Escuela de
Ingeniería y Arquitectura
Universidad Zaragoza

ESCUELA DE INGENIERÍA Y ARQUITECTURA UNIVERSIDAD DE ZARAGOZA

INGENIERÍA SUPERIOR EN INFORMÁTICA

PLUGIN DE ECLIPSE PARA LA GENERACIÓN DE FORMULARIOS WEB PARA MC-SERVER v5.0

David Antón Lou

12 de febrero de 2013

Escuela de
Ingeniería y Arquitectura
Universidad Zaragoza

ESCUELA DE INGENIERÍA Y ARQUITECTURA
UNIVERSIDAD DE ZARAGOZA

INGENIERÍA SUPERIOR EN INFORMÁTICA

PLUGIN DE ECLIPSE PARA LA GENERACIÓN DE FORMULARIOS WEB PARA MC-SERVER v5.0

- Empresa: TB-Solutions Advanced Technologies
- Director: Miguel Ángel Aranda Alcañiz
- Ponente: José Javier Merseguer Hernáiz
- Autor: David Antón Lou

Zaragoza, 12 de febrero de 2013

Fdo: David Antón Lou

Agradecimientos

He de expresar todo mi agradecimiento a aquellas personas que me han ayudado, de una manera u otra, a llevar a cabo este proyecto, sin ellas no hubiese sido posible:

A David Luna, mi codirector del proyecto, cuya importancia ha sido vital en el desarrollo del mismo. Me ha aportado muchos conocimientos de programación en Java, y me ha ayudado a adquirir experiencia en el campo de la banca electrónica.

A Adrián García y Jorge Cholvis, por su atenta dedicación hacia mi y por sus aportaciones hacia este proyecto.

Y para terminar, en el apartado personal, mi más sentida gratitud a mi familia por apoyarme en todo momento durante estos años tan duros, y a mis amigos por estar siempre a mi lado cuando más lo he necesitado.

Resumen

La utilización de formularios dentro de una página web resulta, actualmente, una tarea muy habitual dentro de las entidades del sector bancario, surgiendo la problemática de tener que estar en continuo contacto con programadores para la realización de pequeños cambios o creación de nuevas páginas. El objetivo que se persigue con el proyecto es el de construir una herramienta que permita la creación de formularios sin necesidad de tener altos conocimientos sobre lenguajes de programación.

El presente proyecto, se va a encargar de realizar dicha tarea y para ello se ha pensado en una herramienta que funcione sobre una plataforma de software libre. Consistirá en un plugin para el entorno de programación Eclipse, y para facilitar su utilización por parte del usuario, se utilizará un entorno gráfico simple y amigable, haciendo uso de las librerías de edición gráfica(GEF), el sistema de modelado de eclipse(EMF) y del lenguaje de programación Java.

El plugin de Eclipse, se subdividirá en dos partes claramente diferenciadas, que corresponden a las capas de controller y view del patrón de diseño MVC. El modelador, que corresponderá al controller y el diseñador al view. Serán dos zonas de edición gráfica diferentes, pero se interconectarán entre si para que se pueda completar toda la trazabilidad de la información. Las estructuras de datos con la información de los componentes gráficos se guardarán en ficheros, y existirá un fichero diferente para cada uno de los dos editores gráficos.

El modelador se encargará de recibir operativas, hacer las operaciones necesarias sobre ellas, y proceder al envío de la información necesaria para la maquetación en la vista de la página web. Finalmente, en el diseñador se definirá el aspecto de la página o formulario para su posterior procesamiento y generación de código HTML.

Esta herramienta, estará integrada dentro de la arquitectura MC-Server de la empresa TB-Solutions. MC-Server es un framework para implementación de aplicaciones web de misión crítica y transaccionales, que puede ser utilizado para dar servicio a distintas aplicaciones del sector de la banca electrónica.

La motivación que me ha llevado a elegir este proyecto es poder realizar una herramienta que pueda ser utilizada por programadores con poca experiencia, para facilitar sus labores. Además, adquirir altos conocimientos en la programación en el lenguaje Java, así como, hacer hincapié en todas las fases de las que consta un proyecto de ingeniería del software de gran envergadura.

Índice general

Índice de figuras

Parte I

Descripción del Proyecto

Capítulo 1

Introducción. Motivación y objetivos.

TB-Solutions, con más de 25 años de experiencia, es una compañía española especializada en el desarrollo de plataformas de comunicación seguras para el uso de firma y certificados electrónicos en el entorno de las Nuevas Tecnologías de la Información. TB-Solutions forma parte del Grupo TB-Solutions, junto con TB-Solutions Technologies Software, S.L., TB-Security, S.A., Sevenclick, y Confettika.

TB-Solutions ofrece soluciones innovadoras en áreas como la banca electrónica, la administración, las empresas, la sanidad o el turismo. TB-Solutions está certificada por ISO-9001 e ISO-14001, además de tener CMMI Nivel III.

Este documento detalla el proceso de construcción de una herramienta para la plataforma MC-Server de dicha empresa, la cual se encargará de cubrir sus principales funcionalidades. A continuación, se describe brevemente cómo se va a estructurar el contenido del mismo.

1. **Descripción del Proyecto**.Esta parte incluye el capítulo de introducción, en el que se explica brevemente y de manera general las motivaciones y los objetivos del proyecto, los antecedentes y el estudio de viabilidad.

2. **Marco Teórico**.Recoge los conocimientos teóricos sobre MC-Server en banca electrónica y las principales tecnologías utilizadas para el desarrollo de la aplicación.

3. **Desarrollo e Implementación**.En esta parte se ilustra el proceso de desarrollo e implementación de las aplicaciones usadas y las principales dificultades que se han encontrado a lo largo del desarrollo de las mismas, con sus respectivas soluciones.

4. **Referencias y Bibliografía**. Contiene las principales referencias bibliográficas y páginas web consultadas.

5. **Apéndices**.Contiene el manual de instalación y el manual de usuario de la aplicación.

1.1. Motivaciones

En la sociedad de hoy en día, la inmensa mayoría de las entidades bancarias dependen de una empresa informática para solucionar cualquier necesidad relacionada con sus aplicaciones.

Existen algunas tareas, que no implican demasiada dificultad y que resultan muy sistemáticas. De esta manera, se podría facilitar la incorporación de nuevos trabajadores a la empresa de desarrollo de software, sin necesidad de tener una experiencia excesiva en la programación web. Por ejemplo, para la creación o edición de formularios bancarios.

1.2. Objetivos y Alcance del proyecto

Una de las soluciones innovadoras, y la cual es el objetivo de este proyecto, es construir una herramienta de generación de formularios para el framework MC-Server v5.0 de la empresa TB-Solutions, que ya tiene desarrollado y puesto en funcionamiento. Se pretende obtener una herramienta que pueda ser utilizada por cualquier tipo de usuario, independientemente del nivel de conocimientos informáticos que posea.

Por lo tanto, la aplicación deberá de ser sencilla de utilizar, a la vez que intuitiva para los usuarios. Debe de ser compatible con el proyecto MC-Server v5.0 que ya está desarrollado y adaptarse a todos los requisitos que este tuviera. Además, será necesario que la aplicación sea multiplataforma para no depender de un único entorno de ejecución.

Para cumplir las premisas anteriores, se pensó en un plugin para la plataforma de código abierto Eclipse, ya que es el entorno de desarrollo que se utiliza para MC-Server v5.0 y el cual permitiría añadirlo con facilidad. El plugin estará desarrollado completamente en Java y se utilizarán las librerías gráficas GEF para obtener una apariencia agradable para los usuarios.

1.3. Calendario del proyecto

Para realizar un seguimiento del desarrollo del proyecto, se ha llevado a cabo un desglose de las tareas que se han realizado. Algunas de estas tareas a su vez, se han dividido en subtareas. A continuación, se enumeran todas ellas:

1. Formación avanzada en Java(Servlets, JSP, WebServices, ...). (35 h.)
2. Adquirir los conocimientos necesarios sobre la plataforma MC-Server y los principales elementos que se utilizan en banca electrónica. (10 h.)
3. Análisis, diseño e implementación del diseñador de formularios. (300 h.)
4. Análisis, diseño e implementación del modelador. (385 h.)
5. Pruebas del sistema. (15 h.)

6. Elaboración de la memoria y de los manuales del proyecto. (100 h.)

El tiempo total de realización del proyecto se estima en unas 835 horas, que se repartieron en jornadas de 7 horas al día. A continuación se muestra una tabla con las tareas que se realizaron.

	Nombre	Duración	Inicio	Fin
1	Formación Java	35h	27/06/2011	01/07/2011
2	Formación MC-Server	10h	01/07/2011	04/07/2011
3	Diseñador de Formularios	300h	04/07/2011	24/08/2011
4	Modelador de Operativas	385h	24/08/2011	31/10/2011
5	Pruebas	15h	31/10/2011	01/11/2011
6	Memoria PFC	100h	01/11/2011	17/11/2011

Figura 1.1: Diagrama de Gantt - *Tareas del proyecto*

El diagrama de Gantt (Figura 1.2) muestra el desarrollo total del proyecto, indicando las principales tareas.

Figura 1.2: Diagrama de Gantt - *Calendario del proyecto*

Capítulo 2

Estado del arte. Antecedentes.

MC-Server es la plataforma multicanal que permite a las entidades financieras desarrollar sus aplicaciones de banca online y poner a disposición de sus clientes cualquier operativa de Banca Electrónica, desde cualquier lugar, durante las 24 horas del día y en cualquier dispositivo(Internet, dispositivos móviles, iPhone, etc.). TB-Solutions, ofrece este producto, que garantiza el correcto funcionamiento de las aplicaciones bancarias sea cual sea el número de usuarios conectados al servidor, al mismo tiempo que le proporciona asistencia técnica ante cualquier incidencia.

En la actualidad, más de 20 Entidades financieras confían en MC Server para sus aplicaciones de banca electrónica, por lo que es necesario desarrollar extensiones de este producto para hacerlo más completo.

Una de las mejoras que se pensó realizar fue una aplicación de autogeneración de código para la plataforma MC-Server. Esta aplicación permitiría generar código HTML para formularios web, clases Java de tipo EJB(Enterprise Java Bean), clases Java para conexión con Web Services o con Host, así como clases Java específicas de la plataforma MC-Server.

A continuación se muestran algunos ejemplos de aplicaciones ya desarrolladas que están relacionadas con la generación de código y podrían resultar complementarias a nuestro proyecto.

1. **Mirabyte Web Architect** - *http://www.mirabyte.com/*:
   Es una aplicación que se utiliza para la creación de páginas web profesionales. Soporta lenguajes como HTML, XHTML, CSS, JavaScript o PHP. Posee una interfaz de usuario muy intuitiva que hace que sea muy fácil familiarizarse con la aplicación. También cuenta con una gran cantidad de utilidades para conseguir que la edición de documentos web sea una tarea fácil y rápida. Por ejemplo, posee la característica de completado de código que ahorra mucho trabajo manual y agiliza tareas relacionadas con el desarrollo web. Además permite la creación de páginas web de manera gráfica facilitando así a los usuarios con pocos conocimientos de programación, generándoles automáticamente el código web.

2. **Lombok Feature** - *http://projectlombok.org/features/index.html*:
   Plugin para Eclipse de generación de código Java. Posee un conjunto de sentencias que

comienzan por el carácter ‘@’ y que posteriormente, Eclipse las tratará y realizará las acciones necesarias para cada una de las sentencias.

3. **Microsoft FrontPage** - *http://office.microsoft.com/es-es/frontpage-help/*:
   Es un editor de páginas web creado por la compañía Microsoft. Al igual que Mirabyte, se encarga de la generación de código web, en este caso HTML. Es muy sencillo de utilizar y permite al usuario generar páginas web de manera gráfica. El único inconveniente que posee, es que genera un código poco optimizado.

Parte II

Marco Teórico

Capítulo 3

Conceptos teóricos de MC-Server en banca electrónica.

TB-Solutions ha realizado numerosos proyectos Web en Java dedicados tanto a banca electrónica como a otros tipos de aplicaciones online. En cada uno de estos desarrollos se han utilizado distintas arquitecturas, aunque siempre basadas en el estándar J2EE.

En este capítulo, se va a intentar explicar el modelo de una nueva arquitectura de trabajo para construir aplicaciones sobre banca electrónica. Esta arquitectura, bautizada como MC-Server, intenta definir los componentes a utilizar en el desarrollo de aplicaciones.

Mediante esta arquitectura y el uso de una metodología claramente definida, se pretende que cualquier técnico de un departamento de desarrollo pueda entrar en la realización de este tipo de proyectos en el menor tiempo posible, incluso sin tener altos conocimientos previos de Java.

En los siguientes apartados, se detalla en profundidad la arquitectura MC-Server, sobre la que se apoyará este proyecto.

3.1. Flujo de Entrada

La entrada a la aplicación es siempre una petición http(request), y la respuesta será el resultado del proceso realizado en la aplicación y dependiendo del canal de salida, su formato será HTML, XML, WML, etc.

3.2. Elementos de la Arquitectura

A continuación se explican brevemente los elementos que entran en juego en el proceso, su funcionalidad y su interrelación. No se entrará al detalle ni de programación, ni de tecnicis-

mos específicos. Se trata, en definitiva, de tener una visión global de la arquitectura MC-Server.

3.2.1. Micro Operative

Las micro operativas son nodos con la marca o tag 'micro-operative', que identifican el nombre de la operativa y un atributo:

- **name**: Nombre de la micro operativa que deberá coincidir con el nombre de la vista(parámetro pageOperation del request).
- **iden**: False/true. Si necesita estar identificado para su ejecución.

En las operativas marcadas como 'micro-operative', se pueden encontrar como nodos hijos de la misma, los tag obligatorios de 'validation', 'error-codes' y opcionalmente 'integration'.

3.2.2. Operative

Las operativas son nodos con la marca o tag 'operative', que identifican el nombre de la operativa y sus atributos, mas otro nodo hijo 'CB-class' con los atributos relativos a la clase y el método a invocar:

- **name**: Nombre de la operativa que deberá coincidir con el nombre de la vista(parámetro pageOperation del request).
- **iden**: False/true. Si necesita estar identificado para su ejecución.
- **public**: False/true. Nos dice si la operativa puede ser invocada desde el exterior. Por defecto todas son públicas.
- **remoteAdress**: Expresiones regulares opcionales separadas por comas que nos indican las IPs desde donde se permite la ejecución de las operativas.
- **validationIPClass**:Clase `JavaBean` externa para validación de IP.
- **validationIPMethod**: Método de la clase `JavaBean` que será ejecutado. Por defecto es el método del mismo nombre que la clase. Los argumentos que recibe son la IP de origen y las IP's del 'remoteAdress' en caso de existir.
- **class**: Clase `ControllerBean` seleccionada para ejecutar.
- **method**: Método de la clase `ControllerBean` que será ejecutado.

En las operativas marcadas como 'operative', se pueden encontrar como nodos hijos de la misma, los tag opcionales de 'validation', 'integration' y 'app-data'.

3.2.3. Guided Operative

Las operativas son nodos con la marca o tag 'guided-operative', que identifican el nombre de la operativa y sus atributos, mas otro nodo hijo 'Guidance-class' con los atributos relativos a la clase y el método a invocar:

- **name**: Nombre de la operativa que deberá coincidir con el nombre de la vista(parámetro pageOperation del request).
- **iden**: False/true. Si necesita estar identificado para su ejecución.
- **public**: false/true. Nos dice si la operativa guiada puede ser invocada desde el exterior. Por defecto todas son públicas.
- **class**: Clase `GuidanceClass` seleccionada para ejecutar.
- **method**: Método de la clase `ControllerBean` que será ejecutado.

