[-\\ /_]\d{1,2})[ ]?[\.]?</Pattern>
        <Pattern>FEC\w*[ ]?[:][ ]?(\d{1,2}[-\\ /_]\d{1,2}[-\\ /_]\d{4})[ ]?[\.]?</Pattern>
        <Pattern>FEC\w*[ ]?[:][ ]?(\d{1,2}[-\\ /_]\d{1,2}[-\\ /_]\d{2})[ ]?[\.]?</Pattern>
        <Pattern>FEC\w*[ ]?[:][ ]?(\d{1,2}[-\\ /_]\d{1,2})[ ]?[\.]?</Pattern>
        <Pattern>(\d{1,2}[-\\/]\d{1,2}[-\\/]\d{4})</Pattern>
        <Pattern>(\d{1,2}[-\\/]\d{1,2}[-\\/]\d{2})</Pattern>
        <Pattern>(\d{1,2}[-\\/]\d{1,2})</Pattern>
    </Date>
    </ESP>
</Grammar>
```

También se puede apreciar que los patrones de título para cada idioma son idénticos, salvo por la parte inicial del patrón donde se encuentra la etiqueta que identifica el campo del evento. Para gestionar mejor el almacenamiento cuando el número de palabras clave crece se aplicará el método para guardado utilizado en el fichero de gramáticas de correos del próximo apartado 3.6.3.

3.6.2.1. Patrones para el título

El título tiene el tipo de patrón por defecto para capturar contenido de tipo texto. Los patrones que existen para este campo son:

- Inglés [ENG]

  S[ ]?[:][ ]?([\w [\p{Punct}&&[^\.:]]]*)[\.]
  SUB\w*[ ]?[:][ ]?([\w [\p{Punct}&&[^\.:]]]*)[\.]
  TIT\w*[ ]?[:][ ]?([\w [\p{Punct}&&[^\.:]]]*)[\.]

- Español [ESP]

  T[ ]?[:][ ]?([\w [\p{Punct}&&[^\.:]]]*)[\.]
  TEM\w*[ ]?[:][ ]?([\w [\p{Punct}&&[^\.:]]]*)[\.]
  TIT\w*[ ]?[:][ ]?([\w [\p{Punct}&&[^\.:]]]*)[\.]

La diferencia entre los patrones anteriores se reduce el inicio del patrón, la parte que hace referencia a la etiqueta. Se detalla el análisis de los dos siguientes patrones:

```
S[ ]?[:][ ]?([\w [\p{Punct}&&[^\.:]]]*)[\.]
SUB\w*[ ]?[:][ ]?([\w [\p{Punct}&&[^\.:]]]*)[\.]
```

Parte inicial, detección de campo (Etiqueta):

S	Literalmente el carácter S, debe aparecer obligatoriamente (inicial forzada)
[ ]?	Espacio opcional (1 espacio o ninguno)
[:]	Carácter : obligatorio
[ ]?	Espacio opcional (1 espacio o ninguno)

SUB	Literalmente los caracteres SUB (etiqueta extendida)
\w*	Cualquier carácter de tipo palabra
[ ]?	Espacio opcional (1 espacio o ninguno)
[:]	Carácter : obligatorio
[ ]?	Espacio opcional (1 espacio o ninguno)

Parte central: obtención del contenido del campo:

```
([\w [\p{Punct}&&[^\.:]]]*)
```

(	Inicio de grupo captable
[	Inicio de agrupación
\w	Cualquier carácter de tipo palabra
	Espacio en blanco
	[Inicio de sub-agrupación
	\p{Punct} Cualquier carácter del listado: !"#$%&'()*+,-./:;<=>?@[\]^_`{|}~
	&& Y además
	[Inicio de sub-agrupación
	^ Que no sea
	\. Ni punto
	: Ni dos puntos
	] Fin de sub-agrupación
	] Fin de suba-grupación
]	Fin de agrupación
*****	Ninguna o más veces
)	Fin de grupo captable

Para poder obtener el contenido del grupo, éste es de tipo captable (sin “?:” en el inicio).

Final del campo:

```
[\.]
```

[\.]	Punto final obligatorio para delimitar el campo.

- Etiquetas:

Tabla 3.6: Etiquetas del título para SMS

Idioma	Etiqueta Corta	Etiqueta extendida	Palabras clave
Inglés	S	SUB* / TIT*	Subject / Title
Castellano	T	TIT* / TEM*	Título / Tema

3.6.2.2. Patrones para la calle y la ciudad

Los patrones para los campos calle y ciudad comparten el mismo patrón, por lo que pueden escribirse ambos campos con el mismo identificador. Para diferenciar el final del primero e inicio del segundo se debe indicar mediante el carácter coma ‘,’. También se permite la determinación de cada uno por separado mediante etiquetas extendidas.

Patrón genérico para calle:

```
L[ ]?[:][ ]?([\w [\p{Punct}&&[^\.:]][ºª]]*),[\w ]*[\.]
```

Patrón genérico para ciudad:

```
L[ ]?[:][ ]?[\w [\p{Punct}&&[^\.:]][ºª]]*,([\w ]*)[\.]
```

Los patrones son muy similares a los del título, solo que se añaden los caracteres **‘o’** y **‘a’** como opcionales. Se fija que debe aparecer la coma como delimitador **‘,’** y seguidamente se incluye una segunda agrupación de elementos de palabras y espacios de aparición nula o múltiple **([\w]*)** para que se dé la concordancia de la segunda posible aparición del patrón. Mediante los paréntesis se obtiene la parte de la expresión que interesa: la calle o la ciudad, leyendo el grupo que corresponda según el campo del evento que se está buscando. Nótese que si obtenemos el grupo 1 en el patrón de calle obtenemos la concordancia anterior a la coma, mientras que el primer grupo para el patrón de ciudad es el posterior a la coma.

- Etiquetas (patrón compartido para calle y ciudad):

Tabla 3.7: Etiquetas de la calle y la ciudad para SMS.

Idioma	Etiqueta Corta	Etiqueta extendida	Palabras clave
Inglés	L	LOC*	Location
Castellano	L	LUG*	Lugar

El patrón de búsqueda del campo calle o dirección por separado queda:

```
STR\w*[ ]?[:][ ]?([\w [\p{Punct}&&[^\.:]][ºª]]*)[\.]
```

En este caso solo incorpora el primer bloque de captura del patrón conjunto, es decir, la captura de palabras hasta la coma.

- Etiquetas (patrón solo para Calle):

Tabla 3.8: Etiquetas de la calle para SMS.

Idioma	Etiqueta Corta	Etiqueta extendida	Palabras clave
Inglés	-	STR* / ADD*	Street / Address
Castellano	-	CALL* / DIR*	Calle / Dirección

Patrón de búsqueda del campo ciudad (por separado):

```
CIT\w*[ ]?[:][ ]?([\w ]*)[\.]
```

El patrón en este caso solo incorpora el segundo bloque de captura del patrón conjunto, es decir, la captura de palabras de después de la coma.

- Etiquetas (patrón solo para ciudad):

Tabla 3.9: Etiquetas de la ciudad para SMS.

Idioma	Etiqueta Corta	Etiqueta extendida	Palabras clave
Inglés	-	CIT*	City
Castellano	-	CIU*	Ciudad

3.6.2.3. Patrones para el código postal

El patrón del código postal tiene que soportar todos los formatos que incluyen letras y números.

Patrón de búsqueda del código postal:

```
Z[ ]?[:][ ]?([\w-]*)[\.]
ZIP\w*[]?[:][ ]?([\w-]*)[\.]
```

La parte encargada de buscar el contenido está formada por la expresión **([\w-]*)**. Son captados todos los caracteres alfanuméricos y además el guion '**-**'.

- Etiquetas:

Tabla 3.10: Etiquetas del código postal para SMS.

Idioma	Etiqueta Corta	Etiqueta extendida	Palabras clave
Inglés	Z	ZIP* / COD*	ZipCode / Code
Castellano	C	COD*	Código Postal

Nota: En Inglés la 'C' está reservada para el país: "Country".

3.6.2.4. Patrones para el país

El país tiene como patrones:

```
C[ ]?[:][ ]?([\w ]*)[\.]
COU\w*[ ]?[:][ ]?([\w ]*)[\.]
```

Son patrones que capturan texto y permiten espacios.

- Etiquetas:

Tabla 3.11: Etiquetas del país para SMS.

Idioma	Etiqueta Corta	Etiqueta extendida	Palabras clave
Inglés	C	COU*	Country
Castellano	P	PAI*	País

3.6.2.5. Patrones para la fecha

La fecha tiene uno de los patrones más específicos. Tiene como objetivo el encontrar literales de texto expresados en diferentes formatos de tipo fecha. La parte inicial y final del patrón es idéntica. La única variación está en el grupo de captura del contenido del campo.

Patrones de búsqueda de la fecha:

```
D[ ]?[:][ ]?(\d{1,2}[-\\ /_]\d{1,2}[-\\ /_]\d{4})[ ]?[\.]?
D[ ]?[:][ ]?(\d{1,2}[-\\ /_]\d{1,2}[-\\ /_]\d{2})[ ]?[\.]?
D[ ]?[:][ ]?(\d{1,2}[-\\ /_]\d{1,2})[ ]?[\.]?
```

Los mismos patrones los tenemos duplicados con la etiqueta extendida:

```
DAT\w*[ ]?(\d{1,2}[-\\ /_]\d{1,2}[-\\ /_]\d{4})[ ]?[\.]?
DAT\w*[ ]?(\ (\d{1,2}[-\\ /_]\d{1,2}[-\\ /_]\d{2})[ ]?[\.]?
DAT\w*[ ]?(\d{1,2}[-\\ /_]\d{1,2})[ ]?[\.]?
```

El cuerpo del patrón se compone de tres secciones que intentan encontrar cadenas de texto del tipo: “día” <separador> “mes” <separador> “año”. El día, el mes y el año deben ser numéricos y se contemplan como separadores los caracteres ‘-’, ‘ ’ (espacio), ‘\’ y ‘/’. El patrón de día y mes **\d{1,2}** especifica que deben aparecer forzosamente uno o dos dígitos, para el año existe variación que da lugar a los tres diferentes patrones: que sean cuatro dígitos **\d{4}**, dos **\d{2}** o no haya ninguno (incluyendo su separador).

Los tres patrones solo se diferencian en la parte final, el primero casará con años de cuatro cifras, el segundo con años expresados en la terminación de dos cifras y el último para hallar fechas sin año especificado. A parte de los juegos de patrones con etiquetas, se añaden los bloques anteriores de captura sin parte inicial (etiqueta) y final (‘.’). De esta manera es posible hallar fechas en el cuerpo del mensaje dándole un cierto punto de búsqueda automática de fechas.

Patrones sin etiqueta:

```
(\d{1,2}[-\\/]\d{1,2}[-\\/]\d{4})
(\d{1,2}[-\\/]\d{1,2}[-\\/]\d{2})
(\d{1,2}[-\\/]\d{1,2})
```

- Etiquetas:

Tabla 3.12: Etiquetas de la fecha para SMS.

Idioma	Etiqueta Corta	Etiqueta extendida	Palabras clave
Inglés	D	DAT*	Date
Castellano	F	FEC*	Fecha

3.6.2.6. Patrones para la hora

El campo hora es muy similar conceptualmente al de la fecha solo que existe menos variedad en el formato. Tenemos los tres tipos de patrones: el de la inicial, el de etiqueta extendida y un último sin etiqueta análogamente a lo comentado para el campo fecha (permitir encontrar horas sin referencias).

```
T[ ]?[:][ ]?(\d{1,2}[:]\d\d)[ ]?[\.]?
TIM\w*[ ]?[:][ ]?(\d{1,2}[:]\d\d)[ ]?[\.]?
(\d{1,2}[:]\d\d)
```

El cuerpo del patrón se reduce al poder hallar cadenas de texto que representen una hora formateada como “horas” <separador> “minutos”. Para las horas se permite que sean uno o dos dígitos **\d{1,2}**, para los minutos se fuerzan dos dígitos **\d\d**. En el caso de la hora el único separador permitido son los dos puntos ‘:’.

- Etiquetas:

Tabla 3.13: Etiquetas de la hora para SMS.

Idioma	Etiqueta Corta	Etiqueta extendida	Palabras clave
Inglés	T	TIM*	Time
Castellano	H	HOR*	Hora

3.6.2.7. Patrones para la descripción

Los patrones de la descripción permiten capturar texto y algunos signos especiales:

```
X[ ]?[:][ ]?([\w [\p{Punct}&amp;&amp;[^\.:]][ºª]]*)[\.]
DES\w*[ ]?[:][ ]?([\w [\p{Punct}&amp;&amp;[^\.:]][ºª]]*)[\.]
```

- Etiquetas:

Tabla 3.14: Etiquetas de la descripción para SMS.

Idioma	Etiqueta Corta	Etiqueta extendida	Palabras clave
Inglés	X	DES*	Description
Castellano	D	DES*	Descripción

Nota: En Inglés la ‘D’ está reservada para la fecha “Date”.

3.6.2.8. Patrones para la categoría

Son patrones de tipo texto alfanumérico con espacios:

```
N[ ]?[:][ ]?([\w ]*)[\.]
NAT\w*[ ]?[:][ ]?([\w ]*)[\.]
```

- Etiquetas:

Tabla 3.15: Etiquetas de la categoría para SMS.

Idioma	Etiqueta Corta	Etiqueta extendida	Palabras clave
Inglés	N	NAT* / CAT*	Nature / Category
Castellano	N	CAT*	Categoría

Nota: En Castellano la ‘C’ está reservada para el “Código Postal”.

3.6.2.9. Patrones para la privacidad

La privacidad es un alfanumérico, no permite espacios, y sus patrones son:

```
P[ ]?[:][ ]?([\w]*)[\.]
PRI\w*[ ]?[:][ ]?([\w]*)[\.]
```

Para la correcta detección de la privacidad se deben indicar las palabras clave: “PRI”, “PRIVATE” o “PRIVADO” para crear un evento de tipo privado; “REG”, “REGISTERED” o “CERTIFICADOS” para crear un evento para controlar el acceso a usuarios registrados y “PUB”, “PUBLIC” o “PUBLICO” para eventos públicos. En cualquier otro caso se creará el evento con la privacidad por defecto de la aplicación que dependerá de su política de privacidad (por defecto “PRIVADO”).

Tabla 3.16: Etiquetas de la privacidad para SMS.

Idioma	Etiqueta Corta	Etiqueta extendida	Palabras clave
Inglés	P	PRI*	Privacity
Castellano	A	ACC*	Acceso

Nota: En Castellano la ‘P’ está reservada para el “País”.

3.6.3. Patrones para e-mails

Los patrones para mails son más extensos y siguen las mismas particularidades que para los SMS. Para este caso resulta más interesante el formato XML que almacena la estructura y solo veremos algunos ejemplos de patrones de ejemplo explicando las mayores diferencias con respecto a los SMS. La totalidad de los patrones se encuentra en el apéndice B.3.

En el siguiente fragmento solo mostraremos parte de la estructura nueva, la base de la estructura de nodo principal y estructuración del fichero por idioma y campos a buscar se mantiene.

Código 3.30: Extracto de gramáticas de e-mail – keywords

```
<Title>
    <Keyword id="#K1#">Titulo, Tema, Evento, Espectaculo</Keyword>
    <Pattern>#K1#[ ]?#?[:=][ ]?([\w[\p{Punct}&amp;&amp;[^\.:#=,;]]][\w
[\p{Punct}&amp;&amp;[^\.:#=,;]]]*)[\.,#;[\s&amp;&amp;[^ ]]]</Pattern>
    <Pattern>#K1#[ ]?#?[:=]?[ ]?([\w[\p{Punct}&amp;&amp;[^\.:#=,;]]][\w
[\p{Punct}&amp;&amp;[^\.:#=,;]]]*)</Pattern>
</Title>
```

La primera diferencia es que mediante el tag `<Keyword>` podremos definir un conjunto de palabras clave referenciadas por un texto identificador, en este caso `#K1#`. Cuando se lean los patrones se crearán tantos duplicados como claves haya, reemplazando la etiqueta `#K1#`. De esta manera con esta estructura de almacenamiento tenemos ocho patrones especificados en dos líneas resultando mucho más fácil de mantener.

Si comparamos el patrón de título del SMS:

T[ ]?[:][ ]?([\w [\p{Punct}&&[^\.:]]]*)[\.]

Con el del correo:

#K1#[ ]?#?[:=][ ]?([\w[\p{Punct}&&[^\.:#=,;]]][\w
[\p{Punct}&&[^\.:#=,;]]]*)[\.,#;[\s&&[^]]]

Observamos que el patrón es mucho más complejo. Inicialmente tenemos la parte de las etiquetas que serán cumplimentadas substituyendo `#K1#`. Vemos también que se aceptan algunos separadores extra como '#', ':', y '='. Algunos de estos separadores son introducidos en la transformación del correo de HTML a texto plano (como el reemplazo de múltiples espacios por ':'). Dentro de los caracteres que deben cumplir la parte central introducimos los caracteres que pueden ser separadores como caracteres excluidos. En la parte central obligamos que la expresión **[\w[\p{Punct}&&[^\.:#=,;]]]** aparezca como mínimo una vez. No se utiliza el operador '+' porque la segunda parte de la expresión mencionada incorpora el espacio en blanco.

Finalmente no solo se acepta el '.' como finalizador sino otros caracteres, incluido cualquier separador espacial (salto de línea, tabulador, etc.) excepto el espacio. Otras de las mejoras introducidas es la determinación de la expresión devuelta si el patrón es encontrado:

Código 3.31: Extracto de gramáticas de mail – grupos de retorno

```
<Keyword id="#K1#">Rent Date, Return Date</Keyword>
<Keyword id="#K01#">Jan,January</Keyword>
<Pattern result="#G2#-01-#G1#">#K1#[ ]?#?[:=][ ]?(20\d\d)[-\\ /_]?#K01#[-\\
/_]?([012]?\d)\D</Pattern>

<Keyword id="#K1#">Deporte, Deportes, Partido, Deportivo, Futbol, Football, Balonpie,
Baloncesto, Basquet, Basquetball, Tenis, Tennis</Keyword>
<Pattern result="Sport">\W#K1#\W</Pattern>
```

Dentro de la etiqueta `result` podemos especificar literalmente el resultado obtenido por el patrón independientemente de lo que sea hallado. Por norma general todos los patrones tienen que tener un único grupo para ser devuelto siempre encerrado entre paréntesis. Mediante las etiquetas especiales `#G1#` y `#G2#` podemos especificar el número del grupo que se utilizará. En el primer patrón de Código 3.31 tenemos un patrón de fecha donde se utilizan dos palabras clave, una para especificar cadenas de texto de apertura y la segunda el mes textualmente. El primer grupo de captura hallará el año y el segundo el día por estar en ese orden. Podemos ordenar el resultado para que sea devuelto como `dd-MM-yyyy` que es compatible con el formato de fecha que tiene que ser asignado a un evento.

En el segundo ejemplo el sencillo patrón `\W#K1#\W` busca cualquiera de las palabras definidas en el `Keyword` con identificador `#K1#` y devolverá siempre la palabra "Sport", que es justamente la categoría que debe asignarse al evento si alguna de esas palabras son halladas, independientemente de qué palabra clave sea encontrada.

Los patrones para correos siguen la misma base que los de SMS solo que son mucho más extensos por la variedad de palabras clave que contienen y separadores o delimitadores.

Estos métodos adicionales de construcción de resultado y generación múltiple de patrones tienen que ser contemplados en las clases `XMLGrammarReader` y `Finder`.

3.7. Aplicación multilingüe

En el apartado de análisis hemos visto que el Generador de Eventos sufrió una restructuración para poder trabajar con contenidos multilenguaje. Los factores más importantes resumidos son:

- La aplicación web pasa a tener un idioma definitivo en habla inglesa. Justo en ese punto se tenía un entorno web pensado para una comunidad de habla inglesa y un Generador de Eventos pensado para correos en español. El idioma de los mensajes influye mayormente en los siguientes casos:
  - El formato de la fecha puede variar de una región a otra, el formato *dd/mm/aaaa* puede ser interpretado como *mm/dd/aaaa*.
  - A la hora de determinar la categoría del evento se utiliza una búsqueda de palabras claves. Estas palabras son diferentes según el idioma.
- La incorporación de un correo de pruebas en inglés. Como era de esperar, no se encontraron todos los campos para generar el evento. Las expresiones regulares no fallaron, simplemente lo que falló es que no eran las expresiones regulares adecuadas.
- Dentro del Generador de Eventos había partes de código que fijaban un cierto texto, a la hora de completar descripción, adecuación del título del evento, mensajes de notificación de creación correcta, etc.

La aplicación pasa a dar soporte al español y al inglés para la generación de eventos. Las acciones que se han realizado para un soporte multilingüe son:

- Creación de un módulo diccionario para determinar el idioma de un texto.
- Creación de expresiones regulares para cada idioma.
- Traducción de literales al idioma de la Agenda Social en su día y para el Generador de Eventos posteriormente.
- Adecuación de palabras clave al inglés, como el nombre de las categorías. En correos españoles las categorías encontradas en castellano son traducidas a su análogo del idioma inglés ya que la aplicación trabaja con categorías definidas siempre en inglés.
- Preparación de las clases y métodos para trabajar con una nueva variable, el idioma.
- Creación de claves o identificadores para los idiomas. Las claves que actualmente son utilizadas son “ESP” para idioma español y “ENG” para inglés.

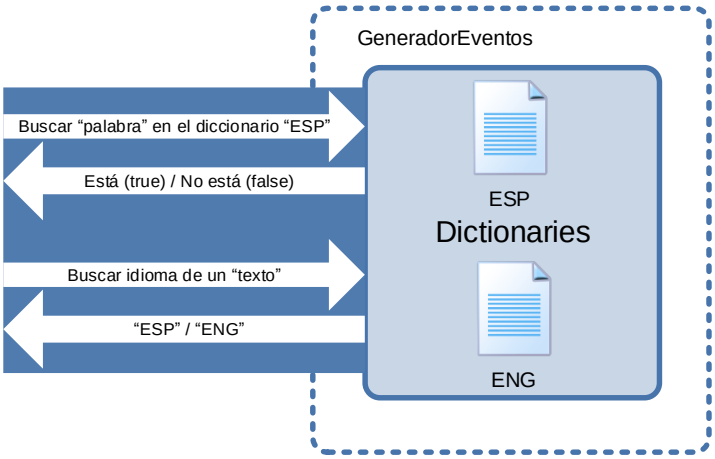
3.7.1. El Diccionario

El Diccionario es un módulo creado para completar la función de creación de eventos multilingüe y solo tiene sentido en el contexto del Generador de Eventos, es por ese motivo que su clases y ficheros de configuración se encapsulan bajo el paquete *eventParser.dictionary*. El diccionario está enfocado sobre todo al contenido de los correos electrónicos porque los SMS pueden albergar faltas, abreviaciones, intercambios de mayúsculas y minúscula, etc. A partir de unos diccionarios almacenados en ficheros de texto se cargan en memoria para acelerar los tiempos de consulta de las palabras. Una vez creado el diccionario se podrá acceder externamente a él mediante dos métodos:

- Buscar una palabra en un diccionario. A partir de la especificación de la palabra y la clave de diccionario buscaremos si está contendia en el diccionario, este método se encuntra descrito en el punto 3.7.5.
- Determinar el idioma de un cierto texto. A partir de un conjunto de palabras se nos devolverá la clave del idioma al que pertenece el texto, este método se describe en el punto 3.7.6.

El módulo Diccionario pertenece conceptualmente al Generador de Eventos y accederemos a él a través de su clase principal *GeneradorEventos* como se observa en la Figura 3.2.

Figura 3.2: Esquema del módulo Diccionario



3.7.2. Definición del fichero de diccionario

Las palabras de los diccionarios deben guardarse siguiendo los siguientes criterios:

- Una palabra por línea.
- Las palabras deben estar en minúsculas.
- No deben existir signos de acentuación o cualquier carácter que no sea ASCII. Se admiten caracteres en el rango de la “a-z” (incluida la ñ) o el apóstrofe ‘'’ inglés.

Una vez definida la estructura de un diccionario, a la hora de buscar palabras en él se realiza una limpieza de la palabra para hallarla en el diccionario. La limpieza del texto antes de buscarla en el diccionario se encuentra definido en el código siguiente:

Código 3.32: Limpieza de una palabra

```
text = text.toLowerCase();
text = StringManipulator.quitarAcentos(text);
text = text.replaceAll("[[^\\p{Alpha}]&&[^ñ]&&[^']]", " ");
text = text.replaceAll("[ ]+", " ");
```

Partiendo de que el texto a analizar está contenido en la variable `text` (que puede contener una única palabra o un conjunto de ellas) se realizan las cuatro acciones de limpieza siguientes:

- Se pasa todo el texto a minúsculas.
- Quitamos todos los acentos mediante el método estático de la clase `StringManiupulator`, `quitarAcentos`. Este método elimina toda acentuación de las vocales incluyendo acento abierto, cerrado, circunflejo y diéresis.
- Cualquier carácter que no sea alfanumérico, 'ñ' o el apóstrofe (para el inglés) es sustituido por un espacio.
- Se reemplazan los múltiples espacios por uno solo.

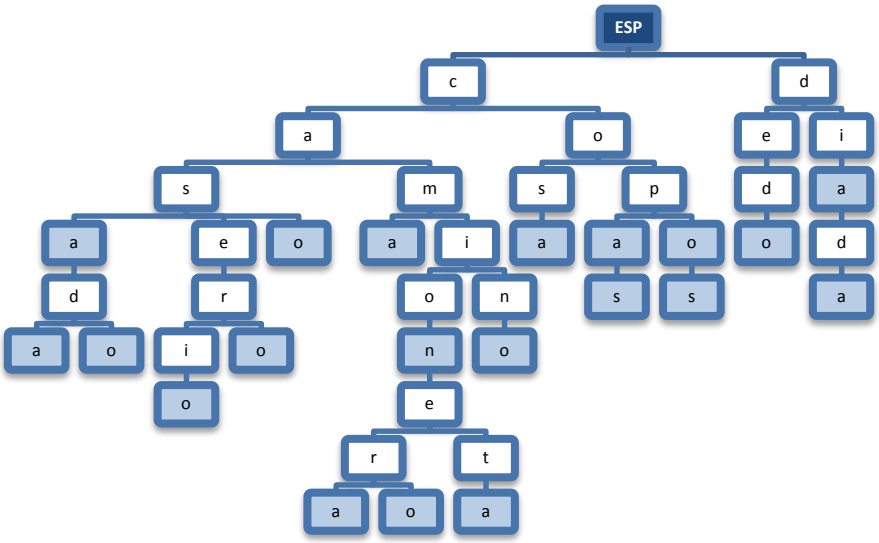
3.7.3. Carga de los diccionarios

Cuando el objeto GeneradorEventos es instanciado, el constructor de dicha clase lanza el proceso de creación de los diccionarios. El objeto `Diccionaries` es un atributo del `GeneradorEventos` y podemos acceder a los diccionarios en cualquier punto de la aplicación a través del `GeneradorEventos` y su método `Dictionaries getDictionaries()`. Los diccionarios creados son guardados en memoria en una tabla (`Hashtable`), donde se guarda la clave del diccionario y la referencia al mismo, de esta manera se accede fácilmente al diccionario almacenado a partir de su clave.

3.7.4. Estructura del diccionario

Los diccionarios han sido modelados siguiendo una estructura en árbol. Estas estructuras están basadas en nodos. A partir de un nodo principal raíz cuelgan de él otros nodos llamados hijos. En la Figura 3.3 se muestra un ejemplo de la estructura del diccionario español con algunas palabras. Empezando por el nivel superior y bajando por cada rama se obtienen las diferentes palabras. Los nodos coloreados representan que ese nodo (y su carácter representado) son final de palabra. Se observa que los nodos de los niveles inferiores de cada rama (hojas) son siempre final de palabra. Las palabras almacenadas en este ejemplo por orden de aparición son: casa, casada, casado, caserío, casero, caso, cama, camino, camionera, camionero, camioneta, camino, cosa, copa, copas, copo, copos, dedo, dia y diada.

Figura 3.3: Esquema de Nodos de un diccionario



3.7.5. Búsqueda de palabras en el diccionario

El proceso de búsqueda de una palabra se hace mediante el método público `boolean findWord(String idDic, String word)`. Se pasa como parámetro el id del diccionario objetivo (“ESP” o “ENG”) y la palabra que deseamos buscar. Se devolverá un booleano indicando si la palabra ha sido hallada o no en el diccionario, `true` para encontrada y `false` en caso contrario.

El método de búsqueda se desglosa en los siguientes pasos:

1) Se obtiene el Nodo principal almacenado en la `Hashtable` a partir del `idDic`.
2) Se desglosa la palabra en un vector de caracteres.
3) Por cada uno de los caracteres miramos si existe el nodo hijo respecto al nodo en el que nos situamos. Se pueden dar dos casuísticas:
    a. Si no existe el nodo hijo para ese carácter se devuelve `false` conforme la palabra no forma parte del diccionario.
    b. Si encontramos el nodo hijo no situamos en él y volvemos a buscar para el siguiente carácter si existe o no otro nodo hijo.
4) El proceso es reiterado hasta llegar al último nodo correspondiente al último carácter. Como es el último carácter de la palabra el nodo debería de estar marcado como “final de palabra”. Se dan dos casuísticas llegado a este punto:
    a. Si el nodo no es marcado como final de palabra se retorna `false`.
    b. Si el nodo es marcado como final de palabra se retorna `true` conforme ha sido encontrada la palabra en el diccionario.

El método `findWord()` debe ejecutarse accediendo al módulo de diccionarios del Generador de Eventos:

```
GeneradorEventos.getInstance().getDictionaries().findWord("ESP", "casa")
```

3.7.6. Determinación del idioma de un contenido

El método para determinar el idioma de un cierto texto está basado en el número máximo de palabras encontradas para cada uno de los diccionarios. Este método tiene sentido cuando el número de diccionarios es reducido, en el caso de haber sido muy elevado habría que haber considerado otro método alternativo.

Nos situamos en el escenario real con dos diccionarios y cierto texto con varias palabras. Mediante el método `public String getLanguage(String content)` se determina el idioma del texto de la siguiente forma:

1) Cogemos cada uno de los diccionarios listados en la `Hashtable`.
2) Se crea un vector de estadísticas y otro de diccionarios.
3) Para cada diccionario se realizan las siguiente acciones:
    a. Extraemos todas las palabras del texto.
    b. Buscamos cada palabra en el diccionario con el que estamos trabajando:

   i. Si la palabra se encuentra incrementamos un contador de aciertos.
   ii. En caso contrario no se incrementa el contador.
  c. Una vez analizadas todas las palabras creamos el porcentaje de palabras encontradas respecto al total del texto.
4) Una vez tenemos todos los porcentajes de aciertos nos quedamos con el del idioma con mayor número.
5) Devolvemos la clave de ese idioma para representar el idioma del contenido.

Puede existir una cierta problemática si el número de palabras que contiene el texto es muy reducido, como es el caso de los SMS. La solución al problema es simple, hay que tener dos métodos de búsqueda a la hora de determinación del idioma. Se crea pues un segundo método para mensajes de tipo SMS, en este caso en vez de utilizar el diccionario se buscan en el texto todas las palabras clave reservadas de inicio de campo de las plantillas o patrones. Al ser un SMS la plantilla debe ser obligatoria por lo que resulta el método es más fiable.

Para obtener el lenguaje de un contenido invocaremos el primer método si partimos de un correo o el segundo si partimos del contenido de un mensaje de texto:

```
String language =
GeneradorEventos.getInstance().getDictionaries().getLanguage(content);

String language =
GeneradorEventos.getInstance().getDictionaries().getXMSLanguage(content);
```

3.8. Módulo de comunicación para SMS

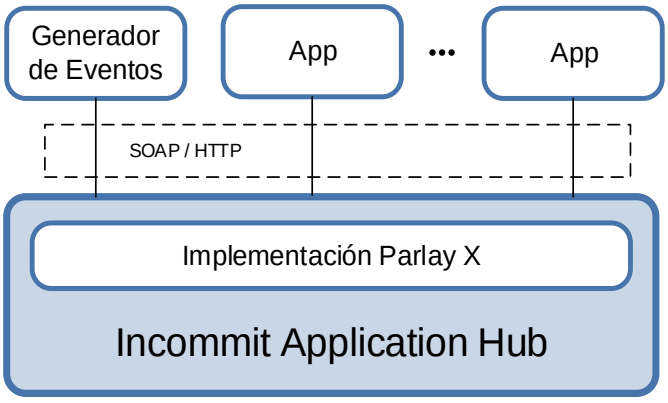
Como hemos visto en el análisis, utilizamos la pasarela de Orange para el envío y recepción de mensajes de texto SMS. Esta plataforma no está preparada exclusivamente para el envío de SMS, sino que ofrece otras muchas funcionalidades como envío de logos, melodías, servicios de localización y envío de mensajes multimedia (MMS). La funcionalidad requerida por el Generador de Eventos es solo el envío y recepción de SMS.

Cabe destacar que la integración de la Agenda Social a la Pasarela Parlay X de Orange venía marcada por unos ciertos parámetros definidos en sus archivos de configuración. Durante la fase de diseño y prueba de la comunicación se mantuvo relación con el personal de Orange para solucionar cualquier conflicto y configurar el acceso para que ambas partes pudiesen comunicar: la Agenda Social y la Plataforma de Orange. El Generador de Eventos mantiene las mismas clases creadas en este proyecto para la Agenda Social.

El módulo de conexión de mensajería móvil ha requerido un desarrollo desde cero, por lo que su desarrollo y sus pruebas se han realizado con una aplicación independiente de tipo J2SE. Posteriormente se ha integrado en el proyecto y se ha deshabilitado su envío y recepción cuando ha dejado de estar operativo el servicio de acceso a la mensajería SMS vía Parlay X.

El nombre real de la pasarela Parlay X de Orange es Incommit Application Hub, y como se observa en la Figura 3.4 modela un dispositivo HUB para atender todos los servicios de las aplicaciones conectadas a él.

Figura 3.4: Esquema de conexión con la Plataforma Parlay X



3.8.1. Comunicación entre el Generador de Eventos y la pasarela Parlay X

La comunicación entre la aplicación y la plataforma de Orange se realiza mediante Web Services. En el presente apartado se detallaran los puntos para crear la conectividad y sus características.

El programa de apoyo a terceros de Orange ofrece varios documentos como guía a los desarrolladores, definición de la API y resolución de dudas, esta documentación está adjunta a esta memoria como anexos digitales.

3.8.1.1. Los Web Services

Un web service es una porción de la lógica de negocio, reside en algún lugar de Internet que es accesible a través de algún protocolo estándar de comunicación de Internet como HTTP o SMTP. Los web services están basados mayormente en la tecnología XML que ofrece una manera neutral para representar datos.

Los servicios web se utilizan para intercambiar datos entre aplicaciones. Estas aplicaciones pueden estar en cualquier parte del mundo, desarrolladas en lenguajes de programación diferentes y ejecutarse en plataformas diferentes. OASIS[46] y W3C[47] son los responsables de la arquitectura y reglamentación de los servicios Web.

Algunos de los estándares utilizados por los servicios web y que afectan a la implementación de los futuros puntos son:

- XML: es un metalenguaje basado en etiqueta. No es un lenguaje en si porque está diseñado para definir las gramáticas de otros lenguajes. Su diseño permite almacenar e interpretar datos y transportarlos entre entidades.
- WSDL: Lenguaje basado en XML para describir servicios y cómo acceder a ellos.
- SOAP: Protocolo basado en XML para permitir a las aplicaciones intercambiar información a través de HTTP (entre otros protocolos). Es el protocolo que se utiliza para codificar la información de los requerimientos de los servicios web y para acceder éstos.
- HTTP: Protocolo estándar para el intercambio de información en la world wide web (WWW).

Para la comprensión de esta tecnología se ha utilizado el libro Java Web Services ([E]) y la página web de W3C ([19]).

3.8.1.2. Parlay X

Parlay X es un conjunto simple de APIs publicadas por Parlay que establece unas definiciones de servicios web. Estos servicios web incluyen control de llamadas a terceros, localización y micro-pagos. Parlay X es un complemento a las APIs de Parlay. La API de Parlay está enfocada a las redes telefónicas y los operadores. Las funciones que incluye Parlay son control de llamadas, conferencias, interacción con los usuarios mediante mensajes de texto, audio y la facturación. Las APIS están especificadas en CORBA[48] y WSDL. Parlay define la API pero no su implementación.

[46] Organization for the Advancement of Structured Information Standards: Organización para el avance de los estándares de información estructurados.
[47] Word Wide Web Consortium: consorcio de WWW
[48] Common Object Request Broker Architecture: Arquitectura común de intermediarios en peticiones a objetos

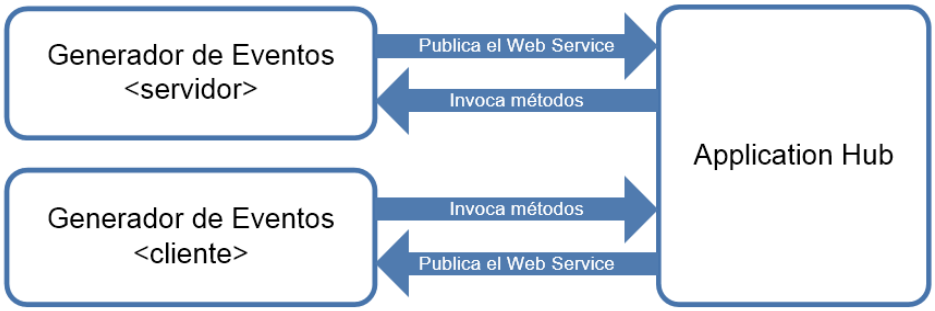
3.8.1.3. Escenarios de comunicación

Existen dos escenarios en el desarrollo de aplicaciones Web Services basados en Parlay X:

- El Application Hub actúa como un servidor y la aplicación desarrollada es el cliente. En este escenario la aplicación utiliza un Web Service proporcionado por el Application Hub.
- La aplicación actúa como servidor y el Application Hub es el cliente. La aplicación es el proveedor del Web Service y el Application Hub invoca métodos en ese Web Service.

La Agenda social implementa los dos escenarios. Su lado servidor se encarga de recibir los mensajes SMS y su lado cliente de generarlos y enviárselos a la plataforma de Orange.

Figura 3.5: Escenarios de comunicación



3.8.1.4. Creación de la API Parlay X

Orange ofrece su API en ficheros WSDL, de esta manera, pueden generarse las clases para los proyectos dependiendo de su lenguaje de programación o plataforma.

En el portal de Orange Business Partners se pueden descargar los ficheros WSDL para crear las clases que implementan y encapsulan las llamadas a los servicios de Parlay X. Las clases de comunicación implementadas harán uso de esta API. Para generar las clases de java mediante estos ficheros WSDL ha sido necesario utilizar la aplicación *wdsl2java*.

A parte de los ficheros WSDL mencionados, se necesitan varias librerías detalladas al final de este capítulo. Al final del proceso se obtiene una API que se renombrado como librería ParlayX.jar incorporada en el proyecto. Hay que mencionar que la estructura de código estaba muy bien documentada en el soporte que daba Orange para el desarrollo de terceros. Por lo que simplemente se ha tenido que adecuar la solución dada por los ejemplos al Generador de Eventos y, anteriormente, a la Agenda Social.

La gestión de comunicaciones de la parte de SMS está ubicada en el paquete del proyecto *xms*. En un principio este paquete se llamaba *sms* pero se cambió a *xms* para indicar que también da soporte a MMS. Aunque la clase de recepción de MMS por pasarela de Orange ha sido implementada, no ha sido probada nunca extremo a extremo, por lo que no será comentada en la presente memoria, aunque si se tiene la clase y métodos de recepción en el código fuente de la aplicación.

Las tres clases que llevan toda la comunicación SMS y ficheros auxiliares están recogidos en la Tabla 3.17.

Tabla 3.17: Clases y ficheros del paquete xms

Paquete	Clase (.java)
xms	RecepcionXMS
xms	SmsNotification
xms	EnvioSMS
xms	deploy_mms.wsdd
xms	undeploy_mms.wsdd
xms	XmsConfig.properties

3.8.1.5. Configuración del intercambio de mensajes

Aunque las clases hayan sido construidas estas carecen de funcionamiento. La plataforma de Orange ofrece el servicio para construir las soluciones de comunicación comentadas. Como servicio que es, tiene que estar dado de alta en Orange y tener tanto Orange como la aplicación de terceros los mismos parámetros para que se garantice el servicio de envío y recepción de SMS estableciéndose la comunicación entre ambas partes.

Tabla 3.18: Parámetros para el intercambio de datos en la pasarela Parlay X

Modulo	API	Información proporcionada por	
		Desarrollador	Proveedor del servicio
Acceso	Access		ID de aplicación. ID de proveedor de servicio. ID de grupo de instancia de aplicación. Contraseña de grupo de intancia de aplicación.
Inicio de conexión	Call	URL de destino.	Número de acceso a la aplicación.
Mensajes SMS	SendMMS		ID MailBox. Contraseña de ID MailBox.
	SMS Notification	URL de destino.	Número de acceso a la aplicación. ID MailBox. Contraseña de ID MailBox.
	Receive SMS		ID MailBox. Contraseña de ID MailBox.
Mensajes Multimedia	Send Message		ID MailBox. Contraseña de ID MailBox.
	Message Notificaciton	URL de destino.	Número de acceso a la aplicación. ID MailBox. Contraseña de ID MailBox.
	Receive Message		ID MailBox. Contraseña de ID MailBox.

Los valores de los campos de configuración se encuentran en el archivo `XmsConfig.properties`. Los proporcionados por el proveedor son ciertos identificadores fijados por él. En cambio, si mencionamos las url de destino de los servicios (en el fragmento

de más abajo), se observa que son las correspondientes a la de los servicios web cargados en el servidor. La <IP> es la ip del equipo que ejecuta el Generador de Eventos o Antigua Agenda Social y el <PUERTO> su puerto HTTP de servidor para atender peticiones HTTP.

