

Programación de plugins en Evolution

Rodrigo Moya

Novell, Inc.
rodrigo@novell.com

1. Introducción

Evolution es un programa de gestión de correo y calendario personal, con funcionalidades que lo hacen ideal para su uso tanto en entornos domésticos como entornos profesionales (al incluir soporte para los servidores de Groupware más usados del mercado, como Microsoft Exchange y Novell Groupwise). **Evolution** inició su camino al ser el desarrollo principal de la empresa **Helix Code** en el momento de su creación, en el año 2.000.

Desde esos primeros instantes, **Evolution** fue usada como terreno de pruebas de muchas de las tecnologías que hoy en día se usan a diario en entornos **GNOME**, ayudando a madurarlas y moldearlas a las necesidades de los desarrolladores de aplicaciones. Una vez completadas las primeras versiones, el desarrollo de **Evolution** se ha ido estabilizando, y, aunque siempre con la idea de ofrecer más y mejores funcionalidades a sus usuarios, en los últimos tiempos, y en especial tras la inclusión oficial de **Evolution** en el escritorio **GNOME**, se ha querido poner más énfasis en facilitar el uso de las librerías que incluye **Evolution** en aplicaciones que deseen compartir los datos que almacena, y la extensión de la aplicación por personas ajenas al equipo de desarrollo de **Evolution**.

Hasta la versión 2.0, **Evolution** tenía una capacidad mínima de extensión, o, en el mejor de los casos, muy compleja, debido al uso directo de **CORBA** para realizar dichas extensiones. Además, había ciertas partes de la aplicación que no podían ser extendidas ni cambiadas sin realizar enormes cambios en el código interno. Por ejemplo, para añadir cuentas en servidores de calendario (sin correo electrónico), había que escribir una extensión para **camel**, la librería de gestión de ¡correo electrónico! que se usa en **Evolution**, por lo que, aparte de ser complejo, no ofrecía posibilidades reales de extensión aparte de las desarrolladas a medida por el equipo de desarrollo de **Evolution**.

La versión 2.2 de **Evolution**, entre otras muchas novedades, incluye un sistema de extensión que favorece la adición de nuevas funcionalidades por medio de la escritura de *plugins*. Este sistema, que ya se ha usado para escribir muchas de las nuevas funcionalidades de **Evolution**, permite añadir, de forma muy sencilla, módulos externos a la aplicación que se cargan en tiempo de ejecución para añadir nuevas funcionalidades y/o modificar el interfaz de la aplicación. Este sistema, denominado *EPlugin*, incluye, de forma general, las siguientes características:

- Mecanismo de invocación independiente del lenguaje de implementación.

- Extensión de partes del interfaz de usuario, así como la captura de notificaciones de los eventos que suceden en la aplicación.
- No requiere cambios en Evolution.
- Requiere un código mínimo para la implementación de las extensiones.
- Versionado de las extensiones.
- Facilidad para ser extendido él mismo.

Otra de las características interesantes de este sistema es su uso para la automatización (*scripting*) de la aplicación, al permitir de una forma sencilla el *enganche* a la aplicación de cualquier script escrito en cualquier lenguaje.

2. Estructura interna

A grosso modo, *EPlugin* consiste en un cargador, que se ocupa de cargar, en el espacio de memoria de Evolution, las distintas extensiones que se encuentren instaladas en el sistema, y que además se ocupa de realizar llamadas a los puntos de entrada de las extensiones cargadas; puntos de entrada, que determinan los distintos interfaces extensibles de la aplicación; contextos, para el paso de parámetros entre la aplicación y las extensiones, y gestores, usados por el corazón del sistema para enlazar entre sí las distintas extensiones. Este sistema está fuertemente inspirado en el sistema de extensión del proyecto Eclipse, aunque, obviamente, tiene una escala mucho menor.

2.1. El cargador

El cargador es parte del corazón del sistema, teniendo como función la carga de las extensiones y la invocación de los puntos de entrada de éstas. Esta carga se hace por medio de librerías enlazadas, en el caso de extensiones escritas en el lenguaje C, de DLLs, en el caso de las escritas con Mono, etc, así como de la lectura de los ficheros XML que definen las extensiones, que permite a la aplicación el conocimiento de todas ellas antes de cargarlas físicamente. La carga física de las extensiones se realiza en el momento en que se vaya a realizar una llamada al punto de entrada de ésta, nunca antes, lo que reduce enormemente el gasto en memoria y recursos.