En las operativas marcadas como 'guided-operative', se pueden encontrar como nodos hijos de la misma, los tag opcionales de 'validation', 'integration', 'app-data', 'breadcrumb', 'preExecution', 'outPutFilter' y 'view'.

3.2.4. OutputFilter

Los filtros de salida son nodos con la marca o tag 'OutPutFilter', que identifican las verificaciones de una serie de parámetros.

- **filterClass**: Clase que contiene el código del filtro de salida.
- **linkedToGuidedOperative**: false/true. Nos dice si está asociado a una operativa guiada.
- **param**: Listado de parámetros que tendrá el filtro.

3.2.5. Validation

La validación de cada operativa, marcada por 'validation', se compone por un atributo de este tag y de sucesivos hijos 'parameter' con los atributos relativos a la validación del parámetro. La lista de atributos de todos ellos es:

- **class-infoset**: Clase que implementa el info de paso de parámetros.
- **name**: Nombre del parámetro. Se admite el literal '*' para validar cualquier parámetro desconocido. Añadiendo el literal '*' al final del nombre del parámetro se valida éste como un plantilla, siendo el literal '*' el sufijo del parámetro en la petición.
- **format**: Formato opcional de expresión regular que ha de tener el valor del parámetro.
- **security-class**: Clase opcional de seguridad externa para la validación del parámetro.

- **security-method**: Método de la clase externa de seguridad a invocar.
- **password**: Marca que nos indica, de cara a su publicación en log, si es una contraseña o parámetro privado. Por omisión no está activo.
- **optional**: False/true. Nos indica que el parámetro es opcional. Su no presencia en la petición o su ausencia de valor es una validación correcta del mismo.
- **compose-new**: Nos indica si se trata de un nuevo parámetro a componer. El método de validación invocado no recibe parámetros.
- **compose-from**: Nos indica la lista de parámetros que componen un nuevo parámetro. Indicando plantillas, un nombre de un parámetro con sufijo '*', en los parámetros del compose-from, se pueden hacer repeticiones de invocaciones al método con cada grupo de parámetros que cumplan la plantilla. El parámetro en el *info* de paso de parámetros ha de extender obligatoriamente de `InfoSetAccesorPlus`, y será en su `ArrayList` donde almacenemos los distintos resultados del método de composición. En los métodos de validación se permite el paso de literales a través del `Controller` marcándolos con 'apóstrofes'.
- **raise-error**: Para parámetros compose-from y compose-new. En caso de fallar la validación compose levanta el error con nombre el valor de este campo.
- **info-error**: False/true. Marca si el parámetro ha de levantar un error en el *info* de paso de parámetros. Por defecto está a cierto.
- return: Nos indica la clase del objeto a guardar en el parámetro del *info* de paso de parámetros.
- **paginated**: Nos indica el número de páginas en que se troceará la respuesta. Página en cuanto al parámetro de page-param, los datos marcados por 'paginated-infoSet'.
- **page-param**: Parámetro que nos indica la página solicitada de caché.
- **paginated-infoSet**: Nombre de las clases de los *InfoSet* a paginar separados por coma.
- **alarm-class**: Clase externa opcional para ejecutar una alarma en caso de que el parámetro no haya sido validado con éxito.
- **alarm-method**: Método a invocar de la clase externa de alarma.

3.2.6. Integration

La integración está marcada por el tag 'integration' y tiene dos atributos relativos al nombre de la clase externa de integración y el método de invocación de la misma. El resto de la estructura del XML es libre, ya que será interpretado por la clase integradora. Sus atributos son:

- **integration-class**: Clase opcional de seguridad externa para la integración de la operativa.
- **integration-method**: Método a invocar de la clase externa de integración.

3.2.7. PreExecution

Las pre ejecuciones, son nodos con la marca o tag 'PreExecution', que realizan la ejecución previa de ciertas operativas.

- **preExecutable**: false/true. Indica si se trata de una pre ejecución.
- **postExecutor**: false/true. Indica si se trata de una post ejecución.
- **operativeToExObject**: Listado de operativas que se ejecutarán.

3.2.8. AppData

El tag 'app-data' identifica una sección para la gestión de los datos capturados por la aplicación. Se pueden tener tantas secciones como necesarias. Se compone de los siguientes atributos mas una lista opcional de nodos hijos 'expand'.

- **key**: Clave que identifica el dato a buscar. Puede hacer referencia a un valor de un info o una concatenación de literales y valores en el info marcado por 'key-from-info' separados por coma. Buscamos la clave 'key' para obtener o guardar el dato en sesión, a nivel de módulo o de aplicación.
- **key-from-info**: Clave para identificar un info de datos en sesión o request donde encontrar la 'key'.
- **field**: Identifica el nombre del campo en el *info* donde buscar las claves por valor de 'open' o 'show' dentro del *info* de 'key-from-info'.
- **no-error**: False/true. Nos indica si permitimos guardar o cargar los datos en caso de que la operativa haya terminado con error. Por defecto no son válidos, false.
- **save**: Directiva para marcar que se van a guardar datos a nivel de sesión del usuario 'session', a nivel de módulo de la aplicación 'module' o a nivel de aplicación para toda la arquitectura 'application'. Las directrices 'save' y 'load' son excluyentes
- **data-infoSet**: Es el nombre de la clase *info* de datos a guardar. Esta ligada al atributo 'save'.
- **load**: Directiva para marcar que se van a leer datos a nivel de sesión del usuario 'session', a nivel de módulo de la aplicación 'module' o a nivel de aplicación para toda la arquitectura 'application'. Las directrices 'save' y 'load' son excluyentes.
- **level**: Indica el nivel de anidamiento sobre la estructura de infos en el que se aplica el 'expand'.
- **show**: All/none/posición/valor. Nos indica qué infos se muestran, todos 'all', ninguno 'none', por posición o por valor o por valor en un info sobre el request o la sesión en 'field'. También se permiten listas de valores o posiciones separadas por coma. Por defecto 'all'.

- **open**: All/none/posición/valor. Nos indica qué infos se abren permitiendo continuar mostrando o abriendo su interior (hijos), todos 'all', ninguno 'none', por posición o por valor o por valor en un info sobre el request o la sesión en 'field'. También se permiten listas de valores o posiciones separadas por coma. Por defecto 'all'. Un info que no ha sido mostrado no puede ser abierto.

3.2.9. ErrorCodes

El tag 'error-codes' identifica el mapeo de errores a realizar sobre la validación para dar la respuesta XML de una micro operativa. Se trata de un *info* o *infoSet* de validación compuesto por el mapeo de errores descritos sobre cada parámetro de la validación. Se compone de un atributo mas un listado de parámetros 'param-error':

- show-multiple-errors: False/true. Indica si se va a levantar más de un error o sólo el primero de la lista.

Para cada parámetro de error 'param-error ', encontraremos lo siguiente:

- name: Nombre del parámetro. Deberá coincidir con un parámetro de la validación de parámetros de entrada del *request*.

- error-code: Error a levantar para el parámetro.

3.2.10. BreadCrumb

El tag 'breadcrumbs' identifica las breadcrumbs (migas de pan). Se trata de una estructura de *infos* que se añade a la respuesta para dar soporte a las breadcrubms en la View. Las breadcrumbs son soportadas también en la View de MC-Server, siendo estas prioritarias sobre las del `Controller` en caso de no poder encadenarse. Se compone de los siguientes atributos:

- **breadcrumb**: True/false. Indica si las breadcrumbs están activas para la operativa.

- **position**: Marca la posición de la breadcrumb o miga a introducir en base a la posición absoluta, mediante un entero sin signo; relativa a la última breadcrumb, mediante un entero con signo '+'/'-'. Por defecto '+', recupera las migas de memoria.

- **load**: Lista de pares valor/enlace de las migas. Cada elemento de la lista corresponde a una miga a cargar a partir de la posición 'position'. Excluye los atributos 'value' y 'link'.

- **value**: Valor/nombre para identificar el breadcrumb.

- **link**: Enlace del breadcrumb.

3.2.11. BL Class

Las clases BL o BussinessLogic son las encargadas de llevar a cabo la lógica de negocio de cada operativa.

El acceso a los datos de la lógica de negocio se puede hacer tanto a través del DB Manager para obtener la información residente en la base de datos, como del Host llamando al Communication Manager. La clase BL puede efectuar tanto llamadas al DB Manager como al Communication Manager.

De su proceso, se debe conseguir como resultado un objeto que extienda de `ResultInformation` y que respete la propiedad de java 'instanceof'. En su contenido, siempre habrá un info de resultado, `ResultInfo`, que informará a la capa superior `Controller` de la correcta ejecución de la operativa. Las listas de *info*, cumplen con el tipo de resultado esperado, `ResultInformation`, con lo que se pueden devolver listas de *infos*, *infoSets*. Si en el proceso de la BussinessLogic, se produjera una anomalía o una necesidad, a partir de la cual se deseara informar al usuario, será en este *info* de resultado donde la BussinessLogic colocará el código de aviso producido.

El *info* devuelto por la clase BussinessLogic, contendrá siempre un *info* de resultado *ResultInfo* que nos indica el estado de la ejecución de la lógica de negocio.

Su contenido es un campo de estado de error, que representa en `ModelEJB` la correcta ejecución de la clase BussinessLogic; dos campos de texto que representan el código de un mensaje y el mensaje en sí, ambos interpretables fuera de la transacción del EJB (para la capa `Controller` o `View`) y un campo de RollBack para marcar a nivel de aplicación al `ModelEJB` que deshaga la transacción.

El módulo de para servicios Web, WebServices, es el encargado de atender las peticiones SOAP a un WebService asociado para encadenarlas bien a la lógica de control, `ControllerBean` en la capa `Controller`, o a la lógica de negocio, BussinessLogic en la capa Model. Para ello se utiliza una clase `WebServiceBean` que será invocada al servicio Web mediante una operativa de servicio Web.

La función de este módulo es el de una capa `Controller`, las peticiones al WebService son tratadas a nivel de operativa por un EJB que atiende el WebService, `WebServiceEJB`, y encadenadas con las clases `WebServiceBean`. Las clases `WebServiceBean` hacen una tarea de `Controller`, bien directa sobre la lógica de negocio a nivel de preparación de datos y recuperación de los mismos para la respuesta; o de una segunda capa de `Controller` en el caso de la invocación a una `ControllerBean`.

El elemento principal es por tanto el EJB `WebServiceEJB`, que se encarga de atender las peticiones especificadas en un fichero XML, denominado `operativas.xml`, que define el comportamiento del EJB. Un conjunto de clases java llamadas WS, WebService, se encargarán de enlazar con el modelo clásico MVC.

3.2.12. CB Class

Las clases CB o ControllerBean, son las encargadas de realizar la comunicación entre una vista y una clase BL.

Se encargará de recibir en una trama de formato de petición, un *infoSet* con los datos necesarios para invocar a la BL. Tras la ejecución de la BL, recogerá la respuesta en una variable de tipo *InfoSetAccessorPlus* y que contendrá información sobre si ha existido algún tipo de error y una variable de tipo *infoSet* con los datos de respuesta. Posteriormente se enviará a otra vista para la maquetación de los datos por pantalla.

3.3. Flujo de Salida

Como ya se indicó en apartados anteriores, todas las capas del sistema tienen un camino de ida y vuelta, excepto la capa View. A continuación, detallaremos cómo se realiza el flujo de vuelta en una aplicación.

Desde la capa Model, en la que nos encontramos cuando se conoce ya el resultado de la ejecución de la operativa, se comienza el retorno hacia la capa Controller.

La clase `ControllerBean` que efectuó la llamada en su momento, recibe la respuesta en un *infoSet*, define la acción para `MCServlet` y le cede el control.

Este *infoSet*, busca la operativa en el fichero `view.xml` y obtiene el nombre del XSL de salida. Transforma el *infoSet* de la aplicación, en un fichero XML de datos, le añade el XML de textos y el XML genérico, y en su caso, el XML de mensajes. Este XML de mensajes sólo contiene el texto definido para el código recibido en el info de resultado, *ResultInfo*.

`MCServlet` cede el control a la capa View transmitiendo el XSL y el XML obtenidos. El transformador construye la información en HTML y el servidor de aplicaciones dispone de esta página para comenzar su tramitación hacia el cliente.

En otro tipo de salidas el flujo en este punto sería distinto, comentamos aquí distintos casos de salidas.

- Si el `view.xml` tiene configurado un atributo 'forward' y un enlace a una aplicación web, se le enviará un request a esa aplicación, para que sea ésta la que elabore la respuesta al usuario.

- Si el `view.xml` tiene configurado un atributo 'redirect' y un enlace a una URL, se le redireccionará con la respuesta de la aplicación componiendo la propia URL, para que sea ésta la que elabore la respuesta al usuario.

Capítulo 4

Tecnologías.

En este capítulo nos centraremos en explicar cuáles han sido las tecnologías utilizadas en el desarrollo del sistema. En la mayoría de los casos, la utilización de tecnologías concretas ha venido impuesto como requisito inicial, en otros casos se ha optado por elegir una opción que se adaptara a las necesidades del propio proyecto.

Las tecnologías que se han utilizado en el desarrollo del proyecto, y que se detallan a continuación, han sido lenguajes de programación, entornos de programación. librerías gráficas y elementos de gestión de proyectos.

4.1. Lenguajes de programación

- Java: Este proyecto se ha programado íntegramente en Java ya que estaba impuesto como requisito inicial. Se ha utilizado la plataforma de compilación Java Standard Edition (J2SE) con la versión JDK6.
- XML: Para el caso particular de los flujos de entrada y salida de datos, se utilizará el lenguaje de etiquetas XML. Concretamente, para la lectura y guardado del controller y del view, se ha utilizado una variante de XML que es con la que trabaja Eclipse, XMI. XMI es un lenguaje basado en XML para compartir modelos UML entre diferentes herramientas de modelado.
- XSL: Se ha utilizado este lenguaje para la maquetación de los resultados por pantalla. Se encarga de tratar las tramas de información en XML y transformarlas en valores que pueda interpretar un explorador en código HTML.
- WSDL: Para la definición del servicio web, se ha utilizado un lenguaje que utiliza XML para la definición de los diferentes servicios.

4.2. Entornos de programación

- Eclipse: Para el desarrollo de la aplicación en Java es necesario utilizar un entorno para compilar y ejecutar el código. En este caso, como necesitamos un plugin para

una plataforma concreta impuesta como requisito inicial, haremos uso de Eclipse para desarrollarlo.

4.3. Librerías gráficas

- GEF: Graphic Editing Framework, es un plugin de eclipse para el desarrollo de editores visuales que pueden ir desde procesadores de texto wysiwyg hasta editores de diagramas UML, interfaces gráficas para el usuario (GUI), etc. Dado que los editores realizados con GEF viven dentro de eclipse, además de poder ser usados conjuntamente con otros plugins, hacen uso de su interfaz gráfica personalizable y profesional. En nuestro caso, se utilizará GEF para la definición de los dos editores visuales.
- EMF: Será necesario utilizar un modelo para representar los datos sobre los que vamos a trabajar. Utilizaremos las librerías EMF, las cuales se basan en dos meta-modelos, el Ecore(contiene información de las clases a definir) y el Genmodel(contiene información adicional para la generación de código). Una de las principales ventajas de utilizar EMF, es que proporciona una visión clara del modelo, generación de interfaces para la creación de objetos y la notificación de cambios en el modelo.
- JDT: El paquete org.eclipse.jdt.core define las clases de las que está compuesto un programa Java. Nos resultará de gran utilidad para cuando sea necesaria la creación de nuevos ficheros o paquetes dentro de un proyecto.