```
Acceso = http://<IP>:<PUERTO>/parlayx/services/Access?
SMS = http://<IP>:<PUERTO>/parlayx/services/SendSmsPort?wsdl
MMS = http://<IP>:<PUERTO>/parlayx/services/ReceiveMmsPort?wsdl
```

3.8.2. Recepción de SMS

La recepción de SMS está gestionada principalmente por la clase `RecepcionSMS`. Esta clase tiene dos métodos, uno para arrancar el servidor y otro para pararlo.

El método `parar()` simplemente para el servidor que es arrancado en el método `arrancar()` y libera el puerto de escucha del servidor.

El método arrancar`()` es utilizado para iniciar el servicio. Creamos un servidor Axis ([20]) que atenderá a las conexiones por parte de la pasarela cuando tenga que enviarnos un SMS (ver Código 3.33). Se crea un Servidor en el puerto especificado en la variable `port`. La variable se obtiene del fichero de configuración `XmsConfig.properties` y por defecto será 9999. Seguidamente se carga como proceso la parametrización del fichero `deploy_sms_wsdd`. En este fichero tenemos la siguiente línea que debe configurarse para la correcta gestión cuando llegue un mensaje:

```
<parameter name="className" value="xms.SmsNotification"/>
```

En esa línea indicamos qué clase será ejecutada cuando nos llegue un SMS.

Código 3.33: Detalle de arranque del servidor Axis

```
String deploymenDescriptorFileName1 = "deploy_sms.wsdd";
// Start SimpleAxisServer
server = new org.apache.axis.transport.http.SimpleAxisServer();

ServerSocket ss = new ServerSocket(port);
server.setServerSocket(ss);
server.start();
GELogger.add(this, "Starting simple axis server...");

// Read the deployment description of the service
InputStream is1 = getClass().getResourceAsStream(deploymenDescriptorFileName1);


// Now deploy our web service
org.apache.axis.client.AdminClient adminClient;
adminClient = new org.apache.axis.client.AdminClient();
GELogger.add(this, "Deploying receiver server web service...");
adminClient.process(new org.apache.axis.utils.Options(new String[] { "-ddd", "-
tlocal" }), is1);

GELogger.add(this, "Axis server started. Waiting for connections on: " + port);
```

La clase `SmsNotification` tiene un único método que es invocado cuando es notificada la llegada de un nuevo mensaje SMS:

La propia clase `SmsNotification` implementa la clase `SmsNotificationPort` contenida dentro de la API de Parlay X. Concretamente se ejecuta el método

`notifySmsReception()` cuando es recibido un mensaje. Al final del método lanzamos el procesado del SMS de la clase GeneradorEventos pasando como parámetros el mensaje y número de teléfono del emisor.

Código 3.34: Procesado de SMS recibido

```
public class SmsNotification implements
org.csapi.www.wsdl.parlayx.sms.v1_0.notification.SmsNotificationPort {

      public void notifySmsReception(java.lang.String registrationIdentifier,
                  java.lang.String smsServiceActivationNumber,
                  org.csapi.www.schema.parlayx.common.v1_0.EndUserIdentifier
senderAddress, java.lang.String message)
                  throws java.rmi.RemoteException,
org.csapi.www.wsdl.parlayx.sms.v1_0.notification.ApplicationException {

            GeneradorEventos.procesarSms(message,
senderAddress.getValue().toString());
      }
}
```

3.8.3. Envío de mensajes

El envío de SMS es gestionado por la clase `EnvioSMS`. Esta clase tiene un diseño basado también en el patrón de diseño de instancia única pues solo queremos que haya un único elemento que gestione la salida de mensajes e interactúe con la plataforma Parlay. Todo el diseño de esta clase está basada en ejemplos de envío de SMS de la plataforma de Orange.

Desde fuera de la clase solo tenemos visibilidad al método:

```
public synchronized static boolean enviar(String message, String
destNumber)
```

Por lo que el envío es transparente, se nos devolverá `true` si el envío se ha efecutado o `false` si ha habido algún problema.

Internamente la clase realiza las siguientes operaciones:

- A la hora de enviar un mensaje se comprueba que hay conexión con la pasarela realizando un `login()` y obteniendo un id de sesión. Si no se consigue identificador entonces no habrá conectividad con la pasarela y se abortará el envío.
- El proceso de `login()` se conecta a la plataforma mediante la configuración parametrizada en el fichero `XmsConfig.xml`
- Finalmente una vez que hay conexión se envía el mensaje siguiendo los pasos indicados por la API Parlay X.
- Aunque no se ha utilizado, adicionalmente se ejecuta una tarea que comprobará el estado de recepción por parte de la pasarela cada pocos segundos. Hay que mencionar que en esta dirección de comunicación, la Pasarela nunca informará que un mensaje ha sido entregado. Es tarea de la aplicación (parte cliente del servicio) ir preguntando a la Pasarela (servidor) el estado de entrega del SMS.

3.9. Módulo de comunicación TCP-IP para Emulador

La pasarela Orange solo estuvo activa unas pocas semanas para probar la recepción y envío de SMS. El parseado de SMS se quedó inhabilitado y muchas de las pruebas se realizaban directamente desde el código parando y arrancando la aplicación web de la Agenda Social, procesando mensajes que estaban escritos directamente en el código. Esto era un proceso costoso en tiempo.

Para solventar esta problemática se creó un servicio alternativo de mensajes de texto que emularía la llegada de un SMS por la plataforma de mensajería. Además el emulador permitiría el análisis de SMS con ficheros, emulando así la recepción de un MMS también.

3.9.1. Estructura del paquete emulator

El Emulador de XMS (SMS y MMS) se encuentra dentro del paquete *xms.emulator* y está formado por las siguientes clases:

Tabla 3.19: Clases y ficheros del paquete xms.emulator

Paquete	Clase (.java)
xms.emulator	ServidorEmulador
xms.emulator	XMSFile

3.9.1.1. La clase XMSFile

Esta clase define una estructura de datos para ser recibida y enviada por el `ServidorEmulador` y el `ClienteEmulador`.

Código 3.35: Clase XMSFile

```
public class XMSFile extends Object implements Serializable {

      private String type;
      private String phone;
      private String content;
      private ArrayList<File> files;

      ...
}
```

El campo `type` es un `String` que indicará si es "SMS" o "MMS". En el campo `phone` tendremos el número de teléfono del emisor, y en la variable *`content`* el contenido del mensaje. También contiene una `ArrayList` de archivos `File` con el conjunto de archivos multimedia que se añadirán en la creación de un evento. Estos cuatro campos forman un objeto `XMSFile`. Para que pueda ser enviado mediante la conexión de un Servidor y Cliente TCP el objeto tiene que ser `Serializable`. De esta manera el objeto puede ser transformado en una cadena de bytes por el Cliente y ser reensamblado por el Servidor si ambas partes tienen y conocen la estructura de un `XMSFile`.

El cliente del servidor ha sido creado como aplicación independiente y tendrá que tener la misma clase `XMSFile` que tiene el Generador de Eventos.

3.9.1.2. La clase ServidorEmulador

La clase `ServidorEmulador` al igual que el Servidor de recepción de SMS tiene dos métodos, uno de arranque y uno de parada. El de parada es necesario utilizarlo para liberar el puerto cuando se cierra el Generador de Eventos. Como es un servidor básico TCP comentaremos simplemente la ejecución de su método de arranque:

- Creamos un objeto `ServerSocket` para atender peticiones en el puerto definido por defecto en el fichero `XmsConfig`, el 9995.
- Entramos en un bucle infinito controlado por una variable booleana, que es puesta a `false` en el método de paro. Mientras el servidor esté en ejecución se mantendrá a la espera de recibir una conexión por parte del cliente.
- Cuando nos llegue una conexión recibiremos un objeto XMSFile enviado por Cliente:

```
inputStream = new ObjectInputStream(SC.getInputStream());
XMSFile archivo = (XMSFile) inputStream.readObject();
```

- Una vez obtenido el objeto `XMSFile` cerraremos el objeto `ObjectInputStream`.
- Con el `XMSFile` recibido procedemos a su procesado por la clase `GeneradorEventos` (en función de si su atributo `type` es SMS o MMS) con los métodos:

```
GeneradorEventos.procesarSms(archivo.getContent(),
archivo.getIdentifier());

GeneradorEventos.procesarMms(archivo.getContent(),
archivo.getIdentifier(), archivo.getFiles());
```

3.10. Módulo de comunicación para e-mails

Para la comunicación con la cuenta de correo de la aplicación es necesaria la utilización de la API de Java *JavaMail* ([21]). Esta API permite realizar todo tipo de operaciones con los correos y también con los servidores de correo. Este módulo es utilizado principalmente para recibir correos, modificar el almacenamiento en la cuenta de correo, analizar e-mails y enviarlos desde la cuenta de la aplicación.

3.10.1. Estructura del paquete mail

El paquete que contiene las clases de este módulo es el paquete *mail* y está formado por las clases que figuran en la siguiente tabla:

Tabla 3.20: Clases y ficheros del paquete mail

Paquete	Clase (.java)
mail	TemporizadorMail
mail	RecepcionMail
mail	ExtractMailTask
mail	MailOperations
mail	EnvioMail
mail	MailConfig.properties

En el fichero de configuración se almacena tanto la cuenta de correo de la aplicación como su contraseña de acceso. También se guarda el tiempo de espaciado entre accesos al correo y un retardo inicial antes de comenzar a conectarse a la cuenta. Este retardo inicial es heredado del arranque de la Agenda Social, que al ser un proyecto web podía comenzar a conectarse a la cuenta de correo y leer correos recibidos sin estar arrancado del todo el contexto de la aplicación, pudiéndose perder la generación de eventos de esos correos.

3.10.1.1. La clase TemporizadorMail

Esta clase contiene los métodos de arranque y parado del proceso que se conecta a la cuenta de correo. Lee la configuración del fichero `MailConfig.properties` y su única función es invocar estos dos métodos:

```
ExtractMailTask temp = null;
temp = new ExtractMailTask(retard, periode, userMail, password);
temp.planificarRecepcio();
```

3.10.1.2. La clase ExtractMailTask

La clase `ExtractMailTask` se encarga de ejecutar una tarea temporizada según los parámetros `periode` y `retard` introducidos en el constructor como hemos visto en el apartado anterior. En el fragmento de código siguiente observamos qué realiza el método `planificarRecepcio()`.

Al ejecutar el método `planificarRecepcio()`, mostrado en el Código 3.36, vemos que se crea un objeto `TimerTask`. En su método se declara la acción que se ejecutará en intervalos definidos por *`rate`* o `periode`. El objeto `temporitzador` es un `Timer` declarado como *daemon*. Esto quiere decir que se cancelará su ejecución si la aplicación principal es finalizada. Si se configurara como no *daemon* entonces incluso parando el Generador de Eventos el hilo que ejecuta la tarea de extracción de eventos continuaría ejecutándose (hasta para la máquina virtual de Java). Para evitar este problema tenemos también el método de parada para cancelar el `Timer`.

Código 3.36: ExtractMailTask

```
this.temporitzador = new Timer(true);
...

public void planificarRecepcio() {
      TimerTask task = new TimerTask() {
            public void run() {
                  recepcioMail.extractEvents(mailUser, password);
            }
      };
      temporitzador.scheduleAtFixedRate(task, delay, rate);
}

public void parar() {
      GELogger.add(this, "Temporizador de consulta de cuenta de correo parado");
      if (temporitzador != null) {
            temporitzador.cancel();
      }
}
```

Observamos que se configura una tarea `task` de ejecución fija con el método `temporitzador.scheduleAtFixedRate(task, delay, rate)`, ejecutándose cada cierto tiempo la función `recepcioMail.extractEvents(mailUser, password)`. Este método pertenece al objeto `RecepcionMail` que veremos a continuación.

3.10.1.3. La clase RecepcionMail

La clase de recepción de e-mails se encarga de conectarse a la cuenta de correo de la aplicación. La cuenta es de Gmail por lo que se utilizará las propiedades de conexión para una cuenta de Gmail. También hay que mencionar que el protocolo utilizado para el acceso al servidor de correo es IMAP. Este protocolo permite realizar operaciones dentro de la cuenta, como mover correos y eliminarlos. También la lectura de los correos no hace que se descarguen y desaparezcan del servidor de correo como es el caso del protocolo POP. Los tres principales métodos y sus características son:

Código 3.37: Recepción de correos electrónicos

```
public void extractEvents(String mailUser, String password) {
      conectarIMAP(mailUser, password);
}

private void conectarIMAP(String mailUser, String password) {
      try {
            GELogger.add(this, "Conectando a " + mailUser + " ...");
            prop = new Properties();
            prop.setProperty("mail.imap.starttls.enable", "false");
            prop.setProperty("mail.imap.socketFactory.class",
"javax.net.ssl.SSLSocketFactory");

(continúa)
```

```
            prop.setProperty("mail.imap.socketFactory.fallback", "false");
            prop.setProperty("mail.imap.port", "993");
            prop.setProperty("mail.imap.socketFactory.port", "993");
            sessio = Session.getInstance(prop);
            emmagatzematge = sessio.getStore("imap");
            emmagatzematge.connect("imap.gmail.com", mailUser, password);
            folder = emmagatzematge.getFolder("INBOX");
            folder.open(Folder.READ_WRITE);
            extractEvents();
      } catch (Exception e) {
            GELogger.add(this, e, "Ha habido problemas de conexión a GMAIL");
      } finally {
            try {
                  if (folder != null) {
                        // no marcamos como expunge los mensajes a eliminar
                        folder.close(false);
                  }
                  if (emmagatzematge != null) {
                        emmagatzematge.close();
                        }
            } catch (Exception e) {
                  GELogger.add(this, e);
            }
      }
}

private void extractEvents() {
      try {
            // Obtenemos los mensajes
            Message[] mails = folder.getMessages();
            GELogger.add(this, "# Mails recibidos: " + mails.length);

            // Movemos los mensajes a la carpeta "PARSING"
            for (int i = 0; i < mails.length; i++) {
                  MailOperations.moveMessageCheck("PARSING", mails[i]);
            }
            /* Actualizamos las referencias de los mensajes ya que ahora estarán
             * en la carpeta PARSING */
            folder = emmagatzematge.getFolder("PARSING");
            folder.open(Folder.READ_WRITE);
            mails = folder.getMessages();

            // Recorremos todos los mensajes intentado crear correos a partir de
            // todos ellos
            for (int i = 0; i < mails.length; i++) {
                  GELogger.add(this, "Processando mail [" + Integer.toString(i + 1) +
" de " + mails.length + "]");
                  GeneradorEventos.procesarMail(mails[i]);
            }
      } catch (Exception e) {
            GELogger.add(this, e);
      }
```

- Método extractEvents público. En este método simplemente llamamos al siguiente método:

- Método conetarIMAP. Las variables de esta clase utilizadas para este método y el siguiente son:

```
protected Properties prop;
protected Session sessio;
protected Store emmagatzematge;
protected Folder folder;
```

Lo primero que se realiza en el método es parametrizar la sesión de conexión a la cuenta de Gmail. Se crea un objeto `Session` a partir de las propiedades de la conexión. Seguidamente creamos un objeto `Store` realizando la conexión directamente utilizando la sesión parametrizada. Una vez establecida la conexión obtenemos la bandeja de

entrada de Gmail asignando a `Folder` la carpeta "INBOX" del correo, y a continuación se abre para operaciones de lectura y escritura.

Seguidamente se ejecuta el método `extractEvents()` para obtener los correos de la carpeta "INBOX" y analizarlos. Una vez finalizada la ejecución de ese método se cierran los recursos del objeto `Folder` y `Store`.

- Método extractEvents privado. Este método obtiene una referencia a todos los correos que están en la bandeja de entrada y los recorre uno a uno moviéndolos a la carpeta "PARSING". Seguidamente se vuelven a recorrer todos los mensajes y se mandan procesar por la clase `GeneradorEventos` con el método `GeneradorEventos.procesarMail(mails[i])`. La operación de traslado es necesaria porque si se volviese a ejecutar el proceso del `TimerTask` antes de analizar todos los mensajes se cogerían mensajes por duplicado. El proceso *n* analizaría todos los correos y el proceso *n+1* analizaría todos los mensajes nuevos y todos aquellos del proceso *n* que no hubiesen sido procesados todavía.

3.10.1.4. La clase MailOperations

Esta clase contiene un conjunto de métodos para trabajar tanto con los correos dentro de las carpetas como para buscar contenidos dentro de un mismo correo. Casi todas las operaciones son utilizadas por los métodos de extracción de evento a partir de un correo de las clases `GeneradorEventos` y `MailEventParser`. Los dos métodos más importantes son:

Código 3.38: Funciones de MailOperations

```
public static Part getPartToProcess(Message message) throws Exception {
	Part part = null;
	part = findPart(message, "text/html");
	if (part != null) {
		return part;
	} else {
		part = findPart(message, "text/plain");
		if (part != null) {
			return part;
		}
	}
	// Opción por defecto part == null
	return part;
}

public static void moveMessageCheck(String folder, Message message) {
	try {
		MailOperations.checkDirectory(folder, message.getFolder().getStore(),
Folder.HOLDS_FOLDERS + Folder.HOLDS_MESSAGES);
		MailOperations.moveMessage(message, folder);
		return;
	} catch (Exception e) {
		GELogger.add(MailOperations.class, e);
	}
}

public static void checkDirectory(String name, Store store, int type) throws
Exception {
	Folder folder = store.getFolder(name);
	if (!folder.exists()) {
		folder.create(type);
	}
}
```

- Método getPartToProcess. A partir de un objeto `Message` que contiene la totalidad de un correo electrónico (contenidos, asuntos direcciones de destinatarios, emisor, etc.). Se utiliza el método `findPart(message, "TIPO MIME")` para hallar un objeto `Part` dentro del mensaje que coincida con el tipo MIME especificado. Primero se buscan contenidos en formato HTML y seguidamente si no se halla ninguno, en formato texto plano. El método es recursivo y recorre toda la estructura interna el correo electrónico hasta que da con la parte que contiene el texto del mensaje. Si no se hallase se devolvería un nulo.

- Métodos moveMessageCheck y checkDirectory. El primer método se utiliza para mover un correo de un directorio a otro dentro de la estructura de correos. Se utiliza el segundo método para comprobar si existe el directorio de destino y en caso de no existir crearlo.

3.10.1.5. La clase EnvioMail

Esta clase contiene métodos para el envío de correos. La estructura es similar a la de recepción de un correo. Se debe crear una sesión con la parametrización adecuada para utilizar el servidor de Gmail para el envío de correos. Para enviar un correo utilizamos las sentencias:

Código 3.39: Envío de un correo

```
MimeMessage message = createMessage(from, to, subject, formatHTML);
// Envío del correo
Transport t = session.getTransport("smtp");
t.connect(sendMailUser, password);
t.sendMessage(message, message.getAllRecipients());
// Cierre de la sesión
t.close();
```

El método `createMessage(from, to, subject, formatHTML)` devuelve un objeto `MimeMessage` construyéndolo de la siguiente forma:

Código 3.40: Construcción de un mensaje de correo

```
private MimeMessage createMessage(String from, String to, String subject, String
formatHTML) {
        MimeMessage message = new MimeMessage(session);
      try {
            message.setFrom(new InternetAddress(from));
            message.addRecipient(Message.RecipientType.TO, new InternetAddress(to));
            message.setSubject(subject);
            message.setContent(formatHTML, "text/html");
      } catch (Exception e) {
            GELogger.add(this, e);
      }
      return message;
}
```

Creamos un objeto `MimeMessage` utilizando su constructor que requiere de la sesión creada con las propiedades adecuadas para el envío. Se especifica quién envía el correo con el campo *from* y quién lo va a recibir con el *to*. Se añade el asunto y el contenido con el tipo MIME adecuado, en este caso `"text/html"`.

3.11. Interfaz gráfica del Generador de Eventos

Para realizar las pruebas del Generador de Eventos se ha implementado una completa interfaz gráfica diseñada en JavaFX. JavaFX es el sustituto de Swing y ha tomado fuerza con la última versión de Java 8. Las fuentes para el estudio de JavaFX han sido el libro Pro JavaFX 8 ([F]) y tutoriales de internet ([22]).

Estas son algunas de las ventajas obtenidas al utilizar JavaFX y algunas consideraciones a tener en cuenta:

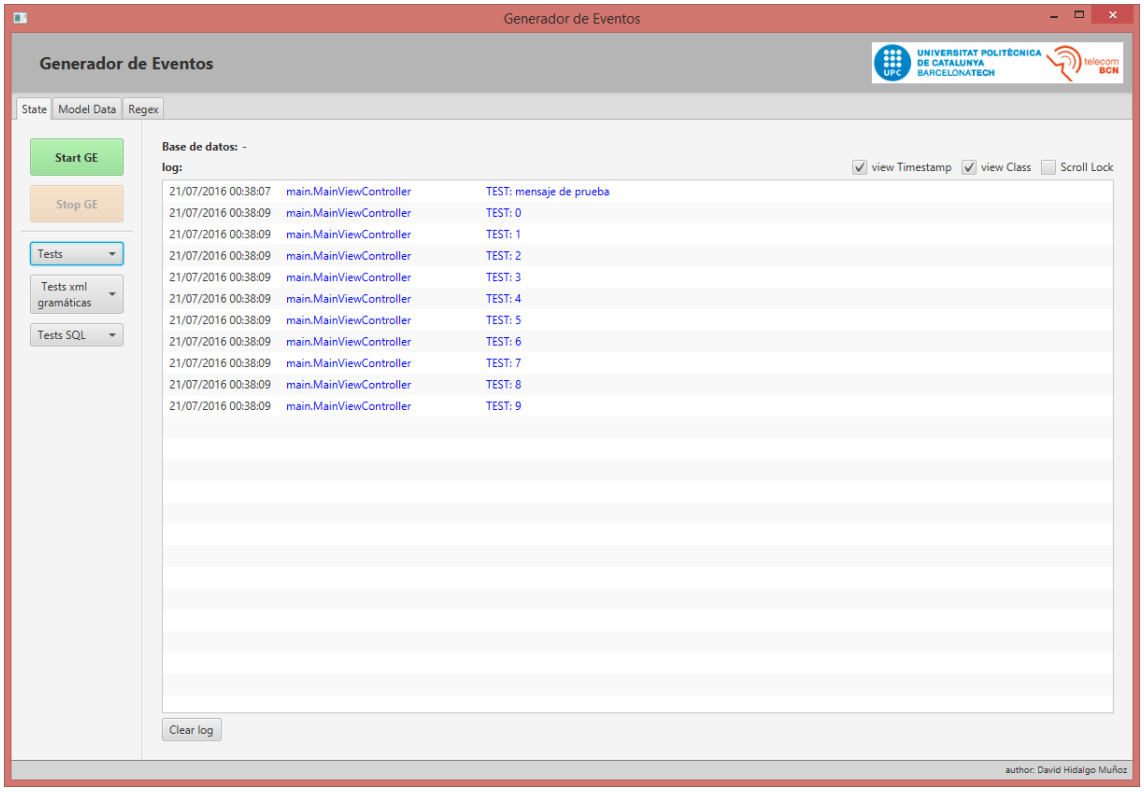
- La interfaz gráfica es sencilla de construir mediante el uso de la aplicación Java Scene Builder. Esta aplicación permite almacenar el diseño gráfico en ficheros fxml. Estos ficheros están basados en XML y detallan tanto los objetos que contiene como su estructura jerárquica.
- JavaFX implementa un modelo de diseño basado en MVC. La vista se guarda en el fichero fxml comentado. El modelo de datos dependerá de cómo se interactúa con los objetos que contengan información y el controlador se especifica en una clase aparte de Java.
- El controlador tiene definidos todos los objetos referenciados en la vista (fichero fxml) que requieran una interactuación con ellos. Además en él se contienen los métodos de acción que se dan lugar en la interfaz, como por ejemplo, presionar un botón.
- Otra de las características importantes es la introducción de clases y variables `Property`. Estas clases pueden ser vinculadas entre ellas para que cuando la propiedad de una variable sea modificada, sea lanzado un método de aviso y actuar en consecuencia. En definitiva, se realiza una notificación ante un cambio (patrón `Observer` y `Observable`)
- Otro aspecto a tener en cuenta es que toda tarea de manipulación de la interfaz gráfica debe realizarse en el `Thread` de la aplicación JavaFX. Tareas de larga duración ejecutadas en este hilo bloquearán el `Thread` y dejará la interfaz sin respuesta. Para ello se crean objetos `Task` y se ejecutan en hilos separados, una vez finalizados podemos actualizar la interfaz gráfica con el resultado obtenido.

3.11.1. Vista de la interfaz

Observamos en la interfaz de la Figura 3.6 una sección a la izquierda con dos botones principales. Uno de “Start GE” y otro de “Stop GE”. Estos dos botones al ser *clickados* ejecutan dos acciones. La de arranque inicia todos los servidores, arranca el `TimerTask` de conexión a la cuenta de correo e instancia el módulo de generación de eventos. También comprueba la correcta conexión a base de datos para determinar el modo de funcionamiento DB_MODE o NO_DB_MODE. El botón de parada detiene todos los servidores y la tarea de consulta de correos. También tenemos unos botones para realizar algunas tareas de prueba:

- Tests para visualización de logs de ejemplo.
- El resultado de la carga de los patrones de búsqueda según el parseador (Mail / SMS) y el idioma (ENG / ESP).
- El resultado de algunas funciones de búsqueda en base de datos utilizadas por los patrones.

Figura 3.6: Interfaz gráfica del Generador de Eventos – pantalla principal



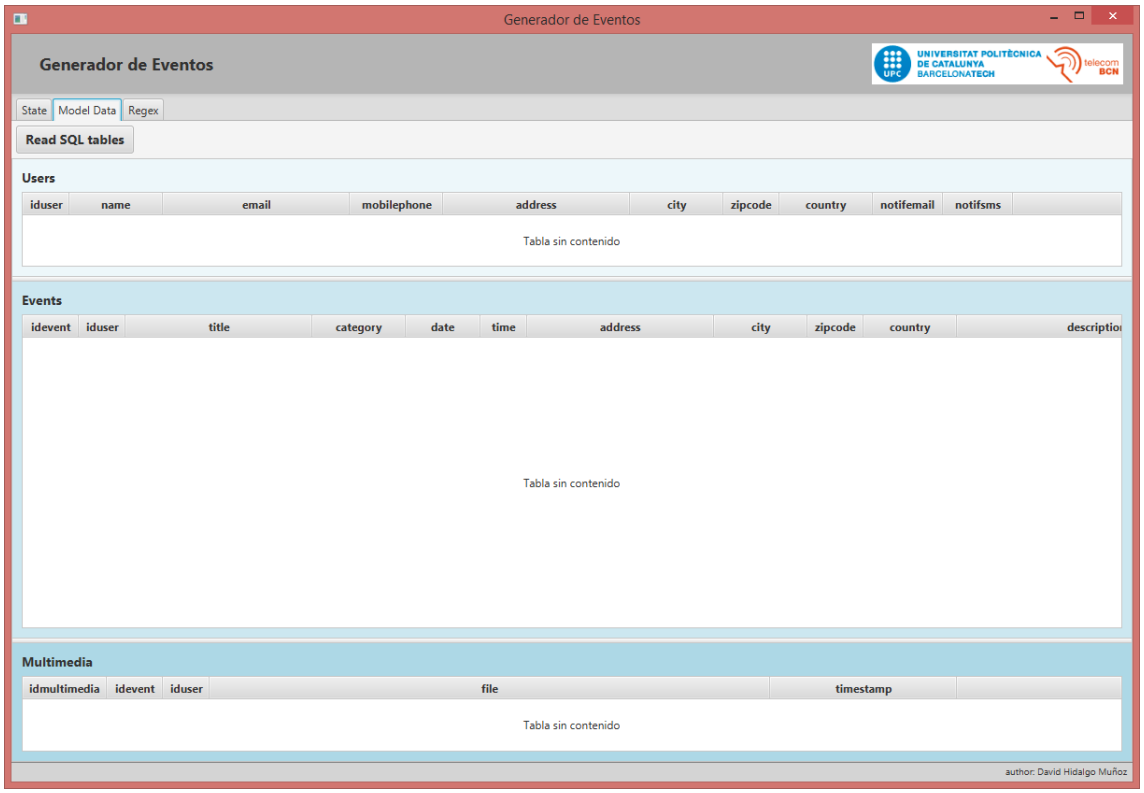
En la parte central tenemos el log. El log es cumplimentando automáticamente cada vez que en la ejecución del Generador de Eventos se guarda algún comentario con el método estático y público `add(Object o, String str)`.

Este `Logger` guarda en un listado interno los mensajes que van llegando y cada cierto intervalo de tiempo los vuelca al listado utilizado por la interfaz gráfica. De esta manera ante avalanchas de escrituras en el log la interfaz gráfica no se ve afectada.

Tenemos arriba a la derecha unos `checkBoxs` para visualizar la hora de generación del log, la clase que lo ha generado (objecto `Object` del método `add`) y otro `checkBox` para parar el desplazamiento hacia abajo del log. En la esquina inferior izquierda el botón "Clear log" borrará el log mostrado por pantalla. Además, si *clickamos* con el botón derecho del ratón en un registro del log nos aparecerá la opción de verlo abriéndose una pantalla para poder copiar su contenido.

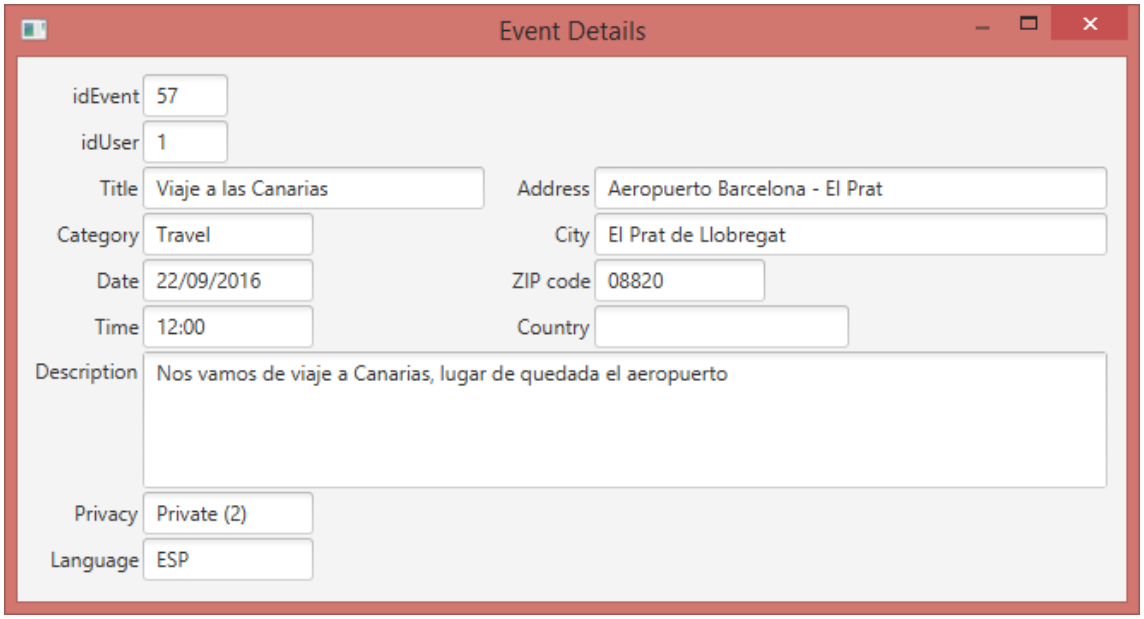
La segunda pestaña (ver Figura 3.7) está diseñada para que al accionar el botón "Read SQL tables" cargue los datos que hay en base de datos. Cada una de las tres secciones representa una tabla y el objeto `User`, `Event` o `Multimedia`. Esta parte de la interfaz gráfica permite que al seleccionar un usuario, se autoseleccionen en las secciones *Events* y *Multimedia* los eventos y archivos multimedia asociados a ese usuario. Siguiendo la misma filosofía, si seleccionamos un evento se nos filtraran automáticamente los archivos multimedia asociados a ese evento.

Figura 3.7: Interfaz gráfica del Generador de Eventos – pantalla de base de datos



Al hacer *dobleclick* sobre un evento de la tabla *Events* se abrirá un pop-up[49] con los campos del evento:

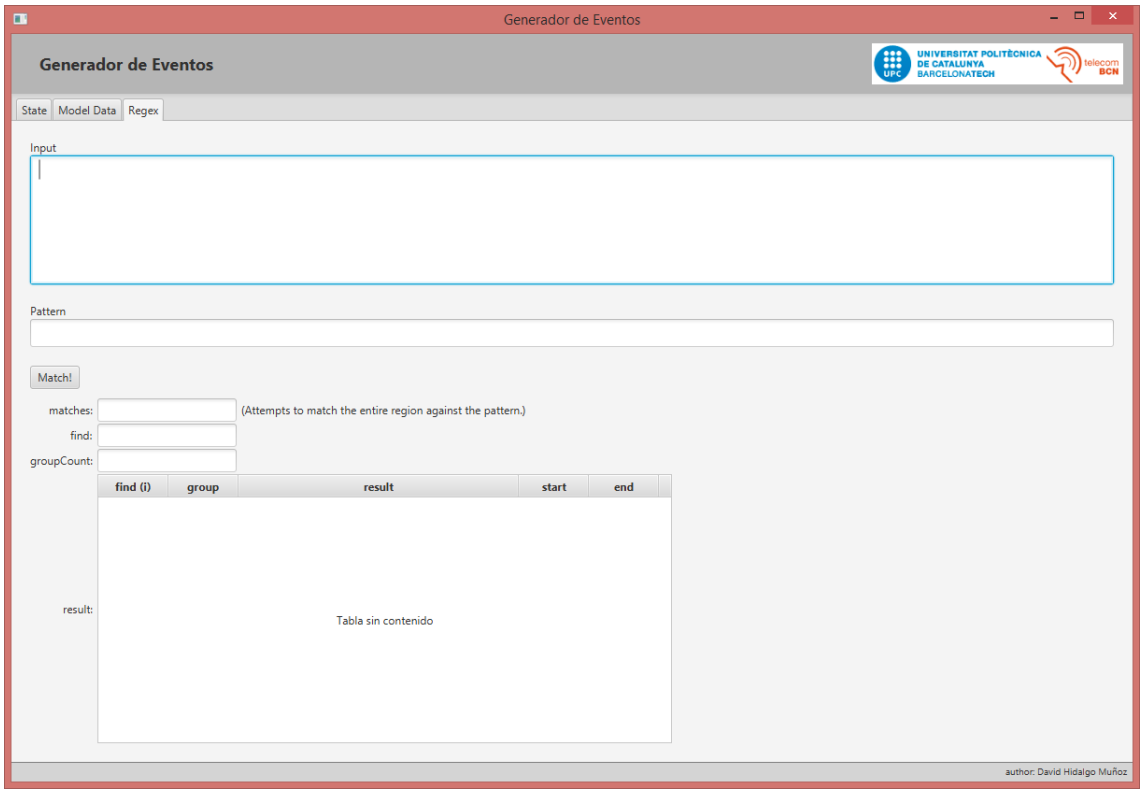
Figura 3.8: Interfaz gráfica del Generador de Eventos – pop-up con detalle de evento



[49] Ventana emergente.

Finalmente hay una tercera pestaña para realizar pequeñas pruebas con expresiones regulares con una sección para introducir un contenido y un patrón, y obtener los grupos que ha encontrado el el objeto `Matcher`.

Figura 3.9: Interfaz gráfica del Generador de Eventos – pantalla de pruebas regex



3.11.2. Clases de la interfaz gráfica

La interfaz gráfica tiene el proceso inicial de ejecución de la aplicación. Sus clases y archivos están contenidos en el paquete *main* y son:

Tabla 3.21: Clases y ficheros del paquete main

Paquete	Clase (.java)
main	Main
main	MainViewController
main	MainView.fxml
main	main.css
main	GELogger
main	GELogField
main.event	EventViewController
main.event	EventView.fxml

El estudio de la interfaz gráfica está fuera del ámbito de este proyecto puesto que ha sido una mejora añadida y no un requisito para su funcionamiento. Aunque la carga de trabajo ha sido considerable para comprobar que todas las acciones e interactuaciones con la interfaz

fuesen fluida. El estudio del código queda relegado a la visualización de éste en el código fuente del programa. Se darán algunas pinceladas de funcionamiento básicas de cada clase:

- La clase `Main` extiende de la clase `Application` de la API de JavaFX. Su método `start()` es invocado nada más comenzar la aplicación. En ella se carga el fichero fxml y se crea un objeto `Stage` asociando la escena contenedora de toda la estructura de datos de la interfaz. Una vez construida la interfaz gráfica se muestra por pantalla. Es importante indicar que al *clickar* en la "X" de cierre de la aplicación se ejecuta automáticamente la parada del módulo de generación de eventos antes de cerrar la aplicación si está arrancado. Adicionalmente también se invoca la desconexión a la base de datos con el método `DBManager.closePools()` visto en el apartado del modelo de datos.
- La interacción con la interfaz la controla la clase `MainViewController`. En ella hay dos acciones ejecutadas por los botones para arrancar y parar la aplicación. Esta clase es muy extensa y contiene todas aquellas acciones programadas para la interfaz gráfica.
- Las clases `GELogField` y `GELogger` son utilizadas por todo el Generador de Eventos y han substituido al típico chivato `System.out.println()`. Cada vez que se añade un registro al log, se inserta a un `ArrayList` de objetos `GELogField`. Estos objetos son volcados periódicamente en el contenedor de `GELogField` de la interfaz gráfica (concretamente cada 200 milisegundos). Esta configuración y la mayoría de inicializaciones de los modelos de datos de los objetos se realizan en el método `initialize()` de la clase `MainViewController`.

  Las operaciones de volcado de un listado a otro del log se realizan síncronamente, por lo que en medio de la operación de `update()` no se aceptarán nuevos registros de log y permanecerán bloqueados los listados hasta que se traspase la información del listado del `Logger` al de la interfaz gráfica. Este listado de la interfaz es el modelo de datos del objeto `ListView<GELogField>` que vemos en medio del Generador de Eventos (lista con el log).
- Por último, la clase `EventViewController` se encarga de cumplimentar con los datos de un evento el pop-up definido en el fichero `EventView.fxml`.

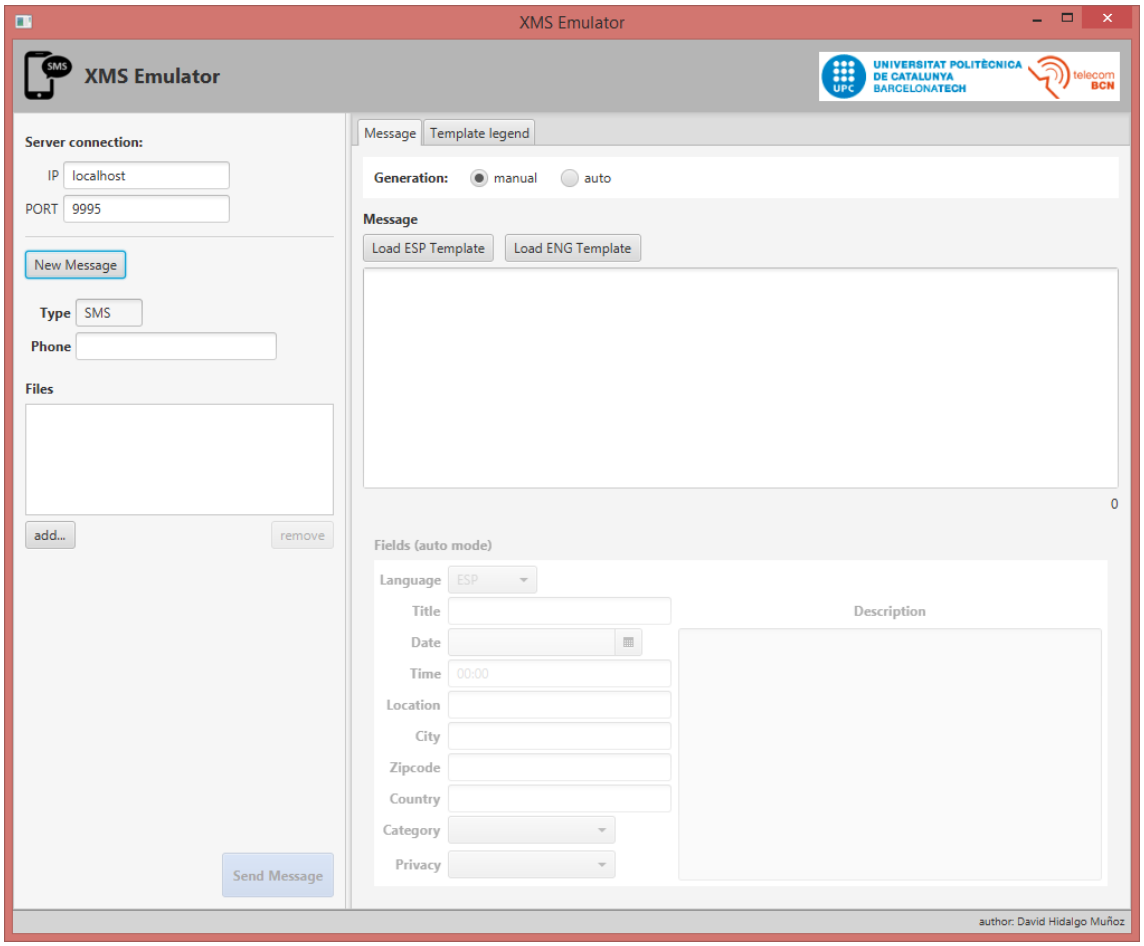
Lo más complejo de desarrollar para la interfaz gráfica han sido los métodos de registro de logs y la interactuación en la sección de base de datos de la selección y filtrado de elementos.

3.12. Emulador TCP para envío de SMS y MMS

Para interactuar con el servidor emulador se ha creado una pequeña aplicación con una simple interfaz gráfica para la ayuda de la edición de mensajes de texto y su envío al Generador de Eventos.

La aplicación permite enviar un mensaje SMS escrito manualmente o utilizando el método *auto,* que va cumplimentando el mensaje en función de los campos del evento que se vayan rellenando. También permite la opción de añadir ficheros al SMS transformándolo en un MMS. Una vez cumplimentado el cuerpo del mensaje e introducido el número de teléfono en la sección *Phone* se nos habilitará el botón "Send Message". El botón "New Message" realiza una limpieza rápida para escribir un nuevo mensaje.

Figura 3.10: Interfaz gráfica del XMS Emulator



Al ser una aplicación diseñada en JavaFX comparte las mismas similitudes que el Generador de Eventos en cuanto a estructura: clase `Main`, un `MainViewController` y un fichero `MainView.fxml` con el diseño gráfico.

Para esta aplicación comentaremos lo que se produce cuando presionamos el botón "Send Message":

Código 3.41: XMS Emulator – envío de mensaje

