Trabajo Fin de Grado

Desarrollo de módulo de plantillas dinámicas para el sistema de gestión de incidencias open-source OTRS

Autor/es

Diego Malo Mateo

Director/es

Fernando Naranjo Palomino

Escuela Universitaria Politécnica de Teruel
2018

Resumen

El proyecto busca la creación de un módulo para la plataforma de código abierto Open-source Ticket Request System (OTRS) que modifique la pantalla de creación de tickets para que los clientes dispongan de campos personalizados en función de la naturaleza del servicio al que estará enviando su ticket.

Tabla de contenido

Índice de imágenes

Índice de tablas

1. Introducción

1.1. Motivación

OTRS (Open-Ticket Request System) es un sistema de código abierto que se utiliza en diversas instituciones para facilitar la comunicación de los clientes con los departamentos oportunos dentro de la institución para obtener una respuesta rápida y adaptada al cliente.

En la Universidad de Zaragoza se utiliza para la herramienta en línea ayudICa (ayudica.unizar.es). En la herramienta se requiere la implantación de un módulo que añada a la interfaz de creación de un ticket un campo desplegable de plantillas según el servicio elegido. Para cada plantilla, se mostrarán unos campos específicos.

A nivel personal este proyecto me otorga nuevos conocimientos sobre el lenguaje de programación Perl, que no ha sido estudiado durante el grado, así como una oportunidad de aplicar los conocimientos obtenidos sobre dicho grado en bases de datos, aplicaciones web y adaptación a un nuevo entorno.

1.2. Objetivos

Los objetivos del proyecto serán los expuestos a continuación:

1. Creación de un módulo para el sistema OTRS que muestre a los usuarios clientes en el formulario de creación de nuevos tickets distintos campos en función de la consulta que deseen realizar.

2. Añadir con el módulo una interfaz con la que los usuarios administradores puedan decidir los grupos de campos que se mostrarán a los usuarios clientes.

2. Tecnologías utilizadas

En este apartado se nombrarán y conocerán todas las tecnologías que han sido utilizadas para la elaboración de este módulo, así como se explicará su aportación al proyecto.

2.1. Perl

Perl [1] es el lenguaje de programación principal sobre el que se ha desarrollado el módulo. Iniciado en 1987 por Larry Wall con la intención de crear un lenguaje de scripting para Unix, Perl ha terminado evolucionando en un lenguaje interpretado multiparadigma de alto nivel para el uso general dividido ahora en dos diferentes lenguajes que avanzan en paralelo, Perl 5 y Perl 6. El sistema OTRS utiliza Perl 5, así que desde ahora se referirá a este como Perl y se dejará apartado Perl 6.

En la Tabla 1 se puede observar una breve vista a la historia de Perl hasta su versión 5.

Versión	*Fecha*	*Breve descripción*
Perl1	1987	Primera versión comercial de Perl
Perl2	1988	Mejora en el motor de expresiones regulares
Perl3	1989	Se añade soporte a los datos binarios
Perl4	1991	Primera versión con documentación oficial. Primera versión de *"Programming Perl"*, también conocido como "Camel Book".
Perl5	1994	Nuevo intérprete y nuevas funciones al lenguaje, como objetos, módulos o variables locales. La versión estable actual es la 5.26.2 [2]

Tabla 1. Versiones de Perl

Perl es un lenguaje de programación multiplataforma, y el estar el sistema OTRS creado sobre una base de Perl le permite tener cobertura multiplataforma.

2.2. JavaScript

JavaScript (JS) [3] es un lenguaje de programación multiparadigma, principalmente utilizado junto con HTML5 y CSS para la creación de contenido para la Web. El uso que ha tenido JavaScript en el desarrollo del módulo ha sido principalmente auxiliar, principalmente para activar interacciones con el contenido de las plantillas.

El uso de JavaScript en OTRS le permite la creación de una interfaz web intuitiva tanto para los clientes como para los agentes, fácil de utilizar y adaptable.

2.3. HTML

El lenguaje de marcado HyperText Markup Language [4] es el lenguaje estandarizado para la creación de páginas web. En el quedan definidas una estructura básica y un código, llamado código HTML, con el que es posible la definición de todos los posibles elementos de una página web, desde texto hasta imágenes y videos entre otros. Esto se logra separando los elementos externos como las imágenes, los videos o los scripts a la página y haciendo referencia a las ubicaciones de estos en el código.

Al ser el estándar todos los navegadores de internet actuales son capaces de interpretarlo, con lo que el sistema OTRS es accesible para todo el mundo con independencia del sistema operativo o navegador de internet que utilicen. Además, el avance en el diseño web para dispositivos móviles añade accesibilidad al sistema al poder ser accesible desde un teléfono móvil actual o una tableta. Actualmente la W3C (World Wide Web Consortium) que es la entidad que mantiene el estándar de HTML recomienda el uso de la versión más actual, que es HTML5.

2.4. MySQL

MySQL [5] es un sistema gestor de bases de datos de código abierto y es una de las soluciones de bases de datos que propone OTRS. En este módulo se han creado cuatro tablas que recogen las nuevas plantillas y sus relaciones con los campos dinámicos y los servicios del sistema.

La elección de MySQL no ha sido azarosa, se ha seleccionado por ser el sistema gestor de bases de datos que actualmente se emplea en el sistema OTRS instalado.

2.5. Ajax

Ajax [6], el acrónimo de Asynchronous JavaScript And XML, es un conjunto de técnicas y tecnologías web usadas en aplicaciones web. Gracias a AJAX, las aplicaciones web son capaces de cambiar el contenido sin necesidad de recargar la página.

En este módulo se usa para cambiar los campos dinámicos en vivo sin necesidad de recargar la página. Por ejemplo, permite cambiar el estado de visibilidad de los campos dinámicos entre visibles y no visibles.

3. Estado del arte

Antes de entrar de lleno en el desarrollo del módulo hay que descubrir el entorno de este, y para ello debemos conocer que es OTRS, cómo se desarrollan módulos en OTRS y si existen otras soluciones alternativas que aporten un resultado similar al que se pretende con este módulo.

3.1. OTRS

En primer lugar, que es OTRS. El producto Open-source Ticket Request System [7] es un sistema de seguimiento de incidencias con tickets. Se trata de un producto que aporta a una empresa u organización un medio con el cual los clientes pueden ponerse en contacto con los empleados para realizar cualquier consulta que consideren apropiada.

OTRS está programado en Perl, usando JavaScript para crear interfaces web adecuadas para un usuario medio y una base de datos de MySQL, aunque en las últimas revisiones se ha añadido la posibilidad de trabajar con bases de datos de PostgreSQL, Oracle, DB2 y MS SQL server.

El producto, como indican sus siglas, es de código abierto y se puede descargar de forma gratuita en su página web.