4.4. Gestión del proyecto

- MS Project: Es un software de administración de proyectos de la compañía Microsoft para asistir a administradores de proyectos en el desarrollo de planes, asignación de recursos, dar seguimiento al progreso, administrar presupuesto y analizar cargas de trabajo. Es el sistema que se utiliza en la empresa, y por tanto, se utilizó para la gestión de dicho proyecto.

Parte III

Desarrollo e Implementación

Capítulo 5

Análisis del sistema.

En este capítulo nos centraremos en realizar la especificación de requisitos del sistema. Diferenciaremos entre requisitos funcionales, no funcionales y algunos aspectos sobre la interfaz de usuario.

5.1. Requisitos funcionales

R.F. 1: Existirán dos editores gráficos: Controller Model Editor(modelador de operativas) y View Model Editor(diseñador de formularios). De esta manera se podrán editar las propiedades de los componentes de una manera más intuitiva.

R.F. 2: El View Model Editor(diseñador de formularios) se encargará de crear la estructura visual del formulario web, previo a la generación del código HTML.

R.F. 3: El Controller Model Editor(modelador de operativas) se encargará de tramitar las operativas de banca electrónica y generar vistas que sean necesarias.

R.F. 4: El diseñador permitirá la creación de vistas asociadas a operativas. Cada operativa podrá tener como hijos un breadcrumb, un resource y/o un form.

R.F. 5: El diseñador permitirá añadir componentes básicos al formulario, editar sus propiedades, cambiar su posición en pantalla y/o eliminarlos del editor.

R.F. 6: Los componentes básicos con los que trabajará el diseñador de formularios serán: Label, TextBox, Password, Button, SubmitButton, ComboBox, RadioButton, CheckBox y FileComponent.

R.F. 7: El diseñador permitirá añadir componentes avanzados al formulario, editar sus propiedades, cambiar su posición en pantalla y/o eliminarlos del editor.

R.F. 8: Estos componentes avanzados serán: DateComponent, Conjuntos de botones y CCC.

R.F. 9: El diseñador permitirá añadir eventos a los componentes del formulario. También dará la posibilidad de asociar una función javascript a los eventos.

R.F. 10: El diseñador permitirá guardar la información en el fichero con extensión .view.

R.F. 11: El modelador permitirá la creación de operativas, micro operativas, operativas guiadas y filtros de salida asociados a un controller.

R.F. 12: El modelador permitirá añadir nuevas micro operativas y/o editar sus propiedades.

R.F. 13: Una micro operativa podrá tener una CB Class como objeto hijo.

R.F. 14: Se permitirá la generación de clases CB(Controller Bean). Además se permitirán generar las clases resultado de tipo Info y el código de la validación relacionada.

R.F. 15: El modelador permitirá añadir objetos a una operativa guiada, editar sus propiedades, cambiar su posición en pantalla y/o eliminarlos del editor.

R.F. 16: Los objetos que se podrán añadir a la estructura de una operativa guiada serán: AppDataObject, GuidedOperativeObject, IntegrationDataObject, OutPutFilterObject, PreExecutionObject, ResultViewObject y ValidationObject.

R.F. 17: El modelador permitirá crear conexiones de tipo OK, ERROR, OTHER, SinEtiqueta o VALIDATION-ERROR entre GuidedOperative y ResultView. Y también permitirá crear conexiones SinEtiqueta desde un GuidedOperative hacia cualquier otro elemento.

R.F. 18: Una operativa guiada podrá tener como objetos hijos: preExecutionObject[0..1], integrationObject[0..N], guidanceClass[0..1], outPutFilter[0..N], appDataObject[0..N], breadcrumb[0..1], validation[0..1] y resultViewObject[0..N].

R.F. 19: Una operativa guiada no podrá tener ningún resultView asociado si no posee un hijo de tipo guidanceClass.

R.F. 20: Un resultView podrá tener una conexión directa con uno o varios guidedOperativeObject.

R.F. 21: Al crear un resultView se podrá seleccionar el tipo de vista(PAGE, BINARY, STRING, XML, FILE o JSP) y el tipo de conexión.

R.F. 22: El modelador permitirá guardar la información en el fichero con extensión .controller. Al proceder al guardado del controller, se guardarán en el fichero .view las vistas que no estuvieran ya creadas.

R.F. 23: El sistema contendrá una opción para validar que los datos introducidos en el controller y en el view son correctos.

R.F. 24: El sistema permitirá la conversión de los ficheros de la versión 4.0 de MCServer a la versión 5.0 y viceversa.

R.F. 25: El sistema permitirá la generación de clases BL(o Bussiness Logic) para la conexión con Host, con una base de datos o con un servicio web. Además de crear la clase BL, se crearán las clases Info e InfoSet asociadas.

R.F. 26: Las conexiones con Host será necesario que contengan una dirección URL y un puerto asociado.

R.F. 27: Las conexiones con bases de datos permitirán seleccionar el driver a utilizar(DB2, MySQL, Oracle o SQL Server).

R.F. 28: Las conexiones con servicios web, permitirán seleccionar el fichero WSDL con el que trabajar mediante una dirección URL, seleccionando una ruta del sistema o un fichero por defecto.

R.F. 29: El sistema permitirá la generación de ficheros EJB(Enterprise Java Bean) de tipo jar para la conexión con diferentes tipos de base de datos.

R.F. 30: Los tipos de EJB que podrán generarse serán de la version 1.3, 2.0 y para el servidor de aplicaciones J2EE webLogic.

5.2. Requisitos no funcionales

R.N.F. 1: La aplicación deberá de funcionar sobre la plataforma de desarrollo Eclipse 3.7 o superiores.

R.N.F. 2: El java development kit(JDK) deberá de ser 1.6 o superior.

R.N.F. 3: La aplicación será multiplataforma para poder ejecutarse en cualquier sistema operativo.

R.N.F. 4: El sistema sobre el que se ejecute deberá de tener un mínimo de 1GB de memoria RAM.

R.N.F. 5: Se informará por pantalla en caso de introducir algún dato erróneo en la aplicación.

R.N.F. 6: En todo momento, el usuario estará informado sobre el lugar en el que se encuentra para facilitar su navegación por la aplicación.

R.N.F. 7: Se podrá llegar a todas las opciones de la aplicación en 3 clics o menos.

5.3. Interfaz de usuario

La usabilidad de la interfaz gráfica de usuario (GUI) es un aspecto muy importante en este proyecto, ya que va a ir destinado a usuarios con diversos tipos de conocimientos informáticos. Los editores contarán con paletas gráficas que facilitarán al usuario a la hora de añadir nuevos elementos. En el editor se mostrarán los elementos con sus propiedades de manera que el usuario se evite hacer demasiados clics. Todas las ventanas de la aplicación contendrán un título y una breve descripción para que el usuario sepa lo que está realizando en cada momento. Cada uno de los elementos tendrá un color representativo para facilitar la interpretación más intuitiva.

Capítulo 6

Diseño.

En este capítulo se presentarán los detalles de diseño del proyecto, basándonos en el análisis del capítulo anterior. Se realizarán una serie de diagramas que ayudarán a la hora de realizar la implementación.

6.1. Casos de uso

Se ha optado por utilizar notación UML para la realización del diagrama de casos de uso del sistema. Representa la funcionalidad más básica y general, y además, define todas las posibles interacciones del sistema con los usuarios. La imagen que mostramos a continuación detalla el diagrama de casos de uso utilizado por el plugin de MCServer y en el que se pueden diferenciar claramente los grandes bloques en los que se subdivide la aplicación.

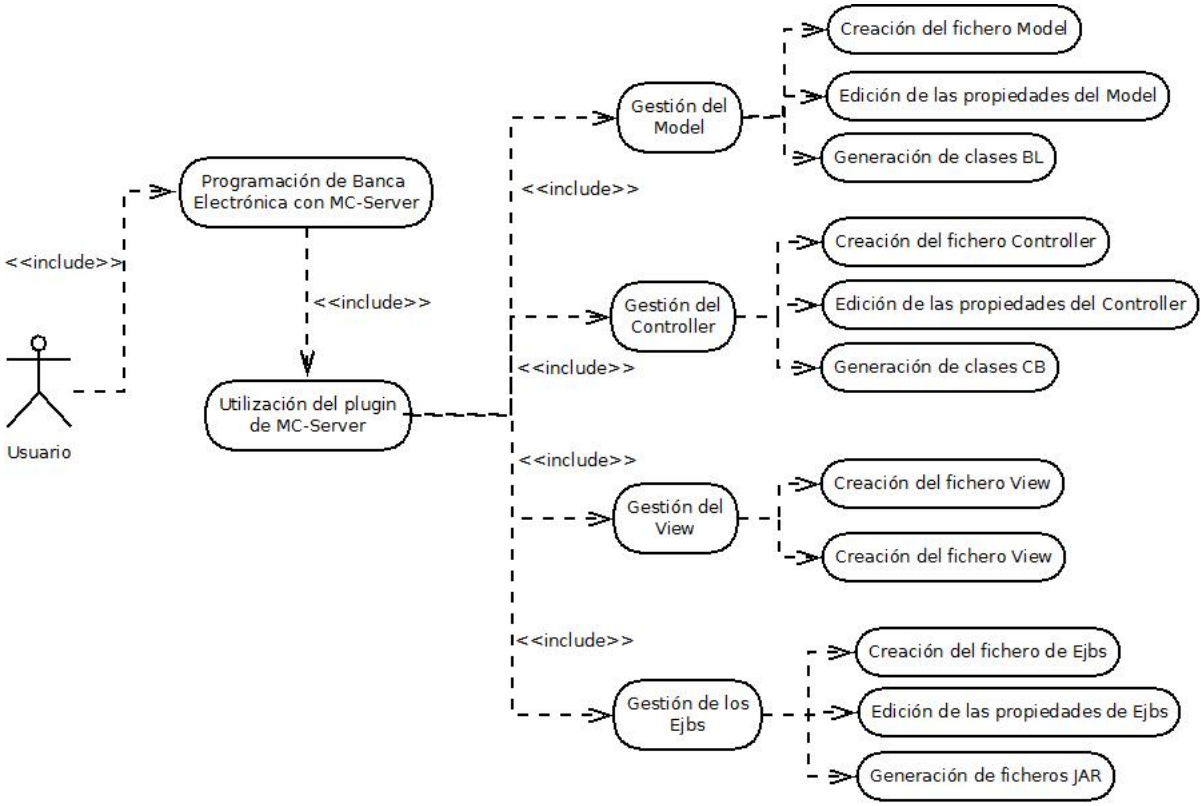


Figura 6.1: Diagrama de casos de uso

6.2. Modelo de clases

A la hora de realizar el diagrama de clases completo, se vio imposible condensar todo el sistema en un único diagrama, por lo que se decidio subdividirlo en varios submodelos. Además, se ha prescindido de representar todas las funciones de cada clase para obtener una mayor claridad y facilidad de comprensión.

6.2.1. Diagrama de clases del diseñador

En el primer diagrama, figura 6.2 representa el modelo de clases del diseñador de formularios. Posee una clase base, *AbstractEditableObject*, de la que heredan las propiedades y métodos las demás clases. A parte, está la clase *AbstractSubcomponent*, de la que extienden todos los subcomponentes ya que poseen muchos atributos similares.

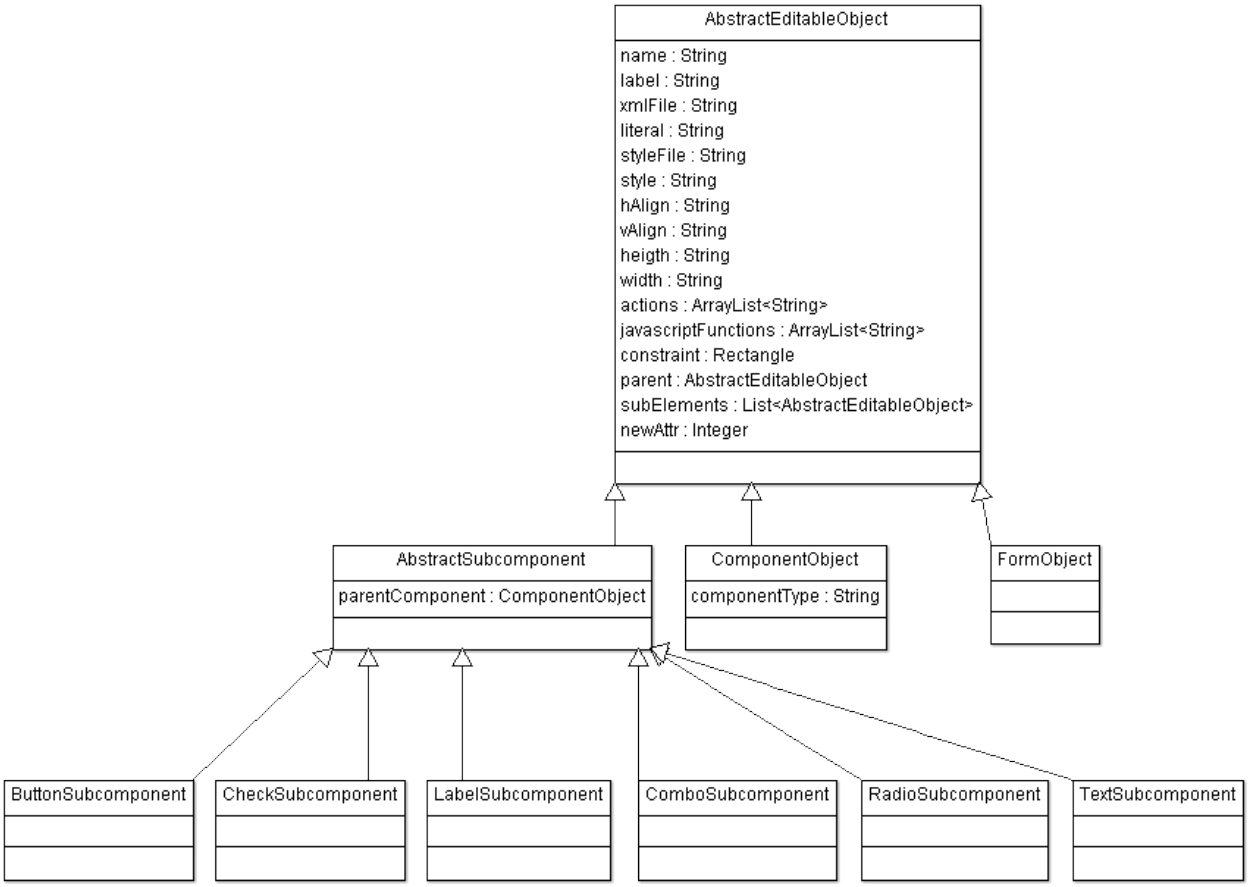


Figura 6.2: Diagrama de clases del diseñador de formularios

6.2.2. Diagrama de clases del modelador

El diagrama de la figura 6.3 representa el modelo de clases del modelador. Posee una clase base, *AbstractConnectableObject*, de la que heredan las propiedades y métodos las demás clases. Se caracteriza por tener una lista de conexiones de origen y de destino para interconectar diferentes objetos. A parte, está la clase *AbstractEditableObject*, de la que extienden todos los objetos que se encuentran internos en algún objeto *AbstractConnectableObject*.