```
@FXML void sendMessageAction(ActionEvent event) {
	System.out.println("type: " + type.get());
	System.out.println("number: " + identifier.get());
	System.out.println("message: " + message.get());
	System.out.println("files: " + files.toString());

	String ip = textFieldIP.getText();
	String port = textFieldPORT.getText();
	if (ip.isEmpty() || port.isEmpty()) {
		Alert alert = new Alert(AlertType.WARNING);
		alert.setContentText("Los campos ip y/o puerto no pueden estar vacíos.");
		alert.showAndWait();
	} else {
		try {
			int int_port = new Integer(port).intValue();
			if (int_port < 1) {
				throw new Exception();
			}
			ClienteEmulador.enviarXMS(ip, int_port, type.get(),
identifier.get(), message.get(), files);
		} catch (Exception e) {
			Alert alert = new Alert(AlertType.WARNING);
			alert.setContentText("El puerto no es un número entero > 0");
			alert.showAndWait();
		}
	}
}
```

Al presionar el botón se desencadena la acción `sendMessageAction` que recoge los diferentes valores de los objetos que conforman la interfaz gráfica y se llama al método estático:

```
ClienteEmulador.enviarXMS(ip, int_port, type.get(),
identifier.get(), message.get(), files);
```

Esta aplicación tiene dos clases encargadas de enviar un mensaje al Generador de Eventos. Una es la clase `XMSFile` comentada anteriormente y que también dispone de ella el servidor TCP, necesaria para la correcta lectura de lo que le es enviado por el cliente.

La otra clase es `ClienteEmulador` que únicamente tiene ese método estático. Ese método realiza las siguientes operaciones:

Código 3.42: XMS Emulator – método enviarXMS

```
Socket S = new Socket(direccion, puerto);
XMSFile fileToTransfer = new XMSFile(tipo, numero, mensaje, new
ArrayList<File>(files));
System.out.println("Mensaje a enviar por el Emulador:");
System.out.println(fileToTransfer.ToString());
ObjectOutputStream outputStream = new ObjectOutputStream(S.getOutputStream());
outputStream.writeObject(fileToTransfer);
outputStream.close();
S.close();
```

Se crea un objeto `XMSFile` a partir de los parámetros `type.get()`, `identifier.get()`, `message.get()` y `files`. Estos objetos representan el tipo de mensaje (SMS o MMS), el número de teléfono, el contenido del mensaje y el `ArrayList` de ficheros `File`. Seguidamente si se consigue establecer conexión con la clase `Socket` con la dirección y puerto del servidor, se escribe por el socket hacia el servidor el objeto `XMSFile` y se cierra la conexión.

3.13. Diferencias de integración respecto a la Agenda Social

El módulo de generación de eventos inicialmente formaba parte del proyecto web Agenda Social, pero para la presentación final del presente trabajo se optó por mostrar una implementación independiente llamada Generador de Eventos.

Las principales diferencias con respecto a la versión integrada en la Agenda Social son:

- Los nombres de clases y métodos se han normalizado: la versión final del Generador de Eventos muestra la mayoría de clases y métodos en inglés.
- Se ha actualizado todo el modelo de datos, por lo que todas las llamadas al modelo para obtener datos de base de datos, modificar o insertar han sido rediseñados por completo. Se ha partido de la base de mantener una estructura de datos similar a la utilizada por la Agenda Social, necesario por el propio diseño del generador. La base de datos, y todas las clases y métodos han sido creados desde cero.
- El arranque y paro del módulo de generación de eventos estaba incrustado en la creación y destrucción del contexto de la aplicación web. Se ha incorporado esta funcionalidad a una interfaz gráfica con botones de arranque y parada.
- Los directorios de guardado de archivos multimedia y el acceso a los recursos se han actualizado, en la Agenda Social el acceso era mediante el contexto de aplicación y url. Además el proyecto estaba cargado y subido al servidor Glassfish. Para el Generador de Eventos los accesos se hacen a directorios del mismo proyecto, por referencia estática de clases para la carga de recursos.

3.14. Incorporación de nuevos idiomas

Aunque solo se ha preparado la aplicación para los idiomas español e inglés. No se tiene limitación para la incorporación de nuevos diccionarios para otros idiomas. A continuación se marcan las modificaciones que deben realizarse para la incorporación de un nuevo idioma, tomando como ejemplo la inclusión de un nuevo diccionario para catalán.

1) Crear una nueva clave manteniendo la estructura en el fichero *dictionaries.properties* del paquete *eventParser.dictionary*:

```
0003=CAT
```

2) Incorporar un nuevo diccionario CAT.dic al paquete *eventParser.dictionary*. Este diccionario debe cumplir las normas descritas en el apartado 3.7.2.

3) Si existe algún carácter especial que debe mantenerse, como 'ç' "ce trencada" o '·' de la "ela geminada" se mantendrá en el diccionario y se añadirán como excepción a la hora de adecuar el texto a buscar en el diccionario. En este caso las acciones de limpieza quedarían así (visto en Código 3.32):

```
text = text.toLowerCase();
text = StringManipulator.quitarAcentos(text);
text = text.replaceAll("[[^\\p{Alpha}]&&[^ñç·']]", " ");
text = text.replaceAll("[ ]+", " ");
```

4) Crear la sección de expresiones regulares correspondientes al idioma <CAT> en los ficheros *eventParser.mail.MailGrammar.xml* y *eventParser.sms.SmsGrammar.xml*. Para una mayor agilidad se puede copiar los del idioma ESP y realizar las traducciones adecuadas de las palabras clave.

5) Crear las plantillas para SMS y correos para creación de eventos semiautomáticos conforme a las palabras clave de los ficheros de gramáticas.

6) En el caso de la Agenda Social: Informar a la comunidad de usuarios de la disponibilidad de las nuevas plantillas y la capacidad de analizar y parsear correos en idioma catalán correctamente.

7) Si el número de diccionarios crece, estudiar la implantación de otro método de carga de los diccionarios en el arranque de la aplicación o el método de determinación de idioma. Un número elevado de idiomas puede afectar a la velocidad de determinación del idioma de un texto puesto que debe analizarse para todos los idiomas declarados.

3.15. Modo de funcionamiento sin base de datos

El funcionamiento por defecto del Generador de Eventos utiliza una base de datos principalmente por tres motivos: consultar si el contenido recibido lo ha enviado un usuario registrado, poder realizar consultas para encontrar campos de evento y poder guardar los resultados obtenidos.

Para poder probar la aplicación sin una conexión a base de datos se ha añadido un control de conexión a ésta. En caso de fallar la conexión se configura el Generador de Eventos en modo de funcionamiento NO_DB_MODE. Las principales afectaciones son:

- Cuando se recibe un contenido, al buscar su usuario propietario siempre se devolverá un usuario virtual, con un identificador de usuario no nulo. Recordemos que si el usuario era nulo (no encontrado), en el modo de funcionamiento normal no se hubiese proseguido con el análisis del contenido. En este nuevo modo, sí acabamos viendo el evento generado aunque no se almacena en base de datos.
- La búsqueda adicional de campos basados en base de datos quedan inutilizados aunque ejecutados, dando como resultado todas las consultas a base de datos un contenido vacío de resultados. En este caso se pierde gran parte de la potencia de obtención de campos de los buscadores.
- Cuando un evento se ha creado no se procede a su guardado en base de datos.

3.16. Librerías utilizadas por el Generador de Eventos

Como la mayoría de desarrollos en Java, se utilizan librerías de terceros de partes de código y funcionalidades que nativamente no están implementadas en Java y son requeridas para implementar rápidamente soluciones de mayor envergadura.

Las librerías utilizadas por el proyecto tienen que estar cargadas en el Entorno de Desarrollo (IDE) para su compilación y ejecución:

- Operaciones con correo electrónico:

```
mail.jar
```

- Librerías de Parlay X para uso de mensajería SMS:

```
ParlayX.jar
activation.jar
axis.jar
commons-discovery-0.2.jar
commons-logging-1.0.4.jar
saaj.jar
wsdl4j-1.5.1.jar
jaxrpc.jar
```

- Acceso a base de datos:

```
bonecp-0.8.0.RELEASE.jar
guava-18.0.jar
slf4j-api-1.7.12.jar
slf4j-simple-1.7.12.jar
mysql-connector-java-5.1.23-bin.jar
```

Las librerías se incorporan junto con las dos aplicaciones en los anexos digitales en el directorio “_libreriasGE/”

Capítulo 4

Pruebas y resultados

En el presente capítulo se detallan las metodologías de pruebas realizadas y se muestran algunos ejemplos. Se analizan también los resultados de la creación de eventos a partir de SMS, MMS y e-mails con formatos predefinidos, escritos por el usuario o reenviados directamente desde su cuenta de correo electrónico.

4.1. Metodología de las pruebas

Como en la mayoría de proyectos de programación con cierta envergadura, las diferentes partes de la aplicación han sido generadas y probadas por separado, creando pequeños proyectos con los métodos a validar. Una vez verificado el correcto funcionamiento se ha unido a bloques mayores creando el conjunto completo que forma el Generador de Eventos. Las principales pruebas de validación de código por áreas han sido las siguientes:

4.1.1. Pruebas de la pasarela Parlay X

Para probar la pasarela de Orange ha sido necesario tener acceso al servicio mediante la pasarela Parlay X. Las pruebas de este bloque han sido sencillas, simplemente ha requerido probar que el método de recepción de mensajes por la pasarela era correcto. Las pruebas han requerido la verificación del arranque del servidor Axis y la recepción de un mensaje de texto enviado desde un terminal móvil. Análogamente, se probó también el envío de mensajes desde el servidor Axis, verificando que eran recibidos por el móvil del usuario.

La pasarela solo estuvo activa unas pocas semanas, quedando la recepción y envío de SMS inhabilitados. Para realizar pruebas de creación de eventos mediante SMS debe utilizarse el emulador TCP. Los usuarios registrados en base de datos deben estar configurados para que no envíen notificación por SMS porque se generarán errores de conexión del servidor Axis.

4.1.2. Pruebas de correo de Gmail

Al igual que con la gestión de la comunicación para mensajería de texto, para las pruebas de comunicación vía correo electrónico se generó un proyecto independiente para finalmente, anexar la funcionalidad de envío y recepción y los demás métodos de procesado de información al proyecto principal. Durante el estudio y pruebas se trabajaron en dos escenarios diferentes:

- Validar el correcto funcionamiento de lectura y envío de correos, verificando también el movimiento de éstos de la bandeja de entrada (INBOX) a otras carpetas según proceda. Esta parte ha requerido de un estudio de la estructura interna de una cuenta de Gmail.
- Análisis de cómo se almacena el contenido de los correos y cómo se anidan sus diferentes partes dentro del contenedor (objeto `Message`). Por ejemplo, un correo en texto plano solo tiene como único elemento el texto del mensaje, mientras que correos más complejos tienen una estructura en árbol con diferentes elementos definidos por sus tipos MIME.

4.1.3. Pruebas de conexión a base de datos

Las pruebas básicas para la utilización de la base de datos han sido la de establecer desde el proyecto una conexión a la base de datos del Generador de Eventos y probar la lectura de tablas y los métodos de consulta e inserción de datos. En la segunda pestaña del Generador de Eventos podemos observar las tres tablas del esquema “ge” que utiliza la aplicación y su correcta cumplimentación al leer las tablas.

4.1.4. Pruebas de patrones de búsqueda

Las pruebas de la correcta ejecución de las expresiones regulares han consumido gran parte de los esfuerzos para que el Generador de Eventos se comportase lo mejor posible ante una gran variedad de contenidos diferentes. Se han ido verificando las diferentes expresiones regulares de los patrones para cada buscador (con su campo de evento) e idioma con contenidos reales (SMS o correos) recibidos por la aplicación. Una de las principales dificultades encontrada es que cualquier ligera modificación en un patrón puede acarrear desastrosos resultados en la generación de todos los eventos futuros, un resultado caótico que tiene una cierta similitud con el conocido "efecto mariposa"[50].

4.1.5. Pruebas de adecuación de contenidos

Una de las partes más complejas ha sido el comprobar que el contenido obtenido en un correo electrónico o SMS era correctamente limpiado por la aplicación. No por el hecho de aplicar las operaciones de limpieza, sino de evaluar cómo debían adaptarse y transformarse los contenidos para la posterior aplicación de los patrones.

4.1.6. Pruebas de métodos del diccionario

Las pruebas del módulo de diccionario se han reducido a la comprobación de un correcto guardado en memoria de las palabras de los idiomas almacenadas en ficheros. Una vez cargados los diccionarios en memoria, se han utilizado los métodos de búsqueda de palabras en el diccionario y comprobado que eran halladas en ellos. Finalmente para identificar el idioma de un contenido se han hecho pruebas directamente con los contenidos enviados (especialmente por correo). Hay que recordar que la detección del idioma de un SMS se efectúa a partir de los identificadores de campo utilizados en la plantilla de envío.

4.1.7. Pruebas de la interfaz gráfica

Las pruebas de la interfaz gráfica se han ido realizando a medida que se iba generando e incrustando las partes de código del Generador de Eventos. Ha sido un método de generación e inmediata validación. Su cometido no es otro que ejecutar pequeños métodos del Generador de Eventos y lanzar los procesos. Las mayores complicaciones a la hora de verificar el correcto funcionamiento han sido la elaboración del log de la aplicación y la visualización de datos filtrados de base de datos.

[50] Efecto mariposa: concepto utilizado en la teoría del caos. Hace referencia a que una mínima variación de las condiciones iniciales provoca que el sistema evolucione en formas completamente diferentes e inesperadas.

4.2. *Prueba global del Generador de Eventos*

En esta sección se detallan las pruebas del proceso de creación de un evento.

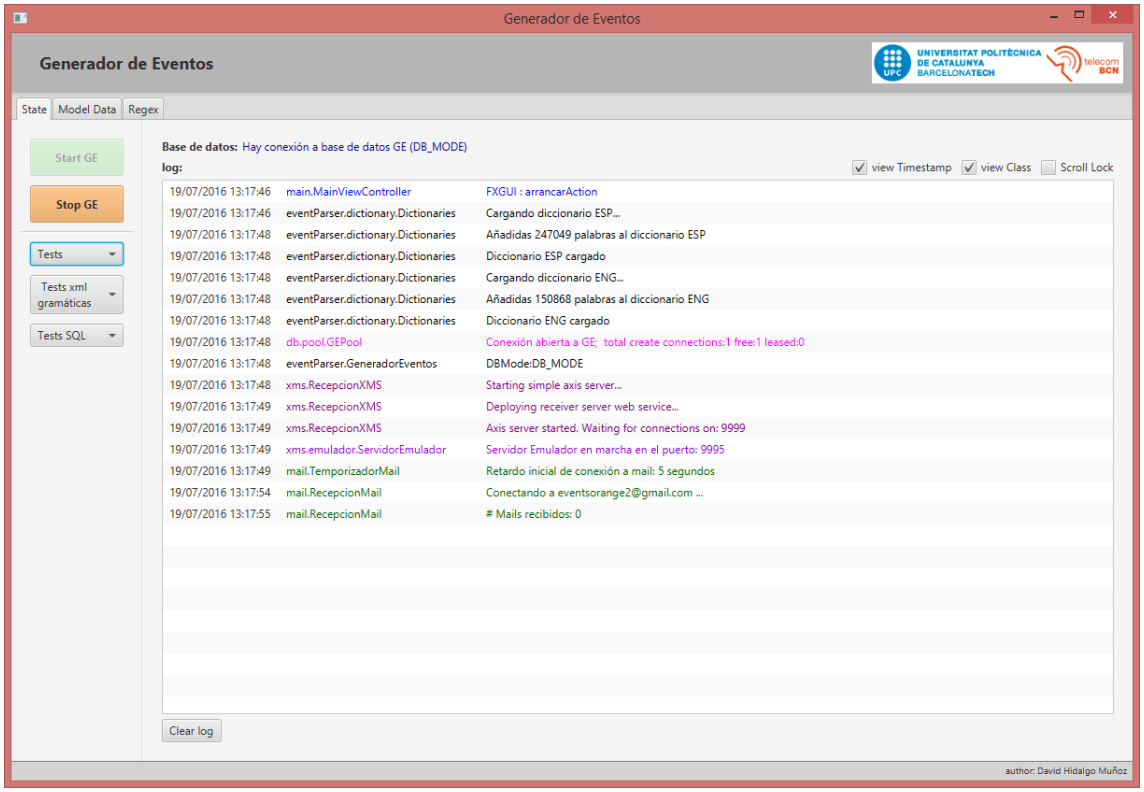
4.2.1. Arranque y parada del Generador de Eventos

La primera acción a realizar es arrancar el módulo de generación, esta acción se divide en tres partes:

- Instanciar la clase `GeneradorEventos`, que implica la carga de diccionarios.
- Crear la conexión a base de datos.
- Arrancar los tres servicios: el servidor Axis, el servidor TCP para el emulador y el temporizador de conexión a la cuenta de Gmail.

En la Figura 4.1 se observan las tres acciones desencadenadas tras presionar el botón “Start GE”. Se observa que se establece conexión a base de datos y se instancia el Generador de Eventos creando los diccionarios. También vemos que el servidor Axis se queda escuchando en el puerto 9999, el servidor TCP del emulador en el 9995 y se realiza la primera conexión a la cuenta de correo *eventsorange2@gmail.com*.

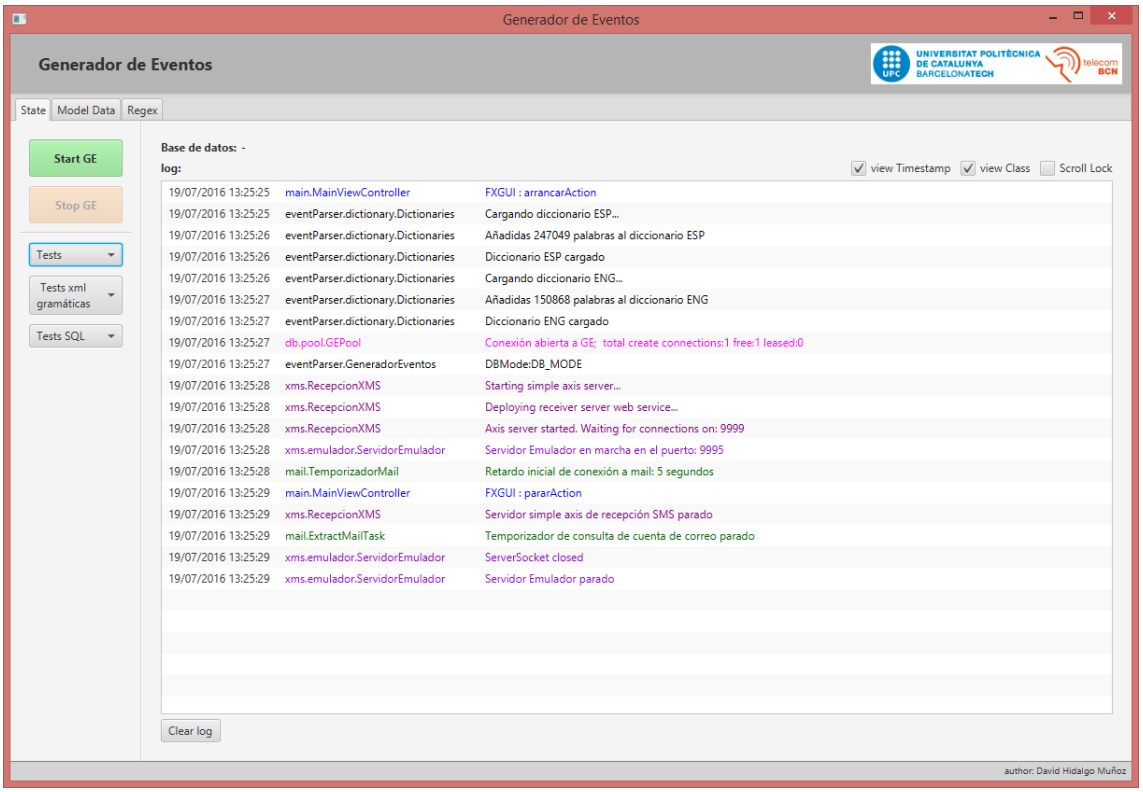
Figura 4.1: Arranque del Generador de Eventos



Es importante no arrancar el generador de eventos de nuevo sin haberlo parado porque los servidores no podrían crearse correctamente al estar ocupados sus puertos, además tendríamos dos procesos de exploración de la cuenta de correo electrónico de la aplicación.

La acción de parado con el botón “Stop GE” detiene los dos servidores y el temporizador de conexión a la cuenta de correo. Tras esta opción vuelve a estar disponible el arranque del Generador de Eventos. Cerrar la aplicación desde el botón de cerrado en la esquina superior derecha ejecuta la misma rutina del botón de parada antes de cerrar la aplicación, liberando los recursos.

Figura 4.2: Parada del Generador de Eventos



4.2.2. Recepción de contenido y generación del evento

Una vez arrancado el Generador de Eventos enviamos un correo de prueba a la cuenta de la aplicación (o un mensaje mediante el emulador de SMS) para verificar su recepción y procesado. Las capturas de recepción de contenidos para e-mail y SMS se pueden ver en el apéndice D, Figura D.1 y Figura D.2 respectivamente. Seguidamente arranca el proceso de parseado del contenido recibido, en la Figura D.3 hay una serie de capturas con el log del flujo de creación de un evento a partir de un SMS de ejemplo. En los logs se ven qué partes generan la información en función de la clase que ha generado el mensaje y a qué hora se ha generado el registro.

Al recibir el mensaje por el servidor se invoca el método de parseado según el tipo de mensaje recibido. En función de si es un correo o un SMS se muestra el mensaje recibido y el contenido final una vez limpiado (en caso de ser un correo en formato HTML).

Mediante el módulo de diccionarios se evalúa el número de palabras de cada idioma que hay en el mensaje consultando al diccionario y se determina el idioma del contenido.

Se detallan también todos los buscadores que van a participar en la creación del evento para cada campo. Siendo específicos si tienen nombres propios o genéricos (`Finder`). Para

cada uno de los Buscadores se informará del campo que está analizándose y de si ha encontrado una coincidencia y con qué patrón la ha obtenido.

Una vez finalizados todos los buscadores se informa de la creación y guardado en base de datos del evento y su identificador asociado (`idevent`). Si hubiese contenidos multimedia a buscar se ejecutarían después del guardado del evento. Se informarían con el mensaje "Extrayendo archivos (n)" en el log.

Si el usuario tiene activada la notificación por correo se le enviará un e-mail con el evento creado. También podemos comprobar que el evento ha sido creado buscándolo en base de datos por su identificador; realizando pruebas, será el evento con identificador más alto si es el último que ha sido creado.

En el proceso de creación de un evento nos podemos encontrar los siguientes casos especiales:

- No existe el usuario o no ha sido hallado mediante su número de teléfono o dirección de correo. En este caso la aplicación, a no ser que esté en modo NO_DB_MODE, no continua con la creación del evento indicando en el log: "Propietario del SMS (o e-mail) no encontrado. El evento no ha podido crearse correctamente: event==null".
- Notificación de envío desactivada. En este caso no aparecerá en el log "Enviado correo de confirmación a: <dirección de correo del usuario>".
- Recepción de datos adjuntos. Tras la creación de un evento con datos adjuntos podremos verificar que los contenidos están almacenados en el directorio raíz del proyecto Generador de Eventos (o ejecutable jar) en la carpeta multimedia.

4.2.3. Envío de verificación por correo

Si el usuario tiene activada en su configuración de base de datos la notificación por e-mail, recibiremos un e-mail enviado desde la cuenta de correo de la aplicación, conteniendo el evento generado. Un ejemplo de correo puede verse más adelante en la Figura 4.5, en el primer ejemplo analizado.

4.3. Prueba de generación de eventos con SMS y MMS

En el Capítulo 3 tres se ha visto el diseño del Emulador de SMS y MMS. Para las pruebas utilizaremos el modo “auto” de generación de mensajes y evaluaremos el comportamiento de los diferentes casos que nos podemos encontrar. En el primer punto veremos un ejemplo representativo de la creación de un evento completo, se ha relegado al segundo punto y apéndices otros ejemplos y casos especiales.

4.3.1. Prueba con mensaje de texto completo

Cumplimentamos el evento a enviar con todos los campos rellenados como se muestra en la Figura 4.3.

Figura 4.3: SMS de prueba – cumplimentación de campos

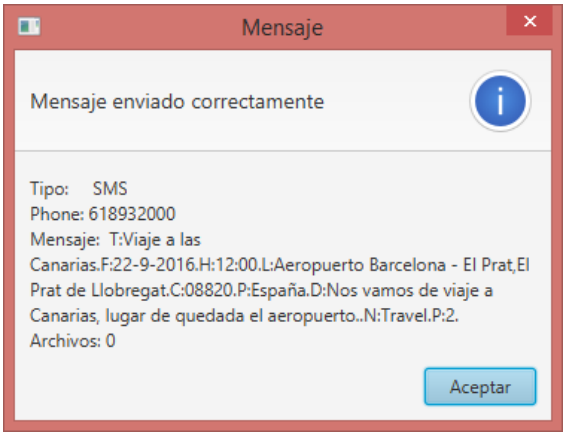
Además en la sección “Phone” debemos introducir el número de teléfono de un usuario existente en base de datos.

El cuerpo del mensaje queda de la siguiente manera:

```
T:Viaje a las Canarias.F:22-9-2016.H:12:00.L:Aeropuerto Barcelona - El Prat,El Prat de Llobregat.C:08820.P:España.D:Nos vamos de viaje a Canarias, lugar de quedada el aeropuerto..N:Travel.P:2.
```

Al enviar el mensaje obtenemos un aviso, mostrado en la Figura 4.4, conforme el mensaje ha sido enviado correctamente a la aplicación.

Figura 4.4: SMS de prueba – confirmación de envío



Tras ser recibido por el Generador de Eventos, procesado y creado el evento correspondiente, obtendremos en el log el mensaje de creación donde se observa la correcta cumplimentación de todos los campos.

Id Event = 59
Id User = 1
Title = Viaje a las Canarias
Date = 22/09/2016
Time = 12:00
Category = Travel
Address = Aeropuerto Barcelona - El Prat
City = El Prat de Llobregat
Zipcode = 08820
Country = 2
Description = Nos vamos de viaje a Canarias, lugar de quedada el aeropuerto
Timestamp = Tue Jul 19 17:13:52 CEST 2016
Privacy = 2

Se comprueba también la llegada del e-mail de confirmación de creación del evento al correo del usuario:

Figura 4.5: SMS de prueba – correo de confirmación de creación de evento

Event created successfully

Title: Viaje a las Canarias

Date / Time: 2016-09-22, 12:00
Location: Aeropuerto Barcelona - El Prat, El Prat de Llobregat, 08820 (2)
Category: Travel
Description: Nos vamos de viaje a Canarias, lugar de quedada el aeropuerto

4.3.2. Pruebas adiconales de SMS

Se han realizado muchas más pruebas con varios SMS poniendo a prueba varias de las funcionalidades y requisitos que debía cumplir el parseado de mensajes de texto. Algunos ejemplos de los casos más representativos se encuentran en el Anexo E.1 y son:

- Mensaje vacío. Este caso estudia la generación de un evento sin ningún campo cumplimentado, de esta manera se pueden ver los campos que son rellenados por defecto.
- SMS con algunos campos rellenos. Este es el caso más común de creación de eventos a partir de SMS. Se comprueba el correcto funcionamiento de la búsqueda indirecta de información, cumplimentando campos de evento a partir de la información que hay en base de datos.
- Diferentes idiomas. Se evalúa la capacidad del Parseador para determinar el idioma del mensaje a partir de las etiquetas utilizadas.
- Archivos multimedia. En este ejemplo se observa el análisis de un MMS, es decir, un SMS con archivos multimedia adjuntos. Tras la creación del evento debe verificarse el guardado de los archivos multimedia en el directorio “multimedia” ubicado en la raíz de la aplicación y en la base de datos.
- Casuísticas especiales. Esta sección recoge casos especiales como el intento de creación de un evento cuando el Generador de Eventos recibe un mensaje con un número de teléfono que no pertenece a ningún usuario.

4.4. Prueba de generación de eventos con e-mails

Para las pruebas de creación de eventos a partir de correos electrónicos tenemos tres escenarios:

- Correos creados utilizando plantillas.
- Correos redactados por el usuario utilizando o no etiquetas para algunos campos.
- Correos reenviados por los usuarios a la cuenta de correo de la aplicación, normalmente generados por terceras aplicaciones y recibidos en las cuentas de correo de los usuarios.

Ambos métodos de parseado (para SMS o e-mail) tienen las mismas casuísticas vistas en la generación de eventos a partir de un SMS, por lo que serán tratados de igual forma. Estos casos son el intento de creación de un evento a partir de un correo electrónico de un usuario que no está registrado, o la generación de un evento a partir de un correo vacío.

- En el caso de no existir el usuario se nos indicará en el log con el mensaje tras intentar crear el evento:

  Propietario del e-mail no encontrado
  El evento no ha podido crearse correctamente: event==null

  Además podremos comprobar si nos conectamos a la cuenta de correo del Generador de Eventos que dicho correo estará en la carpeta *UNPARSED*.

- El caso de un evento vacío lo podemos ver en el apéndice E.2.1.

A continuación se examinan los resultados obtenidos tras la creación de eventos en los tres escenarios.

4.4.1. Correo electrónico con plantilla

Los patrones de búsqueda de los correos parten de la misma base que los generados para los SMS, solo que están más ampliados para encontrar resultados en una mayor variedad de contenidos. Las plantillas utilizadas se encuentran en el apéndice C.2 y su funcionamiento es idéntico a la generación por SMS. Cabe destacar que el parseador de correos se ha diseñado para poder utilizar plantillas pero no es su objetivo final, por lo que tiene algunas diferencias con respecto al de SMS.

La diferencia más notable es que tenemos más libertad a la hora de rellenar los campos y siempre se intentarán aplicar otros patrones antes que los definidos para las búsquedas de contenido por etiquetas. Es más, el concepto de etiqueta pasa de ser un único carácter a una palabra. Eso sí, la estructura de delimitación de etiqueta, separador y punto final se mantiene a la hora de aplicarse, siendo más permisiva que la de mensajes de texto.

Cogiendo como ejemplo el siguiente evento:

Mensaje:
Asunto: Quedada
TITULO : Reunión de amigos
FECHA : 13 de mayo
HORA : 17:00
CALLE : Plaza de Sants
CIUDAD : Barcelona
CODIGO POSTAL : 08014
PAIS : España
CATEGORIA : Social
PRIVACIDAD: Registrado
DESCRIPCION : Nos reunimos lo amigos para pasar la tarde.

Evento generado:
Id Event = 80
Id User = 1
Title = Reunión de amigos
Date = 13/05/2016
Time = 17:00
Category = Social
Address = Plaza de Sants
City = Barcelona
Zipcode = 08014
Country = España
Description = Nos reunimos lo amigos para pasar la tarde.
Timestamp = Tue Jul 19 20:24:15 CEST 2016
Privacy = 1

Comentario:
Se observa que el funcionamiento utilizando plantillas es similar al obtenido con los SMS. Los campos son encontrados y cumplimentados sin mayor problema.

4.4.2. Correos redactados

La principal ventaja del parseador de e-mails es la gran cantidad y variedad de expresiones regulares que tiene para crear eventos a partir de texto sin una estructura definida. Hay que tener en cuenta que la *inteligencia* del Generador de Eventos es limitada y en según qué casos habrá que indicarle algunos campos para que los encuentre.

Por ejemplo, hay campos más fáciles de encontrar como fechas y horas, nombres de ciudades o países y determinar una categoría. En cambio campos como el código postal o incluso el título o descripción son mucho más difíciles de hallar. Cabe destacar que aunque a simple vista algo parezca obvio, sin referencias de ningún tipo el Generador de Eventos no puede saber si una simple cadena de texto es un título, dirección o una ciudad. Es por ello que es necesario que en los patrones y en sus expresiones regulares haya unas ciertas referencias para poder hallar un campo del evento.

También hay que destacar que el análisis es secuencial, tanto de recorrido del texto como en la aplicación de patrones. Por lo que si en el correo hay dos fechas siempre escogerá la

primera y ante varias palabras clave, determinará la categoría a partir de la primera encontrada si ningún patrón es hallado.

Evaluaremos unos cuantos correos de prueba para ver qué parámetros es capaz de encontrar y qué resultados se obtienen:

4.4.2.1. Correo redactado simple

Mensaje:
Asunto: Asistencia al monólogo Risas
Asistimos a un monólogo el día 24 de agosto a las 10:00 pm en La Chocita del Loro

Evento generado:
Id Event = 86
Id User = 1
Title = Comedy Event (Title not found)
Date = 24/08/2016
Time = 22:00
Category = Comedy
Address = null
City = null
Zipcode = null
Country = null
Description = AutoGenerated Event - (Mail Subject: Asistencia al monólogo Risas)
Timestamp = Tue Jul 19 22:35:55 CEST 2016
Privacy = 2

Comentario:
En este mensaje sencillo de correo escrito por el usuario se indica que va a asistir a un monologo un cierto día en lo que parece ser alguna sala llamada “La Chocita del Loro”. Observamos que el parseador ha sido capaz de hallar correctamente la fecha en formato escrito y la hora incluso con el indicador *pm*. Mediante la palabra clave “monólogo” ha podido determinar que el evento tenía una categoría de tipo “Comedia”. En cambio lo siguiente que le ha quedado por hallar ha sido lo que posiblemente sea el lugar. Tampoco se ha especificado ningún título ni descripción.

4.4.2.2. Correo redactado ampliado

Para este caso vamos a utilizar un híbrido entre un correo redactado y la utilización de algunas etiquetas de referencia para que el parseador sea capaz de hallar mejor los resultados.

Mensaje:
Asunto: Asistencia al monólogo Risas
Evento: Monologo Risas
Asistimos a un monólogo el día 24 de agosto a las 10:00 pm en el local: La Chocita del Loro
Descripción: El próximo miércoles día 24 de agosto asistiremos a las diez de la noche al monólogo Risas en el local La Chocita del Loro.

Evento generado:
Id Event = 89
Id User = 1
Title = Monologo Risas
Date = 24/08/2016
Time = 22:00
Category = Comedy
Address = La Chocita del Loro
City = null
Zipcode = null
Country = null
Description = El próximo miércoles día 24 de agosto asistiremos a las diez de la noche al monólogo Risas en el local La Chocita del Loro.
Timestamp = Tue Jul 19 22:53:08 CEST 2016
Privacy = 2

Comentario:
En la creación del evento vemos que al indicar mejor el título, lugar y descripción con una etiqueta indicadora seguida de un separador ':' obtenemos los resultados deseados. En este caso como no se han indicado la ciudad no es capaz de hallar nada más porque no está presente en el correo. Tengamos en cuenta que a partir de una dirección no se puede hallar en base de datos su ciudad o código postal correspondiente, porque varias ciudades pueden tener el mismo nombre de calle.

4.4.2.3. Correo redactado con búsquedas en base de datos

Modificamos el correo para indicar la ciudad pero sin utilizar ninguna etiqueta:

Mensaje:
Asunto: Asistencia al monólogo Risas
Evento: Monologo Risas
Asistimos a un monólogo el día 24 de agosto a las 10:00 pm en el local: La Chocita del Loro
Descripción: El próximo miércoles día 24 de agosto asistiremos a las diez de la noche al monólogo Risas en el local La Chocita del Loro, en Madrid.

Evento generado:
Id Event = 91
Id User = 1
Title = Monologo Risas
Date = 24/08/2016
Time = 22:00
Category = Comedy
Address = La Chocita del Loro
City = Madrid
Zipcode = 28160
Country = España
Description = El próximo miércoles día 24 de agosto asistiremos a las diez de la noche al monólogo Risas en el local La Chocita del Loro, en Madrid.
Timestamp = Tue Jul 19 23:04:59 CEST 2016
Privacy = 2

Comentario:

En este correo hemos añadido al final de la descripción la ciudad “Madrid”. En este caso no va a ser hallado por ningún patrón porque carece de ninguna referencia cercana para que sea hallado como ciudad, sino que se van a buscar las ciudades que hay en base de datos ordenadas por frecuencia de aparición en ella en el correo. Como la ciudad es hallada, también se busca en base de datos el código postal asociado y el país si están en base de datos. Para este correo nos encontramos que en la base de datos, la ciudad Madrid tiene mayormente el código postal 28160 y como país España.

Para el código postal se han contemplado dos opciones: descartar si hay más de uno o elegir el de mayor aparición. En caso de poblaciones con un único código postal tiene sentido la segunda opción por si hay algún error en base de datos. En cambio para ciudades con varios códigos postales aparecerá erróneamente. En este caso se ha decantado el diseño para que siempre sea cumplimentado con el código postal de mayor aparición.

4.4.3. Correo reenviado de billete de confirmación de Renfe

Este escenario contempla correos que suelen enviarse automáticamente a la cuenta del usuario, y que éste puede reenviar directamente a la aplicación. Lo ideal sería configurar un filtro de reenvío en el correo del usuario para que ciertos mensajes se reenviasen a la aplicación, es el escenario ideal para el Generador de Eventos, pero en este caso reenviaremos manualmente algunos correos de ejemplo. Este es el escenario más difícil para probar la eficacia del Generador de Eventos.

Mensaje:

renfe

Hola David.
Su compra se ha realizado correctamente. Compruebe que los datos sean correctos.
Localizador: AST5ZR

Ida : Viernes 18-04-2016 Origen : Madrid-Puerta De Atocha Destino : Barcelona-Sants

Tren	Salida	Llegada	Clase	Coche	Plaza	Tarifa	Precio (Euros)
AVE 03169	19:30	21:27	Turista	004	14C	Ida y Vuelta	79,00
			Turista	004	14D	Ida y Vuelta	79,00

Importe Total del Viaje de Ida : 158,00EUR

Ida : Domingo 20-04-2016 Origen : Barcelona-Sants Destino : Madrid-Puerta De Atocha

Tren	Salida	Llegada	Clase	Coche	Plaza	Tarifa	Precio (Euros)
AVE 03214	12:30	15:23	Turista	010	23D	Ida y Vuelta	79,00
			Turista	010	24D	Ida y Vuelta	79,00

Importe Total del Viaje de Vuelta: 158,00EUR

Importe Total de la Compra : 316,00 EUR

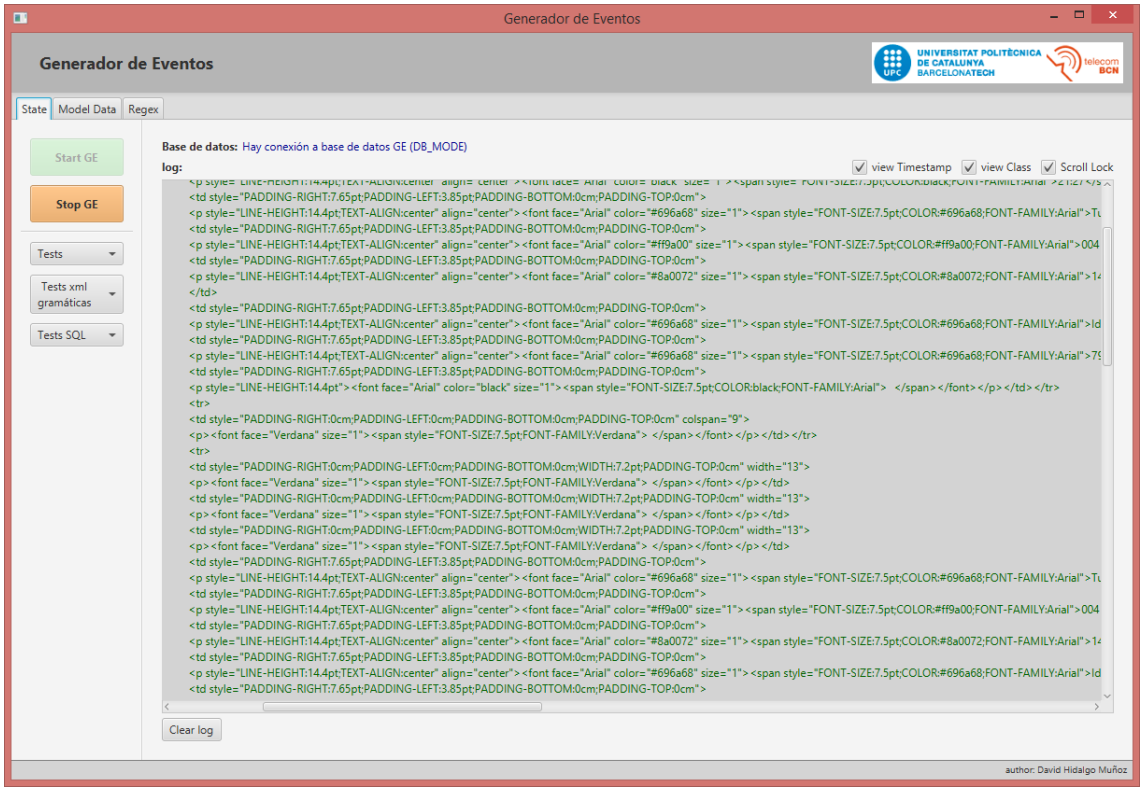
Le deseamos un feliz viaje.

Evento generado:
Id Event = 97
Id User = 1
Title = Travel Event (Title not found)
Date = 18/04/2016
Time = 19:30
Category = Travel
Address = Barcelona-Sants
City = Barcelona
Zipcode = 08830
Country = España
Description = AutoGenerated Event - (Mail Subject: Confirmacion de venta Renfe)
Timestamp = Wed Jul 20 00:04:49 CEST 2016
Privacy = 2

Comentario:

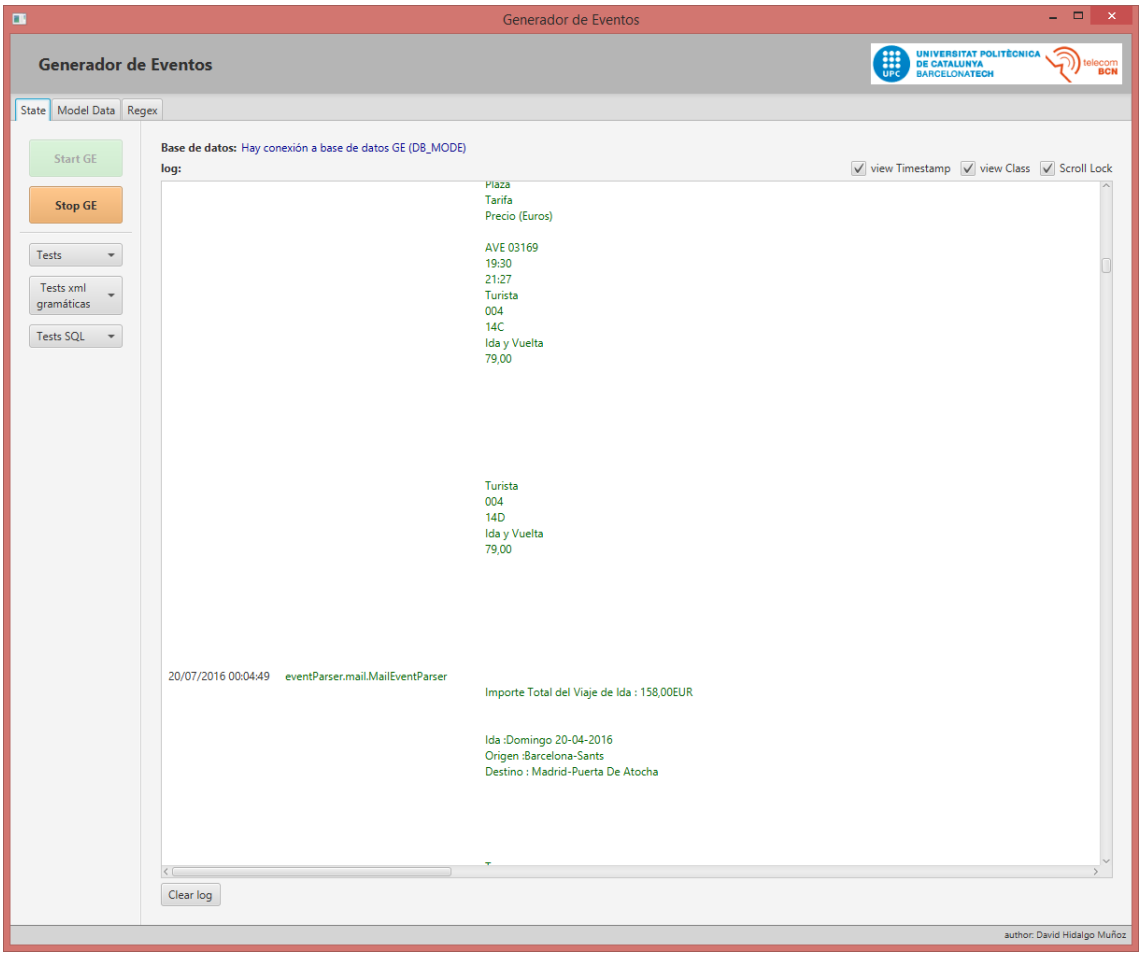
Antes de nada mostraremos en dos figuras tanto el formato del correo recibido como del correo limpio para la aplicación de los patrones. Nótese que sin el proceso de limpieza resulta imposible analizar nada incluso por un ser humano. Es importante tener en cuenta que aunque visualmente puede parecer fácil de entender de qué trata el evento que se quiere crear, el Generador de Eventos trabaja con cadenas de caracteres que tiene que analizar aplicándole patrones con expresiones regulares.

Figura 4.6: E-mail recibido en bruto con su formato en HTML



En Figura 4.6 se observa toda la estructura HTML que forma el correo y es limpiada dando como resultado lo que se muestra en la Figura 4.7.

Figura 4.7: E-mail recibido limpiado y listo para procesar



Aunque el texto se ha espaciado, resulta mucho más comprensible intentar analizar ahora el contenido aplicando patrones de búsqueda. Este proceso de limpieza es necesario y además debe ser igual para todos los correos, por lo que algunos incluso después de ser limpiados puede que tengan una estructura de texto que no permita el hallazgo de la información.

Los patrones han hallado los siguientes resultados (extraído del log):

Patrón encontrado en Date: \D([012]?\d[-\\ /_]0?\d[-\\ /_]20\d\d)\D
Patrón encontrado en Time: \D((?:2[0123])|(?:[01]?\d))[:]([012345]\d)\D
Patrón encontrado en Address: DESTINO[ ]?#?[:=][ ]?([\p{Alpha}[\p{Punct}&&[^\.:#=;]]][\w[ºª][\p{Punct}&&[^\.:#=;]]])[\.#;[\s&&[^]]]*
Patrón encontrado en Category: \WVIAJE\W

Se han obtenido la fecha, la hora, la dirección y la categoría directamente del correo utilizando patrones de búsqueda con expresiones regulares. El título y descripción han sido generados por el parseador indicando que no han sido hallados pero con alguna información básica. La ciudad en este caso ha sido obtenida gracias a una función de búsqueda auxiliar en la que una vez hallada la dirección, se han buscado las ciudades de base de datos en su

contenido. En este caso se ha hallado que la ciudad de Barcelona está contenida en Barcelona – Sants. Tanto el código postal como el país se han encontrado en base de datos a partir de la ciudad "Barcelona". Para este correo también se obtiene la imagen del logotipo de Renfe incrustado en el e-mail como archivo multimedia.

4.4.4. Pruebas adiconales de e-mails

La cantidad de pruebas que se pueden realizar son infinitas, ya que el objetivo del Generador de Eventos es poder crear un evento a partir de cualquier correo. En los puntos anteriores se han mostrado los ejemplos más representativos de la creación de un evento. Se dejan para el apéndice E.2 un conjunto de correos de prueba adicionales:

- E-mail vacío. Ejemplo con evento creado sin información, con campos por defecto.
- E-mail con plantilla en inglés. En este ejemplo se utiliza una plantilla con etiquetas en inglés. Se comprueba que la creación del evento utiliza el juego de patrones para ese idioma.
- E-mail redactado con privacidad pública y datos adjuntos. En este ejemplo se prueba la funcionalidad de especificación de privacidad del evento y la búsqueda y guardado de datos adjuntos contenidos en el correo.
- E-mail redactado ambiguo. En este caso se evalúa la creación de un evento cuando la información contenida en el correo puede ser malinterpretada, mostrándose un caso en el que los campos no se cumplimentan con lo esperado.
- E-mail reenviado de entradas a un concierto. Este caso es del mismo estilo que el correo reenviado de la compra de billetes de Renfe. Se analiza la creación de un evento a partir de un correo de compra de entradas para un concierto.
- E-mail reenviado de alquiler de películas. Este es un tercer ejemplo de correo reenviado a la aplicación. Se trata de un e-mail recibido por un usuario de la Agenda Social con la información del alquiler de una película que ha realizado. Este correo además de contener contenidos multimedia también está en inglés.

4.5. Prueba del modo NO_DB_MODE

Para probar rápidamente el funcionamiento, antes de iniciar la aplicación cambiaremos los parámetros de configuración de la conexión al servidor MySQL en vez de parar su servicio. Por ejemplo, al cambiar la url de conexión de `jdbc:mysql://localhost:3306/ge` a `jdbc:mysql://localhost:3307/ge` no se establecerá conexión y se entrará en el modo de trabajo sin base de datos. En la esquina superior izquierda del log se observará la indicación del modo NO_DB_MODE en naranja.

En este modo de funcionamiento comprobaremos que cuando se reciba un e-mail o SMS de un usuario no registrado en base de datos intentará crear su evento igualmente. En este caso si intentamos generar un evento a partir de un mensaje incompleto obtenemos lo siguiente:

Mensaje:
"T:Viaje a Canarias.F:1-7-2016.H:08:45.L:Algún lugar,Madrid."

Evento generado:
Id Event = null
Id User = -1
Title = Viaje a Canarias
Date = 01/07/2016
Time = 08:45
Category = Travel
Address = Algún lugar
City = Madrid
Zipcode = null
Country = null
Description = null
Timestamp = Tue Jul 19 19:30:01 CEST 2016
Privacy = 2

Comentario:
Observamos que en este caso no se ha encontrado el usuario y se ha continuado con la creación, otorgando por defecto el id de usuario -1. El valor nulo del id de usuario se reserva para identificar que el usuario no ha sido hallado y no debe proceder a crearse el evento. El id del evento sí pasa a ser nulo. También se observa que al no poder utilizar las funciones de búsqueda en base de datos no se han encontrado ni el país, ni el código postal asociado a "Madrid". No ocurre lo mismo con la categoría, ya que su búsqueda en el contenido del mensaje no se realiza mediante consultas a base de datos, sino aplicando un listado con palabras clave predefinidas.

Para volver a activar el modo de funcionamiento normal volveremos a indicar la url correcta y arrancaremos la aplicación de nuevo. Es importante comentar que si la conexión a la base de datos se pierde a mitad de ejecución, como resultado obtendremos en el log errores de ejecución (`Exceptions`), debidos al intento de acceso a base de datos. Estos errores solo influirán en que las consultas a base de datos devolverán resultados vacíos y los insertados en base de datos no se realizarán. No se cambiará a mitad de ejecución de un modo a otro.

4.6. Evaluación de las pruebas

Tras los resultados obtenidos de todos los correos y SMSs de prueba, podemos determinar que en gran medida se encuentran los resultados esperados para contenidos con una estructura a la que se le puede aplicar patrones basados en expresiones regulares. Para todos los demás correos habría que aplicar otros métodos diferentes.

El principal objetivo que debía cumplir el Generador de Eventos era poder analizar el mayor número de correos posibles partiendo de un ecosistema de patrones, expresiones regulares y métodos de consulta fijos.

Hemos de tener en cuenta que si adaptamos los métodos para el correcto hallazgo de información para un único correo perderemos la generalización y seguramente las modificaciones añadidas hagan que correos que sí se analizaban correctamente con anterioridad, dejen de analizarse correctamente después de añadir los cambios. Es asumible pues, que en algunas circunstancias la generación del evento no se comportará según lo esperado.

Para el hallazgo de algunos campos y bajo la premisa de intentar hallar el máximo número de campos posibles nos encontramos que algunas veces, las funciones complementarias utilizadas cuando las expresiones regulares no encuentran nada por no haber referencias, causen un incorrecto cumplimentado del campo con información errónea.

No debemos identificar la incorrecta creación de un evento con el malfuncionamiento del proceso de generación, puesto que estamos partiendo de la base que tiene que poder crear eventos a partir una población infinita de diferentes tipos de correos o menajes de texto, todos redactados de diferente forma y con estructuras diferentes.

En definitiva, después de analizar algunos mensajes de prueba, el proceso de creación de un evento es satisfactorio, cumplimentándose normalmente un gran número de campos del evento de forma correcta, adaptándose a la circunstancias de cada SMS, MMS o e-mail recibido por la aplicación.

Capítulo 5

Conclusiones

Este capítulo muestra una autoevaluación del trabajo realizado, ofreciendo un breve resumen de este proyecto final de carrera, su realización y lo extraído del desarrollo de la aplicación implementada y las pruebas realizadas.

Por último, se comenta qué mejoras podrían haberse implementado y cuáles se han quedado fuera por ir más allá del alcance de este proyecto.

5.1. Desarrollo del proyecto

Durante la realización del análisis del proyecto se contempló cómo debía enfocarse la elaboración de las diferentes partes para obtener la solución de manera óptima. Se dedicó en esa fase de desarrollo mucho tiempo en realizar el modelo conceptual para la creación de los módulos de la aplicación y su flujo de datos. El análisis y desarrollo fue realimentándose a medida que la creación de la aplicación avanzaba en el tiempo y se mostraba un incremento o variación de las funcionalidades junto con el surgir de nuevas especificaciones.

El objetivo principal y más costoso, y donde se han centrado muchos de los esfuerzos, es sin duda la adecuación del Generador de Eventos a la interpretación automática de correos. Para ello se estudiaron varios reales como muestra de posibles correos enviados por los usuarios, así como la creación de unos patrones de búsqueda que pudiesen encontrar el mayor número de campos de un evento.

El incorporar el módulo de generación de eventos a la Agenda Social planteó todo un reto, porque requería adquirir conocimientos de programación orientados a desarrollo de proyectos web, así como trabajar junto con un equipo de trabajo y adaptarse a los objetivos y funcionalidades cambiantes.

Finalmente, la separación del Generador de Eventos, y creación como aplicación independiente, ha permitido la adquisición de una amplia variedad de conocimientos de programación, incluyendo diseño de aplicaciones gráficas y bases de datos.

Junto con las conclusiones y resultados de las pruebas realizadas podemos concluir que los objetivos han sido cumplidos; teniéndose un módulo, que ha sido implementado dentro de una aplicación web y posteriormente separado en una aplicación independiente, totalmente funcional, capaz de interpretar mensajes entrantes y analizarlos paso a paso para poder extraer toda la información posible de ellos.

5.2. Mejoras y trabajos futuros

Son muchas las mejoras, nuevas funcionalidades y rediseños que puede sufrir una aplicación en el proceso de realización de los trabajos. En el mundo de la programación hay muchas formas de llegar a la solución. Puedes dar más o menos rodeos, hacer las cosas de manera más eficiente o mejorar y optimizar lo que ya se tiene.

Muchas de las mejoras post-análisis que se me ocurrieron intenté introducirlas en el proyecto, entre éstas están las ya comentadas a lo largo de esta memoria y que ya formaron parte de la Agenda Social dentro del módulo de generación de eventos:

- Aplicación multilenguaje.
- Mejora en la generación de expresiones regulares a partir de un fichero XML.
- Ampliar el abanico de expresiones regulares para encontrar la información de los campos de un evento en muchos más formatos.
- Soporte para recepción de MMS.
- Crear una tercera vía de comunicación vía TCP-IP con un emulador de SMS.

Ya en la creación de la aplicación de escritorio del Generador de Eventos se incorporaron mejoras de diseño y reestructuración de código y trabajo con bases de datos. Aunque el núcleo de la aplicación no se remodeló si se añadieron nuevas opciones de parseado para el emulador de SMS. Las principales mejoras han sido:

- Interfaz gráfica para el Generador de Eventos basada en JavaFX.
- Interfaz gráfica para el Emulador de SMS y MMS basada en JavaFX.
- Crear un método de parseado de MMS para el emulador.
- Crear una base de datos independiente y el modelo de datos del Generador de Eventos.
- Mejora de los métodos de búsqueda de los patrones en base de datos.

Las principales mejoras que se han obviado por necesidad de un retoque íntegro del núcleo de la aplicación y queda aplazado para futuros trabajos son:

- Darle una mayor robustez a las funciones de los Buscadores, que mediante el modelo de aplicación obtienen información implícita para un evento. Como se ha visto en las pruebas, la cumplimentación de campos a través de estos métodos puede comprometer la creación del evento con información no correcta. Potenciar estas funciones para descartar información incorrecta le hubiese dado al Generador de Eventos un extra para detectar posible eventos mal cumplimentados por los usuarios y no arrastrarlos a futuros eventos.
- Mejorar la búsqueda de algunos campos más complejos conceptualmente como son el título o la descripción dentro de los contenidos.
- Generación múltiple de eventos. Se valoró la posibilidad de crear varios eventos a partir de un único correo. Mediante la generación semiautomática, hubiese sido sencillo incorporar alguna marca por el usuario o plantilla para determinar cuando terminaba un evento y comenzaba el siguiente. Se descartó por no complicar las reglas de la

generación semiautomática de cara a los usuarios y escaparse del concepto original de un evento creado por contenido recibido.

Continuando en la línea de generación múltiple. De cara a eventos automáticos se planteó la idea de poder crear dos eventos diferentes a partir de un correo de categoría Viaje: un evento para la ida y otro para la vuelta. Esta opción se descartó por falta de tiempo y porque se alejaba del criterio acordado para la generación de eventos automáticamente: la generalización. Crear este tipo de multi-eventos especificaría una creación muy concreta de eventos para un determinado tipo de correos.

- Por último, quizás la característica que más hubiese revolucionado al Generador de Eventos es la de otorgarle inteligencia artificial, tener la habilidad de poder ir aprendiendo para poder encontrar campos que no hubiesen sido posibles con expresiones regulares fijas. Esto implicaba hacer un estudio para crear dinámicamente expresiones regulares adaptables y cómo adaptarlas a los contenidos; actualmente solo lo veo posible con la colaboración de los propios usuarios, al menos marcando donde no se ha encontrado un campo o cuando éste no tiene el valor esperado en la fuente original. Introducir este concepto no hubiese sido trivial, hubiese requerido de un fuerte análisis y rediseño, por no decir, partida desde cero para hacer frente a este problema.

Esta última idea me atrajo, pero en el ámbito del proyecto para generar eventos automáticos, veía fuera de lugar implementar dicha mejora. Esto se debe mayormente por los siguientes tres motivos:

- Población infinita y heterogénea de formatos de correos electrónicos y contenidos.
- Las expresiones regulares son muy susceptibles a pequeñas variaciones, siendo difícil encontrar un equilibrio de acierto para todos los contenidos.
- Crear un método de interactuación con el usuario para tener un indicador de que el evento o campo ha sido creado correctamente.

Además cabe destacar que mucha de la información que un usuario puede extraer de un correo a simple vista está muy relacionada con el contexto. Muchas de las expresiones no se podrían relacionar, y si se pudiese extraer algún método fiable para algún contenido seguramente no sería correcto para otro.

Si en un futuro se definiese uno o varios estándares (algún tipo de meta-información) para los contenidos de los correos enviados automáticamente por terceras aplicaciones (como actualmente hay definidos para muchas otras fuentes de información), con ligeras adaptaciones de los patrones, todo correo electrónico podría ser tratado de forma fiable, como un correo listo para una generación semiautomática pero generalizada. Esto es debido a que uno de los mayores inconvenientes de los mensajes, su estructura sintáctica, ya no sería un obstáculo.

5.3. Autoevaluación

Ya desde los inicios de los estudios de la carrera, encontré en la programación una de mis aficiones, aunque en un principio lo vi como algo complejo, me acabó agradando y encontré facilidad en esta materia, en su estudio y en su aplicación a la resolución de problemas. Es por ese motivo que cursé todas las asignaturas de programación y busqué realizar un proyecto final de carrera basado en ello.

Desde un principio la temática del proyecto me pareció un buen reto. Me permitía poder profundizar en el mundo de la programación y formar parte de un equipo de desarrolladores en una aplicación de ámbito profesional. Ha sido mucho el tiempo dedicado a buscar información, programar soluciones parciales, solventar problemáticas y sobre todo ampliar conocimientos y ganar experiencia.

La realización de este proyecto desde un inicio la enfoqué como una oportunidad profesional, más que como una obligación y requerimiento para finalizar los estudios. Cabe destacar que he podido aplicar varios de los conocimientos adquiridos en el mundo laboral, como el diseño de interfaces gráficas o diseño de bases de datos. A fecha del presente documento sigo programando en éste y otros lenguajes y me ha sido muy útil sobre todo la comprensión y dominio de las expresiones regulares, cosa que sigo utilizando a día de hoy.

Han sido unos años duros con altibajos en la elaboración de la memoria y la puesta final de la aplicación para una presentación. Aunque la elaboración del trabajo se hizo en los meses fijados, en la elaboración de la memoria me he demorado demasiado tiempo. Los motivos son varios pero el más destacado ha sido el de la entrada en el mundo laboral. Se ha hecho muy difícil el compaginar el trabajo y los quehaceres cuotidianos con la elaboración del documento y desarrollo de mejoras para la aplicación.

Me quedo sobre todo con la idea de que un trabajo no se puede estar mejorando eternamente, cada vez que he realizado un parón y he retomado este trabajo he dedicado más tiempo en cambiar cosas que ya estaban hechas para intentar mejorarlas que en crear nuevas y cerrar las partes ya realizadas.

Después de todo el tiempo empleado en la realización de un trabajo de la envergadura de un proyecto final de carrera se le queda a uno un buen sabor de boca. Uno mira atrás y recuerda todo el esfuerzo dedicado y al final del camino se obtiene un producto acabado, con un funcionamiento correcto que cumple los objetivos marcados.

Bibliografía

- ***Libros.***

[A] Core Java 2
Cay S. Horstmann y Gary Cornell. Publicado por Prentice Hall PTR.

[B] Java Network Programming, 3rd Edition
Elliotte Rusty Harold. Publicado por O'Reilly.

[C] Java Design Pattern Essentials
Tony Bevis. Publicado por AbilityFIRST

[D] Java Regular Expressions: Taming the java.util.regex Engine
Mehran Habibi. Publicado por Apress.

[E] Java Web Services
David Chappell, Tyler Jewell. Publicado por O'Reilly.

[F] Pro JavaFX 8
Johan Vos, Stephen Chin, Dean Iverson, Weiqi Gao, James L. Weaver. Publicado por Apress.

- Enlaces web.

Los enlaces son vigentes a fecha de entrega del presente documento, septiembre de 2016.

[1] Web 2.0
http://www.oreilly.com/pub/a//web2/archive/what-is-web-20.html

[2] Comparativa de la web 1.0, 1.5 y 2.0
http://e-global.es/b2b-blog/2005/11/23/caracteristicas-principales-de-web-1_0-web-1_5-y-web-2_0/

[3] La Web 3.0, añade significado
http://www.maestrosdelweb.com/la-web-30-anade-significado/

[4] Plataformas Java
http://www.oracle.com/technetwork/java/javase/overview/index.html

[5] Tutoriales Java
http://docs.oracle.com/javase/tutorial/

[6] API Java
https://docs.oracle.com/javase/8/docs/api/

[7] Google Mail
http://mail.google.com

[8] Tipos MIME
http://www.htmlquick.com/es/reference/mime-types.html

[9] Parlay X
https://docs.oracle.com/cd/E13205_01/wcp/wng10/devparlayx/parlayXfor3GSM_api_general.html

[10] NetBeans IDE
https://netbeans.org/index_es.html

[11] Glassfish
http://glassfish.java.net/

[12] MySQL
http://www.mysql.com/

[13] API Persistencia Java
http://www.oracle.com/technetwork/articles/javaee/jpa-137156.html

[14] Eclipse
https://eclipse.org/luna/

[15] e(fx)clipse
http://www.eclipse.org/efxclipse/index.html

[16] Java Scene Builder
http://www.oracle.com/technetwork/java/javase/downloads/sb2download-2177776.html

[17] BoneCP
http://www.jolbox.com/

[18] Expresiones Regulares
http://download.oracle.com/javase/tutorial/essential/regex/

[19] Webservices
https://www.w3.org/TR/wsdl

[20] Apache Axis
http://axis.apache.org/axis/

[21] API mail
http://www.oracle.com/technetwork/java/javamail/index.html

[22] JavaFX
http://docs.oracle.com/javase/8/javase-clienttechnologies.htm

APÉNDICES

A. Entorno de desarrollo

En este anexo se detalla la instalación del conjunto de aplicaciones necesarias para utilizar el entorno de desarrollo y poder ejecutar y testear el Generador de Eventos.

A.1. Entorno de desarrollo del Generador de Eventos

El Generador de Eventos ha requerido el siguiente software:

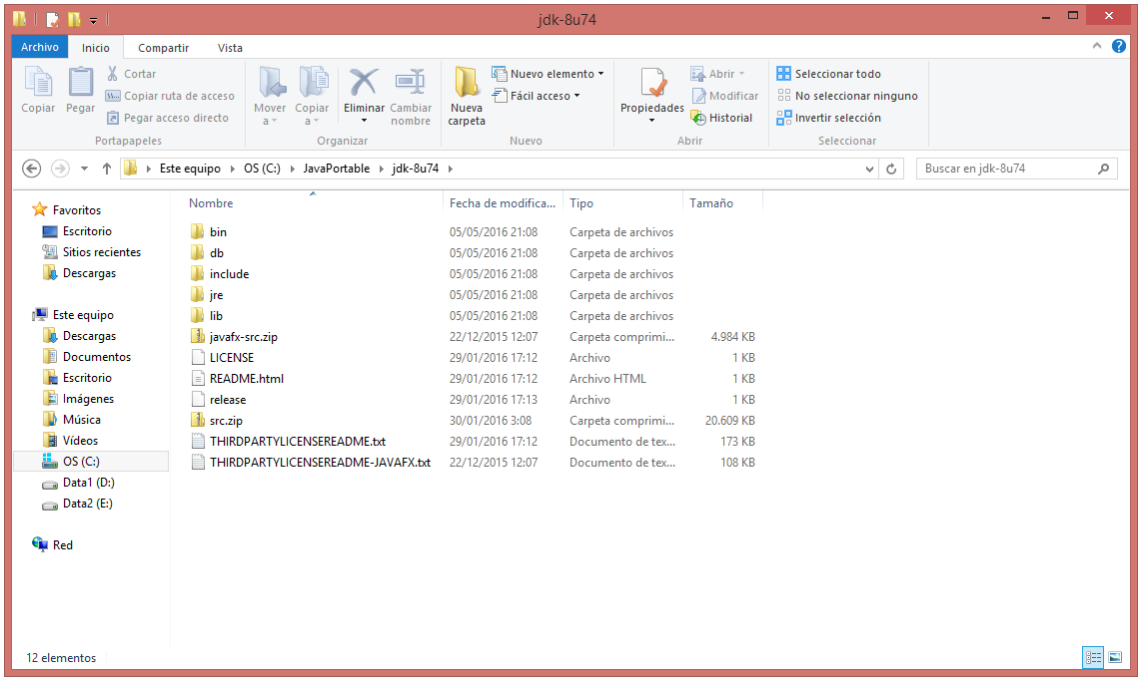
- Java SE Development Kit 8 Update 74.

En vez de utilizar una versión instalada del JDK se ha utilizado una versión portable. Para crear la versión portable (si no se desea instalar java en la máquina) se debe descargar de la página de Java de Oracle la versión comprimida en zip en vez del instalador. Descomprimir el archivo y con la línea de comandos en el mismo directorio de los ficheros ejecutar:

```
for /r %x in (*.pack) do C:\jdk-1.7u45\bin\unpack200 -r "%x" "%~dpnx.jar"
```

Tendremos un duplicado de la carpeta de instalación que crea Java, como se puede ver en la Figura A.1. Al no ser una versión instalada no se han modificado los registros del sistema operativo y no se han creado las variables *path*[51] en el sistema.

Figura A.1: Creación de una versión portable de Java



[51] Variable del sistema operativo que guarda la direcciones de los directorios de algunas aplicaciones, pudiéndose acceder a ellas en línea de comandos sin escribir la ruta completa.

- IDE Eclipse versión Luna SR1 (4.4.1).

El entorno de desarrollo Eclipse no necesita ser instalado. Se ha utilizado una versión limpia portable y las utilidades e(fx)clipse para desarrollar en JavaFX. Para iniciar el IDE con el JDK correspondiente en la carpeta de la aplicación indicamos en el fichero eclipse.ini la ruta de la máquina virtual a utilizar:

```
-startup
plugins/org.eclipse.equinox.launcher_1.3.0.v20140415-2008.jar
--launcher.library
plugins/org.eclipse.equinox.launcher.win32.win32.x86_1.1.200.v20140603-1326
-showsplash
org.eclipse.platform
--launcher.XXMaxPermSize
256m
--launcher.defaultAction
openFile
--launcher.appendVmargs
-vm
C:/JavaPortable/jdk-8u74/bin/javaw.exe
-vmargs
-Xms40m
-Xmx1024m
```

- JavaFX Scene Builder 2.0.

Para un rápido desarrollo de la interfaz gráfica del Generador de Eventos y el Emulador de envíos de SMS se ha utilizado la aplicación JavaFX Scene Builder. Para poder abrir directamente los archivos FXML se necesita especificar la ruta de la aplicación en Preferences / JavaFX del Eclipse. También se ha optado por una versión portable para la instalación de la aplicación.

- Servidor MySQL y Workbench.

Es importante que durante la instalación se anote el usuario y contraseña que se le da al usuario root (normalmente root / root). Para la creación del esquema de base de datos (ge) se pude utilizar el fichero anexado en el directorio "_importSQL/ge.sql". Seleccionando el fichero en la sección de *imports* añadirá en base de datos el esquema con todas las tablas que necesita el generador de eventos con un mínimo de datos para pruebas.

Para trabajar cómodamente con los datos, se puede utilizar el Workbench de MySQL o utilizar otros programas para la visualización y edición de los campos de la base de datos. Por ejemplo adicionalmente se ha utilizado el programa Ms Access creando una vinculación de datos externos, para ello es necesario crear una conexión ODBC[52] con los parámetros de acceso a la base de datos:

- Direccionamiento: ip (localhost si está en el mismo equipo) y puerto (3306).
- Definición de usuario y password de conexión a base de datos.
- Esquema a visualizar, en este caso "ge".

[52] Open DataBase Connectivity: Estándar de conexión a bases de datos.

A.2. Configuración de la cuenta de correo de Gmail

El Generador de Eventos necesita una cuenta de correo de Gmail para poder crear eventos a partir de correos electrónicos. El único requisito es configurar dentro del proyecto la cuenta de usuario y contraseña para el acceso remoto desde la aplicación. No es necesario configurar nada más. Las carpetas que se utilizan para la gestión de correos se crean desde la lógica de acceso y procesado de correos dentro del Generador de Eventos.

Es posible que la cuenta de Gmail pueda ser bloqueada para su acceso con la API de mails de Java. Al intentar conectar nos dará un error de este estilo:

```
[ALERT] Please log in via your web browser: https://support.google.com/mail/a
ccounts/answer/78754 (Failure)
```

En el enlace proporcionado nos redirige en este punto a otro enlace para permitir el acceso de aplicaciones menos seguras:

Es posible que tu aplicación no sea compatible con los últimos estándares de seguridad.
Prueba a cambiar algunos ajustes para permitir un acceso menos seguro a tu cuenta.

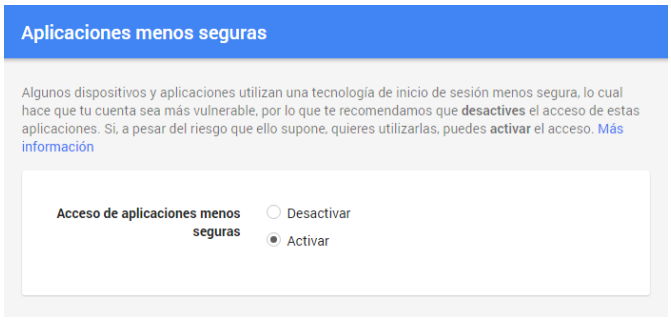
A través del enlace llegamos a la selección de opciones donde elegiremos la segunda:

Opción 2: Cambia la configuración para permitir que las aplicaciones menos seguras accedan a tu cuenta. Esta opción no es recomendable porque podría facilitar el acceso a tu cuenta a otra persona. Si quieres permitirlo de todas formas, sigue estos pasos:

1. En "Mi cuenta", ve a la sección Aplicaciones menos seguras.
2. Junto a "Acceso de aplicaciones menos seguras", selecciona **Activar**. **Nota para usuarios de Google Apps:** Esta opción está oculta si tu administrador ha bloqueado el acceso de las aplicaciones menos seguras a la cuenta.

Este es el acceso directo al cambio de configuración con la sesión de Gmail abierta: https://www.google.com/settings/security/lesssecureapps

Figura A.2: Permiso de Gmail para el acceso de aplicaciones menos seguras



B. Expresiones regulares

B.1. Resumen de operadores

Tabla B.1: Caracteres delimitadores de expresiones regulares básicas

Carácter	Descripción
.	Cualquier carácter individual
$	Fin de entrada o línea
^	Principio de entrada o línea
{	Define inicio de un cuantificador
[	Define inicio de una clase de caracteres
(	Define inicio de un grupo
|	Símbolo de OR
}	Define fin de un cuantificador
]	Define fin de una clase de caracteres
)	Define fin de un grupo
*****	Lo anterior se repite 0 o más veces
+	Lo anterior se repite 1 o más veces
?	Lo anterior se repite 0 o 1 vez
\	Lo siguiente no se considera un meta-carácter

Tabla B.2: Cuantificadores

Regex	Descripción
?	Lo anterior se repite 0 o 1 vez
*****	Lo anterior se repite 0 o más veces
+	Lo anterior se repite 1 o más veces
{n}	Lo anterior se repite exactamente n veces
{n,}	Lo anterior se repite como mínimo n veces
{n,m}	Lo anterior se repite como mínimo n y como máximo m veces
|	Símbolo de OR

Tabla B.3: Clases de Caracteres

Patrón	Descripción
[abc]	*a*, *b*, o *c*. (Puede ser cualquier carácter, no solo *a*, *b*, o *c*.)
[^abc]	Cualquier carácter excepto *a*, *b*, o *c*.
[a-zA-Z]	De la *a* a la *z* o de la *A* a la *Z*.
[a-d[m-p]]	De la *a* a la *d* o de la *m* a la *p*: ***[a-dm-p]***.
[a-z&&[def]]	Cualquier cosa que exista en ambos conjuntos, en resumen *d*, *e*, o *f*.
[a-z&&[^bc]]	De la *a* a la *z*, excepto *b* y *c*: ***[ad-z]***.
[a-z&&[^m-p]]	De la *a* a la *z*, pero no de la *m* a la *p*: ***[a-lq-z]***.

Tabla B.4: Caracteres Comunes y Frontera

Carácter	Descripción
.	Coincide con cualquier carácter, incluso terminadores de línea.
\d	Un dígito ***[0-9]***. Encuentra cualquier dígito del *0* al *9*. Para la entrada *19* se necesita que coincida dos veces: una para el *1* y otra para el *9*.
\D	Un no-dígito ***[^0-9]***. Coincide con cualquier cosa que no sea un dígito, incluido los caracteres de tipo espacio en blanco.
\w	Un carácter alfanumérico ***[a-zA-Z_0-9]***. Coincide con cualquier carácter de la *a* a la *z* o de la *A* a la *Z*, guión bajo, o cualquier dígito del *0* al *9*.
\W	Un carácter no-alfanumérico ***[^\w]***.
\t	El carácter tabulador.
\n	Representa la "nueva línea". En Windows se necesita \r\n, en Unix \n y en Mac_OS \r para representar los saltos de línea.
\r	Representa el "retorno de carro" o "regreso al inicio".
\f	Representa un salto de página.
\s	Un espacio en blanco. Incluye el salto de línea, retorno de carro, tabulador, salto de página y final de línea.
\S	Un no-espacio en blanco ***[^\s]***. Cualquiera cosa no descrita con \s
^	El comienzo de una línea.
$	El final de una línea.
\b	Marca el principio y final de una palabra.
\B	Marca la posición entre dos caracteres alfanuméricos o dos no-alfanuméricos.

Tabla B.5: Clases POSIX de caracteres

Patrón	Descripción
\p{Lower}	Cualquier letra en minúsculas: ***[a-z]***
\p{Upper}	Cualquier letra en mayúsculas: ***[A-Z]***
\p{ASCII}	Cualquier símbolo ASCII: ***[\x00-\x7F]***
\p{Alpha}	Cualquier letra en minúsculas y mayúsculas: ***[\p{Lower}\p{Upper}]***
\p{Digit}	Cualquier dígito: ***[0-9]***
\p{Alnum}	Cualquier letra o dígito: ***[\p{Alpha}\p{Digit}]***
\p{Punct}	Puntuación: cualquiera de: ***!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~***
\p{Graph}	Cualquier carácter visible: ***[\p{Alnum}\p{Punct}]***
\p{Print}	Cualquier carácter imprimible: ***[\p{Graph}]***
\p{Blank}	Un tabulador o un espacio
\p{Cntrl}	Un carácter de control: ***[\x00-\x1F\x7F]***
\p{XDigit}	Un dígito hexadecimal: ***[0-9a-fA-F]***
\p{Space}	Un carácter de espacio en blanco: ***[\t\n\x0B\f\r]***

Tabla B.6: Grupos y referencias a grupos

Regex	Descripción
(	Un grupo que consiste en …
…	Cualquier expresión regular
)	Cierre de grupo
\1	Primer grupo del patrón
\2	Segundo grupo del patrón
\n	n-ésimo grupo del patrón

B.2. Fichero de gramáticas SMS

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!--
        <   &lt;
        >   &gt;
        &   &amp;
        "   &quot;
        '   &apos;

NOTAS:
        - Los patrones tienen que tener un único grupo de resultado
        !Este documento tiene que coincidir con los Strings de la clase Items

        - Portable Operating System Interface for UNIX (POSIX) character classes
        \p{Punct} == !"#$%&'()*+,-./:;<=>?@[\]^_`{|}~

Los dos puntos son obligatorios aunque se pueden incorporar otros separadores
Si se añaden nuevos identificadores y campos corregir el fichero de propiedades
dicConfig.properties en eventParser.dictionary -->

<Grammar defaultLanguage="ESP">
    <ENG>
    <Title>
        <Pattern>S[ ]?[:][ ]?([\w [\p{Punct}&amp;&amp;[^\.:]]]*)[\.]</Pattern>
        <Pattern>SUB\w*[ ]?[:][ ]?([\w [\p{Punct}&amp;&amp;[^\.:]]]*)[\.]</Pattern>
        <Pattern>TIT\w*[ ]?[:][ ]?([\w [\p{Punct}&amp;&amp;[^\.:]]]*)[\.]</Pattern>
    </Title>
    <Date>
        <Pattern>D[ ]?[:][ ]?(\d{1,2}[-\\ /_]\d{1,2}[-\\ /_]\d{4})[ ]?[\.]?</Pattern>
        <Pattern>D[ ]?[:][ ]?(\d{1,2}[-\\ /_]\d{1,2}[-\\ /_]\d{2})[ ]?[\.]?</Pattern>
        <Pattern>D[ ]?[:][ ]?(\d{1,2}[-\\ /_]\d{1,2})[ ]?[\.]?</Pattern>
        <Pattern>DAT\w*[ ]?[:][ ]?(\d{1,2}[-\\ /_]\d{1,2}[-\\ /_]\d{4})[ ]?[\.]?</Pattern>
        <Pattern>DAT\w*[ ]?[:][ ]?(\d{1,2}[-\\ /_]\d{1,2}[-\\ /_]\d{2})[ ]?[\.]?</Pattern>
        <Pattern>DAT\w*[ ]?[:][ ]?(\d{1,2}[-\\ /_]\d{1,2})[ ]?[\.]?</Pattern>
        <Pattern>(\d{1,2}[-\\/]\d{1,2}[-\\/]\d{4})</Pattern>
        <Pattern>(\d{1,2}[-\\/]\d{1,2}[-\\/]\d{2})</Pattern>
        <Pattern>(\d{1,2}[-\\/]\d{1,2})</Pattern>
    </Date>
    <Time>
        <Pattern>T[ ]?[:][ ]?(\d{1,2}[:]\d\d)[ ]?[\.]?</Pattern>
        <Pattern>TIM\w*[ ]?[:][ ]?(\d{1,2}[:]\d\d)[ ]?[\.]?</Pattern>
        <Pattern>(\d{1,2}[:]\d\d)</Pattern>
    </Time>
    <Address>
        <Pattern>L[ ]?[:][ ]?([\w [\p{Punct}&amp;&amp;[^\.:]][ºª]]*),[\w ]*[\.]</Pattern>
        <Pattern>LOC\w*[ ]?[:][ ]?([\w [\p{Punct}&amp;&amp;[^\.:]][ºª]]*),[\w ]*[\.]</Pattern>
        <Pattern>STR\w*[ ]?[:][ ]?([\w [\p{Punct}&amp;&amp;[^\.:]][ºª]]*)[\.]</Pattern>
        <Pattern>ADD\w*[ ]?[:][ ]?([\w [\p{Punct}&amp;&amp;[^\.:]][ºª]]*)[\.]</Pattern>
    </Address>
    <City>
        <Pattern>L[ ]?[:][ ]?[\w [\p{Punct}&amp;&amp;[^\.:]][ºª]]*,([\w ]*)[\.]</Pattern>
        <Pattern>LOC\w*[ ]?[:][ ]?[\w [\p{Punct}&amp;&amp;[^\.:]]]*,([\w ]*)[\.]</Pattern>
        <Pattern>CIT\w*[ ]?[:][ ]?([\w ]*)[\.]</Pattern>
    </City>
    <ZipCode>
        <Pattern>Z[ ]?[:][ ]?([\w-]*)[\.]</Pattern>
        <Pattern>ZIP\w*[ ]?[:][ ]?([\w-]*)[\.]</Pattern>
        <Pattern>COD\w*[ ]?[:][ ]?([\w-]*)[\.]</Pattern>
    </ZipCode>
    <Country>
        <Pattern>C[ ]?[:][ ]?([\w ]*)[\.]</Pattern>
        <Pattern>COU\w*[ ]?[:][ ]?([\w ]*)[\.]</Pattern>
    </Country>
    <Description>
        <Pattern>X[ ]?[:][ ]?([\w [\p{Punct}&amp;&amp;[^\.:]][ºª]]*)[\.]</Pattern>
        <Pattern>DES\w*[ ]?[:][ ]?([\w [\p{Punct}&amp;&amp;[^\.:]][ºª]]*)[\.]</Pattern>
    </Description>
    <Category>
        <Pattern>N[ ]?[:][ ]?([\w ]*)[\.]</Pattern>
        <Pattern>NAT\w*[ ]?[:][ ]?([\w ]*)[\.]</Pattern>
        <Pattern>CAT\w*[ ]?[:][ ]?([\w ]*)[\.]</Pattern>
    </Category>
    <Privacy>
        <Pattern>P[ ]?[:][ ]?([\w]*)[\.]</Pattern>
        <Pattern>PRI\w*[ ]?[:][ ]?([\w]*)[\.]</Pattern>
    </Privacy>
    </ENG>

    <ESP>
    <Title>
        <Pattern>T[ ]?[:][ ]?([\w [\p{Punct}&amp;&amp;[^\.:]]]*)[\.]</Pattern>
        <Pattern>TEM\w*[ ]?[:][ ]?([\w [\p{Punct}&amp;&amp;[^\.:]]]*)[\.]</Pattern>
        <Pattern>TIT\w*[ ]?[:][ ]?([\w [\p{Punct}&amp;&amp;[^\.:]]]*)[\.]</Pattern>
    </Title>
    <Date>
        <Pattern>F[ ]?[:][ ]?(\d{1,2}[-\\ /_]\d{1,2}[-\\ /_]\d{4})[ ]?[\.]?</Pattern>
        <Pattern>F[ ]?[:][ ]?(\d{1,2}[-\\ /_]\d{1,2}[-\\ /_]\d{2})[ ]?[\.]?</Pattern>
        <Pattern>F[ ]?[:][ ]?(\d{1,2}[-\\ /_]\d{1,2})[ ]?[\.]?</Pattern>
        <Pattern>FEC\w*[ ]?[:][ ]?(\d{1,2}[-\\ /_]\d{1,2}[-\\ /_]\d{4})[ ]?[\.]?</Pattern>
        <Pattern>FEC\w*[ ]?[:][ ]?(\d{1,2}[-\\ /_]\d{1,2}[-\\ /_]\d{2})[ ]?[\.]?</Pattern>
        <Pattern>FEC\w*[ ]?[:][ ]?(\d{1,2}[-\\ /_]\d{1,2})[ ]?[\.]?</Pattern>
        <Pattern>(\d{1,2}[-\\/]\d{1,2}[-\\/]\d{4})</Pattern>
        <Pattern>(\d{1,2}[-\\/]\d{1,2}[-\\/]\d{2})</Pattern>
        <Pattern>(\d{1,2}[-\\/]\d{1,2})</Pattern>
    </Date>
```

```
    <Time>
        <Pattern>H[ ]?[:][ ]?(\d{1,2}[:]\d\d)[ ]?[\.]?</Pattern>
        <Pattern>HOR\w*[ ]?[:][ ]?(\d{1,2}[:]\d\d)[ ]?[\.]?</Pattern>
        <Pattern>(\d{1,2}[:]\d\d)</Pattern>
    </Time>
    <Address>
        <Pattern>L[ ]?[:][ ]?([\w [\p{Punct}&amp;&amp;[^\.:]][ºª]]*),[\w ]*[\.]</Pattern>
        <Pattern>LUG\w*[ ]?[:][ ]?([\w [\p{Punct}&amp;&amp;[^\.:]][ºª]]*),[\w ]*[\.]</Pattern>
        <Pattern>CALL\w*[ ]?[:][ ]?([\w [\p{Punct}&amp;&amp;[^\.:]][ºª]]*)[\.]</Pattern>
        <Pattern>DIR\w*[ ]?[:][ ]?([\w [\p{Punct}&amp;&amp;[^\.:]][ºª]]*)[\.]</Pattern>
    </Address>
    <City>
        <Pattern>L[ ]?[:][ ]?[\w [\p{Punct}&amp;&amp;[^\.:]][ºª]]*,([\w ]*)[\.]</Pattern>
        <Pattern>LUG\w*[ ]?[:][ ]?[\w [\p{Punct}&amp;&amp;[^\.:]]]*,([\w ]*)[\.]</Pattern>
        <Pattern>CIU\w*[ ]?[:][ ]?([\w ]*)[\.]</Pattern>
    </City>
    <ZipCode>
        <Pattern>C[ ]?[:][ ]?([\w-]*)[\.]</Pattern>
        <Pattern>COD\w*[ ]?[:][ ]?([\w-]*)[\.]</Pattern>
    </ZipCode>
    <Country>
        <Pattern>P[ ]?[:][ ]?([\w ]*)[\.]</Pattern>
        <Pattern>PAI\w*[ ]?[:][ ]?([\w ]*)[\.]</Pattern>
    </Country>
    <Description>
        <Pattern>D[ ]?[:][ ]?([\w [\p{Punct}&amp;&amp;[^\.:]][ºª]]*)[\.]</Pattern>
        <Pattern>DES\w*[ ]?[:][ ]?([\w [\p{Punct}&amp;&amp;[^\.:]][ºª]]*)[\.]</Pattern>
    </Description>
    <Category>
        <Pattern>N[ ]?[:][ ]?([\w ]*)[\.]</Pattern>
        <Pattern>CAT\w*[ ]?[:][ ]?([\w ]*)[\.]</Pattern>
    </Category>
    <Privacy>
        <Pattern>A[ ]?[:][ ]?([\w]*)[\.]</Pattern>
        <Pattern>ACC\w*[ ]?[:][ ]?([\w]*)[\.]</Pattern>
        <Pattern>R[ ]?[:][ ]?([\w]*)[\.]</Pattern>
        <Pattern>REG\w*[ ]?[:][ ]?([\w]*)[\.]</Pattern>
        <Pattern>PRI\w*[ ]?[:][ ]?([\w]*)[\.]</Pattern>
    </Privacy>
    </ESP>
</Grammar>
```

B.3. Fichero de gramáticas e-mail

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<Grammar defaultLanguage="ESP">
    <ENG>
        <Title>
            <Keyword id="#K1#">Title,Subject,Event,Show</Keyword>
            <Pattern>#K1#[ ]?#?[:=][ ]?([\w[\p{Punct}&amp;&amp;[^\.:#=,;]]][\w
[\p{Punct}&amp;&amp;[^\.:#=,;]]]*)[\.,#;[\s&amp;&amp;[^ ]]]</Pattern>
        </Title>
        <Date>
            <Keyword id="#K1#">Rent Date, Return Date</Keyword>
            <Keyword id="#K2#">Date</Keyword>
            <Keyword id="#K01#">Jan,January</Keyword>
            <Keyword id="#K02#">Feb,February</Keyword>
            <Keyword id="#K03#">Mar,March</Keyword>
            <Keyword id="#K04#">Apr,April</Keyword>
            <Keyword id="#K05#">May,May</Keyword>
            <Keyword id="#K06#">Jun,June</Keyword>
            <Keyword id="#K07#">Jul,July</Keyword>
            <Keyword id="#K08#">Auo,August</Keyword>
            <Keyword id="#K09#">Sep,September</Keyword>
            <Keyword id="#K10#">Oct,October</Keyword>
            <Keyword id="#K11#">Nov,November</Keyword>
            <Keyword id="#K12#">Dec,December</Keyword>

            <!-- patrones con palabra clave 1 -->
            <Pattern>#K1#[ ]?#?[:=][ ]?(20\d\d[-\\ /_]0?\d[-\\ /_][012]?\d)\D</Pattern>
            <Pattern>#K1#[ ]?#?[:=][ ]?(20\d\d[-\\ /_]1[012][-\\ /_][012]?\d)\D</Pattern>
            <Pattern result="#G2#-01-#G1#">#K1#[ ]?#?[:=][ ]?(20\d\d)[-\\ /_]?#K01#[-\\ /_]?([012]?\d)\D</Pattern>
            <Pattern result="#G2#-02-#G1#">#K1#[ ]?#?[:=][ ]?(20\d\d)[-\\ /_]?#K02#[-\\ /_]?([012]?\d)\D</Pattern>
            <Pattern result="#G2#-03-#G1#">#K1#[ ]?#?[:=][ ]?(20\d\d)[-\\ /_]?#K03#[-\\ /_]?([012]?\d)\D</Pattern>
            <Pattern result="#G2#-04-#G1#">#K1#[ ]?#?[:=][ ]?(20\d\d)[-\\ /_]?#K04#[-\\ /_]?([012]?\d)\D</Pattern>
            <Pattern result="#G2#-05-#G1#">#K1#[ ]?#?[:=][ ]?(20\d\d)[-\\ /_]?#K05#[-\\ /_]?([012]?\d)\D</Pattern>
            <Pattern result="#G2#-06-#G1#">#K1#[ ]?#?[:=][ ]?(20\d\d)[-\\ /_]?#K06#[-\\ /_]?([012]?\d)\D</Pattern>
            <Pattern result="#G2#-07-#G1#">#K1#[ ]?#?[:=][ ]?(20\d\d)[-\\ /_]?#K07#[-\\ /_]?([012]?\d)\D</Pattern>
            <Pattern result="#G2#-08-#G1#">#K1#[ ]?#?[:=][ ]?(20\d\d)[-\\ /_]?#K08#[-\\ /_]?([012]?\d)\D</Pattern>
            <Pattern result="#G2#-09-#G1#">#K1#[ ]?#?[:=][ ]?(20\d\d)[-\\ /_]?#K09#[-\\ /_]?([012]?\d)\D</Pattern>
            <Pattern result="#G2#-10-#G1#">#K1#[ ]?#?[:=][ ]?(20\d\d)[-\\ /_]?#K10#[-\\ /_]?([012]?\d)\D</Pattern>
            <Pattern result="#G2#-11-#G1#">#K1#[ ]?#?[:=][ ]?(20\d\d)[-\\ /_]?#K11#[-\\ /_]?([012]?\d)\D</Pattern>
            <Pattern result="#G2#-12-#G1#">#K1#[ ]?#?[:=][ ]?(20\d\d)[-\\ /_]?#K12#[-\\ /_]?([012]?\d)\D</Pattern>
            <Pattern>#K1#[ ]?#?[:=][ ]?(20\d\d[-\\ /_]0?\d[-\\ /_]3[01])\D</Pattern>
            <Pattern>#K1#[ ]?#?[:=][ ]?(20\d\d[-\\ /_]1[012][-\\ /_]3[01])\D</Pattern>
            <Pattern result="#G2#-01-#G1#">#K1#[ ]?#?[:=][ ]?(20\d\d)[-\\ /_]?#K01#[-\\ /_]?(3[01])\D</Pattern>
            <Pattern result="#G2#-02-#G1#">#K1#[ ]?#?[:=][ ]?(20\d\d)[-\\ /_]?#K02#[-\\ /_]?(3[01])\D</Pattern>
            <Pattern result="#G2#-03-#G1#">#K1#[ ]?#?[:=][ ]?(20\d\d)[-\\ /_]?#K03#[-\\ /_]?(3[01])\D</Pattern>
            <Pattern result="#G2#-04-#G1#">#K1#[ ]?#?[:=][ ]?(20\d\d)[-\\ /_]?#K04#[-\\ /_]?(3[01])\D</Pattern>
            <Pattern result="#G2#-05-#G1#">#K1#[ ]?#?[:=][ ]?(20\d\d)[-\\ /_]?#K05#[-\\ /_]?(3[01])\D</Pattern>
            <Pattern result="#G2#-06-#G1#">#K1#[ ]?#?[:=][ ]?(20\d\d)[-\\ /_]?#K06#[-\\ /_]?(3[01])\D</Pattern>
            <Pattern result="#G2#-07-#G1#">#K1#[ ]?#?[:=][ ]?(20\d\d)[-\\ /_]?#K07#[-\\ /_]?(3[01])\D</Pattern>
            <Pattern result="#G2#-08-#G1#">#K1#[ ]?#?[:=][ ]?(20\d\d)[-\\ /_]?#K08#[-\\ /_]?(3[01])\D</Pattern>
            <Pattern result="#G2#-09-#G1#">#K1#[ ]?#?[:=][ ]?(20\d\d)[-\\ /_]?#K09#[-\\ /_]?(3[01])\D</Pattern>
            <Pattern result="#G2#-10-#G1#">#K1#[ ]?#?[:=][ ]?(20\d\d)[-\\ /_]?#K10#[-\\ /_]?(3[01])\D</Pattern>
            <Pattern result="#G2#-11-#G1#">#K1#[ ]?#?[:=][ ]?(20\d\d)[-\\ /_]?#K11#[-\\ /_]?(3[01])\D</Pattern>
            <Pattern result="#G2#-12-#G1#">#K1#[ ]?#?[:=][ ]?(20\d\d)[-\\ /_]?#K12#[-\\ /_]?(3[01])\D</Pattern>

            <Pattern>#K1#[ ]?#?[:=][ ]?([012]?\d[-\\ /_]0?\d[-\\ /_]20\d\d)\D</Pattern>
            <Pattern>#K1#[ ]?#?[:=][ ]?([012]?\d[-\\ /_]1[012][-\\ /_]20\d\d)\D</Pattern>
            <Pattern result="#G1#-01-#G2#">#K1#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K01#[ ]?,?[-\\ /_]?(20\d\d)\D</Pattern>
            <Pattern result="#G1#-02-#G2#">#K1#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K02#[ ]?,?[-\\ /_]?(20\d\d)\D</Pattern>
            <Pattern result="#G1#-03-#G2#">#K1#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K03#[ ]?,?[-\\ /_]?(20\d\d)\D</Pattern>
            <Pattern result="#G1#-04-#G2#">#K1#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K04#[ ]?,?[-\\ /_]?(20\d\d)\D</Pattern>
            <Pattern result="#G1#-05-#G2#">#K1#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K05#[ ]?,?[-\\ /_]?(20\d\d)\D</Pattern>
            <Pattern result="#G1#-06-#G2#">#K1#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K06#[ ]?,?[-\\ /_]?(20\d\d)\D</Pattern>
            <Pattern result="#G1#-07-#G2#">#K1#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K07#[ ]?,?[-\\ /_]?(20\d\d)\D</Pattern>
            <Pattern result="#G1#-08-#G2#">#K1#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K08#[ ]?,?[-\\ /_]?(20\d\d)\D</Pattern>
            <Pattern result="#G1#-09-#G2#">#K1#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K09#[ ]?,?[-\\ /_]?(20\d\d)\D</Pattern>
            <Pattern result="#G1#-10-#G2#">#K1#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K10#[ ]?,?[-\\ /_]?(20\d\d)\D</Pattern>
            <Pattern result="#G1#-11-#G2#">#K1#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K11#[ ]?,?[-\\ /_]?(20\d\d)\D</Pattern>
            <Pattern result="#G1#-12-#G2#">#K1#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K12#[ ]?,?[-\\ /_]?(20\d\d)\D</Pattern>
            <Pattern>#K1#[ ]?#?[:=][ ]?(3[01][-\\ /_]0?\d[-\\ /_]20\d\d)\D</Pattern>
            <Pattern>#K1#[ ]?#?[:=][ ]?(3[01][-\\ /_]1[012][-\\ /_]20\d\d)\D</Pattern>
            <Pattern result="#G1#-01-#G2#">#K1#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K01#[-\\ /_]?(20\d\d)\D</Pattern>
            <Pattern result="#G1#-02-#G2#">#K1#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K02#[-\\ /_]?(20\d\d)\D</Pattern>
            <Pattern result="#G1#-03-#G2#">#K1#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K03#[-\\ /_]?(20\d\d)\D</Pattern>
            <Pattern result="#G1#-04-#G2#">#K1#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K04#[-\\ /_]?(20\d\d)\D</Pattern>
            <Pattern result="#G1#-05-#G2#">#K1#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K05#[-\\ /_]?(20\d\d)\D</Pattern>
            <Pattern result="#G1#-06-#G2#">#K1#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K06#[-\\ /_]?(20\d\d)\D</Pattern>
            <Pattern result="#G1#-07-#G2#">#K1#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K07#[-\\ /_]?(20\d\d)\D</Pattern>
            <Pattern result="#G1#-08-#G2#">#K1#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K08#[-\\ /_]?(20\d\d)\D</Pattern>
            <Pattern result="#G1#-09-#G2#">#K1#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K09#[-\\ /_]?(20\d\d)\D</Pattern>
            <Pattern result="#G1#-10-#G2#">#K1#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K10#[-\\ /_]?(20\d\d)\D</Pattern>
            <Pattern result="#G1#-11-#G2#">#K1#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K11#[-\\ /_]?(20\d\d)\D</Pattern>
            <Pattern result="#G1#-12-#G2#">#K1#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K12#[-\\ /_]?(20\d\d)\D</Pattern>

            <Pattern>#K1#[ ]?#?[:=][ ]?([012]?\d[-\\ /_]0?\d[-\\ /_]\d{2})\D</Pattern>
            <Pattern>#K1#[ ]?#?[:=][ ]?([012]?\d[-\\ /_]1[012][-\\ /_]\d{2})\D</Pattern>
            <Pattern result="#G1#-01-#G2#">#K1#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K01#[-\\ /_]?(\d{2})\D</Pattern>
            <Pattern result="#G1#-02-#G2#">#K1#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K02#[-\\ /_]?(\d{2})\D</Pattern>
            <Pattern result="#G1#-03-#G2#">#K1#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K03#[-\\ /_]?(\d{2})\D</Pattern>
            <Pattern result="#G1#-04-#G2#">#K1#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K04#[-\\ /_]?(\d{2})\D</Pattern>
            <Pattern result="#G1#-05-#G2#">#K1#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K05#[-\\ /_]?(\d{2})\D</Pattern>
            <Pattern result="#G1#-06-#G2#">#K1#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K06#[-\\ /_]?(\d{2})\D</Pattern>
            <Pattern result="#G1#-07-#G2#">#K1#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K07#[-\\ /_]?(\d{2})\D</Pattern>
            <Pattern result="#G1#-08-#G2#">#K1#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K08#[-\\ /_]?(\d{2})\D</Pattern>
            <Pattern result="#G1#-09-#G2#">#K1#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K09#[-\\ /_]?(\d{2})\D</Pattern>
```

```
<Pattern result="#G1#-10-#G2#">#K1#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K10#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-11-#G2#">#K1#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K11#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-12-#G2#">#K1#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K12#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern>#K1#[ ]?#?[:=][ ]?(3[01][-\\ /_]0?\d[-\\ /_]\d{2})\D</Pattern>
<Pattern>#K1#[ ]?#?[:=][ ]?(3[01][-\\ /_]1[012][-\\ /_]\d{2})\D</Pattern>
<Pattern result="#G1#-01-#G2#">#K1#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K01#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-02-#G2#">#K1#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K02#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-03-#G2#">#K1#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K03#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-04-#G2#">#K1#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K04#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-05-#G2#">#K1#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K05#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-06-#G2#">#K1#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K06#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-07-#G2#">#K1#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K07#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-08-#G2#">#K1#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K08#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-09-#G2#">#K1#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K09#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-10-#G2#">#K1#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K10#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-11-#G2#">#K1#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K11#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-12-#G2#">#K1#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K12#[-\\ /_]?(\d{2})\D</Pattern>

<Pattern>#K1#[ ]?#?[:=][ ]?([012]?\d[-\\ /_]0?\d)\D</Pattern>
<Pattern>#K1#[ ]?#?[:=][ ]?([012]?\d[-\\ /_]1[012])\D</Pattern>
<Pattern result="#G1#-01">#K1#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K01#\W</Pattern>
<Pattern result="#G1#-02">#K1#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K02#\W</Pattern>
<Pattern result="#G1#-03">#K1#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K03#\W</Pattern>
<Pattern result="#G1#-04">#K1#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K04#\W</Pattern>
<Pattern result="#G1#-05">#K1#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K05#\W</Pattern>
<Pattern result="#G1#-06">#K1#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K06#\W</Pattern>
<Pattern result="#G1#-07">#K1#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K07#\W</Pattern>
<Pattern result="#G1#-08">#K1#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K08#\W</Pattern>
<Pattern result="#G1#-09">#K1#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K09#\W</Pattern>
<Pattern result="#G1#-10">#K1#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K10#\W</Pattern>
<Pattern result="#G1#-11">#K1#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K11#\W</Pattern>
<Pattern result="#G1#-12">#K1#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K12#\W</Pattern>
<Pattern>#K1#[ ]?#?[:=][ ]?(3[01][-\\ /_]0?\d)\D</Pattern>
<Pattern>#K1#[ ]?#?[:=][ ]?(3[01][-\\ /_]1[012])\D</Pattern>
<Pattern result="#G1#-01">#K1#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K01#\W</Pattern>
<Pattern result="#G1#-02">#K1#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K02#\W</Pattern>
<Pattern result="#G1#-03">#K1#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K03#\W</Pattern>
<Pattern result="#G1#-04">#K1#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K04#\W</Pattern>
<Pattern result="#G1#-05">#K1#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K05#\W</Pattern>
<Pattern result="#G1#-06">#K1#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K06#\W</Pattern>
<Pattern result="#G1#-07">#K1#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K07#\W</Pattern>
<Pattern result="#G1#-08">#K1#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K08#\W</Pattern>
<Pattern result="#G1#-09">#K1#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K09#\W</Pattern>
<Pattern result="#G1#-10">#K1#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K10#\W</Pattern>
<Pattern result="#G1#-11">#K1#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K11#\W</Pattern>
<Pattern result="#G1#-12">#K1#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K12#\W</Pattern>

<!-- patrones con palabra clave 2 -->
<Pattern>#K2#[ ]?#?[:=][ ]?(20\d\d[-\\ /_]0?\d[-\\ /_][012]?\d)\D</Pattern>
<Pattern>#K2#[ ]?#?[:=][ ]?(20\d\d[-\\ /_]1[012][-\\ /_][012]?\d)\D</Pattern>
<Pattern result="#G2#-01-#G1#">#K2#[ ]?#?[:=][ ]?(20\d\d)[-\\ /_]?#K01#[-\\ /_]?([012]?\d)\D</Pattern>
<Pattern result="#G2#-02-#G1#">#K2#[ ]?#?[:=][ ]?(20\d\d)[-\\ /_]?#K02#[-\\ /_]?([012]?\d)\D</Pattern>
<Pattern result="#G2#-03-#G1#">#K2#[ ]?#?[:=][ ]?(20\d\d)[-\\ /_]?#K03#[-\\ /_]?([012]?\d)\D</Pattern>
<Pattern result="#G2#-04-#G1#">#K2#[ ]?#?[:=][ ]?(20\d\d)[-\\ /_]?#K04#[-\\ /_]?([012]?\d)\D</Pattern>
<Pattern result="#G2#-05-#G1#">#K2#[ ]?#?[:=][ ]?(20\d\d)[-\\ /_]?#K05#[-\\ /_]?([012]?\d)\D</Pattern>
<Pattern result="#G2#-06-#G1#">#K2#[ ]?#?[:=][ ]?(20\d\d)[-\\ /_]?#K06#[-\\ /_]?([012]?\d)\D</Pattern>
<Pattern result="#G2#-07-#G1#">#K2#[ ]?#?[:=][ ]?(20\d\d)[-\\ /_]?#K07#[-\\ /_]?([012]?\d)\D</Pattern>
<Pattern result="#G2#-08-#G1#">#K2#[ ]?#?[:=][ ]?(20\d\d)[-\\ /_]?#K08#[-\\ /_]?([012]?\d)\D</Pattern>
<Pattern result="#G2#-09-#G1#">#K2#[ ]?#?[:=][ ]?(20\d\d)[-\\ /_]?#K09#[-\\ /_]?([012]?\d)\D</Pattern>
<Pattern result="#G2#-10-#G1#">#K2#[ ]?#?[:=][ ]?(20\d\d)[-\\ /_]?#K10#[-\\ /_]?([012]?\d)\D</Pattern>
<Pattern result="#G2#-11-#G1#">#K2#[ ]?#?[:=][ ]?(20\d\d)[-\\ /_]?#K11#[-\\ /_]?([012]?\d)\D</Pattern>
<Pattern result="#G2#-12-#G1#">#K2#[ ]?#?[:=][ ]?(20\d\d)[-\\ /_]?#K12#[-\\ /_]?([012]?\d)\D</Pattern>
<Pattern>#K2#[ ]?#?[:=][ ]?(20\d\d[-\\ /_]0?\d[-\\ /_]3[01])\D</Pattern>
<Pattern>#K2#[ ]?#?[:=][ ]?(20\d\d[-\\ /_]1[012][-\\ /_]3[01])\D</Pattern>
<Pattern result="#G2#-01-#G1#">#K2#[ ]?#?[:=][ ]?(20\d\d)[-\\ /_]?#K01#[-\\ /_]?(3[01])\D</Pattern>
<Pattern result="#G2#-02-#G1#">#K2#[ ]?#?[:=][ ]?(20\d\d)[-\\ /_]?#K02#[-\\ /_]?(3[01])\D</Pattern>
<Pattern result="#G2#-03-#G1#">#K2#[ ]?#?[:=][ ]?(20\d\d)[-\\ /_]?#K03#[-\\ /_]?(3[01])\D</Pattern>
<Pattern result="#G2#-04-#G1#">#K2#[ ]?#?[:=][ ]?(20\d\d)[-\\ /_]?#K04#[-\\ /_]?(3[01])\D</Pattern>
<Pattern result="#G2#-05-#G1#">#K2#[ ]?#?[:=][ ]?(20\d\d)[-\\ /_]?#K05#[-\\ /_]?(3[01])\D</Pattern>
<Pattern result="#G2#-06-#G1#">#K2#[ ]?#?[:=][ ]?(20\d\d)[-\\ /_]?#K06#[-\\ /_]?(3[01])\D</Pattern>
<Pattern result="#G2#-07-#G1#">#K2#[ ]?#?[:=][ ]?(20\d\d)[-\\ /_]?#K07#[-\\ /_]?(3[01])\D</Pattern>
<Pattern result="#G2#-08-#G1#">#K2#[ ]?#?[:=][ ]?(20\d\d)[-\\ /_]?#K08#[-\\ /_]?(3[01])\D</Pattern>
<Pattern result="#G2#-09-#G1#">#K2#[ ]?#?[:=][ ]?(20\d\d)[-\\ /_]?#K09#[-\\ /_]?(3[01])\D</Pattern>
<Pattern result="#G2#-10-#G1#">#K2#[ ]?#?[:=][ ]?(20\d\d)[-\\ /_]?#K10#[-\\ /_]?(3[01])\D</Pattern>
<Pattern result="#G2#-11-#G1#">#K2#[ ]?#?[:=][ ]?(20\d\d)[-\\ /_]?#K11#[-\\ /_]?(3[01])\D</Pattern>
<Pattern result="#G2#-12-#G1#">#K2#[ ]?#?[:=][ ]?(20\d\d)[-\\ /_]?#K12#[-\\ /_]?(3[01])\D</Pattern>

<Pattern>#K2#[ ]?#?[:=][ ]?([012]?\d[-\\ /_]0?\d[-\\ /_]20\d\d)\D</Pattern>
<Pattern>#K2#[ ]?#?[:=][ ]?([012]?\d[-\\ /_]1[012][-\\ /_]20\d\d)\D</Pattern>
<Pattern result="#G1#-01-#G2#">#K2#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K01#[ ]?,?[-\\ /_]?(20\d\d)\D</Pattern>
<Pattern result="#G1#-02-#G2#">#K2#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K02#[ ]?,?[-\\ /_]?(20\d\d)\D</Pattern>
<Pattern result="#G1#-03-#G2#">#K2#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K03#[ ]?,?[-\\ /_]?(20\d\d)\D</Pattern>
<Pattern result="#G1#-04-#G2#">#K2#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K04#[ ]?,?[-\\ /_]?(20\d\d)\D</Pattern>
<Pattern result="#G1#-05-#G2#">#K2#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K05#[ ]?,?[-\\ /_]?(20\d\d)\D</Pattern>
<Pattern result="#G1#-06-#G2#">#K2#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K06#[ ]?,?[-\\ /_]?(20\d\d)\D</Pattern>
<Pattern result="#G1#-07-#G2#">#K2#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K07#[ ]?,?[-\\ /_]?(20\d\d)\D</Pattern>
<Pattern result="#G1#-08-#G2#">#K2#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K08#[ ]?,?[-\\ /_]?(20\d\d)\D</Pattern>
<Pattern result="#G1#-09-#G2#">#K2#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K09#[ ]?,?[-\\ /_]?(20\d\d)\D</Pattern>
<Pattern result="#G1#-10-#G2#">#K2#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K10#[ ]?,?[-\\ /_]?(20\d\d)\D</Pattern>
<Pattern result="#G1#-11-#G2#">#K2#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K11#[ ]?,?[-\\ /_]?(20\d\d)\D</Pattern>
<Pattern result="#G1#-12-#G2#">#K2#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K12#[ ]?,?[-\\ /_]?(20\d\d)\D</Pattern>
<Pattern>#K2#[ ]?#?[:=][ ]?(3[01][-\\ /_]0?\d[-\\ /_]20\d\d)\D</Pattern>
<Pattern>#K2#[ ]?#?[:=][ ]?(3[01][-\\ /_]1[012][-\\ /_]20\d\d)\D</Pattern>
<Pattern result="#G1#-01-#G2#">#K2#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K01#[ ]?,?[-\\ /_]?(20\d\d)\D</Pattern>
<Pattern result="#G1#-02-#G2#">#K2#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K02#[ ]?,?[-\\ /_]?(20\d\d)\D</Pattern>
<Pattern result="#G1#-03-#G2#">#K2#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K03#[ ]?,?[-\\ /_]?(20\d\d)\D</Pattern>
<Pattern result="#G1#-04-#G2#">#K2#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K04#[ ]?,?[-\\ /_]?(20\d\d)\D</Pattern>
<Pattern result="#G1#-05-#G2#">#K2#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K05#[ ]?,?[-\\ /_]?(20\d\d)\D</Pattern>
<Pattern result="#G1#-06-#G2#">#K2#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K06#[ ]?,?[-\\ /_]?(20\d\d)\D</Pattern>
```