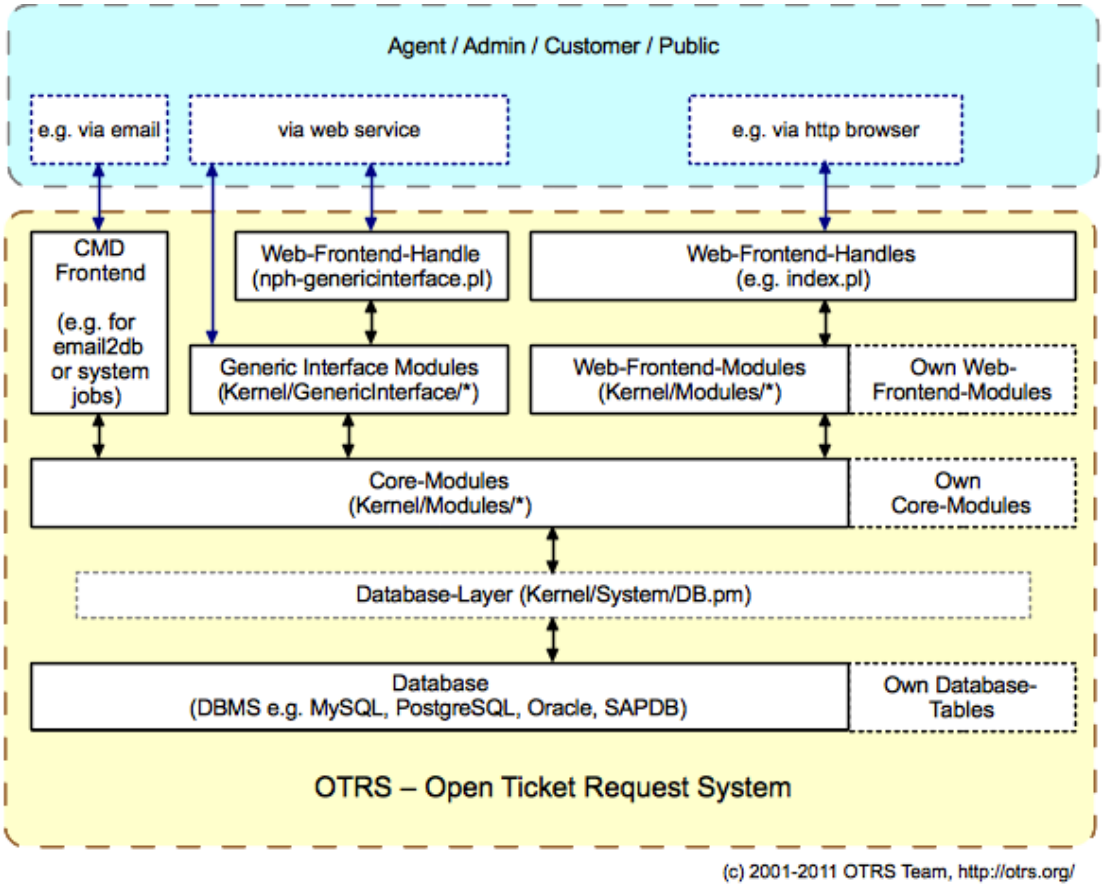

Imagen 1. Esquema básico de la arquitectura de OTRS

OTRS sigue una arquitectura modular, tal y como podemos observar en la Imagen 1. Este tipo de arquitectura favorece mucho la incorporación de nuevos elementos al sistema como el proyecto propuesto.

El funcionamiento de OTRS es una relación entre dos entidades, cliente y agente. Cuando un cliente inicia sesión en el portal de clientes aparece en un listado de todos sus tickets de incidencias abiertos y un botón con el que crear un nuevo ticket para notificar una incidencia.

Para crear un ticket nuevo, el cliente debe añadir un nombre y seleccionar, dependiendo de la configuración del sistema donde cree la incidencia, un servicio, tipo de ticket o cola de tickets. Con estos tres valores el sistema se encargará de encontrar un agente que pueda solventar la incidencia del cliente.

A continuación, el usuario rellena el asunto que identifique la incidencia y un cuerpo con el que la explique. También dispone de la opción de añadir cualquier archivo adjunto que considere necesario para alcanzar una resolución exitosa en la incidencia a tratar.

Estos son los campos básicos de un ticket en OTRS, pero se ofrecen los campos dinámicos, que permiten añadir nuevos campos para que el cliente aporte información adicional. Estos campos pueden ser textos de variados tamaños, fechas, casillas de verificación o seleccionables de entre una lista de valores. En la funcionalidad básica de OTRS no existe limitación a la hora de crear y añadir campos dinámicos a un ticket, sin embargo, todos los tickets deben tener la misma configuración de campos dinámicos.

Al terminar el cliente de crear el primer ticket de la incidencia, el sistema OTRS lo añade a una bandeja de entrada con al menos un agente capaz de resolver la incidencia. Desde el punto de vista del cliente, la incidencia recientemente abierta le aparecerá en un listado con todas sus demás incidencias aun por ser resueltas, y desde ahí podrá acceder a la página de su incidencia donde podrá repasar todos los tickets que ha enviado y todos los tickets que pudiera haber recibido como respuesta por parte de algún agente, y se le ofrece la posibilidad de crear un nuevo ticket asignado a su incidencia.

Cuando un agente recibe una nueva incidencia comenzará un proceso de paso de mensajes con el cliente mediante tickets, que eventualmente terminará resolviendo la incidencia, con un resultado beneficioso para ambas partes o no. Cuando la incidencia se da por resuelta, el agente la marca como tal y se mueve el registro de todos los tickets enviados por ambas partes a un registro, accesible tanto desde el portal del cliente en cuestión como desde el de los agentes que intervinieron en la resolución o que habrían sido válidos para la resolución de la incidencia, aunque no intervinieran en ella.

3.2. Soluciones similares a la aportada

Al comenzar el proyecto, en primer lugar, busqué soluciones ya creadas cuyo fin fuese similar al buscado con este módulo. Tras varias búsquedas por internet, se encontraron una solución en un foro de desarrollo de OTRS, donde se proponía una solución realizada con JavaScript. Además de esta existe un módulo oficial de OTRS al cual se pretende sustituir con este proyecto.

3.2.1. Solución con JavaScript

La solución se encontró en los foros OtterHub [8], una página donde usuarios de OTRS intercambian experiencias y soluciones con este sistema.

La solución propuesta comienza añadiendo todos los posibles campos dinámicos a la vista principal de creación de tickets de forma normal, y mediante un sencillo script y en función de la cola que el cliente seleccione para su ticket muestra todos los campos dinámicos o los oculta mediante la adición de la clase '*Hidden'* a los campos.

Esta solución, muy sencilla en para el proyecto solicitado, podía ser fácilmente ampliable cambiando las colas por servicios y consultando en la base de datos que campos debían ocultarse y cuáles no.

No obstante, se descartó porque la consulta en la base de datos mediante un script que se ejecutaba en otra máquina resultaba un desafío a la seguridad innecesario.

Aun siendo una solución no fiable en términos de seguridad, de esta resolución se obtienen datos tales como una forma de mostrar y ocultar campos utilizando un estilo ya implementado en OTRS, así como el evento de AJAX que se debe utilizar para ordenar una actualización en los campos dinámicos.

3.2.2. Solución oficial

La solución oficial planteada por OTRS [9] contiene ideas de entre las cuales algunas se han tomado como base sobre donde desarrollar. El módulo se basa en modificar el módulo interno de creación de tickets para ofrecer el cambio de visibilidad mediante la selección de una combinación de tipo de ticket y servicio. Para ello utiliza dos módulos, *OTRSDynamicTicketTemplates* y *OTRSHideShowDynamicFields*.

El primero de ellos, *OTRSDynamicTicketTemplates*, se encarga de toda la gestión de las plantillas. De forma resumida, este módulo añade una nueva opción de administración al menú de los administradores del sistema que permite la creación de estas plantillas, crea las funciones de acceso a la base de datos y modifica la interfaz de creación de tickets para obtener los valores de la plantilla seleccionada y aplicarlos.

El segundo de ellos, *OTRSHideShowDynamicFields*, tienen como su nombre indica la función de mostrar y ocultar los campos dinámicos que el módulo anterior se ha encargado de obtener.

De comprender esta solución no solo se ha extraído una base sobre dónde comenzar a implementar mi solución, si no que se han obtenido conocimientos de sobre cómo ampliar el sistema base.

4. Análisis

4.1. Requisitos

4.1.1. Requisitos Funcionales

- El módulo deberá integrarse en el sistema OTRS
- Los campos dinámicos tendrán tres estados de visibilidad: oculto, visible y obligatorio
- El módulo deberá utilizar unas plantillas que relacionan un estado de visibilidad con cada campo dinámico del sistema.
- Toda la información contenida en las plantillas se almacenará en una base de datos a la que accederá únicamente el servidor.
- Las plantillas estarán asignadas a un servicio de colas.
- En las plantillas deberá quedar reflejado quien realizó el último cambio y cuando.
- Las plantillas podrán añadir parte del cuerpo del mensaje al seleccionarlas.