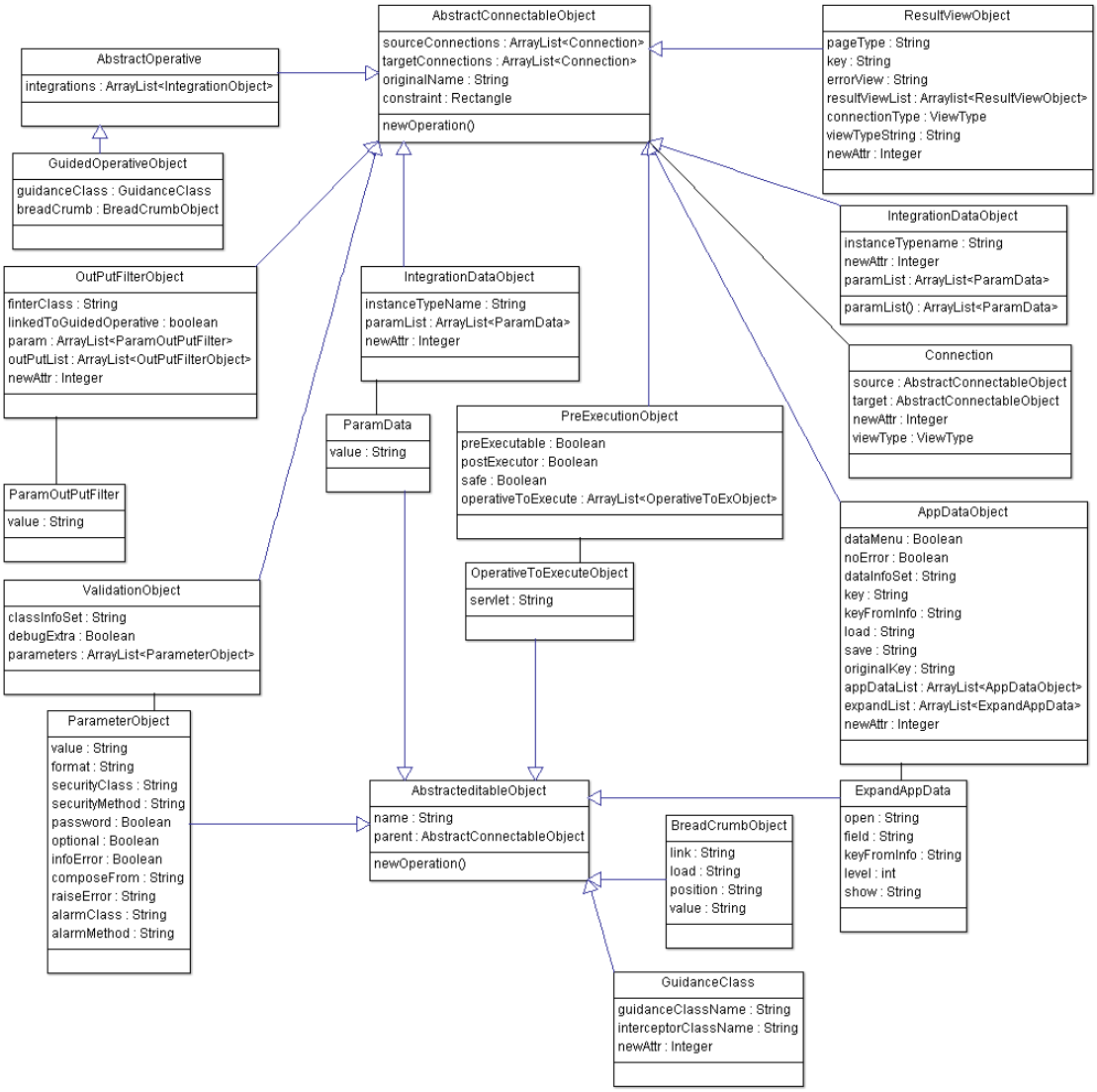


Figura 6.3: Diagrama de clases del modelador

6.2.3. Diagrama de clases de las partes editables del diseñador

El diagrama de la figura 6.4 representa las clases de los componentes editables del formulario. Extienden de la clase abstracta *AbstractGraphicalEditPart* que contiene la figura que será editable y las conexiones que tiene con otros elementos. Además, implementa el método *getCommand()* que será el que se ejecute cuando se produzcan los eventos de creación, edición o borrado de un elemento editable.

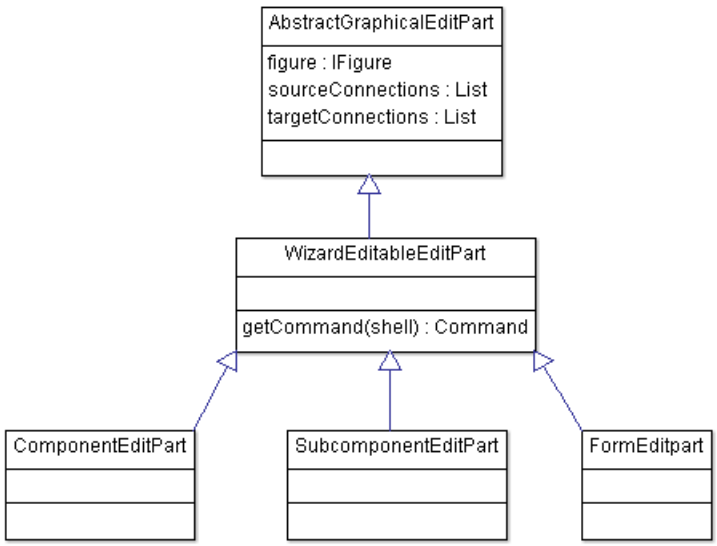


Figura 6.4: Diagrama de clases del diseñador de formularios

6.2.4. Diagrama de clases de las partes editables del modelador

El diagrama de la figura 6.5 representa las clases de los componentes editables del modelador de operativas. Extienden de la clase *AbstractGraphicalEditPart*. La clase *AbstractContainerEditPart* posee un atributo que representa el color del elemento por pantalla, y otro atributo que nos indica si posee subcomponentes internas. Como ocurría en el apartado anterior, también implementa el método *getCommand()* que será el que se ejecute cuando se produzcan los eventos de creación, edición o borrado de un elemento editable. Además, hay una clase *ConnectionEditPart*, para el caso particular de la edición de las líneas de conexión entre objetos, ya que no posee las mismas propiedades que los objetos contenedores.

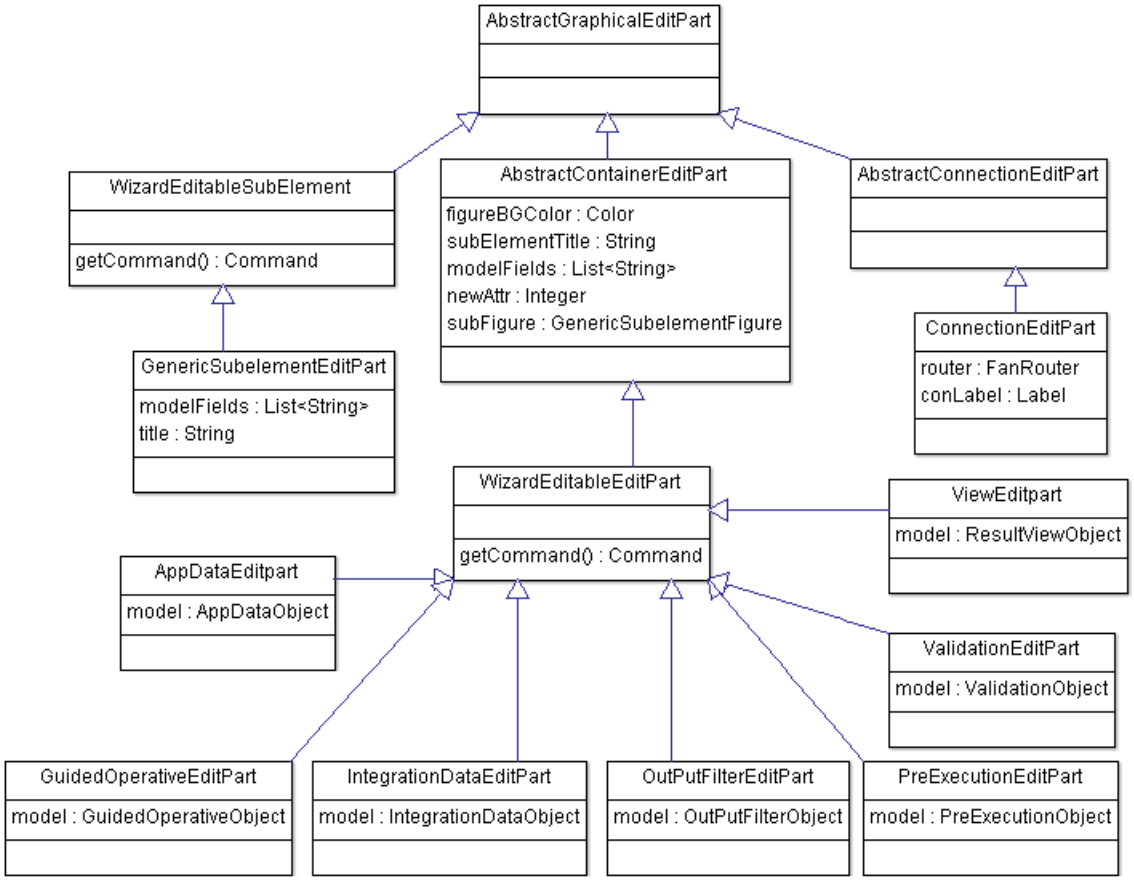


Figura 6.5: Diagrama de clases del modelador de formularios

6.2.5. Diagrama de clases de la vista del diseñador

El diagrama de la figura 6.6 representa las clases que se encargan de dibujar los diferentes elementos en el editor de Eclipse. La clase *AbstractControlFigure* implementa el método *paintFigure* que pintará los rectángulos que representan los componentes del formulario. Todas las demás clases, poseen como atributos los objetos del modelo con todas las propiedades asociadas y que dependiendo del tipo se dibujarán unos elementos u otros por pantalla.

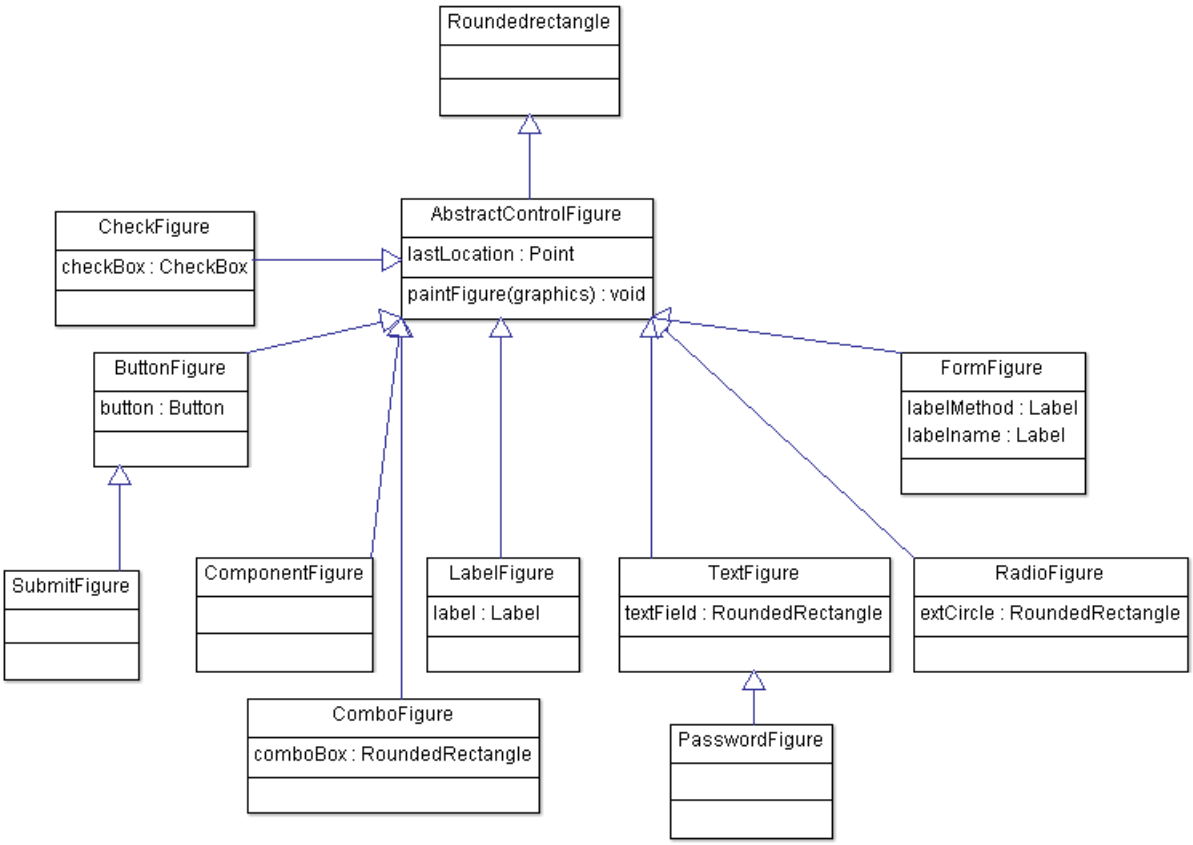


Figura 6.6: Diagrama de clases de la vista del diseñador

6.2.6. Diagrama de clases de la vista del modelador

El diagrama de la figura 6.7 representa las clases que se encargan de dibujar los diferentes elementos en el modelador. La clase *AbstractRectangleFigure*, como en el apartado anterior, implementa el método *paintFigure* que pintará los rectángulos que representan los diferentes componentes.

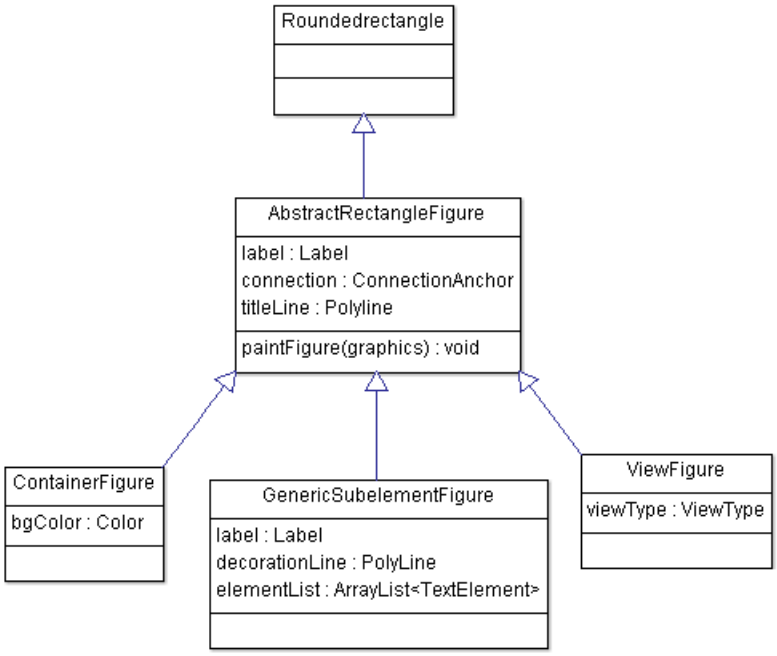


Figura 6.7: Diagrama de clases de la vista del modelador

6.3. Diseño Visual

El aspecto visual de la aplicación resulta de gran importancia en este proyecto ya que podría ser utilizado por cualquier tipo de usuarios. De este modo, se intentó que resultara atractivo y evitando un diseño sobrecargado.