```
<Pattern result="#G1#-07-#G2#">#K2#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K07#[ ]?,?[-\\ /_]?(20\d\d)\D</Pattern>
<Pattern result="#G1#-08-#G2#">#K2#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K08#[ ]?,?[-\\ /_]?(20\d\d)\D</Pattern>
<Pattern result="#G1#-09-#G2#">#K2#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K09#[ ]?,?[-\\ /_]?(20\d\d)\D</Pattern>
<Pattern result="#G1#-10-#G2#">#K2#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K10#[ ]?,?[-\\ /_]?(20\d\d)\D</Pattern>
<Pattern result="#G1#-11-#G2#">#K2#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K11#[ ]?,?[-\\ /_]?(20\d\d)\D</Pattern>
<Pattern result="#G1#-12-#G2#">#K2#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K12#[ ]?,?[-\\ /_]?(20\d\d)\D</Pattern>

<Pattern>#K2#[ ]?#?[:=][ ]?([012]?\d[-\\ /_]0?\d[-\\ /_]\d{2})\D</Pattern>
<Pattern>#K2#[ ]?#?[:=][ ]?([012]?\d[-\\ /_]1[012][-\\ /_]\d{2})\D</Pattern>
<Pattern result="#G1#-01-#G2#">#K2#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K01#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-02-#G2#">#K2#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K02#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-03-#G2#">#K2#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K03#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-04-#G2#">#K2#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K04#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-05-#G2#">#K2#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K05#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-06-#G2#">#K2#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K06#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-07-#G2#">#K2#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K07#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-08-#G2#">#K2#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K08#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-09-#G2#">#K2#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K09#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-10-#G2#">#K2#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K10#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-11-#G2#">#K2#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K11#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-12-#G2#">#K2#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K12#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern>#K2#[ ]?#?[:=][ ]?(3[01][-\\ /_]0?\d[-\\ /_]\d{2})\D</Pattern>
<Pattern>#K2#[ ]?#?[:=][ ]?(3[01][-\\ /_]1[012][-\\ /_]\d{2})\D</Pattern>
<Pattern result="#G1#-01-#G2#">#K2#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K01#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-02-#G2#">#K2#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K02#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-03-#G2#">#K2#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K03#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-04-#G2#">#K2#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K04#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-05-#G2#">#K2#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K05#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-06-#G2#">#K2#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K06#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-07-#G2#">#K2#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K07#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-08-#G2#">#K2#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K08#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-09-#G2#">#K2#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K09#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-10-#G2#">#K2#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K10#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-11-#G2#">#K2#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K11#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-12-#G2#">#K2#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K12#[-\\ /_]?(\d{2})\D</Pattern>

<Pattern>#K2#[ ]?#?[:=][ ]?([012]?\d[-\\ /_]0?\d)\D</Pattern>
<Pattern>#K2#[ ]?#?[:=][ ]?([012]?\d[-\\ /_]1[012])\D</Pattern>
<Pattern result="#G1#-01">#K2#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K01#\W</Pattern>
<Pattern result="#G1#-02">#K2#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K02#\W</Pattern>
<Pattern result="#G1#-03">#K2#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K03#\W</Pattern>
<Pattern result="#G1#-04">#K2#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K04#\W</Pattern>
<Pattern result="#G1#-05">#K2#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K05#\W</Pattern>
<Pattern result="#G1#-06">#K2#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K06#\W</Pattern>
<Pattern result="#G1#-07">#K2#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K07#\W</Pattern>
<Pattern result="#G1#-08">#K2#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K08#\W</Pattern>
<Pattern result="#G1#-09">#K2#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K09#\W</Pattern>
<Pattern result="#G1#-10">#K2#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K10#\W</Pattern>
<Pattern result="#G1#-11">#K2#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K11#\W</Pattern>
<Pattern result="#G1#-12">#K2#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K12#\W</Pattern>
<Pattern>#K2#[ ]?#?[:=][ ]?(3[01][-\\ /_]0?\d)\D</Pattern>
<Pattern>#K2#[ ]?#?[:=][ ]?(3[01][-\\ /_]1[012])\D</Pattern>
<Pattern result="#G1#-01">#K2#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K01#\W</Pattern>
<Pattern result="#G1#-02">#K2#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K02#\W</Pattern>
<Pattern result="#G1#-03">#K2#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K03#\W</Pattern>
<Pattern result="#G1#-04">#K2#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K04#\W</Pattern>
<Pattern result="#G1#-05">#K2#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K05#\W</Pattern>
<Pattern result="#G1#-06">#K2#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K06#\W</Pattern>
<Pattern result="#G1#-07">#K2#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K07#\W</Pattern>
<Pattern result="#G1#-08">#K2#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K08#\W</Pattern>
<Pattern result="#G1#-09">#K2#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K09#\W</Pattern>
<Pattern result="#G1#-10">#K2#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K10#\W</Pattern>
<Pattern result="#G1#-11">#K2#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K11#\W</Pattern>
<Pattern result="#G1#-12">#K2#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K12#\W</Pattern>

<!-- patrones sin palabra clave -->
<Pattern>\D(20\d\d[-\\ /_]0?\d[-\\ /_][012]?\d)\D</Pattern>
<Pattern>\D(20\d\d[-\\ /_]1[012][-\\ /_][012]?\d)\D</Pattern>
<Pattern result="#G2#-01-#G1#">\D(20\d\d)[-\\ /_]?#K01#[-\\ /_]?([012]?\d)\D</Pattern>
<Pattern result="#G2#-02-#G1#">\D(20\d\d)[-\\ /_]?#K02#[-\\ /_]?([012]?\d)\D</Pattern>
<Pattern result="#G2#-03-#G1#">\D(20\d\d)[-\\ /_]?#K03#[-\\ /_]?([012]?\d)\D</Pattern>
<Pattern result="#G2#-04-#G1#">\D(20\d\d)[-\\ /_]?#K04#[-\\ /_]?([012]?\d)\D</Pattern>
<Pattern result="#G2#-05-#G1#">\D(20\d\d)[-\\ /_]?#K05#[-\\ /_]?([012]?\d)\D</Pattern>
<Pattern result="#G2#-06-#G1#">\D(20\d\d)[-\\ /_]?#K06#[-\\ /_]?([012]?\d)\D</Pattern>
<Pattern result="#G2#-07-#G1#">\D(20\d\d)[-\\ /_]?#K07#[-\\ /_]?([012]?\d)\D</Pattern>
<Pattern result="#G2#-08-#G1#">\D(20\d\d)[-\\ /_]?#K08#[-\\ /_]?([012]?\d)\D</Pattern>
<Pattern result="#G2#-09-#G1#">\D(20\d\d)[-\\ /_]?#K09#[-\\ /_]?([012]?\d)\D</Pattern>
<Pattern result="#G2#-10-#G1#">\D(20\d\d)[-\\ /_]?#K10#[-\\ /_]?([012]?\d)\D</Pattern>
<Pattern result="#G2#-11-#G1#">\D(20\d\d)[-\\ /_]?#K11#[-\\ /_]?([012]?\d)\D</Pattern>
<Pattern result="#G2#-12-#G1#">\D(20\d\d)[-\\ /_]?#K12#[-\\ /_]?([012]?\d)\D</Pattern>
<Pattern>\D(20\d\d[-\\ /_]0?\d[-\\ /_]3[01])\D</Pattern>
<Pattern>\D(20\d\d[-\\ /_]1[012][-\\ /_]3[01])\D</Pattern>
<Pattern result="#G2#-01-#G1#">\D(20\d\d)[-\\ /_]?#K01#[-\\ /_]?(3[01])\D</Pattern>
<Pattern result="#G2#-02-#G1#">\D(20\d\d)[-\\ /_]?#K02#[-\\ /_]?(3[01])\D</Pattern>
<Pattern result="#G2#-03-#G1#">\D(20\d\d)[-\\ /_]?#K03#[-\\ /_]?(3[01])\D</Pattern>
<Pattern result="#G2#-04-#G1#">\D(20\d\d)[-\\ /_]?#K04#[-\\ /_]?(3[01])\D</Pattern>
<Pattern result="#G2#-05-#G1#">\D(20\d\d)[-\\ /_]?#K05#[-\\ /_]?(3[01])\D</Pattern>
<Pattern result="#G2#-06-#G1#">\D(20\d\d)[-\\ /_]?#K06#[-\\ /_]?(3[01])\D</Pattern>
<Pattern result="#G2#-07-#G1#">\D(20\d\d)[-\\ /_]?#K07#[-\\ /_]?(3[01])\D</Pattern>
<Pattern result="#G2#-08-#G1#">\D(20\d\d)[-\\ /_]?#K08#[-\\ /_]?(3[01])\D</Pattern>
<Pattern result="#G2#-09-#G1#">\D(20\d\d)[-\\ /_]?#K09#[-\\ /_]?(3[01])\D</Pattern>
<Pattern result="#G2#-10-#G1#">\D(20\d\d)[-\\ /_]?#K10#[-\\ /_]?(3[01])\D</Pattern>
<Pattern result="#G2#-11-#G1#">\D(20\d\d)[-\\ /_]?#K11#[-\\ /_]?(3[01])\D</Pattern>
<Pattern result="#G2#-12-#G1#">\D(20\d\d)[-\\ /_]?#K12#[-\\ /_]?(3[01])\D</Pattern>

<Pattern>\D([012]?\d[-\\ /_]0?\d[-\\ /_]20\d\d)\D</Pattern>
<Pattern>\D([012]?\d[-\\ /_]1[012][-\\ /_]20\d\d)\D</Pattern>
<Pattern result="#G1#-01-#G2#">\D([012]?\d)[-\\ /_]?#K01#[ ]?,?[-\\ /_]?(20\d\d)\D</Pattern>
<Pattern result="#G1#-02-#G2#">\D([012]?\d)[-\\ /_]?#K02#[ ]?,?[-\\ /_]?(20\d\d)\D</Pattern>
```

```
            <Pattern result="#G1#-03-#G2#">\D([012]?\d)[-\\ /_]?#K03#[ ]?,?[-\\ /_]?(20\d\d)\D</Pattern>
            <Pattern result="#G1#-04-#G2#">\D([012]?\d)[-\\ /_]?#K04#[ ]?,?[-\\ /_]?(20\d\d)\D</Pattern>
            <Pattern result="#G1#-05-#G2#">\D([012]?\d)[-\\ /_]?#K05#[ ]?,?[-\\ /_]?(20\d\d)\D</Pattern>
            <Pattern result="#G1#-06-#G2#">\D([012]?\d)[-\\ /_]?#K06#[ ]?,?[-\\ /_]?(20\d\d)\D</Pattern>
            <Pattern result="#G1#-07-#G2#">\D([012]?\d)[-\\ /_]?#K07#[ ]?,?[-\\ /_]?(20\d\d)\D</Pattern>
            <Pattern result="#G1#-08-#G2#">\D([012]?\d)[-\\ /_]?#K08#[ ]?,?[-\\ /_]?(20\d\d)\D</Pattern>
            <Pattern result="#G1#-09-#G2#">\D([012]?\d)[-\\ /_]?#K09#[ ]?,?[-\\ /_]?(20\d\d)\D</Pattern>
            <Pattern result="#G1#-10-#G2#">\D([012]?\d)[-\\ /_]?#K10#[ ]?,?[-\\ /_]?(20\d\d)\D</Pattern>
            <Pattern result="#G1#-11-#G2#">\D([012]?\d)[-\\ /_]?#K11#[ ]?,?[-\\ /_]?(20\d\d)\D</Pattern>
            <Pattern result="#G1#-12-#G2#">\D([012]?\d)[-\\ /_]?#K12#[ ]?,?[-\\ /_]?(20\d\d)\D</Pattern>
            <Pattern>\D(3[01][-\\ /_]0?\d[-\\ /_]20\d\d)\D</Pattern>
            <Pattern>\D(3[01][-\\ /_]1[012][-\\ /_]20\d\d)\D</Pattern>
            <Pattern result="#G1#-01-#G2#">\D(3[01])[-\\ /_]?#K01#[ ]?,?[-\\ /_]?(20\d\d)\D</Pattern>
            <Pattern result="#G1#-02-#G2#">\D(3[01])[-\\ /_]?#K02#[ ]?,?[-\\ /_]?(20\d\d)\D</Pattern>
            <Pattern result="#G1#-03-#G2#">\D(3[01])[-\\ /_]?#K03#[ ]?,?[-\\ /_]?(20\d\d)\D</Pattern>
            <Pattern result="#G1#-04-#G2#">\D(3[01])[-\\ /_]?#K04#[ ]?,?[-\\ /_]?(20\d\d)\D</Pattern>
            <Pattern result="#G1#-05-#G2#">\D(3[01])[-\\ /_]?#K05#[ ]?,?[-\\ /_]?(20\d\d)\D</Pattern>
            <Pattern result="#G1#-06-#G2#">\D(3[01])[-\\ /_]?#K06#[ ]?,?[-\\ /_]?(20\d\d)\D</Pattern>
            <Pattern result="#G1#-07-#G2#">\D(3[01])[-\\ /_]?#K07#[ ]?,?[-\\ /_]?(20\d\d)\D</Pattern>
            <Pattern result="#G1#-08-#G2#">\D(3[01])[-\\ /_]?#K08#[ ]?,?[-\\ /_]?(20\d\d)\D</Pattern>
            <Pattern result="#G1#-09-#G2#">\D(3[01])[-\\ /_]?#K09#[ ]?,?[-\\ /_]?(20\d\d)\D</Pattern>
            <Pattern result="#G1#-10-#G2#">\D(3[01])[-\\ /_]?#K10#[ ]?,?[-\\ /_]?(20\d\d)\D</Pattern>
            <Pattern result="#G1#-11-#G2#">\D(3[01])[-\\ /_]?#K11#[ ]?,?[-\\ /_]?(20\d\d)\D</Pattern>
            <Pattern result="#G1#-12-#G2#">\D(3[01])[-\\ /_]?#K12#[ ]?,?[-\\ /_]?(20\d\d)\D</Pattern>

            <Pattern>\D([012]?\d[-\\ /_]0?\d[-\\ /_]\d{2})\D</Pattern>
            <Pattern>\D([012]?\d[-\\ /_]1[012][-\\ /_]\d{2})\D</Pattern>
            <Pattern result="#G1#-01-#G2#">\D([012]?\d)[-\\ /_]?#K01#[-\\ /_]?(\d{2})\D</Pattern>
            <Pattern result="#G1#-02-#G2#">\D([012]?\d)[-\\ /_]?#K02#[-\\ /_]?(\d{2})\D</Pattern>
            <Pattern result="#G1#-03-#G2#">\D([012]?\d)[-\\ /_]?#K03#[-\\ /_]?(\d{2})\D</Pattern>
            <Pattern result="#G1#-04-#G2#">\D([012]?\d)[-\\ /_]?#K04#[-\\ /_]?(\d{2})\D</Pattern>
            <Pattern result="#G1#-05-#G2#">\D([012]?\d)[-\\ /_]?#K05#[-\\ /_]?(\d{2})\D</Pattern>
            <Pattern result="#G1#-06-#G2#">\D([012]?\d)[-\\ /_]?#K06#[-\\ /_]?(\d{2})\D</Pattern>
            <Pattern result="#G1#-07-#G2#">\D([012]?\d)[-\\ /_]?#K07#[-\\ /_]?(\d{2})\D</Pattern>
            <Pattern result="#G1#-08-#G2#">\D([012]?\d)[-\\ /_]?#K08#[-\\ /_]?(\d{2})\D</Pattern>
            <Pattern result="#G1#-09-#G2#">\D([012]?\d)[-\\ /_]?#K09#[-\\ /_]?(\d{2})\D</Pattern>
            <Pattern result="#G1#-10-#G2#">\D([012]?\d)[-\\ /_]?#K10#[-\\ /_]?(\d{2})\D</Pattern>
            <Pattern result="#G1#-11-#G2#">\D([012]?\d)[-\\ /_]?#K11#[-\\ /_]?(\d{2})\D</Pattern>
            <Pattern result="#G1#-12-#G2#">\D([012]?\d)[-\\ /_]?#K12#[-\\ /_]?(\d{2})\D</Pattern>
            <Pattern>\D(3[01][-\\ /_]0?\d[-\\ /_]\d{2})\D</Pattern>
            <Pattern>\D(3[01][-\\ /_]1[012][-\\ /_]\d{2})\D</Pattern>
            <Pattern result="#G1#-01-#G2#">\D(3[01])[-\\ /_]?#K01#[-\\ /_]?(\d{2})\D</Pattern>
            <Pattern result="#G1#-02-#G2#">\D(3[01])[-\\ /_]?#K02#[-\\ /_]?(\d{2})\D</Pattern>
            <Pattern result="#G1#-03-#G2#">\D(3[01])[-\\ /_]?#K03#[-\\ /_]?(\d{2})\D</Pattern>
            <Pattern result="#G1#-04-#G2#">\D(3[01])[-\\ /_]?#K04#[-\\ /_]?(\d{2})\D</Pattern>
            <Pattern result="#G1#-05-#G2#">\D(3[01])[-\\ /_]?#K05#[-\\ /_]?(\d{2})\D</Pattern>
            <Pattern result="#G1#-06-#G2#">\D(3[01])[-\\ /_]?#K06#[-\\ /_]?(\d{2})\D</Pattern>
            <Pattern result="#G1#-07-#G2#">\D(3[01])[-\\ /_]?#K07#[-\\ /_]?(\d{2})\D</Pattern>
            <Pattern result="#G1#-08-#G2#">\D(3[01])[-\\ /_]?#K08#[-\\ /_]?(\d{2})\D</Pattern>
            <Pattern result="#G1#-09-#G2#">\D(3[01])[-\\ /_]?#K09#[-\\ /_]?(\d{2})\D</Pattern>
            <Pattern result="#G1#-10-#G2#">\D(3[01])[-\\ /_]?#K10#[-\\ /_]?(\d{2})\D</Pattern>
            <Pattern result="#G1#-11-#G2#">\D(3[01])[-\\ /_]?#K11#[-\\ /_]?(\d{2})\D</Pattern>
            <Pattern result="#G1#-12-#G2#">\D(3[01])[-\\ /_]?#K12#[-\\ /_]?(\d{2})\D</Pattern>

            <Pattern>\D([012]?\d[-\\ /_]0?\d)\D</Pattern>
            <Pattern>\D([012]?\d[-\\ /_]1[012])\D</Pattern>
            <Pattern result="#G1#-01">\D([012]?\d)[-\\ /_]?#K01#\W</Pattern>
            <Pattern result="#G1#-02">\D([012]?\d)[-\\ /_]?#K02#\W</Pattern>
            <Pattern result="#G1#-03">\D([012]?\d)[-\\ /_]?#K03#\W</Pattern>
            <Pattern result="#G1#-04">\D([012]?\d)[-\\ /_]?#K04#\W</Pattern>
            <Pattern result="#G1#-05">\D([012]?\d)[-\\ /_]?#K05#\W</Pattern>
            <Pattern result="#G1#-06">\D([012]?\d)[-\\ /_]?#K06#\W</Pattern>
            <Pattern result="#G1#-07">\D([012]?\d)[-\\ /_]?#K07#\W</Pattern>
            <Pattern result="#G1#-08">\D([012]?\d)[-\\ /_]?#K08#\W</Pattern>
            <Pattern result="#G1#-09">\D([012]?\d)[-\\ /_]?#K09#\W</Pattern>
            <Pattern result="#G1#-10">\D([012]?\d)[-\\ /_]?#K10#\W</Pattern>
            <Pattern result="#G1#-11">\D([012]?\d)[-\\ /_]?#K11#\W</Pattern>
            <Pattern result="#G1#-12">\D([012]?\d)[-\\ /_]?#K12#\W</Pattern>
            <Pattern>\D(3[01][-\\ /_]0?\d)\D</Pattern>
            <Pattern>\D(3[01][-\\ /_]1[012])\D</Pattern>
            <Pattern result="#G1#-01">\D(3[01])[-\\ /_]?#K01#\W</Pattern>
            <Pattern result="#G1#-02">\D(3[01])[-\\ /_]?#K02#\W</Pattern>
            <Pattern result="#G1#-03">\D(3[01])[-\\ /_]?#K03#\W</Pattern>
            <Pattern result="#G1#-04">\D(3[01])[-\\ /_]?#K04#\W</Pattern>
            <Pattern result="#G1#-05">\D(3[01])[-\\ /_]?#K05#\W</Pattern>
            <Pattern result="#G1#-06">\D(3[01])[-\\ /_]?#K06#\W</Pattern>
            <Pattern result="#G1#-07">\D(3[01])[-\\ /_]?#K07#\W</Pattern>
            <Pattern result="#G1#-08">\D(3[01])[-\\ /_]?#K08#\W</Pattern>
            <Pattern result="#G1#-09">\D(3[01])[-\\ /_]?#K09#\W</Pattern>
            <Pattern result="#G1#-10">\D(3[01])[-\\ /_]?#K10#\W</Pattern>
            <Pattern result="#G1#-11">\D(3[01])[-\\ /_]?#K11#\W</Pattern>
            <Pattern result="#G1#-12">\D(3[01])[-\\ /_]?#K12#\W</Pattern>

        </Date>
        <Time>
            <Keyword id="#K1#">Time, Leaving, Start, Arrival</Keyword>
            <Pattern result="#G1#:#G2#:PM">#K1#[ ]?#?[:=][ ]*((?:2[0123])|(?:[01]?\d))[:]([012345]\d)[ ]?PM\D</Pattern>
            <Pattern result="#G1#:#G2#:PM">\D((?:2[0123])|(?:[01]?\d))[:]([012345]\d)[ ]?PM\D</Pattern>
            <Pattern result="#G1#:#G2#">#K1#[ ]?#?[:=][ ]*((?:2[0123])|(?:[01]?\d))[:]([012345]\d)\D</Pattern>
            <Pattern result="#G1#:#G2#">\D((?:2[0123])|(?:[01]?\d))[:]([012345]\d)\D</Pattern>
        </Time>
        <Address>
            <Keyword id="#K1#">Street, Location, Place, Origin, Leaving, Destination</Keyword>
            <Pattern>#K1#[ ]?#?[:=][ ]?([\p{Alpha}[\p{Punct}&amp;&amp;[^\.:#=;]]][\w
[ºª][\p{Punct}&amp;&amp;[^\.:#=;]]]*)[\.#;[\s&amp;&amp;[^ ]]]</Pattern>
        </Address>
        <City>
            <Keyword id="#K1#">City, Town, Destination</Keyword>
            <Pattern>#K1#[ ]?#?[:=][ ]?(\p{Alpha}[\w ]*)[\.,#;[\s&amp;&amp;[^ ]]]</Pattern>
```

```
        </City>
        <ZipCode>
            <Keyword id="#K1#">Zip[ ]?Code, Z[ ]?[\.]?C[ ]?[\.]?</Keyword>
            <Pattern>#K1#[ ]?#?[:=][ ]? (\w[\w ]*)[\.,#;\s]</Pattern>
        </ZipCode>
        <Country>
            <Pattern>COUNTRY[ ]?#?[:=][ ]?(\p{Alpha}[\w ]*)[\.,#;[\s&amp;&amp;[^ ]]]</Pattern>
        </Country>
        <Description>
            <Keyword id="#K1#">Description, Summary</Keyword>
            <Pattern>#K1#[ ]?#?[:=][ ]?([\w [ºª][\p{Punct}]]+(?:[\s&amp;&amp;[^ ]][\w [ºª][\p{Punct}]]+)*)[\s&amp;&amp;[^
]][\s&amp;&amp;[^ ]]</Pattern>
        </Description>
        <Category>
            <Keyword id="#K0#">Category, Nature, Type</Keyword>
            <Keyword id="#K1#">Sport, Sports, Match, Sportive, Football, Basquetball, Tennis</Keyword>
            <Keyword id="#K2#">Cinema, Cinemas, Video, Videos, Cinema/Video, Film, Films, Actor, Actors, Actriz, Actress,
Actresses, Protagonist, Protagonists, Video Club, VideoClub, Hire</Keyword>
            <Keyword id="#K3#">Music, Musician, Musical, Singer, Singers, Musical, Artist, Artists, Concert, Concerts, Song,
Songs, Disk, Disks, Disc, Discs</Keyword>
            <Keyword id="#K4#">Art, Arts</Keyword>
            <Keyword id="#K5#">Comedy, Comedies, Comedian, Comedians, Monologue, Monologues</Keyword>
            <Keyword id="#K6#">Comercial</Keyword>
            <Keyword id="#K7#">Festival, Festivals</Keyword>
            <Keyword id="#K8#">Education</Keyword>
            <Keyword id="#K9#">Travel, Travels, Ticket, Origin, Destination, Booking, Train, Palin, Airplane, Leaving, Arrival,
Trip, Round Trip, One Way</Keyword>
            <Keyword id="#K10#">Family, Families</Keyword>
            <Keyword id="#K11#">Politics, Politician, Politicians</Keyword>
            <Keyword id="#K12#">Social</Keyword>

            <Pattern>#K0#[ ]?#?[:=][ ]?(\w[\w ]*)[\.,#;[\s&amp;&amp;[^ ]]]</Pattern>
            <Pattern result="Sport">\W#K1#\W</Pattern>
            <Pattern result="Cinema/Video">\W#K2#\W</Pattern>
            <Pattern result="Music">\W#K3#\W</Pattern>
            <Pattern result="Arts">\W#K4#\W</Pattern>
            <Pattern result="Comedy">\W#K5#\W</Pattern>
            <Pattern result="Commercial">\W#K6#\W</Pattern>
            <Pattern result="Festivals">\W#K7#\W</Pattern>
            <Pattern result="Education">\W#K8#\W</Pattern>
            <Pattern result="Travel">\W#K9#\W</Pattern>
            <Pattern result="Family">\W#K10#\W</Pattern>
            <Pattern result="Politics">\W#K11#\W</Pattern>
            <Pattern result="Social">\W#K12#\W</Pattern>
        </Category>
        <Privacy>
            <Pattern>PRIVACY[ ]?#?[:=][ ]?([\w]+)[\.,#;[\s&amp;&amp;[^ ]]]</Pattern>
        </Privacy>
    </ENG>

    <ESP>
        <Title>
            <Keyword id="#K1#">Titulo, Tema, Evento, Espectaculo</Keyword>
            <Pattern>#K1#[ ]?#?[:=][ ]?([\w[\p{Punct}&amp;&amp;[^\.:#=,;]]][\w
[\p{Punct}&amp;&amp;[^\.:#=,;]]]*)[\.,#;[\s&amp;&amp;[^ ]]]</Pattern>
            <Pattern>#K1#[ ]?#?[:=]?[ ]?([\w[\p{Punct}&amp;&amp;[^\.:#=,;]]][\w
[\p{Punct}&amp;&amp;[^\.:#=,;]]]*)</Pattern>
        </Title>
        <Date>
            <Keyword id="#K1#">Devolucion, Llegada, Salida</Keyword>
            <Keyword id="#K2#">Fecha</Keyword>
            <Keyword id="#K01#">Ene,Enero,[ ]?de Enero[ ]?,[ ]?de Enero de[ ]?</Keyword>
            <Keyword id="#K02#">Feb,Febrero,[ ]?de Febrero[ ]?,[ ]?de Febrero de[ ]?</Keyword>
            <Keyword id="#K03#">Mar,Marzo,[ ]?de Marzo[ ]?,[ ]?de Marzo de[ ]?</Keyword>
            <Keyword id="#K04#">Abr,Abril,[ ]?de Abril[ ]?,[ ]?de Abril de[ ]?</Keyword>
            <Keyword id="#K05#">May,Mayo,[ ]?de Mayo[ ]?,[ ]?de Mayo de[ ]?</Keyword>
            <Keyword id="#K06#">Jun,Junio,[ ]?de Junio[ ]?,[ ]?de Junio de[ ]?</Keyword>
            <Keyword id="#K07#">Jul,Julio,[ ]?de Julio[ ]?,[ ]?de Julio de[ ]?</Keyword>
            <Keyword id="#K08#">Ago,Agosto,[ ]?de Agosto[ ]?,[ ]?de Agosto de[ ]?</Keyword>
            <Keyword id="#K09#">Sep,Septiembre,[ ]?de Septiembre[ ]?,[ ]?de Septiembre de[ ]?</Keyword>
            <Keyword id="#K10#">Oct,Octubre,[ ]?de Octubre[ ]?,[ ]?de Octubre de[ ]?</Keyword>
            <Keyword id="#K11#">Nov,Noviembre,[ ]?de Noviembre[ ]?,[ ]?de Noviembre de[ ]?</Keyword>
            <Keyword id="#K12#">Dic,Diciembre,[ ]?de Diciembre[ ]?,[ ]?de Diciembre de[ ]?</Keyword>

            <!-- patrones con palabra clave 1 -->
            <Pattern>#K1#[ ]?#?[:=][ ]?(20\d\d[-\\ /_]0?\d[-\\ /_][012]?\d)\D</Pattern>
            <Pattern>#K1#[ ]?#?[:=][ ]?(20\d\d[-\\ /_]1[012][-\\ /_][012]?\d)\D</Pattern>
            <Pattern result="#G2#-01-#G1#">#K1#[ ]?#?[:=][ ]?(20\d\d)[-\\ /_]?#K01#[-\\ /_]?([012]?\d)\D</Pattern>
            <Pattern result="#G2#-02-#G1#">#K1#[ ]?#?[:=][ ]?(20\d\d)[-\\ /_]?#K02#[-\\ /_]?([012]?\d)\D</Pattern>
            <Pattern result="#G2#-03-#G1#">#K1#[ ]?#?[:=][ ]?(20\d\d)[-\\ /_]?#K03#[-\\ /_]?([012]?\d)\D</Pattern>
            <Pattern result="#G2#-04-#G1#">#K1#[ ]?#?[:=][ ]?(20\d\d)[-\\ /_]?#K04#[-\\ /_]?([012]?\d)\D</Pattern>
            <Pattern result="#G2#-05-#G1#">#K1#[ ]?#?[:=][ ]?(20\d\d)[-\\ /_]?#K05#[-\\ /_]?([012]?\d)\D</Pattern>
            <Pattern result="#G2#-06-#G1#">#K1#[ ]?#?[:=][ ]?(20\d\d)[-\\ /_]?#K06#[-\\ /_]?([012]?\d)\D</Pattern>
            <Pattern result="#G2#-07-#G1#">#K1#[ ]?#?[:=][ ]?(20\d\d)[-\\ /_]?#K07#[-\\ /_]?([012]?\d)\D</Pattern>
            <Pattern result="#G2#-08-#G1#">#K1#[ ]?#?[:=][ ]?(20\d\d)[-\\ /_]?#K08#[-\\ /_]?([012]?\d)\D</Pattern>
            <Pattern result="#G2#-09-#G1#">#K1#[ ]?#?[:=][ ]?(20\d\d)[-\\ /_]?#K09#[-\\ /_]?([012]?\d)\D</Pattern>
            <Pattern result="#G2#-10-#G1#">#K1#[ ]?#?[:=][ ]?(20\d\d)[-\\ /_]?#K10#[-\\ /_]?([012]?\d)\D</Pattern>
            <Pattern result="#G2#-11-#G1#">#K1#[ ]?#?[:=][ ]?(20\d\d)[-\\ /_]?#K11#[-\\ /_]?([012]?\d)\D</Pattern>
            <Pattern result="#G2#-12-#G1#">#K1#[ ]?#?[:=][ ]?(20\d\d)[-\\ /_]?#K12#[-\\ /_]?([012]?\d)\D</Pattern>
            <Pattern>#K1#[ ]?#?[:=][ ]?(20\d\d[-\\ /_]0?\d[-\\ /_]3[01])\D</Pattern>
            <Pattern>#K1#[ ]?#?[:=][ ]?(20\d\d[-\\ /_]1[012][-\\ /_]3[01])\D</Pattern>
            <Pattern result="#G2#-01-#G1#">#K1#[ ]?#?[:=][ ]?(20\d\d)[-\\ /_]?#K01#[-\\ /_]?(3[01])\D</Pattern>
            <Pattern result="#G2#-02-#G1#">#K1#[ ]?#?[:=][ ]?(20\d\d)[-\\ /_]?#K02#[-\\ /_]?(3[01])\D</Pattern>
            <Pattern result="#G2#-03-#G1#">#K1#[ ]?#?[:=][ ]?(20\d\d)[-\\ /_]?#K03#[-\\ /_]?(3[01])\D</Pattern>
            <Pattern result="#G2#-04-#G1#">#K1#[ ]?#?[:=][ ]?(20\d\d)[-\\ /_]?#K04#[-\\ /_]?(3[01])\D</Pattern>
            <Pattern result="#G2#-05-#G1#">#K1#[ ]?#?[:=][ ]?(20\d\d)[-\\ /_]?#K05#[-\\ /_]?(3[01])\D</Pattern>
            <Pattern result="#G2#-06-#G1#">#K1#[ ]?#?[:=][ ]?(20\d\d)[-\\ /_]?#K06#[-\\ /_]?(3[01])\D</Pattern>
            <Pattern result="#G2#-07-#G1#">#K1#[ ]?#?[:=][ ]?(20\d\d)[-\\ /_]?#K07#[-\\ /_]?(3[01])\D</Pattern>
            <Pattern result="#G2#-08-#G1#">#K1#[ ]?#?[:=][ ]?(20\d\d)[-\\ /_]?#K08#[-\\ /_]?(3[01])\D</Pattern>
```

```
<Pattern result="#G2#-09-#G1#">#K1#[ ]?#?[:=][ ]?(20\d\d)[-\\ /_]?#K09#[-\\ /_]?(3[01])\D</Pattern>
<Pattern result="#G2#-10-#G1#">#K1#[ ]?#?[:=][ ]?(20\d\d)[-\\ /_]?#K10#[-\\ /_]?(3[01])\D</Pattern>
<Pattern result="#G2#-11-#G1#">#K1#[ ]?#?[:=][ ]?(20\d\d)[-\\ /_]?#K11#[-\\ /_]?(3[01])\D</Pattern>
<Pattern result="#G2#-12-#G1#">#K1#[ ]?#?[:=][ ]?(20\d\d)[-\\ /_]?#K12#[-\\ /_]?(3[01])\D</Pattern>

<Pattern>#K1#[ ]?#?[:=][ ]?([012]?\d[-\\ /_]0?\d[-\\ /_]20\d\d)\D</Pattern>
<Pattern>#K1#[ ]?#?[:=][ ]?([012]?\d[-\\ /_]1[012][-\\ /_]20\d\d)\D</Pattern>
<Pattern result="#G1#-01-#G2#">#K1#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K01#[-\\ /_]?(20\d\d)\D</Pattern>
<Pattern result="#G1#-02-#G2#">#K1#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K02#[-\\ /_]?(20\d\d)\D</Pattern>
<Pattern result="#G1#-03-#G2#">#K1#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K03#[-\\ /_]?(20\d\d)\D</Pattern>
<Pattern result="#G1#-04-#G2#">#K1#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K04#[-\\ /_]?(20\d\d)\D</Pattern>
<Pattern result="#G1#-05-#G2#">#K1#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K05#[-\\ /_]?(20\d\d)\D</Pattern>
<Pattern result="#G1#-06-#G2#">#K1#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K06#[-\\ /_]?(20\d\d)\D</Pattern>
<Pattern result="#G1#-07-#G2#">#K1#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K07#[-\\ /_]?(20\d\d)\D</Pattern>
<Pattern result="#G1#-08-#G2#">#K1#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K08#[-\\ /_]?(20\d\d)\D</Pattern>
<Pattern result="#G1#-09-#G2#">#K1#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K09#[-\\ /_]?(20\d\d)\D</Pattern>
<Pattern result="#G1#-10-#G2#">#K1#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K10#[-\\ /_]?(20\d\d)\D</Pattern>
<Pattern result="#G1#-11-#G2#">#K1#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K11#[-\\ /_]?(20\d\d)\D</Pattern>
<Pattern result="#G1#-12-#G2#">#K1#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K12#[-\\ /_]?(20\d\d)\D</Pattern>
<Pattern>#K1#[ ]?#?[:=][ ]?(3[01][-\\ /_]0?\d[-\\ /_]20\d\d)\D</Pattern>
<Pattern>#K1#[ ]?#?[:=][ ]?(3[01][-\\ /_]1[012][-\\ /_]20\d\d)\D</Pattern>
<Pattern result="#G1#-01-#G2#">#K1#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K01#[-\\ /_]?(20\d\d)\D</Pattern>
<Pattern result="#G1#-02-#G2#">#K1#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K02#[-\\ /_]?(20\d\d)\D</Pattern>
<Pattern result="#G1#-03-#G2#">#K1#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K03#[-\\ /_]?(20\d\d)\D</Pattern>
<Pattern result="#G1#-04-#G2#">#K1#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K04#[-\\ /_]?(20\d\d)\D</Pattern>
<Pattern result="#G1#-05-#G2#">#K1#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K05#[-\\ /_]?(20\d\d)\D</Pattern>
<Pattern result="#G1#-06-#G2#">#K1#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K06#[-\\ /_]?(20\d\d)\D</Pattern>
<Pattern result="#G1#-07-#G2#">#K1#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K07#[-\\ /_]?(20\d\d)\D</Pattern>
<Pattern result="#G1#-08-#G2#">#K1#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K08#[-\\ /_]?(20\d\d)\D</Pattern>
<Pattern result="#G1#-09-#G2#">#K1#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K09#[-\\ /_]?(20\d\d)\D</Pattern>
<Pattern result="#G1#-10-#G2#">#K1#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K10#[-\\ /_]?(20\d\d)\D</Pattern>
<Pattern result="#G1#-11-#G2#">#K1#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K11#[-\\ /_]?(20\d\d)\D</Pattern>
<Pattern result="#G1#-12-#G2#">#K1#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K12#[-\\ /_]?(20\d\d)\D</Pattern>