4.1.2. Requisitos No Funcionales

- El módulo deberá realizarse de acuerdo con la guía de desarrollo de software para OTRS
- El módulo deberá quedar integrado en la interfaz del cliente en forma de un nuevo campo que permita seleccionar la plantilla
- El módulo deberá tener una interfaz de administración de plantillas accesible solo para administradores del sistema
- Los agentes podrán ver que campos dinámicos ha completado el cliente
- Los agentes serán completamente ajenos a la elección de plantilla realizada por el cliente.

4.2. Estructura de los módulos de OTRS

El desarrollo de módulos [10] para OTRS se realiza en Perl principalmente, con algunas adicciones de JavaScript y HTML para las interfaces web y XML para las opciones de configuración del módulo. Un nuevo módulo consta de los siguientes archivos:

- Archivo de configuración
- Archivo de interfaz
- Archivo de núcleo
- Plantilla HTML
- Un archivo de idiomas para cada idioma adicional en los que estará disponible el módulo

A continuación, se explicará brevemente cada uno de los archivos anteriores.

4.2.1. Archivo de configuración

En este archivo, escrito en XML, se crea un nuevo elemento para la configuración del sistema al cual los administradores podrán acceder desde el menú correspondiente. En este archivo se debe registrar el nuevo módulo, añadiendo un nombre y una breve descripción, y se podrán añadir variables globales para el nuevo módulo, como se puede observar en la Imagen 2.

```
<?xml version="1.0" encoding="UTF-8" ?>
<otrs_config version="1.0" init="Application">
        <ConfigItem Name="Frontend::Module###AgentHelloWorld"
Required="1" Valid="1">
        <Description Translatable="1">FrontendModuleRegistration
for HelloWorld module.</Description>
        <Group>HelloWorld</Group>
        <SubGroup>Frontend::Agent::ModuleRegistration</SubGroup>
        <Setting>
            <FrontendModuleReg>
                <Title>HelloWorld</Title>
                <Group>users</Group>
                <Description>HelloWorld</Description>
                <NavBarName>HelloWorld</NavBarName>
                <NavBar>
                    <Description>HelloWorld</Description>
                    <Name>HelloWorld</Name>
                    <Link>Action=AgentHelloWorld</Link>
                    <NavBar>HelloWorld</NavBar>
                    <Type>Menu</Type>
                    <Prio>8400</Prio>
                    <Block>ItemArea</Block>
                </NavBar>
            </FrontendModuleReg>
        </Setting>
    </ConfigItem>
</otrs_config>
```

Imagen 2. Archivo de configuración

4.2.2. Archivo de interfaz

El archivo de interfaz es el archivo que se carga al iniciar un módulo. En este archivo, el cual está escrito en Perl, se recogen, calculan y aplican las variables relacionadas con la vista HTML y se aplican a esta los cambios generados por el usuario mediante AJAX. Está compuesto por dos funciones: *new* y *Run*. La función new se cargará con el módulo y prepara todas las variables que se puedan utilizar durante la ejecución de este, así como carga todos los módulos adicionales de los cuales hará uso. Por otra parte, la función Run se ejecutará con cada carga de la página, ya sea por solicitud del navegador como por un cambio producido por AJAX.

En este archivo se cargará una plantilla HTML a la que se le pasarán los resultados de todas las operaciones y consultas a la base de datos que realice para representarlos ante el usuario.

4.2.3. Archivo de núcleo

En este archivo se crean todas las funciones que el módulo necesitará para interactuar con el sistema OTRS, por ejemplo, todas las funciones para gestionar los datos almacenados en la base de datos. Los archivos de núcleo utilizan la extensión `.pm` y se encuentran en el directorio `$OTRS_HOME/Kernel/System/`.

La estructura principal de un archivo de núcleo es una función new que inicializa las variables del archivo y carga otros archivos de núcleo necesarios para la ejecución del presente, y una subrutina para cada una de las funcionalidades que se le quieran dar al archivo.

Como ejemplo de archivos de núcleo están:

- `Config`. Permite la lectura y escritura de los archivos de configuración del sistema. En el desarrollo del proyecto se utiliza para comprobar si el módulo está activo.
- `Log`. Contiene todas las funciones necesarias para documentar los registros de los módulos. Se ha utilizado para comunicarse con el usuario en caso de error.
- `DB`. Contiene las subrutinas genéricas de acceso a base de datos.
- `User`. Permite la gestión de usuarios desde el código fuente.

4.2.4. Plantilla HTML

En el último archivo necesario para el módulo se crea una plantilla que genere dinámicamente los archivos HTML que se visualizarán en el navegador. Esto se realiza mediante la unión de HTML para la presentación al usuario y Perl para la lógica de programa. El archivo resultante para esta plantilla será en apariencia un HTML con bloques de código Perl donde el motor de plantillas de OTRS se encargará de operar según se indique en el archivo de interfaz y generará los archivos finales que podrá ver el usuario.

Las plantillas deben ser introducidas entre corchetes con un símbolo de porcentaje en el interior, como se puede observar en el ejemplo `[% Data.Nombre %]`.

Existen diversos tipos de plantillas según el tipo de información que se quiera añadir y se explicarán a continuación:

- Inserción de datos dinámicos. Sirven, como su nombre indica para introducir datos. Las fuentes de los datos pueden ser un parámetro del código Perl con la lógica del programa, una variable de entorno o una variable de la configuración.

Este tipo de plantillas suelen estar seguidas de un tag de procesado de datos que, aunque es opcional, es muy recomendable añadir para evitar problemas de seguridad como, por ejemplo, la introducción de código ajeno usando los tags de HTML `<script>`. Además de la seguridad que aportan, estos tags se encargan de formatear levemente la salida de las plantillas para adaptarlas al bloque donde están insertadas. Los posibles tags a utilizar son html, json y uri, para código HTML, JSON y enlaces URL respectivamente.

Los parámetros tienen la forma del ejemplo anterior, `[% Data.Nombre %]`. Sus contenidos se extraen del código Perl con la lógica de programa del módulo correspondiente. En un ejemplo de uso como el siguiente, `[% Data.Archivo | uri %]`, la plantilla recoge el valor de la variable `Archivo` y lo insertará en una dirección URL.

Las variables de entorno tienen la forma `[% Env() %]`, con el nombre de la variable dentro del paréntesis y entre comillas. Proporcionan información proveniente del objeto LayoutObject que contiene todas las variables de la vista web actual. Un ejemplo de uso sería: `[% Env("UserFullname") %].`

Finalizando con los datos dinámicos, están las variables de configuración, que funcionan de forma similar a las variables de entorno, pero utilizando la palabra clave `Config`, por ejemplo, `[% Config("ServiceActive" %]`.

- Localización. Para la localización se dispone de dos plantillas.

  `[% Translate() %]`. Permite la traducción al idioma seleccionado por el usuario de un texto. Funciona igual que las plantillas anteriores, añadiendo dentro del paréntesis el texto a traducir entre comillas. Al igual que con los datos dinámicos, cuando se traduce un texto es recomendado añadir tags para mejorar la seguridad.

  `[% Localize() %]`. Esta segunda plantilla permite añadir fechas y horas en la zona horaria seleccionada por el usuario. Para que funcione se ha de introducir los datos en el formato UTC.

  Para facilidad del programador, estas dos plantillas se pueden usar en conjunción con las anteriores permitiendo la traducción o localización directa de un dato dinámico, por ejemplo, `[% Data.Opcion | Translate | html %].`

- Comandos de procesado de plantillas. Para finalizar los archivos de plantillas, existen tres tipos más de configuraciones que se pueden añadir.

  [% InsertTemplate() %]. Este comando de plantillas permite insertar un archivo de plantilla dentro del actual. El uso más común de esta funcionalidad es insertar la información de copyright en la página.

[% RenderBlockStart() %] [% RenderBlockEnd() %]. Permiten especificar partes separadas del código de la plantilla como un bloque, de forma que el código Perl puede decidir añadir estos bloques una vez, muchas o ninguna con independencia del resto del código. El ejemplo más representativo de esta funcionalidad se puede observar en la creación de tablas HTML. Como se puede ver en la Imagen 3, extraída del manual de desarrollador de OTRS, se crean dos bloques para completar las filas de la tabla, uno donde se tratan todos los datos y se asignan a su columna correspondiente, y otro en el que se informa al usuario que no se han encontrado datos, y será el código Perl el que decida añadir las filas necesarias llamando al bloque correspondiente.