Las *interfaces gráficas* tenían que mostrar toda la información necesaria intentando ser lo más minimalista posible y por su puesto buscando la claridad. Para ello se decidió utilizar ventanas típicas del sistema operativo a la hora de trabajar con los elementos del editor gráfico. En estas pantallas se han distribuido los diferentes campos de introducción de datos de una manera que resulte intuitivo para el usuario. Por otra parte, las pantallas de edición de elementos se realizaron de manera similar a las de inserción pero rellenadas con los datos actuales del elemento en cuestión. También se decidió que existiese una navegación sencilla entre pantallas, para evitar sobrecargar una sola con demasiada información.

La *usabilidad* fue uno de los puntos clave a considerar para abarcar la mayoría de los usuarios posible. Entre las características más importantes a destacar, la existencia de una paleta gráfica en el lado izquierdo. Resulta muy sencillo añadir nuevos elementos al editor únicamente realizando un clic sobre el deseado. Otro aspecto destacable fue la utilización de colores distintivos para los diferentes elementos. De esta manera los usuarios pueden identificar de manera más rápida los tipos de elementos en pantalla. Por último se intentó que la creación de elementos se realizará en un máximo de 3 clics.

Capítulo 7

Implementación y pruebas.

Como ya se ha comentado en capítulos anteriores, la implementación del plugin para Eclipse será programada principalmente en Java, utilizando otros lenguajes auxiliares para su correcto funcionamiento.

En este apartado, se describirá la estructura interna de la aplicación, así como los aspectos más destacables en la implementación de la misma y el plan de pruebas seguido para verificar su correcto funcionamiento.

7.1. Implementación

El desarrollo del plugin de Eclipse se va a subdividir en varios proyectos que dependen unos de otros, pero así se consigue no sobrecargar toda la implementación en un único proyecto.

De esta manera, la estructura general que se va a utilizar es la que aparece a continuación en la imágen y de la cual explicaremos cada proyecto con más detalle.

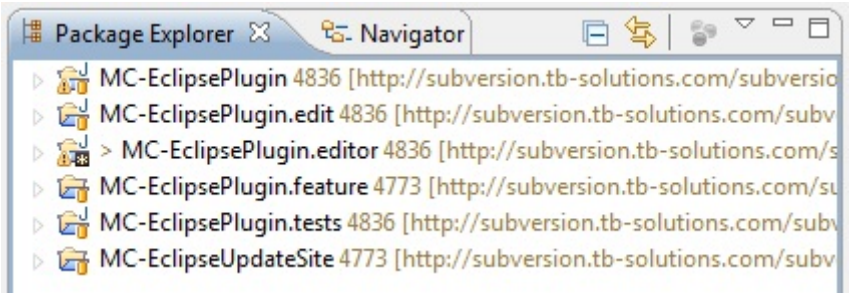


Figura 7.1: Estructura de proyectos en Eclipse

7.1.1. MC-EclipsePlugin

Es un proyecto de tipo 'Java Project', y es el encargado de almacenar los ficheros de configuración del modelo del plugin, y fue la primera de las tareas que hubo que realizar.

Estos ficheros son de tipo Ecore(Eclipse Core) y permiten definir de manera arborescente cómo va a ser la estructura de los elementos. Esta estructura será utilizada por la librería EMF(Eclipse Modeling Framework) de Eclipse para la generación de código que pueda utilizarse para implementar el plugin. Además, tendremos otro tipo de ficheros GenModel, que se asociará al fichero Ecore correspondiente y que utilizaremos para la autogeneración de código del editor, del modelo y para las pruebas unitarias. A continuación mostramos una imagen con el menú desplegable de la generación de codigo a partir del fichero GenModel.

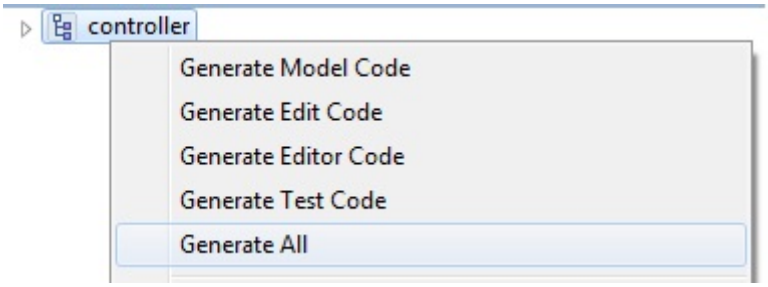


Figura 7.2: Fichero GenModel

Para la autogeneración de código, habrá que editar una serie de propiedades en el fichero GenModel como mostramos a continuación.

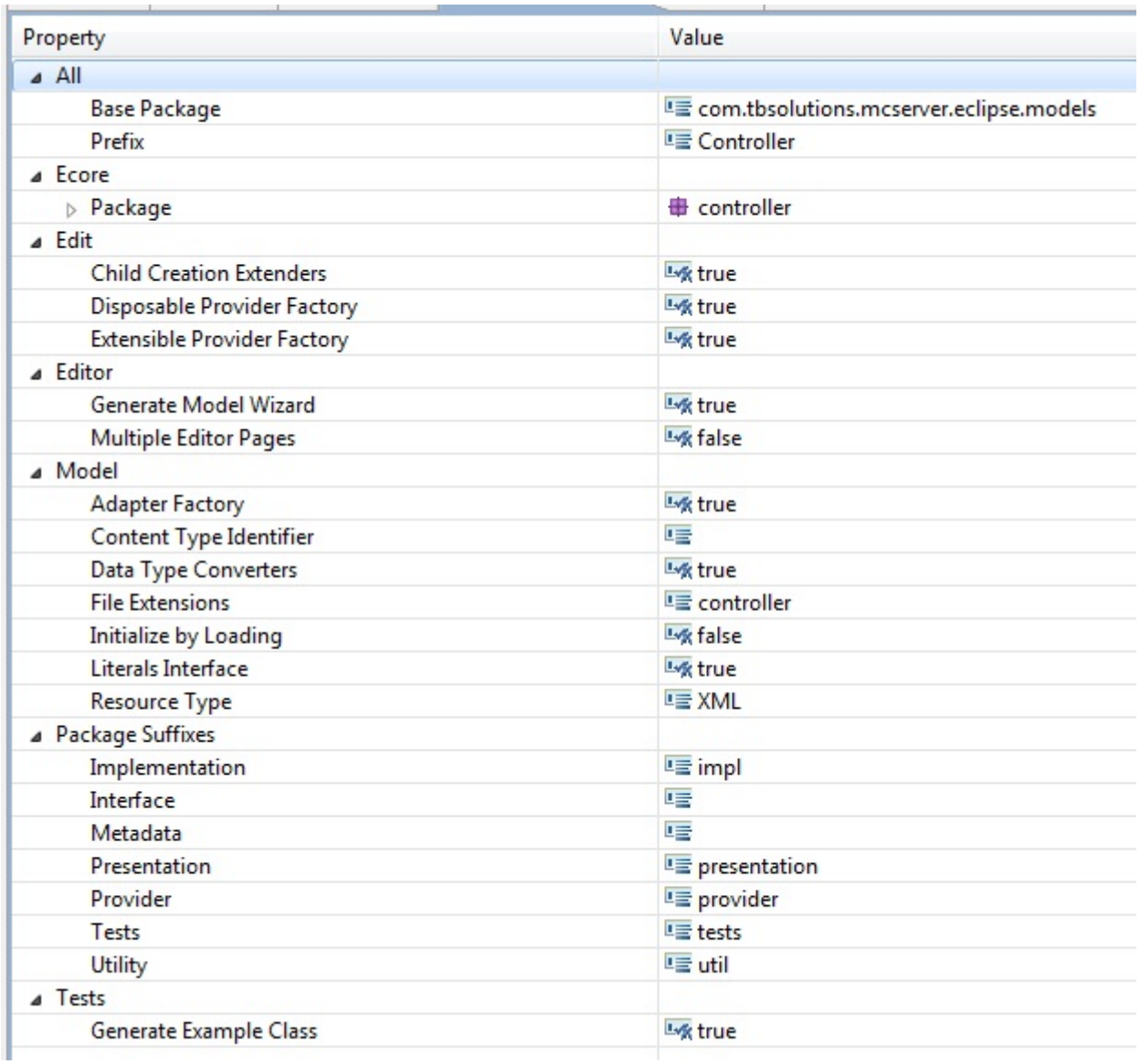


Figura 7.3: Propiedades del fichero GenModel

En este proyecto, estará tambien el fichero plugin.xml. Podremos editar una serie de propiedades del mismo, dependencias entre los diferentes proyectos, y librerías, que nos servirán para la creación global de todo el plugin. Mostramos a continuación una imagen de la edición de este fichero.

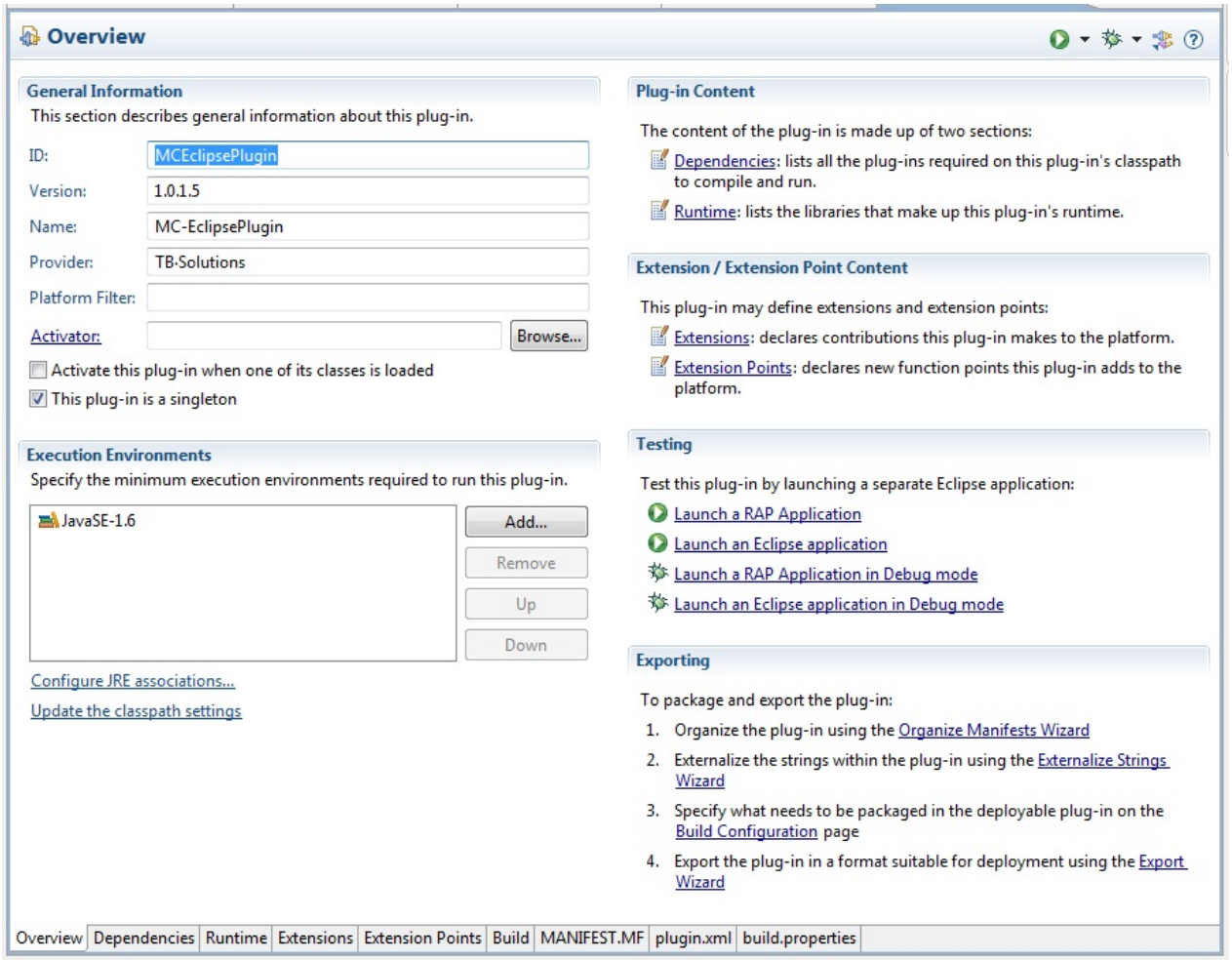


Figura 7.4: Fichero plugin.xml

7.1.2. MC-EclipsePlugin.edit

Es un proyecto de tipo 'Java Project', y es el encargado de almacenar los ficheros que se han autogenerado con el fichero GenModel explicado en el apartado anterior. Estas clases nos servirán para obtener las instancias a los objetos del modelo y para almacenar o modificar sus propiedades. Mostramos una imagen con la estructura de los paquetes generados en este proyecto.

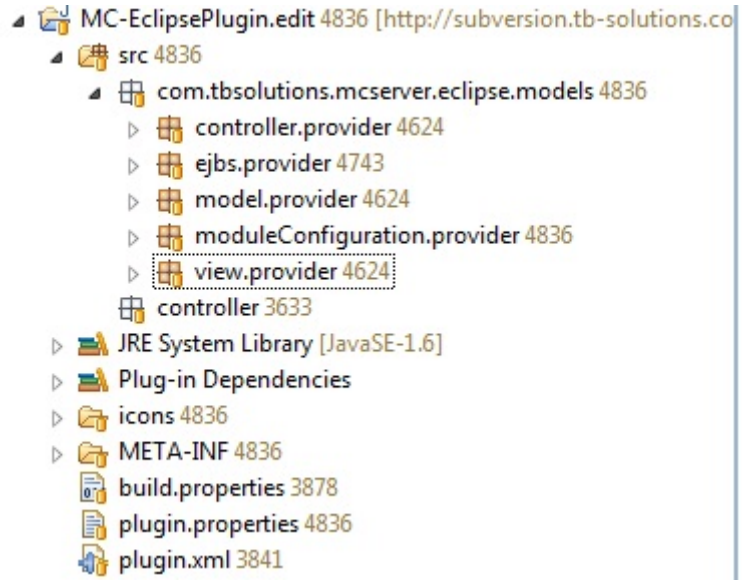


Figura 7.5: Fichero plugin.xml

7.1.3. MC-EclipsePlugin.editor

Es un proyecto de tipo 'Java Project', y se podría decir que abarca el grueso de la implementación del plugin. Tendrá los paquetes referentes a la implementación de los dos editores gráficos(designer y modeler), otro paquete con las clases autogeneradas para mostrar la estructura arborescente de los ficheros(models) y otro paquete para acciones variadas del plugin(editor.actions).