<Pattern>#K1#[ ]?#?[:=][ ]?([012]?\d[-\\ /_]0?\d[-\\ /_]\d{2})\D</Pattern>
<Pattern>#K1#[ ]?#?[:=][ ]?([012]?\d[-\\ /_]1[012][-\\ /_]\d{2})\D</Pattern>
<Pattern result="#G1#-01-#G2#">#K1#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K01#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-02-#G2#">#K1#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K02#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-03-#G2#">#K1#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K03#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-04-#G2#">#K1#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K04#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-05-#G2#">#K1#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K05#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-06-#G2#">#K1#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K06#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-07-#G2#">#K1#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K07#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-08-#G2#">#K1#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K08#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-09-#G2#">#K1#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K09#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-10-#G2#">#K1#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K10#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-11-#G2#">#K1#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K11#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-12-#G2#">#K1#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K12#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern>#K1#[ ]?#?[:=][ ]?(3[01][-\\ /_]0?\d[-\\ /_]\d{2})\D</Pattern>
<Pattern>#K1#[ ]?#?[:=][ ]?(3[01][-\\ /_]1[012][-\\ /_]\d{2})\D</Pattern>
<Pattern result="#G1#-01-#G2#">#K1#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K01#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-02-#G2#">#K1#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K02#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-03-#G2#">#K1#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K03#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-04-#G2#">#K1#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K04#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-05-#G2#">#K1#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K05#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-06-#G2#">#K1#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K06#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-07-#G2#">#K1#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K07#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-08-#G2#">#K1#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K08#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-09-#G2#">#K1#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K09#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-10-#G2#">#K1#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K10#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-11-#G2#">#K1#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K11#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-12-#G2#">#K1#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K12#[-\\ /_]?(\d{2})\D</Pattern>

<Pattern>#K1#[ ]?#?[:=][ ]?([012]?\d[-\\ /_]0?\d)\D</Pattern>
<Pattern>#K1#[ ]?#?[:=][ ]?([012]?\d[-\\ /_]1[012])\D</Pattern>
<Pattern result="#G1#-01">#K1#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K01#\W</Pattern>
<Pattern result="#G1#-02">#K1#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K02#\W</Pattern>
<Pattern result="#G1#-03">#K1#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K03#\W</Pattern>
<Pattern result="#G1#-04">#K1#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K04#\W</Pattern>
<Pattern result="#G1#-05">#K1#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K05#\W</Pattern>
<Pattern result="#G1#-06">#K1#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K06#\W</Pattern>
<Pattern result="#G1#-07">#K1#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K07#\W</Pattern>
<Pattern result="#G1#-08">#K1#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K08#\W</Pattern>
<Pattern result="#G1#-09">#K1#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K09#\W</Pattern>
<Pattern result="#G1#-10">#K1#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K10#\W</Pattern>
<Pattern result="#G1#-11">#K1#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K11#\W</Pattern>
<Pattern result="#G1#-12">#K1#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K12#\W</Pattern>
<Pattern>#K1#[ ]?#?[:=][ ]?(3[01][-\\ /_]0?\d)\D</Pattern>
<Pattern>#K1#[ ]?#?[:=][ ]?(3[01][-\\ /_]1[012])\D</Pattern>
<Pattern result="#G1#-01">#K1#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K01#\W</Pattern>
<Pattern result="#G1#-02">#K1#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K02#\W</Pattern>
<Pattern result="#G1#-03">#K1#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K03#\W</Pattern>
<Pattern result="#G1#-04">#K1#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K04#\W</Pattern>
<Pattern result="#G1#-05">#K1#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K05#\W</Pattern>
<Pattern result="#G1#-06">#K1#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K06#\W</Pattern>
<Pattern result="#G1#-07">#K1#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K07#\W</Pattern>
<Pattern result="#G1#-08">#K1#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K08#\W</Pattern>
<Pattern result="#G1#-09">#K1#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K09#\W</Pattern>
<Pattern result="#G1#-10">#K1#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K10#\W</Pattern>
<Pattern result="#G1#-11">#K1#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K11#\W</Pattern>
<Pattern result="#G1#-12">#K1#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K12#\W</Pattern>

<!-- patrones con palabra clave 2 -->
<Pattern>#K2#[ ]?#?[:=][ ]?(20\d\d[-\\ /_]0?\d[-\\ /_][012]?\d)\D</Pattern>
<Pattern>#K2#[ ]?#?[:=][ ]?(20\d\d[-\\ /_]1[012][-\\ /_][012]?\d)\D</Pattern>
<Pattern result="#G2#-01-#G1#">#K2#[ ]?#?[:=][ ]?(20\d\d)[-\\ /_]?#K01#[-\\ /_]?([012]?\d)\D</Pattern>
<Pattern result="#G2#-02-#G1#">#K2#[ ]?#?[:=][ ]?(20\d\d)[-\\ /_]?#K02#[-\\ /_]?([012]?\d)\D</Pattern>
<Pattern result="#G2#-03-#G1#">#K2#[ ]?#?[:=][ ]?(20\d\d)[-\\ /_]?#K03#[-\\ /_]?([012]?\d)\D</Pattern>
<Pattern result="#G2#-04-#G1#">#K2#[ ]?#?[:=][ ]?(20\d\d)[-\\ /_]?#K04#[-\\ /_]?([012]?\d)\D</Pattern>
```

```
<Pattern result="#G2#-05-#G1#">#K2#[ ]?#?[:=][ ]?(20\d\d)[-\\ /_]?#K05#[-\\ /_]?([012]?\d)\D</Pattern>
<Pattern result="#G2#-06-#G1#">#K2#[ ]?#?[:=][ ]?(20\d\d)[-\\ /_]?#K06#[-\\ /_]?([012]?\d)\D</Pattern>
<Pattern result="#G2#-07-#G1#">#K2#[ ]?#?[:=][ ]?(20\d\d)[-\\ /_]?#K07#[-\\ /_]?([012]?\d)\D</Pattern>
<Pattern result="#G2#-08-#G1#">#K2#[ ]?#?[:=][ ]?(20\d\d)[-\\ /_]?#K08#[-\\ /_]?([012]?\d)\D</Pattern>
<Pattern result="#G2#-09-#G1#">#K2#[ ]?#?[:=][ ]?(20\d\d)[-\\ /_]?#K09#[-\\ /_]?([012]?\d)\D</Pattern>
<Pattern result="#G2#-10-#G1#">#K2#[ ]?#?[:=][ ]?(20\d\d)[-\\ /_]?#K10#[-\\ /_]?([012]?\d)\D</Pattern>
<Pattern result="#G2#-11-#G1#">#K2#[ ]?#?[:=][ ]?(20\d\d)[-\\ /_]?#K11#[-\\ /_]?([012]?\d)\D</Pattern>
<Pattern result="#G2#-12-#G1#">#K2#[ ]?#?[:=][ ]?(20\d\d)[-\\ /_]?#K12#[-\\ /_]?([012]?\d)\D</Pattern>
<Pattern>#K2#[ ]?#?[:=][ ]?(20\d\d[-\\ /_]0?\d[-\\ /_]3[01])\D</Pattern>
<Pattern>#K2#[ ]?#?[:=][ ]?(20\d\d[-\\ /_]1[012][-\\ /_]3[01])\D</Pattern>
<Pattern result="#G2#-01-#G1#">#K2#[ ]?#?[:=][ ]?(20\d\d)[-\\ /_]?#K01#[-\\ /_]?(3[01])\D</Pattern>
<Pattern result="#G2#-02-#G1#">#K2#[ ]?#?[:=][ ]?(20\d\d)[-\\ /_]?#K02#[-\\ /_]?(3[01])\D</Pattern>
<Pattern result="#G2#-03-#G1#">#K2#[ ]?#?[:=][ ]?(20\d\d)[-\\ /_]?#K03#[-\\ /_]?(3[01])\D</Pattern>
<Pattern result="#G2#-04-#G1#">#K2#[ ]?#?[:=][ ]?(20\d\d)[-\\ /_]?#K04#[-\\ /_]?(3[01])\D</Pattern>
<Pattern result="#G2#-05-#G1#">#K2#[ ]?#?[:=][ ]?(20\d\d)[-\\ /_]?#K05#[-\\ /_]?(3[01])\D</Pattern>
<Pattern result="#G2#-06-#G1#">#K2#[ ]?#?[:=][ ]?(20\d\d)[-\\ /_]?#K06#[-\\ /_]?(3[01])\D</Pattern>
<Pattern result="#G2#-07-#G1#">#K2#[ ]?#?[:=][ ]?(20\d\d)[-\\ /_]?#K07#[-\\ /_]?(3[01])\D</Pattern>
<Pattern result="#G2#-08-#G1#">#K2#[ ]?#?[:=][ ]?(20\d\d)[-\\ /_]?#K08#[-\\ /_]?(3[01])\D</Pattern>
<Pattern result="#G2#-09-#G1#">#K2#[ ]?#?[:=][ ]?(20\d\d)[-\\ /_]?#K09#[-\\ /_]?(3[01])\D</Pattern>
<Pattern result="#G2#-10-#G1#">#K2#[ ]?#?[:=][ ]?(20\d\d)[-\\ /_]?#K10#[-\\ /_]?(3[01])\D</Pattern>
<Pattern result="#G2#-11-#G1#">#K2#[ ]?#?[:=][ ]?(20\d\d)[-\\ /_]?#K11#[-\\ /_]?(3[01])\D</Pattern>
<Pattern result="#G2#-12-#G1#">#K2#[ ]?#?[:=][ ]?(20\d\d)[-\\ /_]?#K12#[-\\ /_]?(3[01])\D</Pattern>

<Pattern>#K2#[ ]?#?[:=][ ]?([012]?\d[-\\ /_]0?\d[-\\ /_]20\d\d)\D</Pattern>
<Pattern>#K2#[ ]?#?[:=][ ]?([012]?\d[-\\ /_]1[012][-\\ /_]20\d\d)\D</Pattern>
<Pattern result="#G1#-01-#G2#">#K2#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K01#[-\\ /_]?(20\d\d)\D</Pattern>
<Pattern result="#G1#-02-#G2#">#K2#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K02#[-\\ /_]?(20\d\d)\D</Pattern>
<Pattern result="#G1#-03-#G2#">#K2#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K03#[-\\ /_]?(20\d\d)\D</Pattern>
<Pattern result="#G1#-04-#G2#">#K2#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K04#[-\\ /_]?(20\d\d)\D</Pattern>
<Pattern result="#G1#-05-#G2#">#K2#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K05#[-\\ /_]?(20\d\d)\D</Pattern>
<Pattern result="#G1#-06-#G2#">#K2#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K06#[-\\ /_]?(20\d\d)\D</Pattern>
<Pattern result="#G1#-07-#G2#">#K2#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K07#[-\\ /_]?(20\d\d)\D</Pattern>
<Pattern result="#G1#-08-#G2#">#K2#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K08#[-\\ /_]?(20\d\d)\D</Pattern>
<Pattern result="#G1#-09-#G2#">#K2#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K09#[-\\ /_]?(20\d\d)\D</Pattern>
<Pattern result="#G1#-10-#G2#">#K2#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K10#[-\\ /_]?(20\d\d)\D</Pattern>
<Pattern result="#G1#-11-#G2#">#K2#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K11#[-\\ /_]?(20\d\d)\D</Pattern>
<Pattern result="#G1#-12-#G2#">#K2#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K12#[-\\ /_]?(20\d\d)\D</Pattern>
<Pattern>#K2#[ ]?#?[:=][ ]?(3[01][-\\ /_]0?\d[-\\ /_]20\d\d)\D</Pattern>
<Pattern>#K2#[ ]?#?[:=][ ]?(3[01][-\\ /_]1[012][-\\ /_]20\d\d)\D</Pattern>
<Pattern result="#G1#-01-#G2#">#K2#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K01#[-\\ /_]?(20\d\d)\D</Pattern>
<Pattern result="#G1#-02-#G2#">#K2#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K02#[-\\ /_]?(20\d\d)\D</Pattern>
<Pattern result="#G1#-03-#G2#">#K2#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K03#[-\\ /_]?(20\d\d)\D</Pattern>
<Pattern result="#G1#-04-#G2#">#K2#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K04#[-\\ /_]?(20\d\d)\D</Pattern>
<Pattern result="#G1#-05-#G2#">#K2#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K05#[-\\ /_]?(20\d\d)\D</Pattern>
<Pattern result="#G1#-06-#G2#">#K2#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K06#[-\\ /_]?(20\d\d)\D</Pattern>
<Pattern result="#G1#-07-#G2#">#K2#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K07#[-\\ /_]?(20\d\d)\D</Pattern>
<Pattern result="#G1#-08-#G2#">#K2#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K08#[-\\ /_]?(20\d\d)\D</Pattern>
<Pattern result="#G1#-09-#G2#">#K2#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K09#[-\\ /_]?(20\d\d)\D</Pattern>
<Pattern result="#G1#-10-#G2#">#K2#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K10#[-\\ /_]?(20\d\d)\D</Pattern>
<Pattern result="#G1#-11-#G2#">#K2#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K11#[-\\ /_]?(20\d\d)\D</Pattern>
<Pattern result="#G1#-12-#G2#">#K2#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K12#[-\\ /_]?(20\d\d)\D</Pattern>

<Pattern>#K2#[ ]?#?[:=][ ]?([012]?\d[-\\ /_]0?\d[-\\ /_]\d{2})\D</Pattern>
<Pattern>#K2#[ ]?#?[:=][ ]?([012]?\d[-\\ /_]1[012][-\\ /_]\d{2})\D</Pattern>
<Pattern result="#G1#-01-#G2#">#K2#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K01#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-02-#G2#">#K2#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K02#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-03-#G2#">#K2#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K03#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-04-#G2#">#K2#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K04#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-05-#G2#">#K2#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K05#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-06-#G2#">#K2#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K06#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-07-#G2#">#K2#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K07#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-08-#G2#">#K2#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K08#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-09-#G2#">#K2#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K09#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-10-#G2#">#K2#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K10#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-11-#G2#">#K2#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K11#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-12-#G2#">#K2#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K12#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern>#K2#[ ]?#?[:=][ ]?(3[01][-\\ /_]0?\d[-\\ /_]\d{2})\D</Pattern>
<Pattern>#K2#[ ]?#?[:=][ ]?(3[01][-\\ /_]1[012][-\\ /_]\d{2})\D</Pattern>
<Pattern result="#G1#-01-#G2#">#K2#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K01#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-02-#G2#">#K2#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K02#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-03-#G2#">#K2#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K03#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-04-#G2#">#K2#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K04#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-05-#G2#">#K2#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K05#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-06-#G2#">#K2#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K06#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-07-#G2#">#K2#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K07#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-08-#G2#">#K2#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K08#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-09-#G2#">#K2#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K09#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-10-#G2#">#K2#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K10#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-11-#G2#">#K2#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K11#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-12-#G2#">#K2#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K12#[-\\ /_]?(\d{2})\D</Pattern>

<Pattern>#K2#[ ]?#?[:=][ ]?([012]?\d[-\\ /_]0?\d)\D</Pattern>
<Pattern>#K2#[ ]?#?[:=][ ]?([012]?\d[-\\ /_]1[012])\D</Pattern>
<Pattern result="#G1#-01">#K2#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K01#\W</Pattern>
<Pattern result="#G1#-02">#K2#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K02#\W</Pattern>
<Pattern result="#G1#-03">#K2#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K03#\W</Pattern>
<Pattern result="#G1#-04">#K2#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K04#\W</Pattern>
<Pattern result="#G1#-05">#K2#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K05#\W</Pattern>
<Pattern result="#G1#-06">#K2#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K06#\W</Pattern>
<Pattern result="#G1#-07">#K2#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K07#\W</Pattern>
<Pattern result="#G1#-08">#K2#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K08#\W</Pattern>
<Pattern result="#G1#-09">#K2#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K09#\W</Pattern>
<Pattern result="#G1#-10">#K2#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K10#\W</Pattern>
<Pattern result="#G1#-11">#K2#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K11#\W</Pattern>
<Pattern result="#G1#-12">#K2#[ ]?#?[:=][ ]?([012]?\d)[-\\ /_]?#K12#\W</Pattern>
<Pattern>#K2#[ ]?#?[:=][ ]?(3[01][-\\ /_]0?\d)\D</Pattern>
<Pattern>#K2#[ ]?#?[:=][ ]?(3[01][-\\ /_]1[012])\D</Pattern>
<Pattern result="#G1#-01">#K2#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K01#\W</Pattern>
<Pattern result="#G1#-02">#K2#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K02#\W</Pattern>
```

```
<Pattern result="#G1#-03">#K2#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K03#\W</Pattern>
<Pattern result="#G1#-04">#K2#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K04#\W</Pattern>
<Pattern result="#G1#-05">#K2#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K05#\W</Pattern>
<Pattern result="#G1#-06">#K2#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K06#\W</Pattern>
<Pattern result="#G1#-07">#K2#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K07#\W</Pattern>
<Pattern result="#G1#-08">#K2#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K08#\W</Pattern>
<Pattern result="#G1#-09">#K2#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K09#\W</Pattern>
<Pattern result="#G1#-10">#K2#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K10#\W</Pattern>
<Pattern result="#G1#-11">#K2#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K11#\W</Pattern>
<Pattern result="#G1#-12">#K2#[ ]?#?[:=][ ]?(3[01])[-\\ /_]?#K12#\W</Pattern>

<!-- patrones sin palabra clave -->
<Pattern>\D(20\d\d[-\\ /_]0?\d[-\\ /_][012]?\d)\D</Pattern>
<Pattern>\D(20\d\d[-\\ /_]1[012][-\\ /_][012]?\d)\D</Pattern>
<Pattern result="#G2#-01-#G1#">\D(20\d\d)[-\\ /_]?#K01#[-\\ /_]?([012]?\d)\D</Pattern>
<Pattern result="#G2#-02-#G1#">\D(20\d\d)[-\\ /_]?#K02#[-\\ /_]?([012]?\d)\D</Pattern>
<Pattern result="#G2#-03-#G1#">\D(20\d\d)[-\\ /_]?#K03#[-\\ /_]?([012]?\d)\D</Pattern>
<Pattern result="#G2#-04-#G1#">\D(20\d\d)[-\\ /_]?#K04#[-\\ /_]?([012]?\d)\D</Pattern>
<Pattern result="#G2#-05-#G1#">\D(20\d\d)[-\\ /_]?#K05#[-\\ /_]?([012]?\d)\D</Pattern>
<Pattern result="#G2#-06-#G1#">\D(20\d\d)[-\\ /_]?#K06#[-\\ /_]?([012]?\d)\D</Pattern>
<Pattern result="#G2#-07-#G1#">\D(20\d\d)[-\\ /_]?#K07#[-\\ /_]?([012]?\d)\D</Pattern>
<Pattern result="#G2#-08-#G1#">\D(20\d\d)[-\\ /_]?#K08#[-\\ /_]?([012]?\d)\D</Pattern>
<Pattern result="#G2#-09-#G1#">\D(20\d\d)[-\\ /_]?#K09#[-\\ /_]?([012]?\d)\D</Pattern>
<Pattern result="#G2#-10-#G1#">\D(20\d\d)[-\\ /_]?#K10#[-\\ /_]?([012]?\d)\D</Pattern>
<Pattern result="#G2#-11-#G1#">\D(20\d\d)[-\\ /_]?#K11#[-\\ /_]?([012]?\d)\D</Pattern>
<Pattern result="#G2#-12-#G1#">\D(20\d\d)[-\\ /_]?#K12#[-\\ /_]?([012]?\d)\D</Pattern>
<Pattern>\D(20\d\d[-\\ /_]0?\d[-\\ /_]3[01])\D</Pattern>
<Pattern>\D(20\d\d[-\\ /_]1[012][-\\ /_]3[01])\D</Pattern>
<Pattern result="#G2#-01-#G1#">\D(20\d\d)[-\\ /_]?#K01#[-\\ /_]?(3[01])\D</Pattern>
<Pattern result="#G2#-02-#G1#">\D(20\d\d)[-\\ /_]?#K02#[-\\ /_]?(3[01])\D</Pattern>
<Pattern result="#G2#-03-#G1#">\D(20\d\d)[-\\ /_]?#K03#[-\\ /_]?(3[01])\D</Pattern>
<Pattern result="#G2#-04-#G1#">\D(20\d\d)[-\\ /_]?#K04#[-\\ /_]?(3[01])\D</Pattern>
<Pattern result="#G2#-05-#G1#">\D(20\d\d)[-\\ /_]?#K05#[-\\ /_]?(3[01])\D</Pattern>
<Pattern result="#G2#-06-#G1#">\D(20\d\d)[-\\ /_]?#K06#[-\\ /_]?(3[01])\D</Pattern>
<Pattern result="#G2#-07-#G1#">\D(20\d\d)[-\\ /_]?#K07#[-\\ /_]?(3[01])\D</Pattern>
<Pattern result="#G2#-08-#G1#">\D(20\d\d)[-\\ /_]?#K08#[-\\ /_]?(3[01])\D</Pattern>
<Pattern result="#G2#-09-#G1#">\D(20\d\d)[-\\ /_]?#K09#[-\\ /_]?(3[01])\D</Pattern>
<Pattern result="#G2#-10-#G1#">\D(20\d\d)[-\\ /_]?#K10#[-\\ /_]?(3[01])\D</Pattern>
<Pattern result="#G2#-11-#G1#">\D(20\d\d)[-\\ /_]?#K11#[-\\ /_]?(3[01])\D</Pattern>
<Pattern result="#G2#-12-#G1#">\D(20\d\d)[-\\ /_]?#K12#[-\\ /_]?(3[01])\D</Pattern>

<Pattern>\D([012]?\d[-\\ /_]0?\d[-\\ /_]20\d\d)\D</Pattern>
<Pattern>\D([012]?\d[-\\ /_]1[012][-\\ /_]20\d\d)\D</Pattern>
<Pattern result="#G1#-01-#G2#">\D([012]?\d)[-\\ /_]?#K01#[-\\ /_]?(20\d\d)\D</Pattern>
<Pattern result="#G1#-02-#G2#">\D([012]?\d)[-\\ /_]?#K02#[-\\ /_]?(20\d\d)\D</Pattern>
<Pattern result="#G1#-03-#G2#">\D([012]?\d)[-\\ /_]?#K03#[-\\ /_]?(20\d\d)\D</Pattern>
<Pattern result="#G1#-04-#G2#">\D([012]?\d)[-\\ /_]?#K04#[-\\ /_]?(20\d\d)\D</Pattern>
<Pattern result="#G1#-05-#G2#">\D([012]?\d)[-\\ /_]?#K05#[-\\ /_]?(20\d\d)\D</Pattern>
<Pattern result="#G1#-06-#G2#">\D([012]?\d)[-\\ /_]?#K06#[-\\ /_]?(20\d\d)\D</Pattern>
<Pattern result="#G1#-07-#G2#">\D([012]?\d)[-\\ /_]?#K07#[-\\ /_]?(20\d\d)\D</Pattern>
<Pattern result="#G1#-08-#G2#">\D([012]?\d)[-\\ /_]?#K08#[-\\ /_]?(20\d\d)\D</Pattern>
<Pattern result="#G1#-09-#G2#">\D([012]?\d)[-\\ /_]?#K09#[-\\ /_]?(20\d\d)\D</Pattern>
<Pattern result="#G1#-10-#G2#">\D([012]?\d)[-\\ /_]?#K10#[-\\ /_]?(20\d\d)\D</Pattern>
<Pattern result="#G1#-11-#G2#">\D([012]?\d)[-\\ /_]?#K11#[-\\ /_]?(20\d\d)\D</Pattern>
<Pattern result="#G1#-12-#G2#">\D([012]?\d)[-\\ /_]?#K12#[-\\ /_]?(20\d\d)\D</Pattern>
<Pattern>\D(3[01][-\\ /_]0?\d[-\\ /_]20\d\d)\D</Pattern>
<Pattern>\D(3[01][-\\ /_]1[012][-\\ /_]20\d\d)\D</Pattern>
<Pattern result="#G1#-01-#G2#">\D(3[01])[-\\ /_]?#K01#[-\\ /_]?(20\d\d)\D</Pattern>
<Pattern result="#G1#-02-#G2#">\D(3[01])[-\\ /_]?#K02#[-\\ /_]?(20\d\d)\D</Pattern>
<Pattern result="#G1#-03-#G2#">\D(3[01])[-\\ /_]?#K03#[-\\ /_]?(20\d\d)\D</Pattern>
<Pattern result="#G1#-04-#G2#">\D(3[01])[-\\ /_]?#K04#[-\\ /_]?(20\d\d)\D</Pattern>
<Pattern result="#G1#-05-#G2#">\D(3[01])[-\\ /_]?#K05#[-\\ /_]?(20\d\d)\D</Pattern>
<Pattern result="#G1#-06-#G2#">\D(3[01])[-\\ /_]?#K06#[-\\ /_]?(20\d\d)\D</Pattern>
<Pattern result="#G1#-07-#G2#">\D(3[01])[-\\ /_]?#K07#[-\\ /_]?(20\d\d)\D</Pattern>
<Pattern result="#G1#-08-#G2#">\D(3[01])[-\\ /_]?#K08#[-\\ /_]?(20\d\d)\D</Pattern>
<Pattern result="#G1#-09-#G2#">\D(3[01])[-\\ /_]?#K09#[-\\ /_]?(20\d\d)\D</Pattern>
<Pattern result="#G1#-10-#G2#">\D(3[01])[-\\ /_]?#K10#[-\\ /_]?(20\d\d)\D</Pattern>
<Pattern result="#G1#-11-#G2#">\D(3[01])[-\\ /_]?#K11#[-\\ /_]?(20\d\d)\D</Pattern>
<Pattern result="#G1#-12-#G2#">\D(3[01])[-\\ /_]?#K12#[-\\ /_]?(20\d\d)\D</Pattern>

<Pattern>\D([012]?\d[-\\ /_]0?\d[-\\ /_]\d{2})\D</Pattern>
<Pattern>\D([012]?\d[-\\ /_]1[012][-\\ /_]\d{2})\D</Pattern>
<Pattern result="#G1#-01-#G2#">\D([012]?\d)[-\\ /_]?#K01#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-02-#G2#">\D([012]?\d)[-\\ /_]?#K02#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-03-#G2#">\D([012]?\d)[-\\ /_]?#K03#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-04-#G2#">\D([012]?\d)[-\\ /_]?#K04#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-05-#G2#">\D([012]?\d)[-\\ /_]?#K05#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-06-#G2#">\D([012]?\d)[-\\ /_]?#K06#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-07-#G2#">\D([012]?\d)[-\\ /_]?#K07#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-08-#G2#">\D([012]?\d)[-\\ /_]?#K08#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-09-#G2#">\D([012]?\d)[-\\ /_]?#K09#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-10-#G2#">\D([012]?\d)[-\\ /_]?#K10#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-11-#G2#">\D([012]?\d)[-\\ /_]?#K11#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-12-#G2#">\D([012]?\d)[-\\ /_]?#K12#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern>\D(3[01][-\\ /_]0?\d[-\\ /_]\d{2})\D</Pattern>
<Pattern>\D(3[01][-\\ /_]1[012][-\\ /_]\d{2})\D</Pattern>
<Pattern result="#G1#-01-#G2#">\D(3[01])[-\\ /_]?#K01#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-02-#G2#">\D(3[01])[-\\ /_]?#K02#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-03-#G2#">\D(3[01])[-\\ /_]?#K03#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-04-#G2#">\D(3[01])[-\\ /_]?#K04#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-05-#G2#">\D(3[01])[-\\ /_]?#K05#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-06-#G2#">\D(3[01])[-\\ /_]?#K06#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-07-#G2#">\D(3[01])[-\\ /_]?#K07#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-08-#G2#">\D(3[01])[-\\ /_]?#K08#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-09-#G2#">\D(3[01])[-\\ /_]?#K09#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-10-#G2#">\D(3[01])[-\\ /_]?#K10#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-11-#G2#">\D(3[01])[-\\ /_]?#K11#[-\\ /_]?(\d{2})\D</Pattern>
<Pattern result="#G1#-12-#G2#">\D(3[01])[-\\ /_]?#K12#[-\\ /_]?(\d{2})\D</Pattern>
```

```
            <Pattern>\D([012]?\d[-\\ /_]0?\d)\D</Pattern>
            <Pattern>\D([012]?\d[-\\ /_]1[012])\D</Pattern>
            <Pattern result="#G1#-01">\D([012]?\d)[-\\ /_]?#K01#\W</Pattern>
            <Pattern result="#G1#-02">\D([012]?\d)[-\\ /_]?#K02#\W</Pattern>
            <Pattern result="#G1#-03">\D([012]?\d)[-\\ /_]?#K03#\W</Pattern>
            <Pattern result="#G1#-04">\D([012]?\d)[-\\ /_]?#K04#\W</Pattern>
            <Pattern result="#G1#-05">\D([012]?\d)[-\\ /_]?#K05#\W</Pattern>
            <Pattern result="#G1#-06">\D([012]?\d)[-\\ /_]?#K06#\W</Pattern>
            <Pattern result="#G1#-07">\D([012]?\d)[-\\ /_]?#K07#\W</Pattern>
            <Pattern result="#G1#-08">\D([012]?\d)[-\\ /_]?#K08#\W</Pattern>
            <Pattern result="#G1#-09">\D([012]?\d)[-\\ /_]?#K09#\W</Pattern>
            <Pattern result="#G1#-10">\D([012]?\d)[-\\ /_]?#K10#\W</Pattern>
            <Pattern result="#G1#-11">\D([012]?\d)[-\\ /_]?#K11#\W</Pattern>
            <Pattern result="#G1#-12">\D([012]?\d)[-\\ /_]?#K12#\W</Pattern>
            <Pattern>\D(3[01][-\\ /_]0?\d)\D</Pattern>
            <Pattern>\D(3[01][-\\ /_]1[012])\D</Pattern>
            <Pattern result="#G1#-01">\D(3[01])[-\\ /_]?#K01#\W</Pattern>
            <Pattern result="#G1#-02">\D(3[01])[-\\ /_]?#K02#\W</Pattern>
            <Pattern result="#G1#-03">\D(3[01])[-\\ /_]?#K03#\W</Pattern>
            <Pattern result="#G1#-04">\D(3[01])[-\\ /_]?#K04#\W</Pattern>
            <Pattern result="#G1#-05">\D(3[01])[-\\ /_]?#K05#\W</Pattern>
            <Pattern result="#G1#-06">\D(3[01])[-\\ /_]?#K06#\W</Pattern>
            <Pattern result="#G1#-07">\D(3[01])[-\\ /_]?#K07#\W</Pattern>
            <Pattern result="#G1#-08">\D(3[01])[-\\ /_]?#K08#\W</Pattern>
            <Pattern result="#G1#-09">\D(3[01])[-\\ /_]?#K09#\W</Pattern>
            <Pattern result="#G1#-10">\D(3[01])[-\\ /_]?#K10#\W</Pattern>
            <Pattern result="#G1#-11">\D(3[01])[-\\ /_]?#K11#\W</Pattern>
            <Pattern result="#G1#-12">\D(3[01])[-\\ /_]?#K12#\W</Pattern>