[% WRAPPER JSOnDocumentComplete %] [% END %]. Como el nombre de este último comando puede indicar, se utiliza para insertar código JavaScript en la plantilla que se ejecutará cuando los archivos CSS, JavaScript u otros hayan terminado de ejecutarse. Al contrario que el caso anterior de los bloques, para el uso de JavaScript no se necesitan crear identificadores porque es totalmente independiente de otros archivos de Perl. La principal utilidad que se le da a estos bloques es crear las funciones AJAX que permitirán la interacción en vivo con el módulo.

```
    <tbody>
[% RenderBlockStart("NoDataFoundMsg") %]
        <tr>
            <td colspan="6">
                [% Translate("No data found.") | html %]
            </td>
        </tr>
[% RenderBlockEnd("NoDataFoundMsg") %]
[% RenderBlockStart("OverviewResultRow") %]
        <tr>
            <td><a class="AsBlock" href="[% Env("Baselink")
%]Action=[% Env("Action") %];Subaction=Change;ID=[% Data.ID | uri
%]">[% Data.Name | html %]</a></td>
            <td>[% Translate(Data.TypeName) | html %]</td>
            <td title="[% Data.Comment | html %]">[% Data.Comment
| truncate(20) | html %]</td>
            <td>[% Translate(Data.Valid) | html %]</td>
            <td>[% Data.ChangeTime | Localize("TimeShort") %]
</td>
            <td>[% Data.CreateTime | Localize("TimeShort") %]
</td>
        </tr>
[% RenderBlockEnd("OverviewResultRow") %]
```

Imagen 3. Uso de los bloques de renderizado en las plantillas HTML

Para finalizar con las plantillas HTML, como asignar una plantilla a un módulo. Cuando se va a mostrar un resultado del módulo en la web se ha de realizar una llamada a la plantilla como se muestra en la Imagen 4 y añadir como parámetros el nombre de la plantilla y una lista con todos los datos que se utilizarán en esta.

```
$Output = $Kernel::OM->Get('Kernel::Output::HTML::Layout')->Output(
    TemplateFile => 'PlantillaHTML',
    Data         => \%Parametros,
);
```

Imagen 4. Inserción de una plantilla HTML

4.2.5. Archivos de idiomas

Para finalizar están los archivos de idiomas, uno por cada idioma adicional que se quiera añadir al módulo. Estos archivos, los cuales son muy sencillos, constan de una tabla de equivalencias entre frases y sus traducciones y una función para buscar la traducción, como se puede comprobar en el ejemplo de la Imagen 5 donde se traduce el clásico “Hello World!” al castellano.

```
sub Data {
    my $Self = shift;

    $Self->{Translation}->{'Hello World!'} = ¡Hola mundo!';

    return 1;
}
1;
```

Imagen 5. Ejemplo de archivo de idioma

Para añadir nuevas cadenas de texto con traducción se deberá añadir a la lista `$Self->{Translation}` un nuevo elemento, donde el identificador será el texto original. Para que el sistema cargue este nuevo archivo de idiomas solo es necesario que el nombre del archivo comience con el código del idioma, en el caso del ejemplo el nombre del archivo sería `es_HolaMundo.pm`.

5. Diseño

En este apartado se añaden los diseños de los pequeños submódulos que forman el módulo y todos los aspectos a tener en cuenta para la futura implementación en el sistema.

5.1. Diagrama de secuencia

Para el diseño se ha realizado un diagrama de secuencia que se muestra en la Imagen 6 con el que comprender mejor el funcionamiento que debe tener el módulo para mostrar al usuario los campos dinámicos que necesita en función de la plantilla seleccionada.

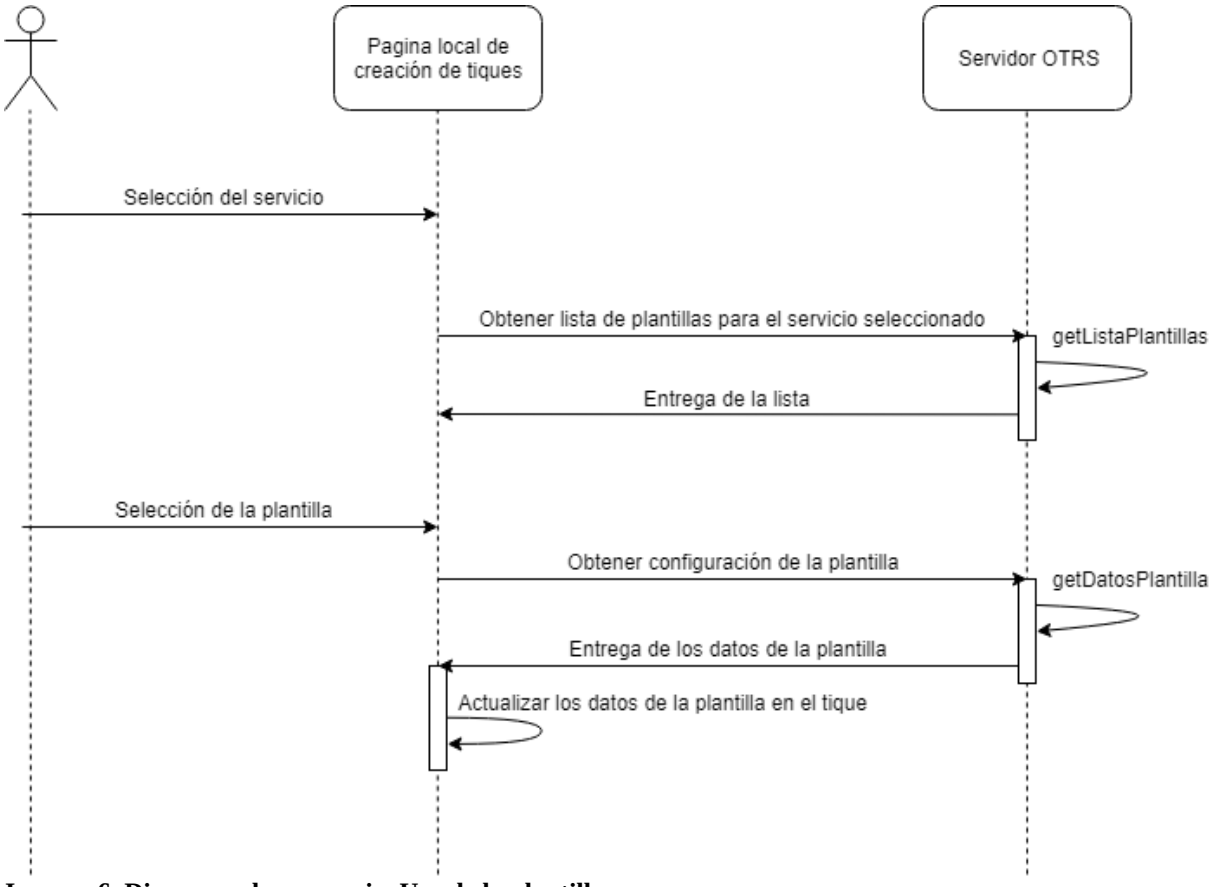

Imagen 6. Diagrama de secuencia. Uso de la plantilla

Lo que se pretende ilustrar con el diagrama son las implicaciones que tendrán las dos únicas acciones del usuario cliente para el sistema.

La primera de ellas será seleccionar un servicio entre una lista. Con ellos el sistema deberá buscar un listado con todas las plantillas que estén conectadas

con dicho servicio. Es una tarea sencilla a simple vista, pero implicara añadir un nuevo campo al formulario, es decir, modificar la plantilla HTML y el archivo de interfaz.

La segunda de las acciones mostradas es la selección de una plantilla, solo disponible previa selección de un servicio. En esta ocasión se tendrá que obtener todos los datos de la plantilla y con ellos editar en vivo el formulario. Para llevar a cabo dicha tarea se aplicarán los cambios en el archivo de interfaz, pero para la actualización correcta será necesaria una nueva llamada de AJAX que tenga en cuenta los nuevos cambios que se realizarán.