Este sistema de carga es, a su vez, totalmente extensible, de forma que se pueden añadir nuevos cargadores, así como nuevos puntos de entrada, de la misma forma que se añade cualquier otra extensión. Así, por ejemplo, el cargador por defecto está orientado a la carga de extensiones en lenguaje C, o sea, librerías compartidas, pero esta extensibilidad permite, por ejemplo, añadir cargadores para otros lenguajes, tales como Perl, Python, Mono, etc.

2.2. Puntos de entrada

Los puntos de entrada son los métodos de acceso a las extensiones (o, lo que es lo mismo, las funciones exportadas por la extensión). El núcleo de *EPlugin*

se encarga, una vez cargadas las extensiones por el cargador, de establecer la relación entre los puntos de entrada definidos en los ficheros XML descriptivos de cada extensión y los puntos de entrada reales en el módulo ejecutable (librería compartida, DLL, etc).

2.3. Gestores

Los gestores ofrecen al núcleo del sistema los puntos de extensión del sistema. Éstos están definidos por *Evolution*, e incluyen cosas como el menú principal y los menús emergentes de la aplicación, distintos eventos, tales como *correo recibido*, *migración de una versión a otra iniciada*, etc. Estos gestores tienen asociado un identificador, que es el que usan los ficheros XML de las extensiones para especificar para qué punto de entrada es la extensión en cuestión. Así, el cargador usa dicho identificador para saber qué extensiones deben ser invocadas para cada punto de entrada.

Los gestores, además, definen lo que se conoce como *targets*, que son las estructuras de datos usados para enviar información desde la aplicación a las extensiones y viceversa. Estos *targets*, en la mayor parte de los casos, son a su vez extensibles, lo que permite la adición de nuevos parámetros sin demasiada dificultad.

3. Implementación de extensiones

El primer paso para la implementación de extensiones es la escritura del fichero XML que describe la extensión. Este fichero contiene una o más definiciones de extensiones, y tiene el siguiente formato:

```
<?xml version='1.0'>
<e-plugin-list>
  <e-plugin id='id_unico'
            type='tipo_de_cargador'
            location='/camino/a/libreria/o/dll'
            domain='dominio_de_traduccion' ?
            name='nombre'
            ...>
    <description>Descripción de la extensión</description> ?
    <hook class='punto de extensión'
          ...>
      ...
    </hook> +
  </e-plugin> +
</e-plugin-list>
```

El identificador (*id*) es una cadena que define a la extensión. Se usa para esta cadena la convención usado en objetos Java, así, para escribir una extensión

para el calendario que permite acceso a información meteorológica, el nombre a usar sería *org.gnome.evolution.calendar.weather*. El tipo (*type*) indica el tipo de cargador a usar (a día de hoy, sólo *shlib*, para extensiones en C, y *mono*, para extensiones en Mono, están soportados). Junto al tipo, es preciso indicar la localización (*location*) de la librería que implementa la extensión. El dominio de traducción (*domain*) es el nombre a usar para la obtención de traducción de mensajes de la extensión a través de *gettext*. El nombre (*name*) es una cadena que describe brevemente la extensión, y es de libre uso. La descripción (*description*), por su parte, permite añadir una descripción más larga de la extensión.

Por último, la parte más importante de todas, la que define los puntos de extensión usados por la extensión. En ella se pueden incluir una o más entradas que definen los partes de la aplicación a las que se aplica la extensión y los puntos de entrada que deberán ser invocados por *EPlugin*. Los puntos de entrada se indican mediante el nombre de la función (en el caso de las extensiones en C) que deberá ser llamada. Dicha función, en la librería, tiene la siguiente forma general:

```
void *funcion (EPlugin *ep, void *data);
```

donde *ep* es un objeto que representa la extensión en uso, y *data* contiene los datos de contexto usados para comunicar a la aplicación con la extensión. Su forma depende del tipo de extensión que se esté implementando. Asimismo, el valor de retorno de la función depende también del tipo de extensión, pero en todos los casos es usado para devolver información desde la extensión a la aplicación una vez concluidas las operaciones realizadas por la extensión.