        </Date>
        <Time>
            <Keyword id="#K1#">Hora, Salida, Comienzo, Llegada</Keyword>
            <Pattern result="#G1#:#G2#:PM">#K1#[ ]?#?[:=][ ]*((?:2[0123])|(?:[01]?\d))[:]([012345]\d)[ ]?PM\D</Pattern>
            <Pattern result="#G1#:#G2#:PM">\D((?:2[0123])|(?:[01]?\d))[:]([012345]\d)[ ]?PM\D</Pattern>
            <Pattern result="#G1#:#G2#">#K1#[ ]?#?[:=][ ]*((?:2[0123])|(?:[01]?\d))[:]([012345]\d)\D</Pattern>
            <Pattern result="#G1#:#G2#">\D((?:2[0123])|(?:[01]?\d))[:]([012345]\d)\D</Pattern>
        </Time>
        <Address>
            <Keyword id="#K1#">Calle, Lugar, Sitio, Salida, Origen, Local, Destino, Direccion</Keyword>
            <Pattern>#K1#[ ]?#?[:=][ ]?([\p{Alpha}[\p{Punct}&amp;&amp;[^\.:#=;]]][\w
[ºª][\p{Punct}&amp;&amp;[^\.:#=;]]]*)[\.#;[\s&amp;&amp;[^ ]]]</Pattern>
        </Address>
        <City>
            <Keyword id="#K1#">Ciudad, Pueblo, Poblacion, Destino</Keyword>
            <Pattern>#K1#[ ]?#?[:=][ ]?(\p{Alpha}[\w ]*)[\.,#;[\s&amp;&amp;[^ ]]]</Pattern>
            <Pattern>#K1#[ ]?de[ ]?(\p{Alpha}[\w ]*)[\.,#;[\s&amp;&amp;[^ ]]]</Pattern>
            <Pattern>#K1#[ ]?(\p{Alpha}[\w ]*)[\.,#;[\s&amp;&amp;[^ ]]]</Pattern>
        </City>
        <ZipCode>
            <Keyword id="#K1#">Codigo[ ]?Postal, C[ ]?[\.]?P[ ]?[\.]?</Keyword>
            <Pattern>#K1#[ ]?#?[:=][ ]?(\w[\w]*)[\.,#;\s]</Pattern>
        </ZipCode>
        <Country>
            <Pattern>COUNTRY[ ]?#?[:=][ ]?(\p{Alpha}[\w ]*)[\.,#;[\s&amp;&amp;[^ ]]]</Pattern>
        </Country>
        <Description>
            <Keyword id="#K1#">Descripcion, Resumen</Keyword>
            <Pattern>#K1#[ ]?#?[:=][ ]?([\w [ºª][\p{Punct}]]+(?:[\s&amp;&amp;[^ ]][\w [ºª][\p{Punct}]]+)*)[\s&amp;&amp;[^
]][\s&amp;&amp;[^ ]]</Pattern>
        </Description>
        <Category>
            <Keyword id="#K0#">Categoria, Naturaleza, Tipo</Keyword>
            <Keyword id="#K1#">Deporte, Deportes, Partido, Deportivo, Futbol, Football, Balonpie, Baloncesto, Basquet,
Basquetball, Tenis, Tennis</Keyword>
            <Keyword id="#K2#">Cine, Cines, Video, Videos, Cine/Video, Pelicula, Peliculas, Actor, Actores, Actriz, Actrices,
Protagonista, Protagonistas, Video Club, Alquiler, Alquilar</Keyword>
            <Keyword id="#K3#">Musica, Musico, Musicos, Cantante, Cantantes, Musical, Artista, Artistas, Concierto, Conciertos,
Cancion, Canciones, Disco, Discos</Keyword>
            <Keyword id="#K4#">Arte, Artes</Keyword>
            <Keyword id="#K5#">Comedia, Comedias, Comico, Comicos, Monologo, Monologos</Keyword>
            <Keyword id="#K6#">Comercial</Keyword>
            <Keyword id="#K7#">Festival, Festivales</Keyword>
            <Keyword id="#K8#">Educacion</Keyword>
            <Keyword id="#K9#">Viaje, Viajes, Billete, Billetes, Origen, Destino, Tren, Avion, Salida, Llegada, Ida,
Vuelta</Keyword>
            <Keyword id="#K10#">Familia, Familias</Keyword>
            <Keyword id="#K11#">Politica, Politicas</Keyword>
            <Keyword id="#K12#">Social</Keyword>
            <Pattern>#K0#[ ]?#?[:=][ ]?(\w[\w ]*)[\.,#;[\s&amp;&amp;[^ ]]]</Pattern>
            <Pattern result="Sport">\W#K1#\W</Pattern>
            <Pattern result="Cinema/Video">\W#K2#\W</Pattern>
            <Pattern result="Music">\W#K3#\W</Pattern>
            <Pattern result="Arts">\W#K4#\W</Pattern>
            <Pattern result="Comedy">\W#K5#\W</Pattern>
            <Pattern result="Commercial">\W#K6#\W</Pattern>
            <Pattern result="Festivals">\W#K7#\W</Pattern>
            <Pattern result="Education">\W#K8#\W</Pattern>
            <Pattern result="Travel">\W#K9#\W</Pattern>
            <Pattern result="Family">\W#K10#\W</Pattern>
            <Pattern result="Politics">\W#K11#\W</Pattern>
            <Pattern result="Social">\W#K12#\W</Pattern>
        </Category>
        <Privacy>
            <Pattern>PRIVACIDAD[ ]?#?#?[:=][ ]?([\w]+)[\.,#;[\s&amp;&amp;[^ ]]]</Pattern>
        </Privacy>
    </ESP>
</Grammar>
```

C. Plantillas de generación de eventos

C.1. Plantillas para SMS

La creación de un evento a partir de un SMS es muy similar a la creación de un evento a partir de un correo electrónico. La limitación del número de caracteres hace que las etiquetas se reduzcan lo máximo posible para acortar el número de caracteres del mensaje a enviar por el usuario.

Para crear un evento, el SMS debe enviarse al número corto (y su centro de mensajes) asociado a la aplicación desde el número de teléfono registrado por el usuario en la aplicación. La estructura del SMS debe ser:

"Etiqueta" "Separador" "Contenido" "Final de contenido"

- Etiqueta.

Por cada etiqueta se soporta la inicial y el texto reducido hasta tres caracteres. Por ejemplo: Para el título su etiqueta puede ser "T" o la palabra clave título recortada hasta tres caracteres (TIT), siendo una etiqueta extendida. Esto es útil para recordar etiquetas de un carácter que difieren de su nombre. Por ejemplo, en inglés la 'D' está reservada para la fecha "Date", la etiqueta de la descripción en cambio es 'X', aunque podemos poner "Des" que es la abreviación de "Description".

- Separador.

El separador debe ser ':'.

- Contenido.

El contenido es el texto con el que el usuario desea rellenar el campo del evento definido por la etiqueta.

- Final de contenido.

El Final del contenido se marca con un '.'.

Las plantillas de SMS a utilizar son:

- Plantilla en inglés:

  S:(contenido para Subject).
  D:(contenido para Date).
  T:(contenido para Time).
  L:(contenido para Address),(contenido para City).
  Z:(contenido para Zip Code).
  C:(contenido para Country).
  X:(contenido para Description).
  N:(contenido para Category).
  P:(public / registered / private).

- Plantilla en castellano:

  T:(contenido para Título).
  F:(contenido para Fecha).
  H:(contenido para Hora).
  L:(contenido para Calle),(contenido para Ciudad).
  C:(contenido para Código Postal).
  P:(contenido para País).
  D:(contenido para Descripción).
  N:(contenido para Categoría).
  A:(publico / registrado/ privado).

Las plantillas y su significado pueden verse en el emulador mediante los botones: “Load ESP Template” y “Load ENG Template”. El significado también puede verse en la sección de leyenda de las plantillas.

C.2. Plantillas para correo electrónico

Para crear un evento se debe enviar el correo a la cuenta de la Agenda Social o Generador de Eventos desde la dirección de correo del usuario que ha sido registrado en la aplicación. Ese correo debe seguir la siguiente estructura:

"Etiqueta" "Separador" "Contenido" "Final de contenido"

- Etiqueta.

Las etiquetas se utilizan para identificar el contenido que se debe almacenar en un cierto campo. Son una serie de palabras reservadas y su nombre hace alusión al campo del evento. Existen etiquetas para cada idioma declarado en el Generador de Eventos. Cuando se implementa un nuevo idioma para la creación de nuevos eventos se deben declarar nuevos patrones de búsqueda. En estos patrones se encuentran implícitamente las etiquetas.

Las etiquetas pueden ser usadas en cualquier orden y no deben estar todas cumplimentadas, ni siquiera deben estar escritas si van a estar vacías. No son susceptibles a las mayúsculas por lo que pueden escribirse en mayúsculas, minúsculas o combinación de ambas.

- Separador.

La separación entre la etiqueta y el contenido se realiza mediante un separador. Este separador podrá ser ':' o '='. Entre la etiqueta y el separador o entre el separador y el contenido puede haber espacio o no.

- Contenido.

El contenido es el texto con el que el usuario desea rellenar el campo marcado por la etiqueta.

- Final de contenido.

El Final del contenido se marca con un '.' o con un salto de línea (Enter, '↵').

Las plantillas de correo electrónico a utilizar son:

- Plantilla en inglés:

TITLE :
DATE :
TIME :
STREET :
CITY :
ZIPCODE :
COUNTRY :
CATEGORY :
PRIVACY : (Public, Registered, Private)
DESCRIPTION :

- Plantilla en castellano:

TITULO :
FECHA :
HORA :
CALLE :
CIUDAD :
CODIGO POSTAL :
PAIS :
CATEGORIA :
PRIVACIDAD: (Publico, Registrado, Privado)
DESCRIPCION :

D. Proceso de creación de un evento

Figura D.1: Recepción de un correo electrónico

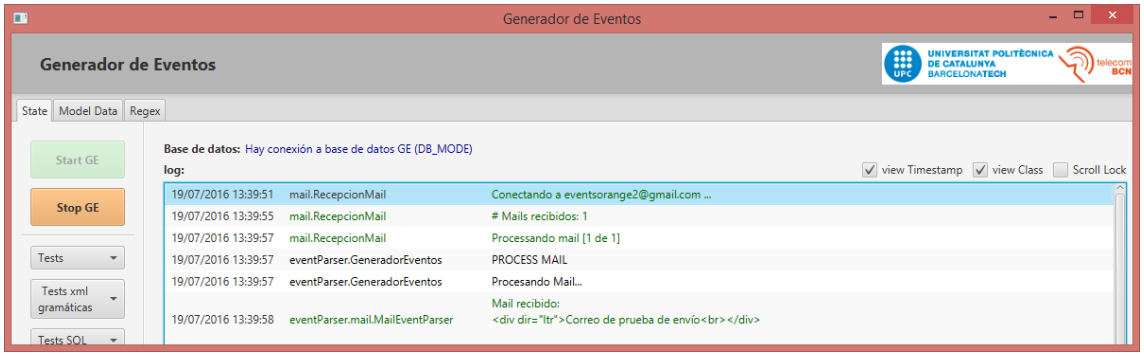


Figura D.2: Recepción de un SMS

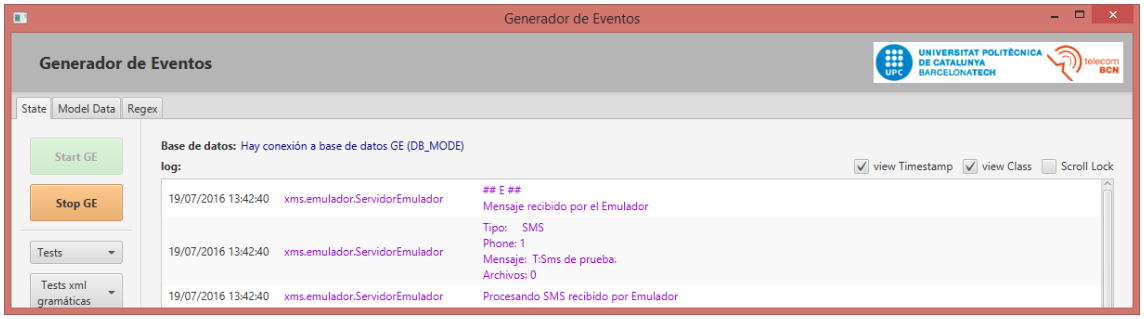
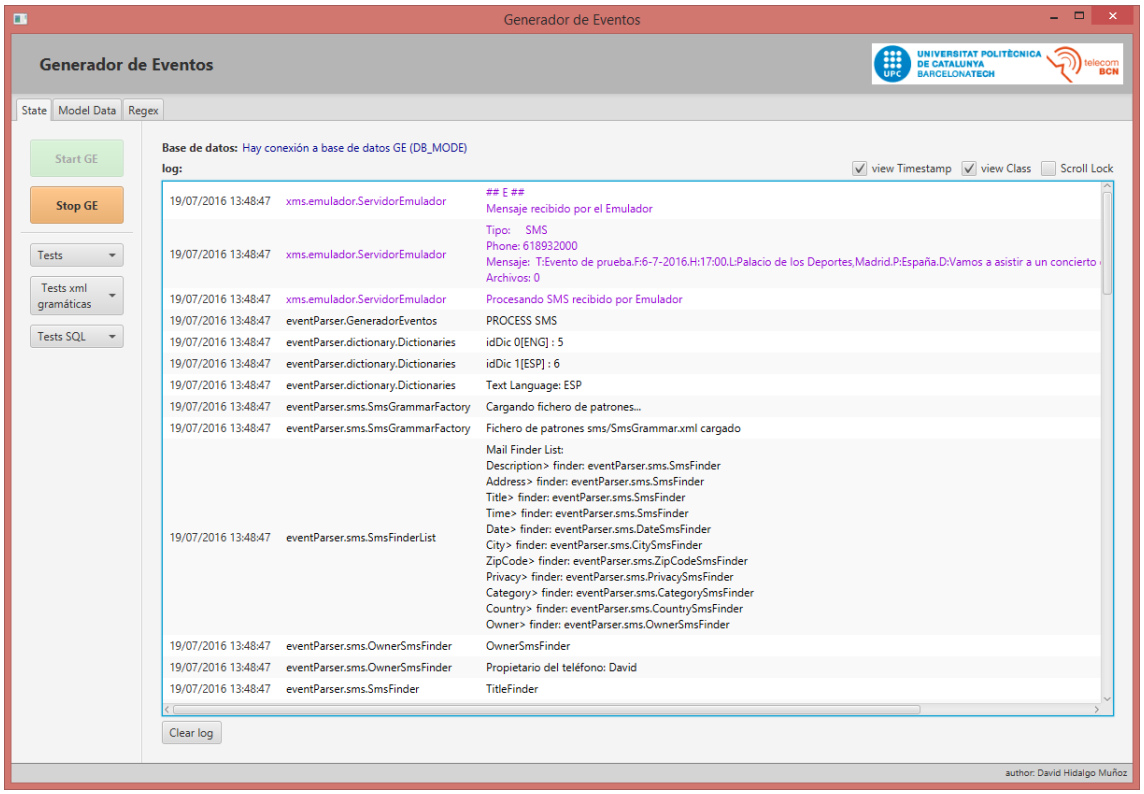
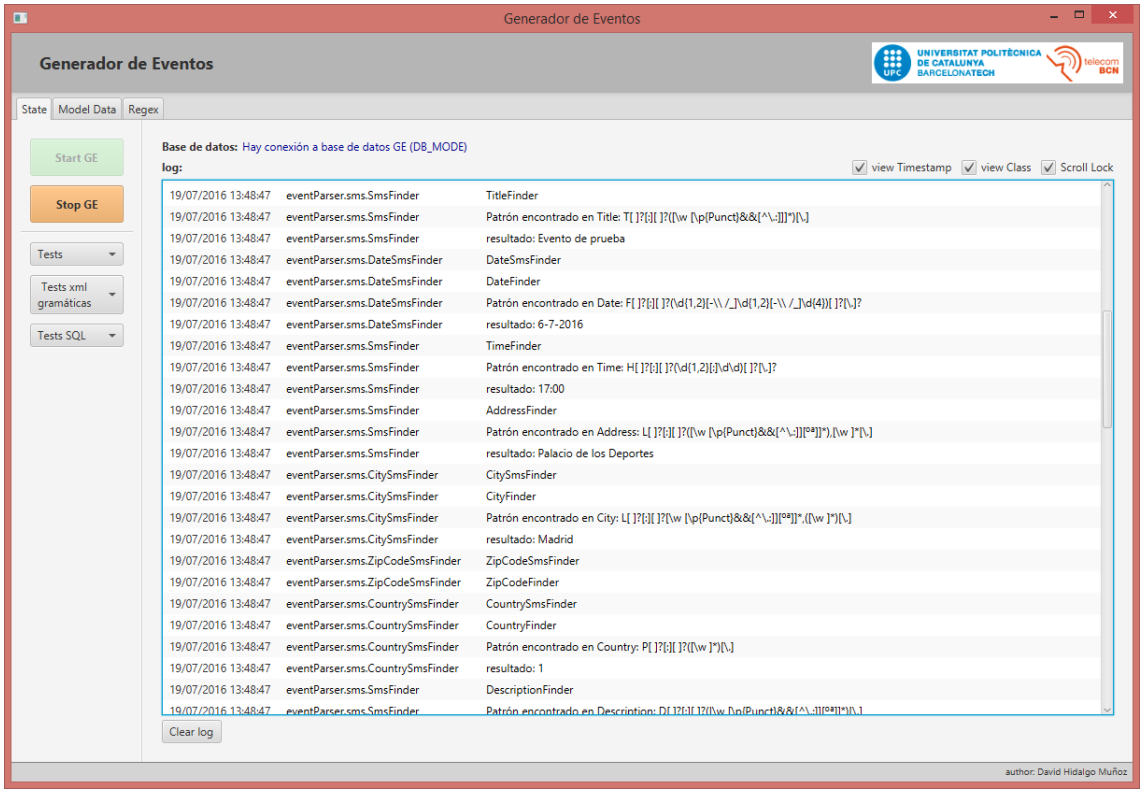
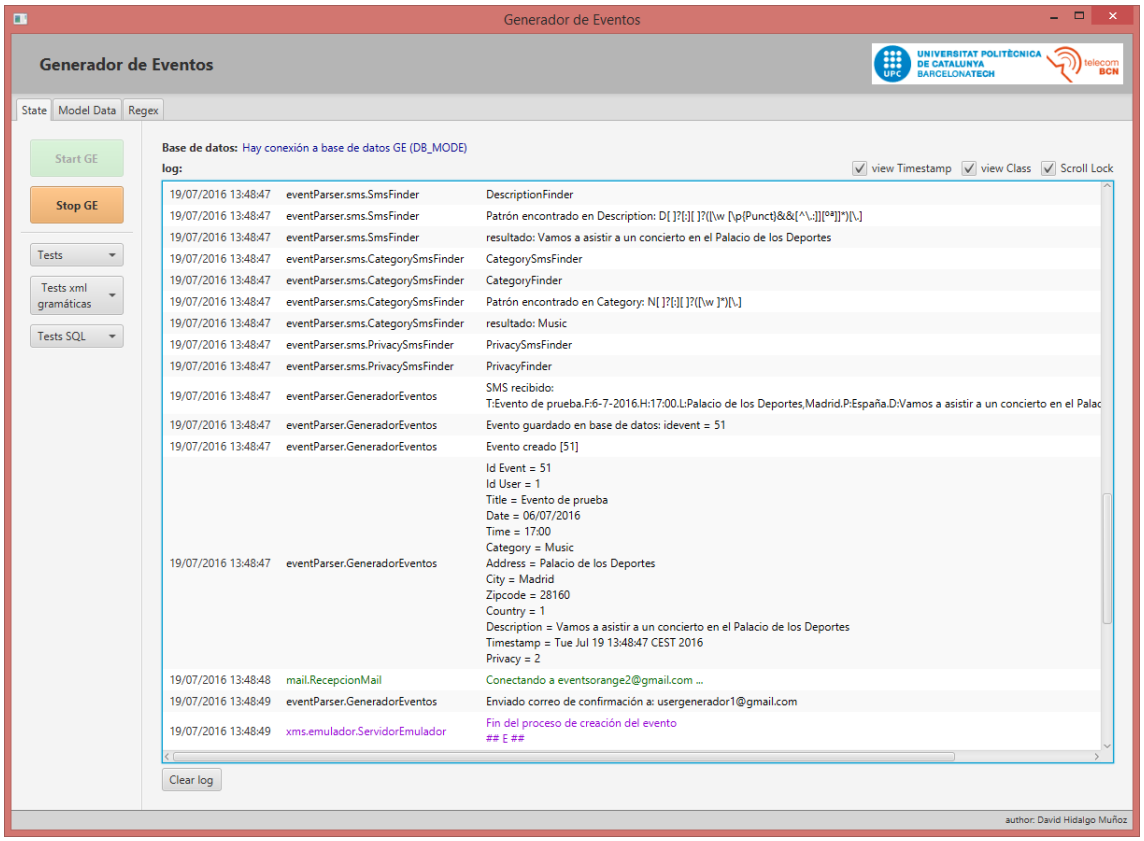


Figura D.3: Detalle de creación de un evento




Generador de Eventos
Generador de Eventos
UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH
telecom BCN
State
Model Data
Regex
Start GE
Stop GE
Tests
Tests xml gramáticas
Tests SQL
Base de datos: Hay conexión a base de datos GE (DB_MODE)
log:
view Timestamp
view Class
Scroll Lock
19/07/2016 13:48:47 eventParser.sms.SmsFinder TitleFinder
19/07/2016 13:48:47 eventParser.sms.SmsFinder Patrón encontrado en Title: T[ ]?[:][ ]?([\w [\p{Punct}&&[^\.:]]]*)[\.]
19/07/2016 13:48:47 eventParser.sms.SmsFinder resultado: Evento de prueba
19/07/2016 13:48:47 eventParser.sms.DateSmsFinder DateSmsFinder
19/07/2016 13:48:47 eventParser.sms.DateSmsFinder DateFinder
19/07/2016 13:48:47 eventParser.sms.DateSmsFinder Patrón encontrado en Date: F[ ]?[:][ ]?(\d{1,2}[-\\ /_]\d{1,2}[-\\ /_]\d{4})[ ]?[\.]?
19/07/2016 13:48:47 eventParser.sms.DateSmsFinder resultado: 6-7-2016
19/07/2016 13:48:47 eventParser.sms.SmsFinder TimeFinder
19/07/2016 13:48:47 eventParser.sms.SmsFinder Patrón encontrado en Time: H[ ]?[:][ ]?(\d{1,2}[:]\d\d)[ ]?[\.]?
19/07/2016 13:48:47 eventParser.sms.SmsFinder resultado: 17:00
19/07/2016 13:48:47 eventParser.sms.SmsFinder AddressFinder
19/07/2016 13:48:47 eventParser.sms.SmsFinder Patrón encontrado en Address: L[ ]?[:][ ]?([\w [\p{Punct}&&[^\.:]]]*),[\w]*[\.]
19/07/2016 13:48:47 eventParser.sms.SmsFinder resultado: Palacio de los Deportes
19/07/2016 13:48:47 eventParser.sms.CitySmsFinder CitySmsFinder
19/07/2016 13:48:47 eventParser.sms.CitySmsFinder CityFinder
19/07/2016 13:48:47 eventParser.sms.CitySmsFinder Patrón encontrado en City: L[ ]?[:][ ]?[\w [\p{Punct}&&[^\.:]]]*,([\w]*)[\.]
19/07/2016 13:48:47 eventParser.sms.CitySmsFinder resultado: Madrid
19/07/2016 13:48:47 eventParser.sms.ZipCodeSmsFinder ZipCodeSmsFinder
19/07/2016 13:48:47 eventParser.sms.ZipCodeSmsFinder ZipCodeFinder
19/07/2016 13:48:47 eventParser.sms.CountrySmsFinder CountrySmsFinder
19/07/2016 13:48:47 eventParser.sms.CountrySmsFinder CountryFinder
19/07/2016 13:48:47 eventParser.sms.CountrySmsFinder Patrón encontrado en Country: P[ ]?[:][ ]?([\w]*)[\.]
19/07/2016 13:48:47 eventParser.sms.CountrySmsFinder resultado: 1
19/07/2016 13:48:47 eventParser.sms.SmsFinder DescriptionFinder
Clear log
author: David Hidalgo Muñoz


Generador de Eventos
Generador de Eventos
UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH
telecom BCN
State
Model Data
Regex
Start GE
Stop GE
Tests
Tests xml gramáticas
Tests SQL
Base de datos: Hay conexión a base de datos GE (DB_MODE)
log:
view Timestamp
view Class
Scroll Lock
19/07/2016 13:48:47 eventParser.sms.SmsFinder DescriptionFinder
19/07/2016 13:48:47 eventParser.sms.SmsFinder Patrón encontrado en Description: D[ ]?[:][ ]?([\w [\p{Punct}&&[^\.:]]]*)[\.]
19/07/2016 13:48:47 eventParser.sms.SmsFinder resultado: Vamos a asistir a un concierto en el Palacio de los Deportes
19/07/2016 13:48:47 eventParser.sms.CategorySmsFinder CategorySmsFinder
19/07/2016 13:48:47 eventParser.sms.CategorySmsFinder CategoryFinder
19/07/2016 13:48:47 eventParser.sms.CategorySmsFinder Patrón encontrado en Category: N[ ]?[:][ ]?([\w]*)[\.]
19/07/2016 13:48:47 eventParser.sms.CategorySmsFinder resultado: Music
19/07/2016 13:48:47 eventParser.sms.PrivacySmsFinder PrivacySmsFinder
19/07/2016 13:48:47 eventParser.sms.PrivacySmsFinder PrivacyFinder
19/07/2016 13:48:47 eventParser.GeneradorEventos SMS recibido:
T:Evento de prueba.F:6-7-2016.H:17:00.L:Palacio de los Deportes,Madrid.P:España.D:Vamos a asistir a un concierto en el Palac
19/07/2016 13:48:47 eventParser.GeneradorEventos Evento guardado en base de datos: idevent = 51
19/07/2016 13:48:47 eventParser.GeneradorEventos Evento creado [51]
19/07/2016 13:48:47 eventParser.GeneradorEventos Id Event = 51
Id User = 1
Title = Evento de prueba
Date = 06/07/2016
Time = 17:00
Category = Music
Address = Palacio de los Deportes
City = Madrid
Zipcode = 28160
Country = 1
Description = Vamos a asistir a un concierto en el Palacio de los Deportes
Timestamp = Tue Jul 19 13:48:47 CEST 2016
Privacy = 2
19/07/2016 13:48:48 mail.RecepcionMail Conectando a eventsorange2@gmail.com ...
19/07/2016 13:48:49 eventParser.GeneradorEventos Enviado correo de confirmación a: usergenerador1@gmail.com
19/07/2016 13:48:49 xms.emulador.ServidorEmulador Fin del proceso de creación del evento
F
Clear log
author: David Hidalgo Muñoz

E. Pruebas adicionales

Este apéndice contiene pruebas complementarias realizadas para la creación de eventos.

E.1. SMS

En este apartado se evalúa el comportamiento en la creación de eventos a partir de SMSs con diferentes casuísicas de cumplimentación de campos.

E.1.1. Mensaje vacío

Mensaje:
" " (al no permitirse el envío de mensaje vacío por el Emulador, se envía un espacio)

Evento generado:
Id Event = 71
Id User = 2
Title = (No Title)
Date = 19/07/2016
Time = null
Category = Other
Address = null
City = null
Zipcode = null
Country = null
Description = null
Timestamp = Tue Jul 19 18:37:15 CEST 2016
Privacy = 2

Comentario:
Para un mensaje vacío se cumplimentarán los campos mínimos necesarios para poder crear un evento. Por defecto se introduce la fecha del envío y un título informativo por defecto "(No Title)". La categoría por defecto si no se encuentra ninguna es "Other" y la privacidad por defecto de todo evento a no ser que se especifique lo contrario es la 2 (evento privado).

E.1.2. SMS con algunos campos rellenos

Realizando sucesivas pruebas con diferentes mensajes con varios campos cumplimentados observamos que el funcionamiento del rellenado de los campos del evento es el correcto. Incluso dejando algunos campos vacíos se rellenan automáticamente utilizando las funciones de búsqueda en base de datos. Como por ejemplo:

Mensaje:
"T:Viaje a Canarias.F:21-7-2016.H:17:00.L:Aeropuerto,Madrid"

Evento generado:
Id Event = 73
Id User = 2
Title = Viaje a Canarias
Date = 21/07/2016
Time = 17:00
Category = Travel
Address = Aeropuerto
City = Madrid
Zipcode = 28160
Country = España
Description = null
Timestamp = Tue Jul 19 18:46:39 CEST 2016
Privacy = 2

Comentario:
En este mensaje de prueba ni la categoría, ni el país, ni el código postal estaban especificados. La categoría "Travel" o viaje ha sido puesta por defecto al detectar en el mensaje la palabra clave "Viaje". A partir de la ciudad "Madrid" se ha determinado consultando en base de datos su país y su código postal.

E.1.3. Diferentes idiomas

En el Emulador podemos seleccionar directamente mediante el desplegable "Language" qué idioma vamos a utilizar para determinar las plantillas del idioma con el que queremos que se cree el evento. Observamos que enviando un mismo mensaje de prueba en ESP o ENG el Generador de Eventos crea el evento correctamente a partir de las etiquetas enviadas.

Coincidencias para un mensaje enviado en español:
idDic 0[ENG] : 2
idDic 1[ESP] : 3
Text Language: ESP

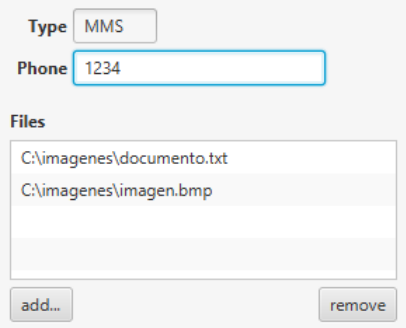
Coincidencias para un mensaje enviado en inglés:
idDic 0[ENG] : 3
idDic 1[ESP] : 2
Text Language: ENG

Nos podemos encontrar que si el número de etiquetas utilizadas hace que se obtenga el mismo número de resultados encontrados se elegirá un idioma al azar en función del sistema de almacenamiento de los diccionarios, puesto que siempre debe elegirse uno. En este caso es muy probable que el evento se genere incorrectamente si se escoge el idioma que no toca. Esta problemática podría solventarse enviando un campo adicional con el idioma. Ese campo sería rellenado por la aplicación que generase el SMS. Si fuese el propio usuario el que lo crease, no tendría sentido que estuviese siempre escribiendo el idioma del mensaje, por ese motivo se ha descartado esa idea.

E.1.4. Archivos multimedia

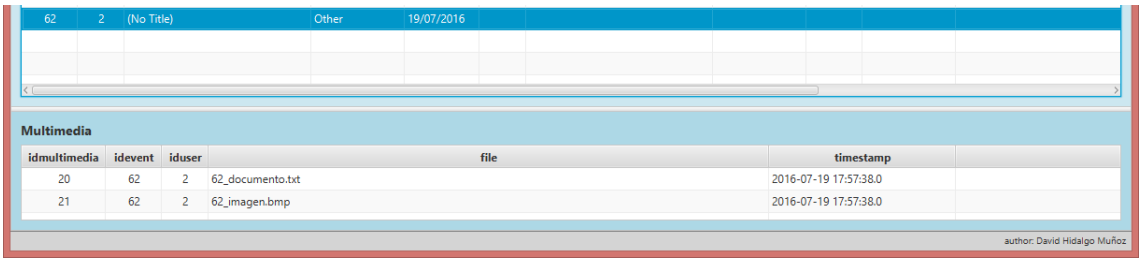
Si a un SMS le añadimos archivos en la sección “Files” irán apareciendo en el listado y observaremos que el tipo de mensaje cambia a MMS.

Figura E.1: Prueba SMS – envío de archivos multimedia.



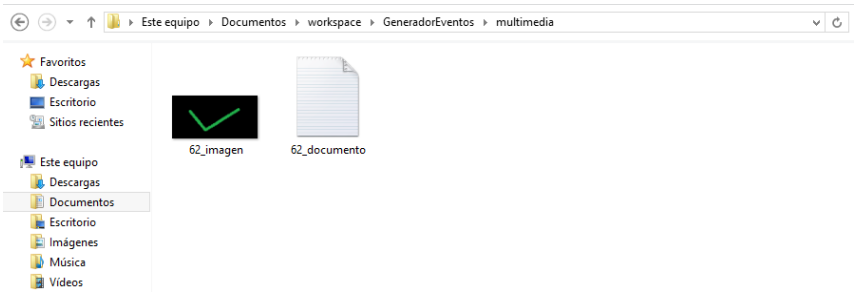
En este caso tras la creación del evento asociado (para esta prueba el id = 62 con un evento sin cumplimentar), tendremos que en la tabla multimedia se han creado los dos registros asociados a los dos archivos multimedia enviados.

Figura E.2: Prueba SMS – tabla de base de datos multimedia



Y además han sido almacenados en la ruta de guardado de archivos multimedia del Generador de Eventos.

Figura E.3: Prueba SMS – almacenamiento de archivos multimedia



E.1.5. Casuísticas especiales

Esta sección engloba las siguientes pruebas:

- Cambios en el formato de número de teléfono.

Se han utilizado formatos diferentes en vez del típico de 9 cifras escrito consecutivamente como 6xxxxxxxx. Los formatos intercalando espacios 6xx xx xx xx son aceptados y tratados como el mismo número de teléfono. No ocurre lo mismo si añadimos un +34 delante, porque aunque las operaciones de limpieza de número de teléfono ignoran todo aquello que no es un dígito, para la aplicación son diferentes los números 346xxxxxxxx y 6xxxxxxxxx. Es responsabilidad de una aplicación de nivel superior que utilize el módulo de generación de eventos el determinar cómo deben guardarse los números de teléfono dentro de la aplicación.

También observamos que el Generador de Eventos por su diseño actual permite números de teléfono de longitudes diferentes a 9 cifras puesto que su comprobación de número de teléfono se reduce a comprobar que existe en base de datos.

- Creación de un evento con un número que no está registrado en base de datos.

Cuando introducimos un número de teléfono incorrecto, el usuario no es encontrado en la base de datos y se aborta el proceso.

- Intento de generación de un evento sin estar arrancado el Generador de Eventos.

Al no poder realizar conexión con el servidor TCP del Generador de Eventos el Emulador de SMS nos informa con un mensaje “Connection refused”.

E.2. E-mail

En este apartado se evalúa el comportamiento en la creación de eventos a partir de e-mails con diferentes casuísicas de cumplimentación de campos.

E.2.1. E-mail vacío

Mensaje:
Asunto: “”
“”

Id Event = 74
Id User = 1
Title = Other Event (Title not found)
Date = 19/07/2016
Time = null
Category = Other
Address = null
City = null
Zipcode = null
Country = null
Description = AutoGenerated Event - (Mail Subject: null)
Timestamp = Tue Jul 19 19:41:45 CEST 2016
Privacy = 2

Comentario:
En este caso aparte de cumplimentarse con un título pro defecto “Other Event (Title not found)” también se cumplimenta una descripción por defecto a “AutoGenerated Event - (Mail Subject: null)” ya que los eventos creados a partir del parseador de e-mails intentan enriquecer la información del campo descripción.

E.2.2. E-mail con plantilla en inglés

Mensaje:
Asunto: Viaje a Futuroscope
Title: Travel to Futuroscope
Date: 31/12/16
Location: Parc d'attractions du Futuroscope
City: Poitiers
Country: Paris
Time: 10:00
Description: We are traveling.

Evento generado:
Id Event = 98
Id User = 1
Title = Travel to Futuroscope
Date = 31/12/2016
Time = 10:00

Category = Travel
Address = Parc d'attractions du Futuroscope
City = Poitiers
Zipcode = null
Country = Paris
Description = We are traveling.
Timestamp = Wed Jul 20 00:35:59 CEST 2016
Privacy = 2

Comentario:
En este ejemplo vemos que la mayoría de palabras son en inglés y son seleccionados para aplicarse los patrones en inglés. En este caso el idioma del evento es "ENG" (english). Este campo no se muestra en el log principal, pero si es almacenado en base de datos en el campo `language`.

E.2.3. E-mail redactado con privacidad pública y datos adjuntos

Mensaje:
Asunto: Fiesta en Benidorm

Título: Low Festival
Descripción: El próximo 29 de julio comienza el festival de Benidorm.
Población: Benidorm, España
C.P.: 03501

privacidad: público

Evento generado:
Id Event = 93
Id User = 1
Title = Low Festival
Date = 29/07/2016
Time = null
Category = Festivals
Address = null
City = Benidorm
Zipcode = 03501
Country = España
Description = AutoGenerated Event - (Mail Subject: Fiesta en Benidorm)
Timestamp = Tue Jul 19 23:23:46 CEST 2016
Privacy = 0

Comentario:
Para este correo al final del proceso de creación se nos indica en el log:

Buscando Ficheros multimedia en el mail...
Parte encontrada con tipo mime: IMAGE/PNG; name=image.png

Es hallada la imagen incrustada en el correo, almacenada como archivo multimedia asociado al evento y guardada en la carpeta de archivos multimedia. También observamos que la hora al no ser indicada no se cumplimenta y la privacidad ha cambiado del valor por defecto 2 (privado) a 0 (público). En este correo observamos que la descripción no ha sido hallada, eso es debido a que el patrón y la expresión regular está creada para que la descripción esté al final del documento y pueda soportar la captura de varias cadenas de caracteres acabadas en punto. Se ha diseñado así para no obtener como descripción en los correos reenviados todo ese texto que incorporan las empresas al final de los correos normalmente en letra pequeña. Para hallar la descripción es necesario también un doble salto de línea, específicado así en las expresiones regulres. Nos encontramos que la descripción junto con el código postal tienen limitaciones, puesto que en unos casos si se encuentra y en otros no. Esto es normal que pase, porque si las reglas son las mismas y los contenidos muy diferentes, es imposible que aplicando las mismas reglas se obtengan los resultados deseados en todo tipo de contenidos.

E.2.4. E-mail redactado ambiguo

Mensaje:
Asunto: Correo ambiguo
En el Palacio de los Deportes vamos a asistir a un concierto. Las entradas las compramos el día 12-6 pero el concierto es el día 18-6. Los que vengan desde Barcelona nos veremos el día de antes.

Evento generado:
Id Event = 95
Id User = 1
Title = Sport Event (Title not found)
Date = 12/06/2016
Time = null
Category = Sport
Address = null
City = Barcelona
Zipcode = 08830
Country = España
Description = AutoGenerated Event - (Mail Subject: Correo ambiguo)
Timestamp = Tue Jul 19 23:43:46 CEST 2016
Privacy = 2

Comentario:
Este es un claro ejemplo de malinterpretación de la información porque el Generador de Eventos hace todo lo posible por encontrar campos pero carece de inteligencia. Leyendo el mensaje vemos que no se utiliza ninguna referencia para declarar campos, y es en estos casos cuando el proceso de parseado hará todo lo posible para encontrar y rellenar los campos en vez de dejarlos vacíos. Leyendo el mensaje observamos que la palabra “deporte” aparece, pero también la palabra “concierto”. Como el patrón de la categoría deporte se busca antes que el de concierto la palabra es hallada y cumplimentada la categoría como “Sport”. Para la fecha pasa algo similar, es cogida la primera fecha que aparece en el correo. En el caso de la ciudad, al no haber especificado ninguna en el correo se buscan todas las que hay en base de datos hallándose Barcelona, cumplimentándose incorrectamente ya que no es el lugar del acontecimiento.

E.2.5. E-mail reenviado de entradas a un concierto

Mensaje:

> Apreciado Cliente
> Te informamos que tu compra se ha realizado con éxito. La
> referencia de la misma es: 36725311301
> A partir de este momento, y cuando te sea más cómodo, puedes
> pasar a recoger tus entradas en cualquiera de nuestros puntos de recogida.
> Te recordamos que puedes consultar el estado de tu compra, así
> como la relación de nuestros puntos de recogida en nuestra sección
> Seguimiento de compras. En caso de cualquier duda puedes enviar un
> e-mail a <a
> href=mailto:"central@ticktackticket.com">central@ticktackticket.com >
>
> INFORMACIÓN DEL EVENTO:
>
> Evento :Marea
> Fecha :08/09/16
> Local :Palacio de Deportes Comunidad de Madrid
> Hora de apertura :20:00
>
> HORARIO:
>
> Evento Hora
> ---
> Artista Invitado 21:00
> Marea 22:30
>
> ENTRADAS COMPRADAS:
> ------ Precio ----- ----
> Subtotal -----
> Entradas Euros Cant.
> Euros
> ---
> ------
> ----------
> Entrada General Pista 27,20 3
> 81,60
> ---
> ------
> ----------
> TOTAL: 3
> 81,60
>
> CONDICIONES DE ENTREGA:
>
> Establecimientos autorizados (hasta 72 horas* antes del evento sin
> incluir fines de semana ni festivos) y (desde 1 hora * antes del
> evento) Taquillas del local
>
> * Tick Tack Ticket se reserva el derecho a modificar los horarios sin
> previo aviso.
>
> La/s entrada/s se entregará/n exclusivamente a la persona que reúna
> las siguientes condiciones:
> 1) Mostrar la referencia de la compra.
> 2) Exhibir la tarjeta utilizada en la compra, así como documento
> acreditativo de la titularidad de la misma (DNI o pasaporte).

> 3) Firmar el recibo que acredite la entrega de la/s entrada/s.
>
> IMPORTANTE
> Para que otra persona distinta al titular de la tarjeta pueda recoger
> las entradas, debe presentar y entregar la siguiente documentación
> (tanto en tiendas autorizadas como en taquilla): el número de
> referencia de la compra; una fotocopia del DNI o pasaporte de la
> persona que recoge las entradas; una hoja que incluya una fotocopia de
> la tarjeta con la que se ha efectuado el pago, otra del DNI o
> pasaporte del titular y una autorización escrita firmada por titular de la tarjeta.
>
>
> Gracias por confiar en Tick Tack Ticket.

Evento generado:
Id Event = 99
Id User = 1
Title = Marea
Date = 08/09/2016
Time = 20:00
Category = Sport
Address = Palacio de Deportes Comunidad de Madrid
City = Madrid
Zipcode = 28160
Country = España
Description = AutoGenerated Event - (Mail Subject: Tick Tack Ticket Marea)
Timestamp = Wed Jul 20 00:47:46 CEST 2016
Privacy = 2

Comentario:
En este correo de prueba observamos que los patrones que han hallado algo son:

Patrón encontrado en Title: EVENTO[ ]?#?[:=][ ]?([\w[\p{Punct}&&[^\.:#=,;]]][\w[\p{Punct}&&[^\.:#=,;]]])[\.,#;[\s&&[^]]]*
Patrón encontrado en Date: FECHA[ ]?#?[:=][ ]?([012]?\d[-\\ /_]0?\d[-\\ /_]\d{2})\D
Patrón encontrado en Time: \D((?:2[0123])|(?:[01]?\d))[:]([012345]\d)\D
Patrón encontrado en Address: LOCAL[ ]?#?[:=][ ]?([\p{Alpha}[\p{Punct}&&[^\.:#=;]]][\w[ºª][\p{Punct}&&[^\.:#=;]]])[\.#;[\s&&[^]]]*

Observamos que el título es hallado mediante la referencia “Evento”, el día y la hora vuelven a encontrarse sin problemas, en este caso la primera aparición del tiempo es la que se queda almacenada siendo esta la apertura del recinto. En este caso la categoría se cumplimenta erróneamente como “Sport” por hallarse la palabra “Deportes” en el correo. Aunque tampoco observamos que haya ninguna otra referencia a que el evento sea de tipo “Music”. La dirección se halla correctamente y encontramos la ciudad contenida en la dirección. El código postal y país vuelven a cumplimentarse automáticamente a partir de la dirección.

E.2.6. E-mail reenviado de alquiler de películas

Mensaje:

Para este correo y debido al formato se inserta como imagen.

Video Club Marilyn

Title: The Bucket List

User: David
User e-mail: usergenerador1@gmail.com
Rent date:2016/6/3 at 17:26
Return date:2016/6/5 at 17:26
Total cost:
6.25€

A confirmation mail is automatically send to your e-mail account, thanks to use this service.

Studio: Warner Home Video
Directed by: Rob Reiner
Featuring: Jack Nicholson, Morgan Freeman, Sean Hayes, Beverly Todd, Rob Morrow

Features:
Keep Case
Widescreen - 2.40
Audio:
Dolby Surround Stereo - English, French
Dolby Surround 5.1 - English, Spanish
Additional Release Material:
Writing A Bucket List - Screenwriter Justin Zackman
Music Video - Say
DVD Rom Features:
Weblink Access

Description
In THE BUCKET LIST, cancer doesn't discriminate in its choice of victims. It's equally eager in its attacks on kindly sage of a mechanic Carter Chambers (Morgan Freeman) and mean-spirited millionaire Edward Cole (Jack Nicholson). When the unlikely pair shares a room at a hospital, they learn that they both have less than a year to live as a result of the deadly disease. Inspired by the words of a college professor, Carter begins to make a bucket list of things he wants to accomplish before he dies. With Edward's limitless funds at their disposal, the men embark on an adventure that takes them from Egypt to France to Hong Kong, crossing items off their list as death grows closer.

Fans of the leading actors will be glad to see them in familiar territory: Freeman plays a narrator who's nearly godlike in his omniscience, while Nicholson stars as the lovable grump. Both men have won Oscars for similar roles (Nicholson for AS GOOD AS IT GETS, and Freeman for MILLION DOLLAR BABY), so they excel in these parts. Director Rob Reiner's career has jumped between comedy and drama, and despite its serious subject matter, THE BUCKET LIST resides firmly in comedic territory. But fans of weepies such as TERMS OF ENDEARMENT and MY LIFE will find much to like in this film that is sure to require a tissue or two.

Evento generado:

Id Event = 100
Id User = 1
Title = The Bucket List
Date = 16/06/2016
Time = 17:26
Category = Cinema/Video
Address = null
City = null
Zipcode = null
Country = null
Description = David is watching The Bucket List

AutoGenerated Event - (Mail Subject: The Bucket List 3)
Timestamp = Wed Jul 20 01:01:46 CEST 2016
Privacy = 2

Comentario:

Este correo está en inglés y se han utilizado los patrones en inglés al detectar el idioma automáticamente. Los patrones que han hallado información son:

Patrón encontrado en Title: TITLE[ ]?#?[:=][ ]?([\w[\p{Punct}&&[^\.:#=,;]]][\w[\p{Punct}&&[^\.:#=,;]]])[\.,#;[\s&&[^]]]*
Patrón encontrado en Date: RENT DATE[ ]?#?[:=][ ]?(20\d\d[-\\ /_]0?\d[-\\ /_][012]?\d)\D
Patrón encontrado en Time: \D((?:2[0123])|(?:[01]?\d))[:]([012345]\d)\D
Patrón encontrado en Category: \WVIDEO\W

Son hallados el título especificado al inicio del correo, la fecha de alquiler y la hora. También es detectada la palabra “VIDEO” en el buscador de categoría, asignando la categoría “Cinema/Video”. En este caso no hay información relacionada con la dirección. La descripción se ha generado por código con la frase: “David is watching The Bucket List”. Para este correo también es almacenada la imagen de la portada de la película como archivo multimedia.

A modo de resumen, cabe destacar que muchos de los patrones se han ido mejorando añadiendo más expresiones regulares y palabras clave en función de los correos de prueba analizados por la aplicación. En la presente memoria solo se han mostrado una pequeña parte representativa de las características de parseado de correos, pero suficientes para contemplar las circunstancias en las que el algoritmo funciona correctamente y en las que flojea por falta de información explícita.

Aunque no se ha mencionado hasta ahora, si llegase un correo en el que toda la información estuviese contenida en un documento pdf o una imagen sería imposible extraer la información ya que el Generador de Eventos solo analiza correos que llegan en formato de texto, ya sean en texto plano o HTML.