5.2. Diseño de interfaz

Para poder crear plantillas y poder editarlas es necesaria la introducción de una nueva interfaz que permita a los usuarios administrador realizar las tareas oportunas. Esta parte del diseño está orientada a los administradores, es una parte formada únicamente por dos interfaces, una para mostrar datos y otra para editarlos, por lo que no se considera necesaria la creación de diagramas.

Como se ha dicho, serán dos interfaces. La primera de ellas mostrará un listado con todas las plantillas ya creadas en una tabla que contenga los datos básicos de cada plantilla y desde la que se podrá acceder a una pantalla de edición donde se pueden comprobar todos los campos de la plantilla al completo.

Imagen 7. Diseño de interfaz para el listado de plantillas

Como podemos observar en la Imagen 7 la primera página con la que se encontrara el administrador es un listado de todas las plantillas creadas en

forma de tabla, con los campos más identificativos de la plantilla y un botón para borrarla. Se incluirá también un botón para acceder a un formulario con el que crear una nueva plantilla en el lateral izquierdo siguiendo de esta forma el diseño de OTRS.

El nombre de las plantillas hará las veces de botón y servirá para poder acceder al formulario con el que se podrán comprobar todos los datos de la plantilla y editarla.

Tanto la edición de plantillas como la creación de nuevas utilizan los mismos campos, así que se utilizará el mismo formulario web para ambas.

En la interfaz de creación de plantillas, Imagen 8, se presentará un formulario para rellenar los datos básicos de una plantilla: nombre, comentario, validez y servicio asociado. Se añaden también datos adicionales de carácter opcional para el funcionamiento de la plantilla que incluyen el asunto y el cuerpo del mensaje, que en caso de estar completados cambiaran el valor inicial de estos campos al seleccionar la plantilla el cliente.

Imagen 8. Diseño de interfaz para el formulario de creación y edición de plantillas

Para finalizar con esta interfaz tenemos lo más importante, la relación de campos dinámicos, en forma de un listado con todos los campos dinámicos que existen en el sistema y un desplegable con las opciones visible, oculto y obligatorio.

Para el diseño de esta interfaz se ha tomado como referencia la interfaz de administración de campos dinámicos, integrada por defecto en OTRS, con lo cual creamos una interfaz que sigue el estilo de diseño de OTRS.

5.3. Diseño de la base de datos

El sistema sobre el que se instalará este módulo ya tiene una base de datos definida, por lo que no es necesario establecer una nueva. Aun así, es necesaria la creación de algunas tablas para almacenar la información nueva que se genere. La base de datos sobre la que se trabajará es MySQL.

Para la base de datos se necesitará una tabla que almacene la plantilla, y sus datos básicos, como un nombre identificativo, una descripción o su estado de activación en el sistema.

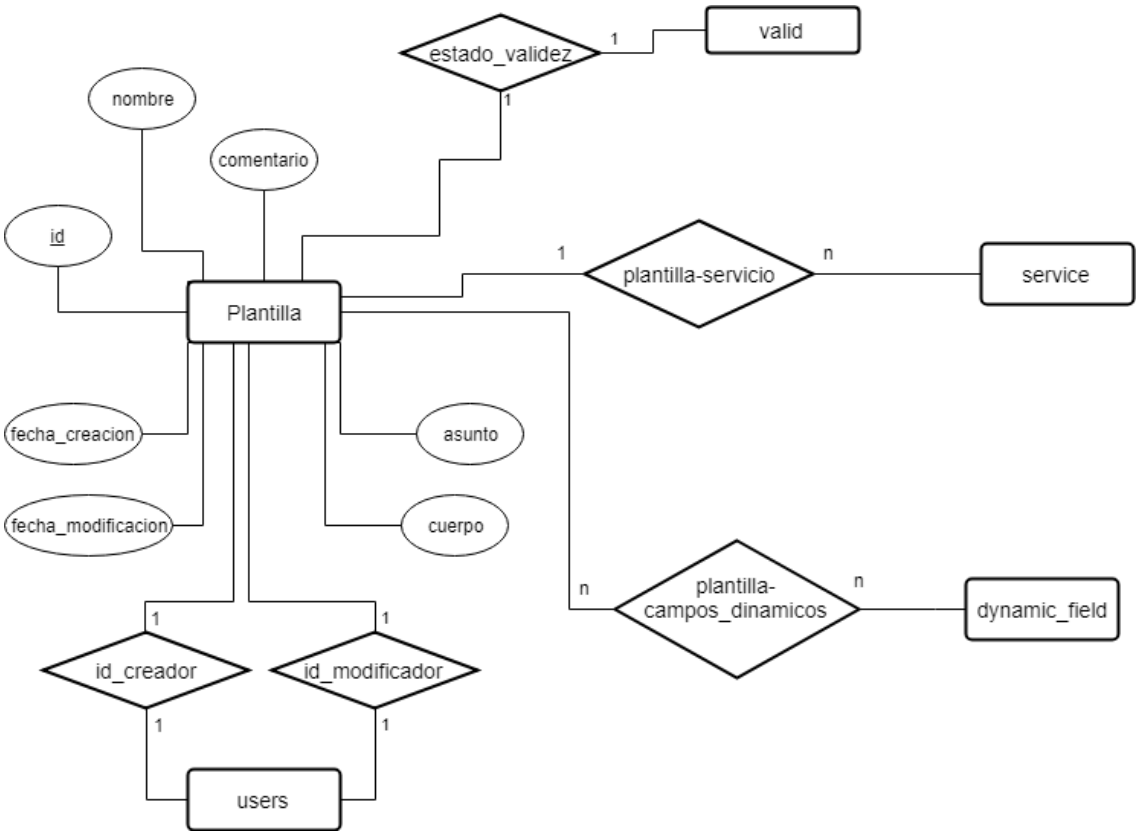

Imagen 9. Diagrama entidad-relación

Se han de añadir tablas que permitan relacionar las plantillas con los campos dinámicos y con los servicios, que poseen sus propias tablas en la base de datos original del sistema. Las relaciones entre la nueva tabla y los campos dinámicos serán de tipo varios-varios, mientras que con el servicio serán de una-varios, estando la unicidad del lado de la plantilla. En la Imagen 9 se puede observar un pequeño diagrama entidad-relación en el que se ha creado la clase principal que se añade con las plantillas y sus relaciones con otras clases ya existentes del sistema.

A continuación, describiré las tablas y atributos de la base de datos.

- Plantilla. La clase plantilla será la clase principal sobre la que se diseña la base de datos. Representará las plantillas y entre sus atributos se encuentra, junto a una id única autoincremental, un nombre de plantilla y un comentario con los que el usuario final las identificará y diferenciará, el comentario será un atributo opcional. Se encuentran también el asunto y cuerpo, ambos opcionales, predefinidos para los tickets que utilicen esa plantilla. Como dos últimos atributos se añaden las fechas de creación y última edición del ticket, que se utilizarán como medidas de control internas, y junto a ellas dos claves ajenas a la tabla del sistema users, con las que almacenaremos los usuarios que crearon y realizaron la última modificación del ticket. Finalmente se añade otra clave ajena a la tabla valid, también del sistema de OTRS, con la que se podrá asignar un valor de activación al ticket.

- Plantilla-servicio. Esta tabla contendrá las relaciones entra las entidades plantilla y service, esta última forma parte del sistema OTRS y contiene toda la información sobre los servicios que serán los que se utilizarán para agrupar las plantillas. Cada plantilla tendrá asignado un único servicio y no existirá ninguna restricción en el número de plantillas que pueda un servicio tener asignadas.

- Plantilla-campos_dinamicos. Esta tabla contendrá las relaciones entra las entidades plantilla y dynamic_field. La entidad dynamic_field está incluida en la base de datos de OTRS y almacena los campos dinámicos creados por los usuarios. Cada plantilla que se crea debe tener una entrada en esta tabla para cada campo dinámico existente.