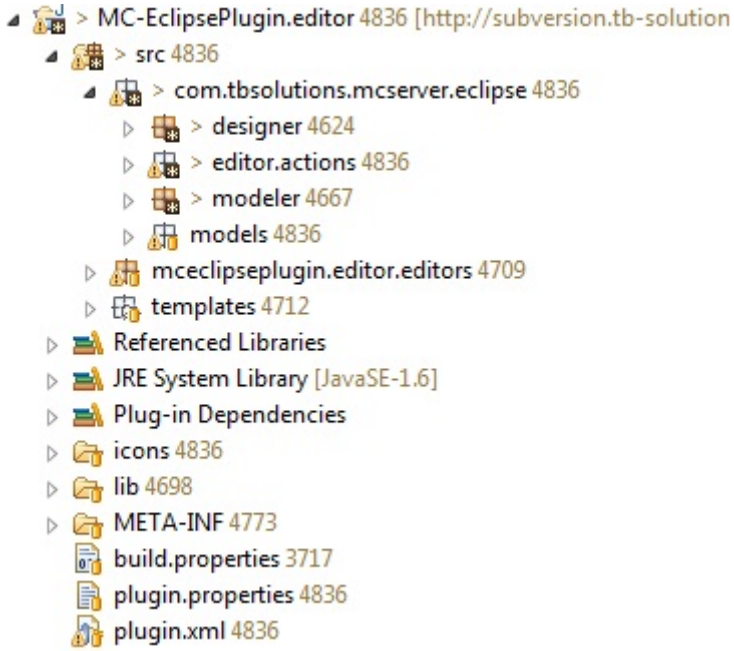


Figura 7.6: Estructura del editor

Explicaremos en más detalle a continuación los aspectos más importantes de la implementación del editor gráfico de la vista y del controlador. Para estos dos editores, existen clases similares, asi que nos centraremos en detallar la estructura general de la implementación independientemente de cual se trate. En el apartado anterior de diseño se puede apreciar con mayor detalle las principales diferencias en las clases de cada uno de los dos editores.

a) Package *com.tbsolutions.mcserver.eclipse.designer.model*

Contiene todas las clases base que utilizará el diseñador(las principales serán FormObject para el formulario, ComponentObject para todas las componentes, AbstractSubcomponent clase abstracta de la que extenderán los subcomponentes). Todas las clases base, extenderán de la clase abstracta AbstractEditableObject, la cual poseerá todas las propiedades comunes (literal, estilo, alineaciones, tamaños, elemento padre, lista de subelementos y el rectángulo gráfico). Además extenderá de la clase Observable para que puedan realizarse las notificaciones de cualquier cambio sobre un elemento y producirse su actualización en pantalla. La relación que existirá entre los elementos será la que se muestra en la imagen a continuación:

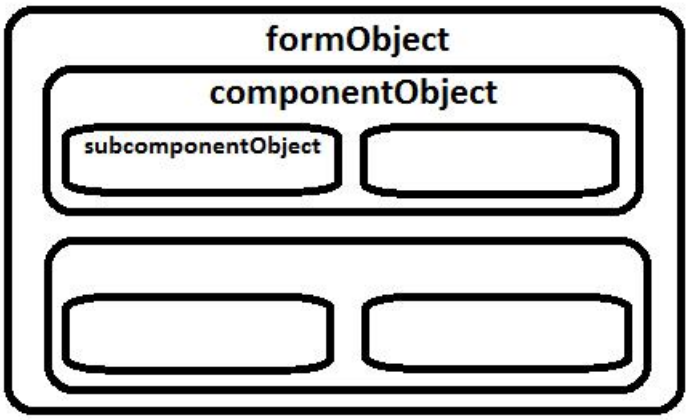


Figura 7.7: Relación entre elementos del diseñador

Además, este package contiene un fichero *componentes.xml* desde el cual se crearán dinámicamente todos los componentes. De esta manera resulta muy cómodo realizar modificaciones sobre los elementos sin tener que hacer excesivos cambios. Se adjunta a continuación un fragmento de este fichero:

```
<!--Fichero XML que contiene la estructura de los subcomponentes -->
<catalog>
    <component name="LabelComponent">
        <subcomponent>Label</subcomponent>
    </component>
    <component name="TextComponent">
        <subcomponent>Label</subcomponent>
        <subcomponent>Text</subcomponent>
    </component>
    <component name="PasswordComponent">
        <subcomponent>Label</subcomponent>
        <subcomponent>Text</subcomponent>
    </component>
    <component name="ButtonComponent">
        <subcomponent>Button</subcomponent>
    </component>
```

Figura 7.8: Fragmento del fichero componentes.xml

b) Package *com.tbsolutions.mcserver.eclipse.designer.view*

Contiene todas las clases necesarias para dibujar por pantalla todos los elementos. Estas clases extienden de *AbstractControlFigure* que a su vez extiende de *RoundedRectangle* y contiene las propiedades comunes de todos ellos. La función principal es la de *paintFigure*, que editará el rectángulo de cada elemento y lo posicionará en el gridData correspondiente. Por claridad a la hora de mostrar el formulario en pantalla se decidió que todos los elementos fueran rectángulos.

c) Package *com.tbsolutions.mcserver.eclipse.controller*

Contiene las clases necesarias para la creación de los editPart(objetos sobre los que está permitido realizar acciones) de las diferentes componentes. Todas las clases extienden de *Observer* y de *NodeEditPart*. La función *createEditPolicies* nos permitirá añadir las acciones que permitimos sobre ese objecto (edición, borrado, cambio de posición, cambio de tamaño,). Mostramos a continuación un ejemplo de editParts:

Figura 7.9: Ejemplo de EditPart

d) Package *com.tbsolutions.mcserver.eclipse.designer.wizard*

Contiene toda la definición de los wizard de creación/edición de elementos del formulario. Se pretende que se pueda configurar de manera sencilla todas las propiedades que tiene asociadas y poder observar el resultado al cerrar el wizard. Los datos introducidos se almacenarán en el fichero .view y se cargarán al abrirlo en el editor gráfico.

e) Package *com.tbsolutions.mcserver.eclipse.designer.command*

Contiene las clases de definición de los comandos de creación y/o edición para cada componente. Son las clases desde donde se invoca a los wizard. Las clases extienden a la clase *Command* y *EditableCreateCommand*, y cada elemento tiene asociado comandos propios de creación, ya que tienen que invocar a wizards diferentes y con propiedades diferentes.

f) Package *com.tbsolutions.mcserver.eclipse.designer.palette*

Contiene la clase OperativeGraphPalette que se encarga de definir la paleta gráfica con las componentes que se podrán insertar en el formulario. El aspecto final de la paleta es como el que mostramos a continuación:

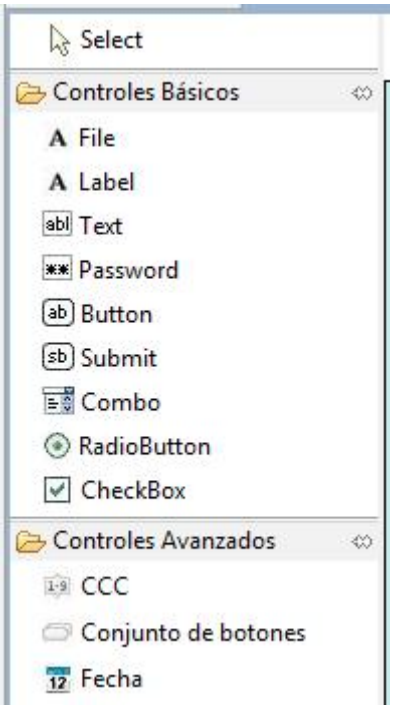


Figura 7.10: Paleta gráfica del diseñador de formularios

7.1.4. MC-EclipsePlugin.feature

Es un proyecto de tipo 'Project' y únicamente contendrá un fichero feature.xml donde se detallan las características de cada versión del plugin y que aparecerán cuando un usuario quiera descargarselo desde el servidor de actualizaciones de Eclipse. Desde este mismo fichero se podrán crear versiones para desplegar en el servidor(deployable features). Mostramos a continuación una imagen con las principales propiedades que se pueden editar del plugin.

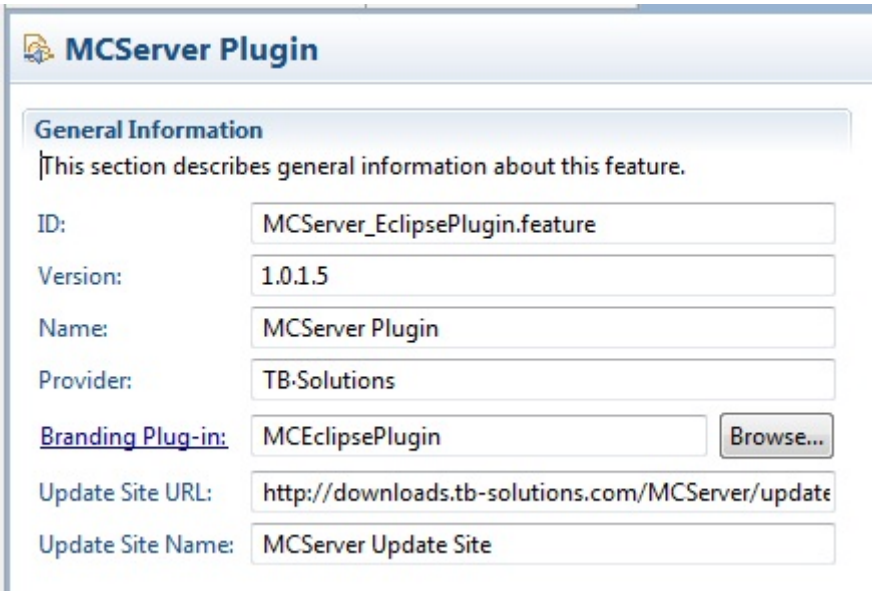


Figura 7.11: Feature

7.1.5. MC-EclipsePlugin.tests

El MC-EclipsePlugin.tests, es un proyecto de tipo 'Java Project' y se encargará de almacenar las clases Java que se utilizarán para la realización de los test unitarios.
Para realizar la ejecución de los mismos, utilizaremos un complemento de Eclipse que se llama *JUnit* y que explicaremos con más detalle en la sección de pruebas.

7.1.6. MC-EclipseUpdateSite

El `MC-EclipseUpdateSite`, es un proyecto de tipo 'Update Site Project' y su principal funcionalidad es la de establecer la estructura necesaria para la plataforma de publicación de plugins Eclipse Update Manager. De esta manera, mediante una url se podrá descargar y actualizar en cualquier momento el plugin.

A este proyecto se le vincula el proyecto `MC-EclipsePlugin.feature` visto en apartados anteriores que tendrá el contenido a publicar. El fichero principal del proyecto es site.xml y nos permite definir las categorías a visualizar en el Eclipse Update Manager. En nuestro caso, únicamente hemos definido una sola categoría como mostramos en la figura.

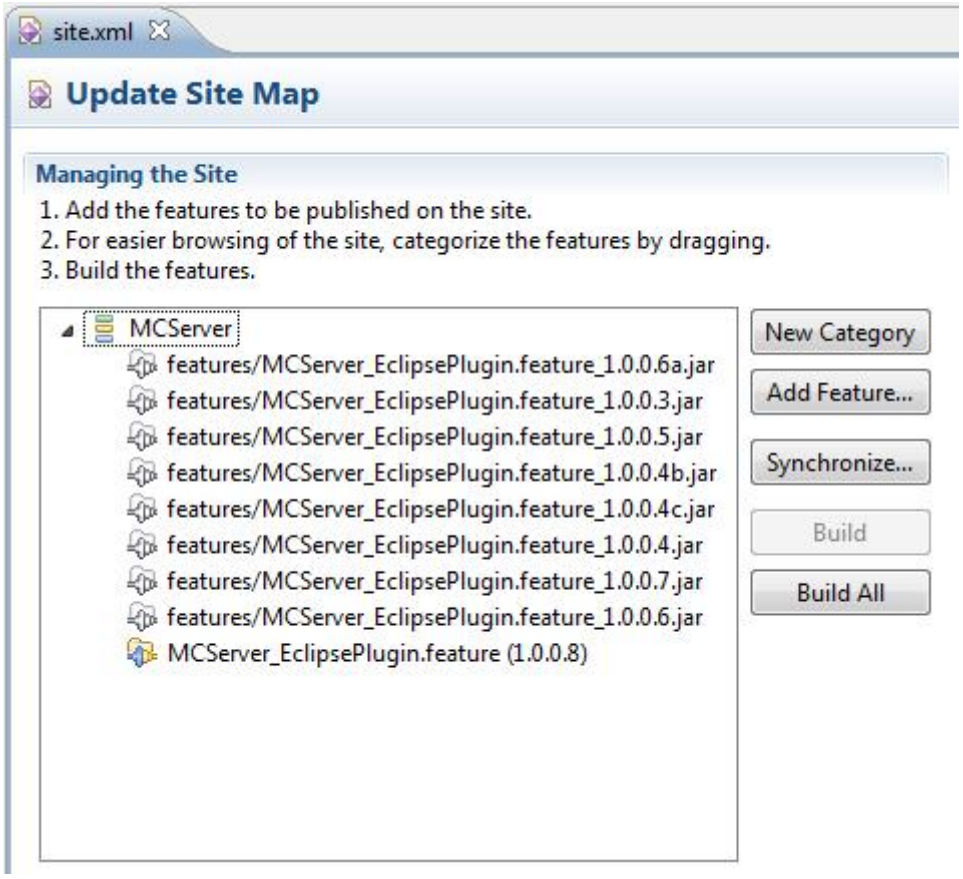


Figura 7.12: Update Site

Finalmente, solo hay que subir al servidor web, mediante un gestor FTP, el fichero features con la última actualización. La url donde se encuentra el repositorio de versiones finales es:

MCServer Update Site - http://downloads.tb-solutions.com/MCServer/update

7.2. Pruebas

La fase de pruebas de un sistema es una de las etapas más importantes del proceso de desarrollo software. Para ello es necesario realizar una batería de pruebas que permitan verificar el correcto funcionamiento de la aplicación. En el caso del este plugin, es difícil realizar un proceso automático que verifique que no existen errores., por ello únicamente se han realizado pruebas unitarias de todas las clases por separado y pruebas globales de todo el sistema.

a) Pruebas Unitarias

En proyectos de gran envergadura y de muchas clases, resulta muy tedioso realizar las pruebas unitarias para cada una de las clases. En nuestro caso, se optó por utilizar JUnit, un complemento para Eclipse que nos permite ejecutar las pruebas de manera sencilla mediante la clase abstracta *TestRunner*.

Para la ejecución de una prueba, seleccionaremos la opción JUnit como mostramos en la imagen a continuación.

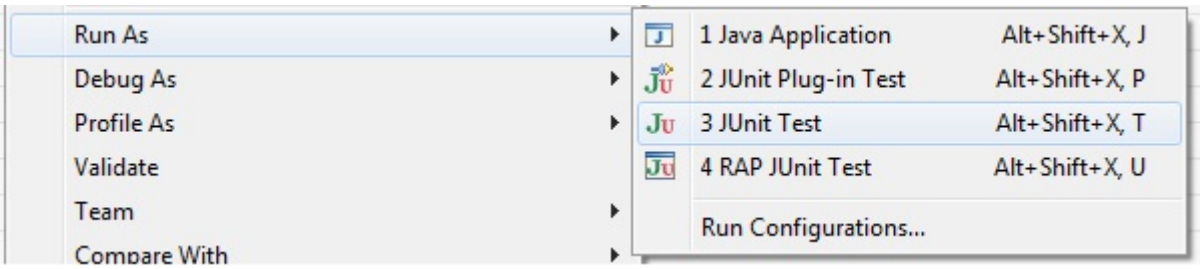


Figura 7.13: Ejecución JUnit

Al realizar la ejecución de una prueba unitaria utilizando JUnit, se mostraría en una pestaña el resultado como mostramos a continuación.