A la hora de escribir extensiones, es necesario tener en cuenta todos los puntos de extensión disponibles. Así, existen los siguientes:

- Menús emergentes: este punto de extensión permite añadir, a todos los menús emergentes de **Evolution**, nuevas opciones de menú, con sus correspondientes acciones.
- Menú principal: al igual que con los menús emergentes, las extensiones pueden añadir nuevas opciones al menú principal de la aplicación.
- Páginas de configuración: la ventana de preferencias de **Evolution** puede también ser extendida. Así, se pueden añadir nuevas páginas de configuración o añadir contenido a las páginas ya existentes.
- Eventos: las distintas partes de **Evolution** envían notificaciones cuando sucesos tales como el envío de un mensaje, la migración de datos, etc. ocurran. Las extensiones pueden usar este sistema de notificación para ejecutar determinadas acciones en el momento en que esos eventos se produzcan.
- Formateador de correo: este punto de extensión permite mostrar, a través de extensiones, partes de un correo electrónico basado en su tipo MIME. Así, se pueden añadir extensiones para mostrar, desde el visor de correo de **Evolution**, archivos de **OpenOffice.org**, **Gnumeric**, etc.

Cada uno de los componentes de **Evolution** (Correo, Calendario, Tareas, Contactos) define su propia nomenclatura para los eventos de cada uno de sus puntos de extensión, así como los parámetros que se usan para los puntos de entrada de las extensiones, aunque todos se basan en el mismo concepto comentado anteriormente de usar estructuras extensibles para pasar los argumentos.

4. Conclusiones

EPlugin ofrece un sistema sencillo de extensión de **Evolution**, del que ya están aprovechándose los propios desarrolladores de la aplicación, pues algunas de las nuevas funcionalidades (así como otras antiguas) en la versión 2.2 han sido implementadas en forma de extensiones. Ejemplos claros de esto son: la opción de salvar un calendario a disco (*save-calendar*), la conversión de correos en tareas (*mail2task*), o la extensión que muestra nuevos campos en la ventana de creación de calendarios cuando se selecciona un tipo distinto al local (*http*:, *weather*:, *groupwise*:, *exchange*:).

En cuanto a los desarrollos futuros, es de suponer que ahora aparezcan extensiones que permitan a **Evolution** el acceso a otro tipo de servidores (la empresa **Scalix**, por ejemplo, está desarrollando tal extensión para su servidor, o el desarrollo de **Evolution-Ogo**, que permite el acceso a servidores **OpenGroupware.org**), pues *EPlugin* unido a la facilidad con la que ahora se pueden escribir *backends* de calendario, tareas, correo o agenda que accedan a otro tipo de servidores, ofrece el marco ideal para que terceras partes puedan usar **Evolution** para sus desarrollos relacionados con la colaboración y gestión de información personal. En este caso, *EPlugin* ofrece todo lo necesario para que estas extensiones adecúen el interfaz gráfico de la aplicación para permitir, por ejemplo, el acceso a opciones específicas de estos servidores.

Otros usos de este sistema permitirían que cualquier persona que quisiera una determinada funcionalidad no tuviera que pedirla a los desarrolladores de **Evolution**. Simplemente tendría que añadir su extensión que haga lo que desea, y publicarla por separado del resto de **Evolution**.

EPlugin es pues, junto con las mejoradas APIs, la bandera del nuevo **Evolution**, un programa ahora amigable para los desarrolladores, no sólo para los usuarios.

Referencias

1. Evolution: <http://www.gnome.org/projects/evolution>
2. Gnome: <http://www.gnome.org>
3. Mono: <http://www.go-mono.com>
4. Manual de EPlugin (en inglés):
<http://www.gnome.org/projects/evolution/developer-doc/eplugin/>
5. Información de Evolution para desarrolladores:
<http://www.gnome.org/projects/evolution/developer.shtml>
6. Wiki de Evolution: <http://go-evolution.org>