6. Desarrollo

El proceso de desarrollo del proyecto se ha dividido en 3 bloques, crear el tipo de dato de las plantillas, introducir las plantillas en la creación de tickets y crear una interfaz para que el usuario administrador pueda crear y editar las plantillas. La realización en primer lugar de la base de datos permite contar con datos con los que contrastar la funcionalidad de las diferentes funciones que forman los siguientes bloques.

6.1. Creación de la base de datos

En primer lugar, se creará en la base de datos las tablas anteriormente diseñadas mediante cualquier gestor de bases de datos válido. Para ello se he hecho uso de la herramienta MySQL Workshop.

Tras esto se continúa creando el archivo de núcleo del módulo, el cual contendrá todas las funciones que gestionan la interacción con la base de datos. El archivo se compone de la función new que carga un listado con todos los campos dinámicos y prepara la caché de base de datos, y una serie de funciones para cada tipo de interacción con la base de datos, en la Imagen 10 se puede observar como ejemplo la función que obtiene una lista con todas las plantillas existentes para un servicio específico.

```
sub DynamicTemplatesSearch {
    my ( $Self, %Param ) = @_;

    my $DBObject = $Kernel::OM->Get('Kernel::System::DB');
    return if !$DBObject->Prepare(
        SQL => 'SELECT id, nombre
                FROM plantilla
                WHERE valid_id = 1 AND id IN (SELECT plantilla_id
                    FROM plantilla-servicio
                    WHERE service_id = ?)',
        Bind => [ \$Param{ServiceID} ],
    );

    my %ListaPlantillas;
    while ( my @Row = $DBObject->FetchrowArray() ) {
        $ListaPlantillas{ $Row[0] } = $Row[1];
    }

    return %ListaPlantillas;

}
```

Imagen 10. Ejemplo función de acceso a la base de datos

Para realizar su objetivo, la función en primer lugar carga el archivo de núcleo DB en la variable `$DBObject`. Se ejecuta el código SQL correspondiente al

servicio necesario, un *Select* en este caso, y se recupera la información fila a fila ejecutando la función `FetchrowArray()`.

Al igual que en el anterior ejemplo se añaden las funciones para obtener los datos de una plantilla, crearla, modificarla y otro listado de plantillas, sin restricción de servicio en este caso.

6.2. Edición de la interfaz de creación de tickets

El siguiente paso es modificar la interfaz que permite crear tickets para poder añadir las nuevas funcionalidades. Se comienza editando la plantilla HTML a la que se añade el nuevo campo que permite seleccionar la plantilla y a continuación modificamos el archivo de interfaz.

En primer lugar, la plantilla HTML, originalmente situada en `Kernel/Output/HTML/Templates/Standard/CustomerTicketMessage.tt`. El primer cambio que se ha de realizar en el archivo es crear el nuevo campo que permita seleccionar la plantilla, como se puede observar en la Imagen 11. Se crea un nuevo bloque que contiene una etiqueta con texto traducible y un campo cuyo contenido se guarda en *DTStrg*. Al final del bloque se añade un wrapper de JavaScript que creará la llamada de AJAX al seleccionar una plantilla.

```
[% RenderBlockStart("DesplegablePlantillasDinamicas") %]
    <div>
        <label for="PlantillaDinamica">
            [% Translate("Dynamic Templates") | html %]:
        </label>
        <div class="Field">
            [% Data.DTStrg %]
            <p class="FieldExplanation">
                [% Translate("Setting a template will overwrite any
                text or attachment.") | html %]
            </p>
            </div>
            <div class="Clear"></div>
        </div>
[% WRAPPER JSOnDocumentComplete %]
        <script type="text/javascript">//<![CDATA[
            $('#PlantillaDinamica').bind('change', function (Event){
                Core.AJAX.FormUpdate(
                    $('#NewCustomerTicket'),
                    'AJAXUpdate',
                    'PlantillaDinamica' );
            });
        //]]></script>
[% END %]
[% RenderBlockEnd("DesplegablePlantillasDinamicas") %]
```

Imagen 11. Bloque de creación del nuevo campo

En la Imagen 12 se añade la segunda modificación de la plantilla. El bloque de campos dinámicos existe previamente, sin embargo, se añade una nueva clase

al listado de clases del bloque HTML. Esta clase será *Hidden* y permitirá que el campo dinámico quede completamente oculto.

```
[% RenderBlockStart("DynamicField") %]
    <div class="Row Row_DynamicField_[% Data.Name | html %] [% Data.Class | html%]">
    [% Data.Label %]
        <div class="Field">
        [% Data.Field %]
        </div>
        <div class="Clear"></div>
    </div>
[% RenderBlockEnd("DynamicField") %]
```

Imagen 12. Modificación para mostrar y ocultar campos dinámicos

Con esto la plantilla HTML está completa. A continuación, se va a comentar las modificaciones realizadas en el archivo de interfaz, localizado en `Kernel/Modules/CustomerTicketMessage.pm`.

Esta parte se trata de la más difícil y comprometida de todo el proyecto al necesitar modificar archivos originales del sistema. Para estas tareas que requieren la modificación de archivos clave del sistema OTRS contiene en su directorio de instalación una carpeta *Custom* donde se pueden crear copias de estos archivos modificadas que serán ejecutadas en primer lugar, dejando el archivo original intacto.

En primer lugar, hay que permitir la carga de todos los campos dinámicos. Por defecto, en OTRS solo se cargan los campos dinámicos que se han añadido en su correspondiente archivo de configuración, haciendo necesario que para poder usar un campo dinámico primero se tenga que haber configurado en una pantalla distinta a la de creación de este, lo que hace que sea una tarea larga y poco intuitiva, sobre todo con grandes cantidades de campos dinámicos. Para solucionar esto hace falta eliminar el filtro que se emplea en el archivo. Esto se consigue realizando un nuevo filtro como se ve en la Imagen 13, donde se hace uso del operador ternario para configurar el filtro como su valor por defecto o vacío según este activo el módulo o no, lo que permitirá que OTRS funcione igualmente si se desea en un futuro desactivar temporalmente el módulo. Complementario a este filtro, es necesario realizar una búsqueda en el archivo y sustituir todas las ocurrencias del filtro por el nuevo creado.

```
my $FieldFilter;
$FieldFilter =
    $ConfigObject->Get('Ticket::PlantillasDinamicas') ? undef : $Config->{DynamicField};
```

Imagen 13. Filtro para campos dinámicos

A partir de ahora la función se divide en función de la “subacción” seleccionada. Las posibles subacciones posibles son *StoreNew* y *AJAXUpdate*, y se ejecutaran cuando se envíe el ticket o se añadan archivos adjuntos, y

cuando se realice un cambio en la página mediante AJAX. La tercera opción es que no se seleccione ninguna subacción, que corresponde a la primera carga de la página, cuando se crea todo. En el caso de no existir subacción no se realiza ningún cambio, pero si se encuentran cambios en la subrutina *_MaskNew* que se ejecuta al finalizar este apartado del código, y en caso de *StoreNew* hay que añadir la lista de plantillas en la nueva carga de la página tras añadir un nuevo archivo adjunto.

La carga de trabajo en este archivo se encuentra, pues, en *AJAXUpdate*. Lo primero que se realiza al actualizar es comprobar si el cambio que ha producido la actualización es la selección de un nuevo servicio, y de ser así se carga el nuevo listado de plantillas. Si al cargar la lista de plantillas solo se obtiene una plantilla, se deja seleccionada esta y en caso contrario se borra la plantilla seleccionada, de la forma que se puede comprobar en la Imagen 14.

```
my $DynamicTemplates = $Self->_ObtenerPlantillasDinamicas(%GetParam);
$GetParam{IDPlantilla} = $ParamObject->GetParam( Param=> 'PlantillaDinamica') || '0';

if ( $GetParam{AntiguoServiceID} ne $GetParam{ServiceID} ) {

    my $CantidadPlantillas = scalar keys %{$DynamicTemplates};
    if ( $CantidadPlantillas == 1 ) {
        $GetParam{IDPlantilla} = (sort keys %{$DynamicTemplates})[0];
    }
    else {
        $GetParam{IDPlantilla} = '';
    }
}
```

Imagen 14. Carga del listado de plantillas

El siguiente paso es recuperar la lista de campos dinámicos con sus valores de visibilidad asociados a la plantilla seleccionada, solicitando la búsqueda en la base de datos y guardando los datos para más adelante en una lista con la id del campo dinámico como índice y su visibilidad como valor, codificada esta última como 0, 1 ó 2, según se quiera marcar como oculto, visible u obligatorio. En caso de no tener ninguna seleccionada se marcan todos como ocultos.