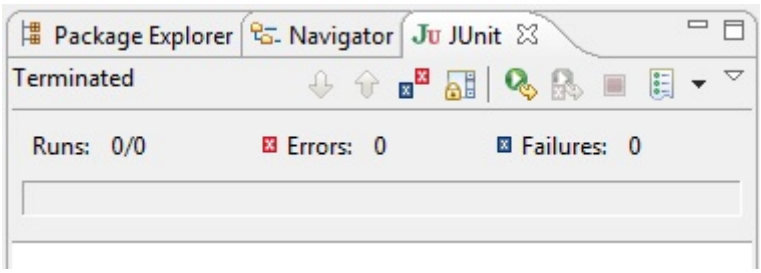


Figura 7.14: Resultados JUnit

b) Pruebas de Sistema

Para probar el plugin completo, se realizó manualmente, ya que resultaba de gran dificultad hacer un fichero que autocomprobara el sistema, ya que la mayor parte es gráfica. Conforme se iba realizando la prueba, se modificaba la implementación para corregir posibles errores.

7.3. Aspecto final del plugin

Para terminar, mostraremos unas cuantas imágenes del aspecto final del editor de formularios y del modelador de operativas, así como algunas de sus funcionalidades más destacables.

El aspecto arborescente de los ficheros .view y .controller se observa a continuación, y desde el cual se podrán editar las propiedades de cada uno de sus componentes y añadir nuevos.

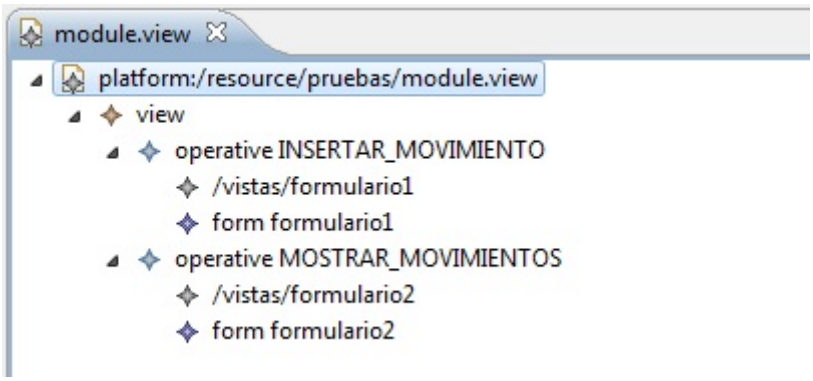


Figura 7.15: Fichero view

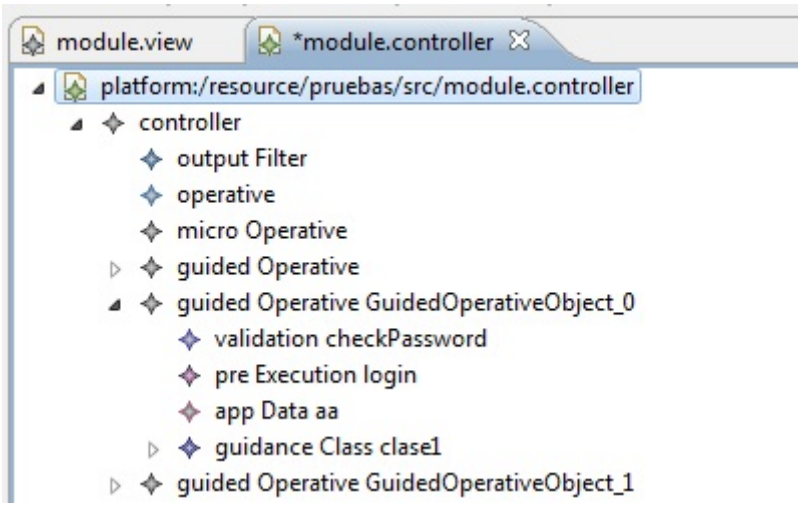


Figura 7.16: Fichero controller

Y el aspecto visual del editor gráfico de formularios ha adquirido el siguiente aspecto, donde apreciaremos la vista preliminar del formulario, y la zona de la paleta para realizar la inserción de nuevos elementos.

Figura 7.17: Editor gráfico de formularios

Y el aspecto visual del modelador de operativas ha adquirido el siguiente aspecto, donde apreciaremos la vista preliminar del controlador, y la zona de la paleta para realizar la inserción de nuevos elementos.

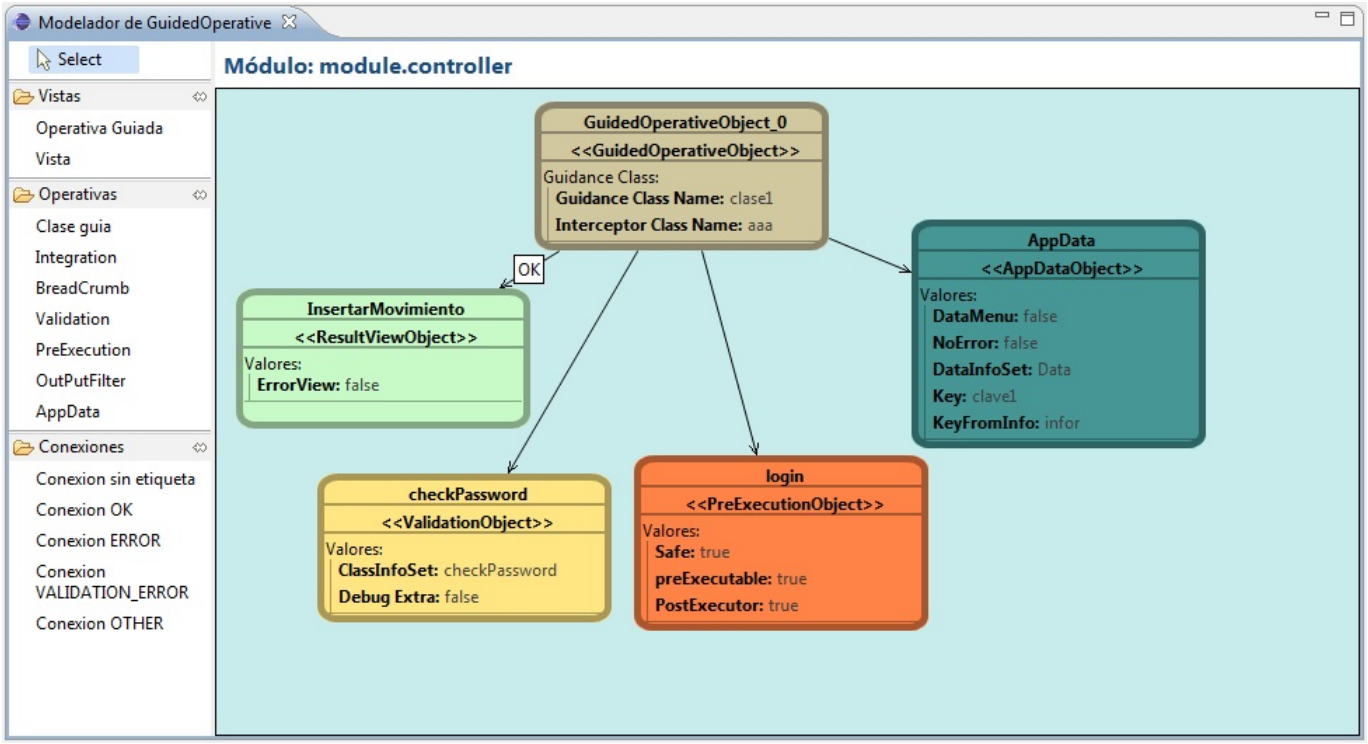


Figura 7.18: Editor gráfico de formularios

A continuación, observamos un formulario con tres componentes y un menú desplegable con las opciones disponibles para ese objeto. El menú desplegable de edición para el modelador de operativas tendrá el mismo aspecto.

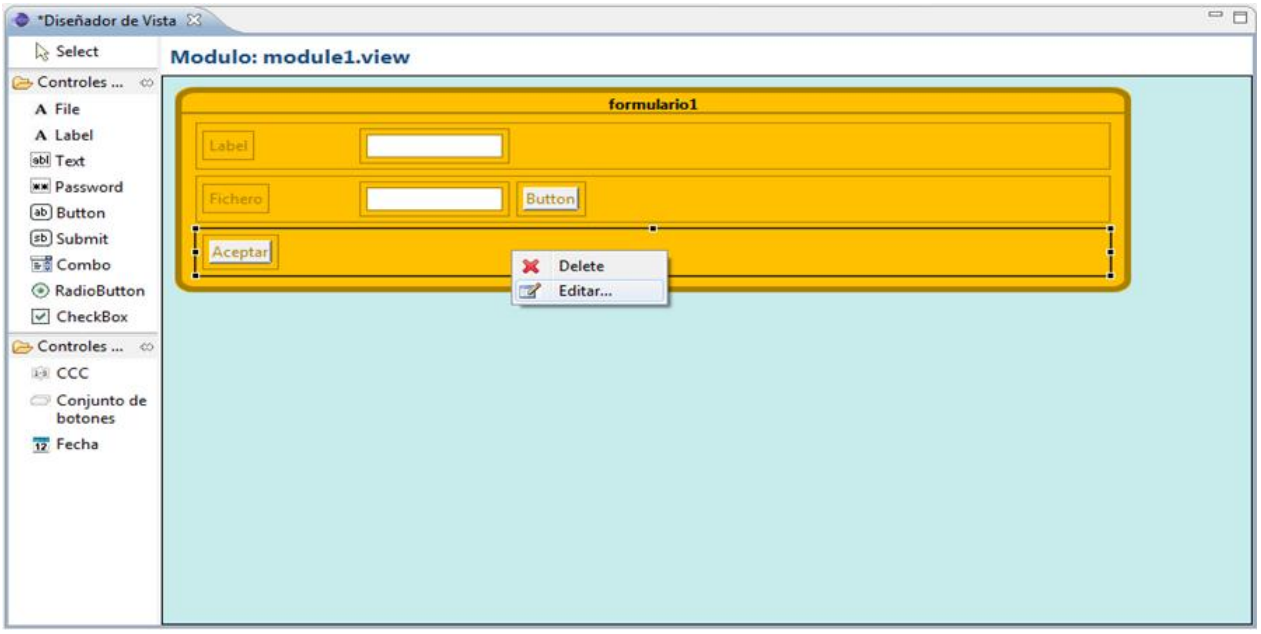


Figura 7.19: Edición de componentes

Y finalmente, un ejemplo de wizard que permite la introducción de las propiedades asociadas al elemento seleccionado. Los wizards para el modelador de operativas tendrán el mismo aspecto.

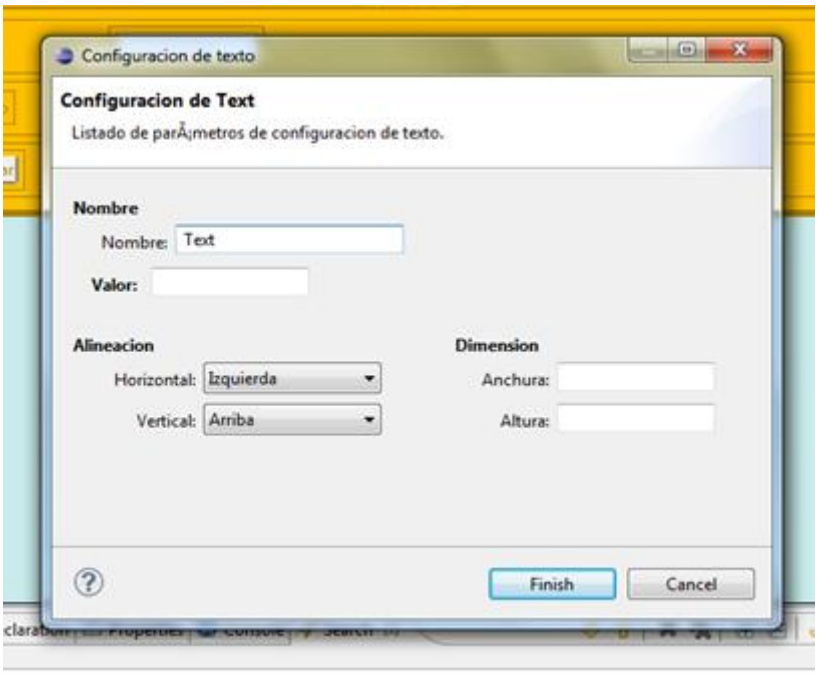


Figura 7.20: Aspecto final wizard

Capítulo 8

Conclusiones. Futuras líneas de trabajo.

La plataforma Eclipse, se encuentra entre las más utilizadas actualmente por las empresas del sector del desarrollo software. Por ello, la realización de plugins para esta plataforma resultan de gran utilidad ya que pueden ser utilizados por multitud de usuarios, además de su cómoda y rápida distribución vía su repositorio de descarga de plugins. Por estos motivos, ha resultado de gran utilidad, tanto personalmente como por parte de la empresa, el trabajar con esta plataforma de software libre.

La principal finalidad que se deseaba obtener con este proyecto era la de comenzar con su utilización de manera inmediata en la propia empresa para el desarrollo de nuevas aplicaciones de banca electrónica que utilizaran la plataforma MC-Server. En la actualidad, el plugin desarrollado en este proyecto, está siendo utilizado por programadores de la empresa TB-Solutions. Se decidió añadirlo en la etapa de implementación para el proyecto de desarrollo de la nueva banca electrónica para la entidad CAN(Caja Navarra). Las primeras impresiones que se obtienen son bastante positivas para ser una primera versión del plugin. Resulta de gran comodidad ya que en tareas muy sistemáticas se ahorra tiempo de producción, y además mayor comodidad a la hora de editar las propiedades de los ficheros propios de la plataforma MC-Server(ficheros .model, .controller y .view).

Para futuras versiones, se tiene pensado en ampliar y/o mejorar alguna de las funcionalidades que ofrece esta primera versión del plugin, así como retomar otras que hayan quedado pendientes de implementación.. También se pretende reforzar el aspecto de las validaciones de datos que resulta imprescindible para el perfecto funcionamiento de la aplicación. Además, mediante su temprana implantación, se desea obtener cualquier tipo de fallo o carencia para poder mejorarla y que se encuentre perfectamente operativa para los usuarios.

Parte IV

Referencias y Bibliografía

Bibliografía

[1] Koen Aers. Developing an editor for direct graphs - gef introduction, 2008. http://www.inf.ed.ac.uk/teaching/courses/ip/resources/GEF/gef.eclipsecon.2008.pdf.

[2] Frank Budinsky. *Eclipse Modeling Framework*. The Eclipse Series, 2003.

[3] The Eclipse Foundation. Eclipse documentation, 2011. http://help.eclipse.org/indigo/index.jsp.

[4] Bill Moore. *Eclipse Development using the Graphical Editing Framework and the Eclipse Modeling Framework*. RedBooks, 2004.

[5] T.J. Watson. Eclipse 2.0 jdt plug-in developer guide, 2002. .