Para crear el código HTML que se insertará en la plantilla más adelante se utiliza la función `EditFieldRender`, localizada en el archivo de núcleo *Backend*. Esta función que se puede ver en la Imagen 15 devuelve el código HTML para un campo dinámico en forma de una lista con dos elementos, el primero de ellos la etiqueta del campo y el segundo el cuerpo. Aquí se configuran ya los campos dinámicos que deban ser obligatorios, pero la función no dispone de funcionalidad para ocultarlos.

```
$DynamicFieldHTML{ $DynamicFieldConfig->{Name} } =
    $BackendObject->EditFieldRender(
    DynamicFieldConfig      => $DynamicFieldConfig,
    PossibleValuesFilter    => $PossibleValuesFilter,
    Mandatory       => $Config->{DynamicField}->{ $DynamicFieldConfig->{Name} } == 2,
    LayoutObject    => $LayoutObject,
    ParamObject     => $ParamObject,
    AJAXUpdate      => 1,
    UpdatableFields => $Self->_GetFieldsToUpdate(OnlyDynamicFields => 1,),
    UseDefaultValue => 1,
);
```

Imagen 15. Creación del cuerpo HTML de los campos dinámicos

Como no se puede aplicar en la función anterior la visibilidad, será este el siguiente paso. Como se puede ver en la Imagen 16, se recorren todos los campos dinámicos, y en caso de pertenecer al grupo de visibilidad 1 ó 2 se almacena su código HTML en la lista *Campo*, que se añadirá más adelante a la lista de datos que se envía a la plantilla. En caso de tener que estar oculto, el campo se añade como una cadena de texto vacía.

```
switch ( $EstadoCamposDinamicos{$Nombre} ) {
    case [1..2] {

        $Campo{ "DynamicField_" . $Nombre } =
            $DynamicFieldHTML{$DynamicFieldConfig->{Name}}{Label};
        $Campo{ "DynamicField_" . $Nombre } .=
            $DynamicFieldHTML{$DynamicFieldConfig->{Name}}{Field};
    }
    else {
        $EstadoCamposDinamicos{$Nombre} = 0;
        $Campo{ "DynamicField_" . $Nombre } = "";
    }
}
```

Imagen 16. Aplicación de visibilidad a los campos dinámicos

En la Imagen 17 se puede comprobar la creación del JSON con todos los cambios que se han realizado en el archivo. En primer lugar, se crea un array que tendrá todos los cambios realizados en forma de listas. En la imagen se muestra como ejemplo la primera lista que se añade, pero se añadirán más entre las que se incluyen la plantilla y el servicio seleccionados, por ejemplo. En segundo lugar, haciendo uso de la función `BuildSelectionJSON`, se añadirán los datos anteriores al JSON que contendrá toda la información a escribir en el formulario web. La función ya era anteriormente utilizada, y en este módulo solo se ha introducido el array *DatosAJAX* al final de esta.

```
my @DatosAJAX;
if ( IsHashRefWithData( \%Campo ) ) {
    push @DatosAJAX,
    {
        Name => 'FormDisplay',
        Data => \%Campo,
        Max => 1000000,
    };
}
my $JSON = $LayoutObject->BuildSelectionJSON(
    [
        ...
        @DatosAJAX,
    ],
);
```

Imagen 17. Creación del JSON con la configuración de visibilidad de los campos dinámicos

Por último, se modifica la subrutina *_MaskNew*, que se ejecuta para crear el formulario web vacío. Para ello se añade el bloque que se encuentra en la Imagen 18. En el hash Param se almacena los datos que permiten crear el campo generados por la función `BuildSelection()` y mediante la función `Block()` se aplicará en la plantilla.

```
if ( $CampoPlantillas ) {
    $Param{DTStrg} = $LayoutObject->BuildSelection(
        Data         => $Plantillas,
        Name         => 'PlantillaDinamica',
        PossibleNone => $Self->{SeleccionObligatoriaPlantillaVacia} || 0,
        Sort         => 'AlphanumericValue',
        Translation  => 0,
        Class        => 'Modernize',
        SelectedID   => $Param{IDPlantilla},
    );
    $LayoutObject->Block(
        Name => 'DesplegablePlantillasDinamicas',
        Data => {
            %Param,
            }
        );
}
```

Imagen 18. Creación del campo de selección de plantillas en el archivo de interfaz

6.3. Creación de interfaz de administración de plantillas

La siguiente parte en el desarrollo de este proyecto es la realización de una interfaz que permita a los administradores tanto crear nuevas plantillas como visualizarlas, editarlas o eliminarlas. Para la realización de esta interfaz de administrador se ha utilizado como base la ya creada interfaz de administración de campos dinámicos.

La creación de esta interfaz comienza con una nueva plantilla HTML en la que se crea, usando lenguaje HTML, la pantalla de visualización del listado de plantillas siguiendo el diseño realizado previamente. Utilizando las funcionalidades del motor de plantillas de OTRS podemos, a continuación, añadir el formulario correspondiente al segundo diseño en el mismo archivo y que el archivo de interfaz se encargue de seleccionar uno u otro.

El primer paso es crear la barra lateral que tendrá las opciones disponibles, en este caso crear una nueva plantilla o volver al listado si el usuario se encuentra en la página con el formulario de las plantillas. Como se observa en la Imagen 19, se crea un bloque con la barra entera y dentro de este otro con el botón concreto. A continuación, se añadirá en el código otro bloque con el botón que permita acceder al formulario de creación y será desde el archivo de interfaz desde donde se seleccionara para mostrar uno u otro.

```
[% RenderBlockStart("ActionList") %]
<div class="WidgetSimple">
    <div class="Header">
        <h2>[% Translate("Actions") | html %]</h2>
    </div>
    <div class="Content">
        <ul class="ActionList">
[% RenderBlockStart("ActionOverview") %]
            <li>
                <a href="[% Env("Baselink") %]Action=[% Env("Action") %]"
                    class="CallForAction Fullsize Center">
                    <span>
                        <i class="fa fa-caret-left"></i>
                        [% Translate("Go to overview") | html %]
                    </span>
                </a>
            </li>
[% RenderBlockEnd("ActionOverview") %]
```

Imagen 19. Creación barra lateral de la interfaz de administración

De forma similar, a continuación se creará un nuevo bloque que contendrá el cuerpo de la página web. Este, al igual que con la barra lateral, tendrá dos bloques en el interior, en uno se añadirá una tabla donde se añadirán todos los campos y en el segundo se creará un formulario web de similar forma a un formulario común, añadiendo como cuerpo de los campos del formulario las referencias para que el archivo de interfaz pueda recoger los valores y aplicarlos en la base de datos.

Con la plantilla terminada pasamos a crear el archivo de interfaz. La función Run se dividirá en seis tareas que se elegirán mediante un bloque if-else, de forma similar a las subacciones del archivo utilizado en el apartado anterior. Las tareas serán *Add, AddAction, Change, ChangeAction, Delete* y *Overview*. Las tareas *AddAction*, *ChangeAction* y *Delete* serán las encargadas de aplicar los cambios realizados en la base de datos, mientras que las otras tres se encargarán de mostrar información por pantalla.

El funcionamiento de estos apartados será similar al observado en el apartado anterior. En caso de los encargados de mostrar información se utiliza el archivo de núcleo *Layout* para crear bloques de contenidos que al final del código se empaqueta y se envía a la plantilla.

En los otros tres casos, utilizando el archivo de núcleo creado anteriormente podemos aplicar los cambios en la base de datos.

Con esto el módulo ya es ejecutable, pero no aparece en la lista de opciones del administrador. Para hacerlo accesible para el usuario se debe añadir primero en un archivo de configuración, tal y como se muestra en la Imagen 20.

```
<ConfigItem Name="Frontend::Module###AdminPlantillasDinamicas" Required="0" Valid="1">
    <Description Translatable="1">
    Frontend module registration for dynamic templates module admin area.
    </Description>
    <Group>PlantillasDinamicas</Group>
    <SubGroup>Frontend::Admin::ModuleRegistration</SubGroup>
    <Setting>
        <FrontendModuleReg>
            <Group>admin</Group>
            <Description Translatable="1">
            This module adds the admin area of dynamic templates module.
            </Description>
            <Title Translatable="1">Admin Dynamic Templates</Title>
            <NavBarName>Admin</NavBarName>
            <NavBarModule>
                <Module>Kernel::Output::HTML::NavBar::ModuleAdmin</Module>
                <Name Translatable="1">Dynamic Templates</Name>
                <Description Translatable="1">Create and manage dynamic templates.</Description>
                <Block>Ticket</Block>
                <Prio>800</Prio>
            </NavBarModule>
        </FrontendModuleReg>
    </Setting>
</ConfigItem>
```

Imagen 20. Archivo de configuración donde se añade la interfaz de administración

6.4. Empaquetamiento del producto final en un instalador e instalación

El módulo ya está terminado, pero para poder instalarlo en otros sistemas hay que crear un instalador, que en este caso es un paquete de OTRS (.opm).

El primer paso es crear un archivo de especificaciones (.sopm) que incluya todas las propiedades del módulo. Este archivo cuya estructura es de XML ha de contener, entre otros datos opcionales, el nombre del módulo, la versión actual, los frameworks de OTRS compatibles, otros paquetes que deben estar instalados previamente, un listado de todos los archivos que incluye el módulo y las sentencias de creación de la base de datos codificadas en XML. En la

Imagen 21, suministrada en el manual de desarrollo de OTRS, está disponible un ejemplo sencillo de cómo es la estructura del archivo .sopm

```
<?xml version="1.0" encoding="utf-8" ?>
<otrs_package version="1.0">
    <Name>Calendar</Name>
    <Version>0.0.1</Version>
    <Framework>3.2.x</Framework>
    <Vendor>OTRS AG</Vendor>
    <URL>http://otrs.org/</URL>
    <License>GNU AFFERO GENERAL PUBLIC LICENSE Version 3, November 2007</License>
    <ChangeLog Version="1.1.2" Date="2013-02-15 18:45:21">Added some feature.</ChangeLog>
    <ChangeLog Version="1.1.1" Date="2013-02-15 16:17:51">New package.</ChangeLog>
    <Description Lang="en">A web calendar.</Description>
    <Description Lang="de">Ein Web Kalender.</Description>
    <IntroInstall Type="post" Lang="en" Title="Thank you!">Thank you for choosing the Calendar module.
</IntroInstall>
    <IntroInstall Type="post" Lang="de" Title="Vielen Dank!">Vielen Dank fuer die Auswahl des Kalender
Modules.</IntroInstall>
    <BuildDate>?</BuildDate>
    <BuildHost>?</BuildHost>
    <Filelist>
        <File Permission="644" Location="Kernel/Config/Files/Calendar.pm"></File>
        <File Permission="644" Location="Kernel/System/CalendarEvent.pm"></File>
        <File Permission="644" Location="Kernel/Modules/AgentCalendar.pm"></File>
        <File Permission="644" Location="Kernel/Language/de_AgentCalendar.pm"></File>
        <File Permission="644" Location="Kernel/Output/HTML/Standard/AgentCalendar.tt"></File>
        <File Permission="644" Location="Kernel/Output/HTML/NotificationCalendar.pm"></File>
        <File Permission="644" Location="var/httpd/htdocs/images/Standard/calendar.png"></File>
    </Filelist>
    <DatabaseInstall>
        <TableCreate Name="calendar_event">
            <Column Name="id" Required="true" PrimaryKey="true" AutoIncrement="true" Type="BIGINT"/>
            <Column Name="title" Required="true" Size="250" Type="VARCHAR"/>
            <Column Name="content" Required="false" Size="250" Type="VARCHAR"/>
            <Column Name="start_time" Required="true" Type="DATE"/>
            <Column Name="end_time" Required="true" Type="DATE"/>
            <Column Name="owner_id" Required="true" Type="INTEGER"/>
            <Column Name="event_status" Required="true" Size="50" Type="VARCHAR"/>
        </TableCreate>
    </DatabaseInstall>
    <DatabaseUninstall>
        <TableDrop Name="calendar_event"/>
    </DatabaseUninstall>
</otrs_package>
```

Imagen 21. Archivo .spom

Con este archivo completado y utilizando en la consola de OTRS el comando mostrado en la Imagen 22 construimos el paquete final que podrá ser instalado en cualquier sistema OTRS actual que cumpla el requisito de versión a través de la gestión de paquetes de los administradores.

```
shell> bin/otrs.Console.pl Dev::Package::Build /path/to/example.sopm /tmp
Building package...
Done.
shell>
```

Imagen 22. Creación del paquete instalable

7. Conclusiones y trabajo futuro

El proyecto partía de una propuesta del Servicio de Informática y Comunicaciones de la Universidad de Zaragoza con el fin de ampliar ayudICa, el sistema de gestión de incidencias de OTRS instalado, mediante la introducción de un módulo que permita la sustitución del módulo oficial *OTRSTicketMaskExtensions* por un módulo personalizado que de cara al usuario final realice una tarea equivalente.

El modulo desarrollado ha cumplido con los requisitos establecidos inicialmente y está listo para su instalación en el sistema y para una mayor accesibilidad, el módulo está disponible tanto en castellano como en inglés.

La realización del módulo comenzó muy lentamente al ser un sistema completamente nuevo y escrito en un lenguaje de programación que no se ha visto a lo largo del grado en ninguna asignatura. No obstante, la capacidad de aprendizaje que se ha entrenado durante los años de estudio correspondientes a esta etapa universitaria ha sido capaz de hacerme evaluar los problemas que se han presentado y adaptarme a ellos para salir satisfactorio en la resolución de este trabajo.

En cuanto a posibles líneas de trabajo futuro en este proyecto se propone lo siguiente:

- Mantenimiento y revisión del módulo frente a posibles incompatibilidades con otros módulos externos que pudiesen ser instalados.
- Actualización del módulo en caso de un gran cambio en el sistema OTRS que provoque que algunas funcionalidades del módulo dejen de funcionar correctamente o provoquen incongruencias en el sistema

Referencias

[1] «Wikipedia Perl,» [En línea]. Available: https://es.wikipedia.org/wiki/Perl. [Último acceso: 15 Mayo 2018].

[2] «Perl,» [En línea]. Available: https://www.perl.org/get.html. [Último acceso: 15 Mayo 2018].

[3] «Wikipedia JavaScript,» [En línea]. Available: https://es.wikipedia.org/wiki/JavaScript. [Último acceso: 15 Mayo 2018].

[4] «Wikipedia HTML,» HTML, [En línea]. Available: https://es.wikipedia.org/wiki/HTML. [Último acceso: 15 Mayo 2018].

[5] «Wikipedia MySQL,» [En línea]. Available: https://es.wikipedia.org/wiki/MySQL. [Último acceso: 15 Mayo 2018].

[6] «Wikipedia AJAX,» [En línea]. Available: https://es.wikipedia.org/wiki/AJAX. [Último acceso: 15 Mayo 2018].

[7] «Wikipedia OTRS,» [En línea]. Available: https://es.wikipedia.org/wiki/Open-source_Ticket_Request_System. [Último acceso: 17 Mayo 2018].

[8] «Otterhub.org,» [En línea]. Available: http://forums.otterhub.org/viewtopic.php?t=24116. [Último acceso: 25 Mayo 2018].

[9] *OTRSTicketMaskExtensions Documentation.*

[10] «OTRS 5 - Developer Manual,» [En línea]. Available: http://doc.otrs.com/doc/manual/developer/5.0/en/html/index.html. [Último acceso: 25 Mayo 2018].