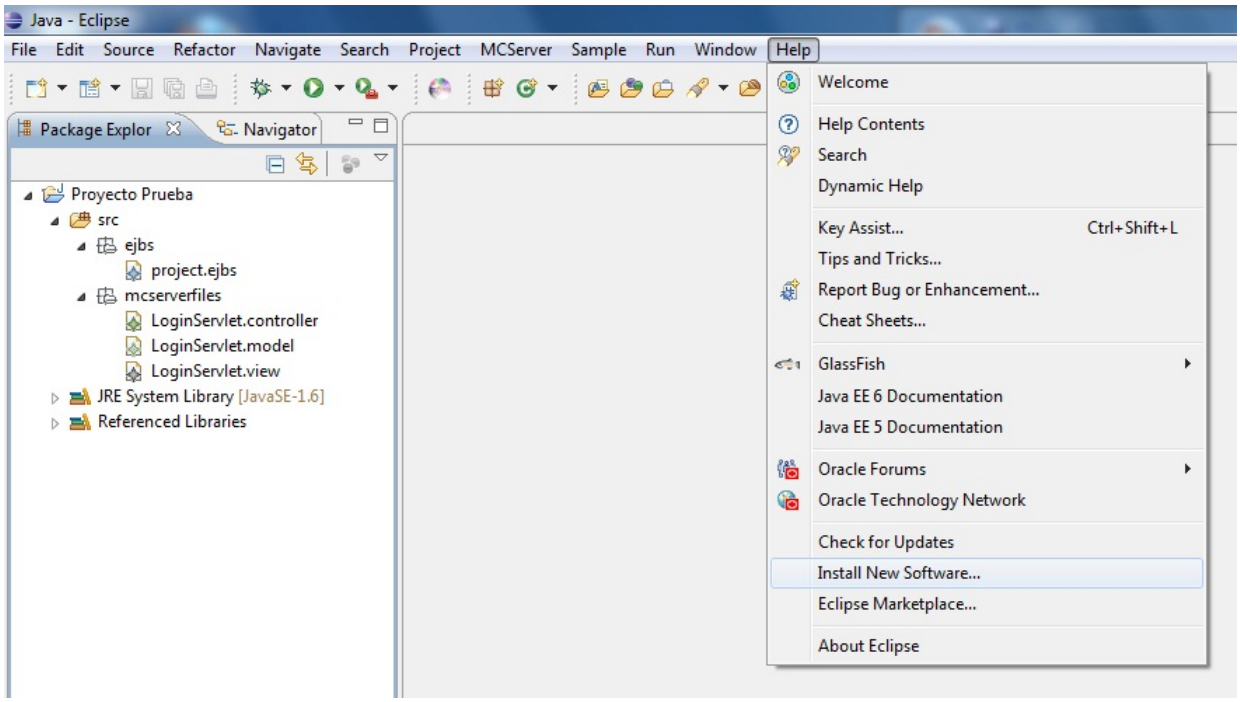
Parte V

Apéndices

Manual de Instalación

1.- Instalación del plugin

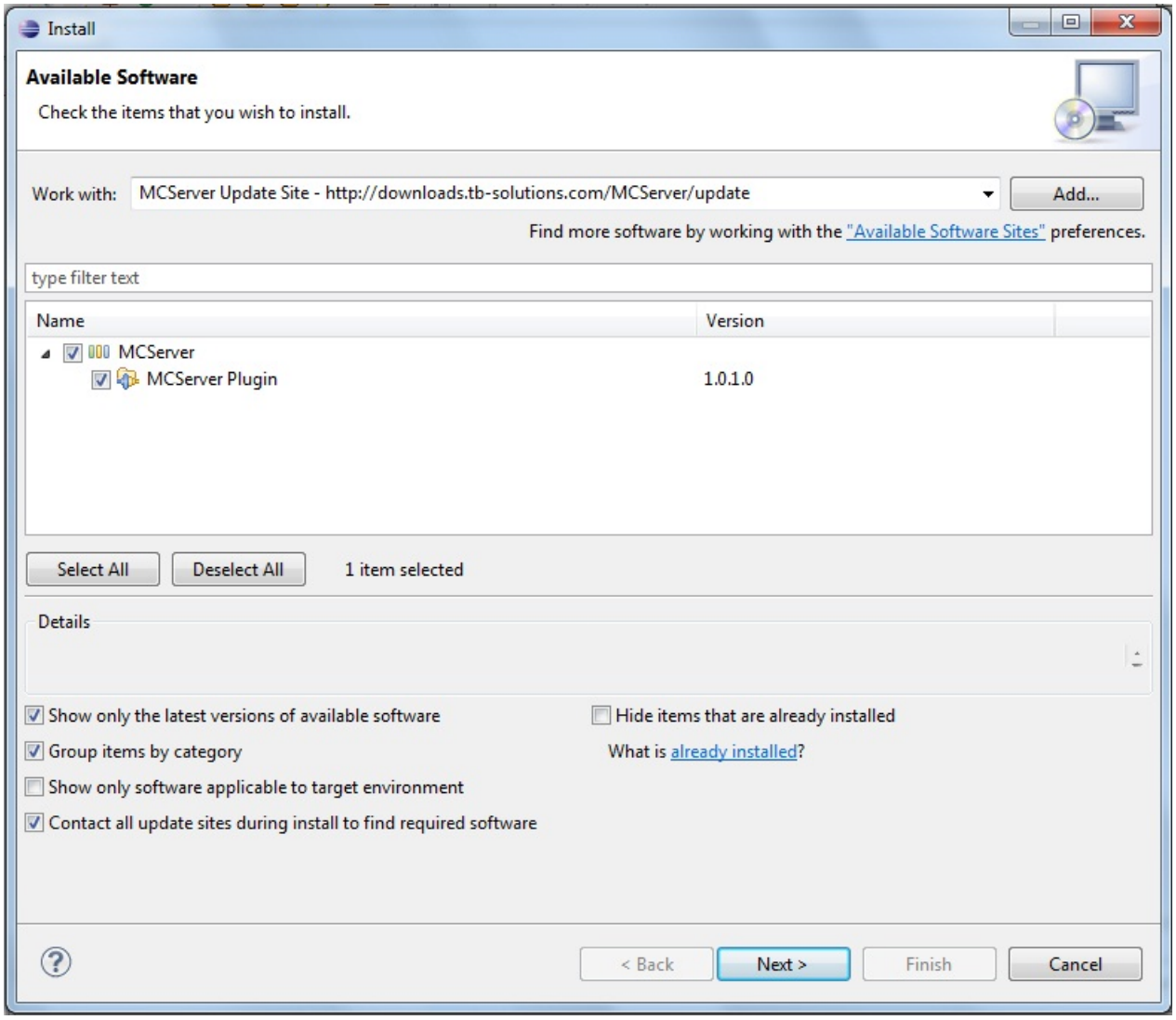
Para realizar la instalación del plugin de MCServer en nuestra API de Eclipse, será necesario seguir una serie de sencillos pasos. Accedemos al menú *Help* y seleccionamos la opción de menú *Install New Software* como indicamos en la imagen a continuación.



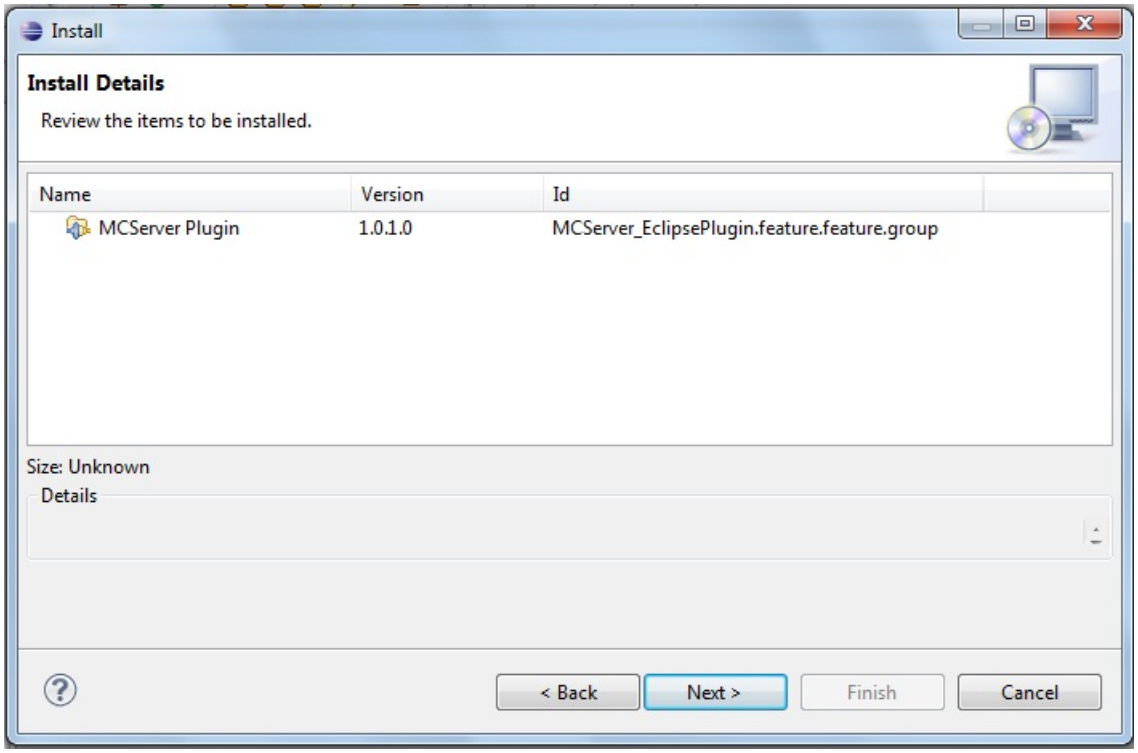
Y se nos abrirá un wizard como el que mostramos en la siguiente imagen. En esta pantalla, deberemos indicar la ruta del repositorio donde se encuentra el plugin:

http://downloads.tb-solutions.com/MCServer/update

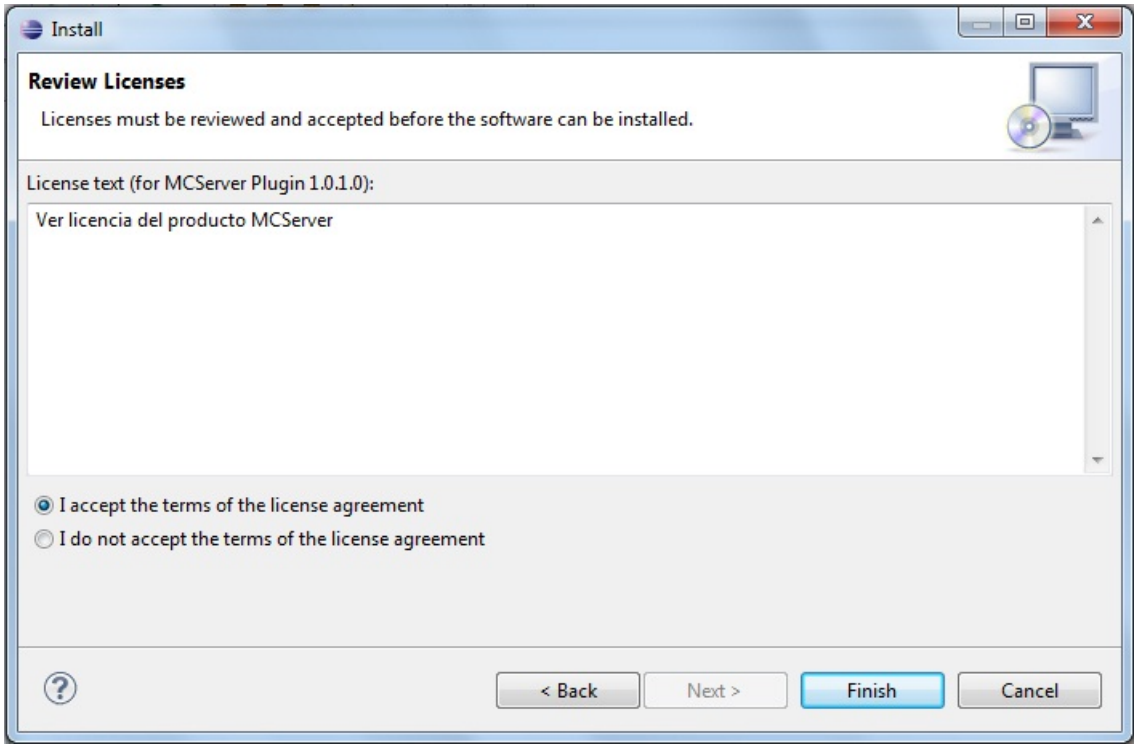
Si presionamos la tecla Enter, nos aparecerá en la pantalla una estructura arborescente con las versiones disponibles del plugin MCServer. Seleccionaremos la versión deseada y presionaremos sobre el botón *Next* para continuar con la instalación.



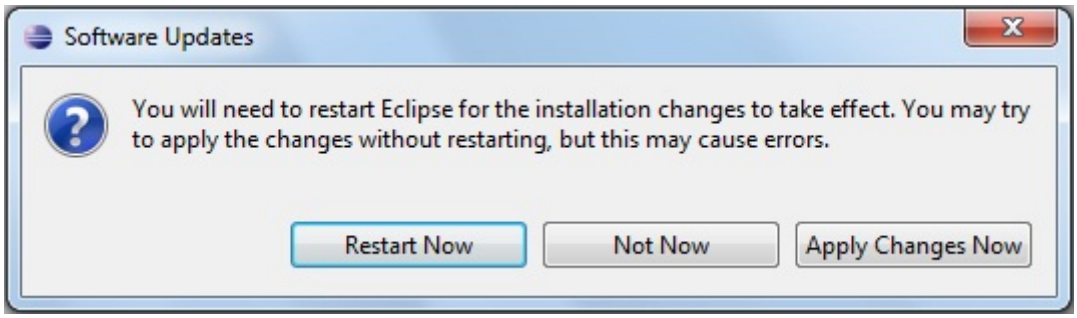
Accederemos a una pantalla, como se muestra en la siguiente imagen, para verificar las licencias del plugin. Tendremos que aceptar los términos que impone la licencia para poder continuar. Por último presionaremos el botón de *Next*.



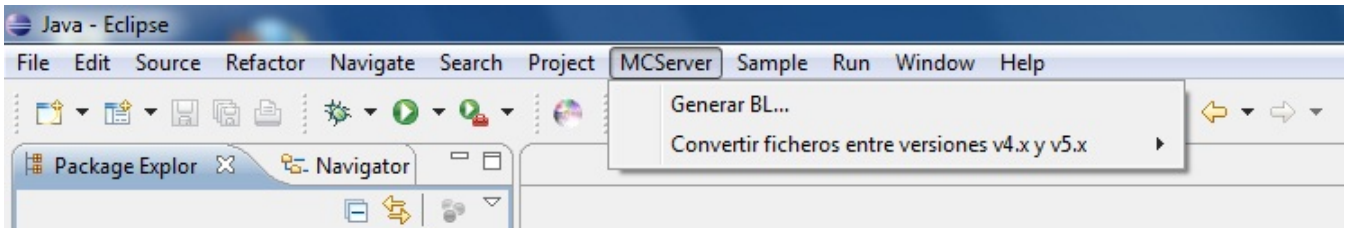
Se mostrará una pantalla con los detalles del plugin, que únicamente deberemos presionar el botón *Next*, como se muestra a continuación.



Para finalizar, después de esperar unos minutos a que termine la instalación del plugin, se mostrará una pantalla como la siguiente, en la que deberemos seleccionar la opción de *Restart Now*, para reiniciar Eclipse y que los cambios se realicen correctamente.

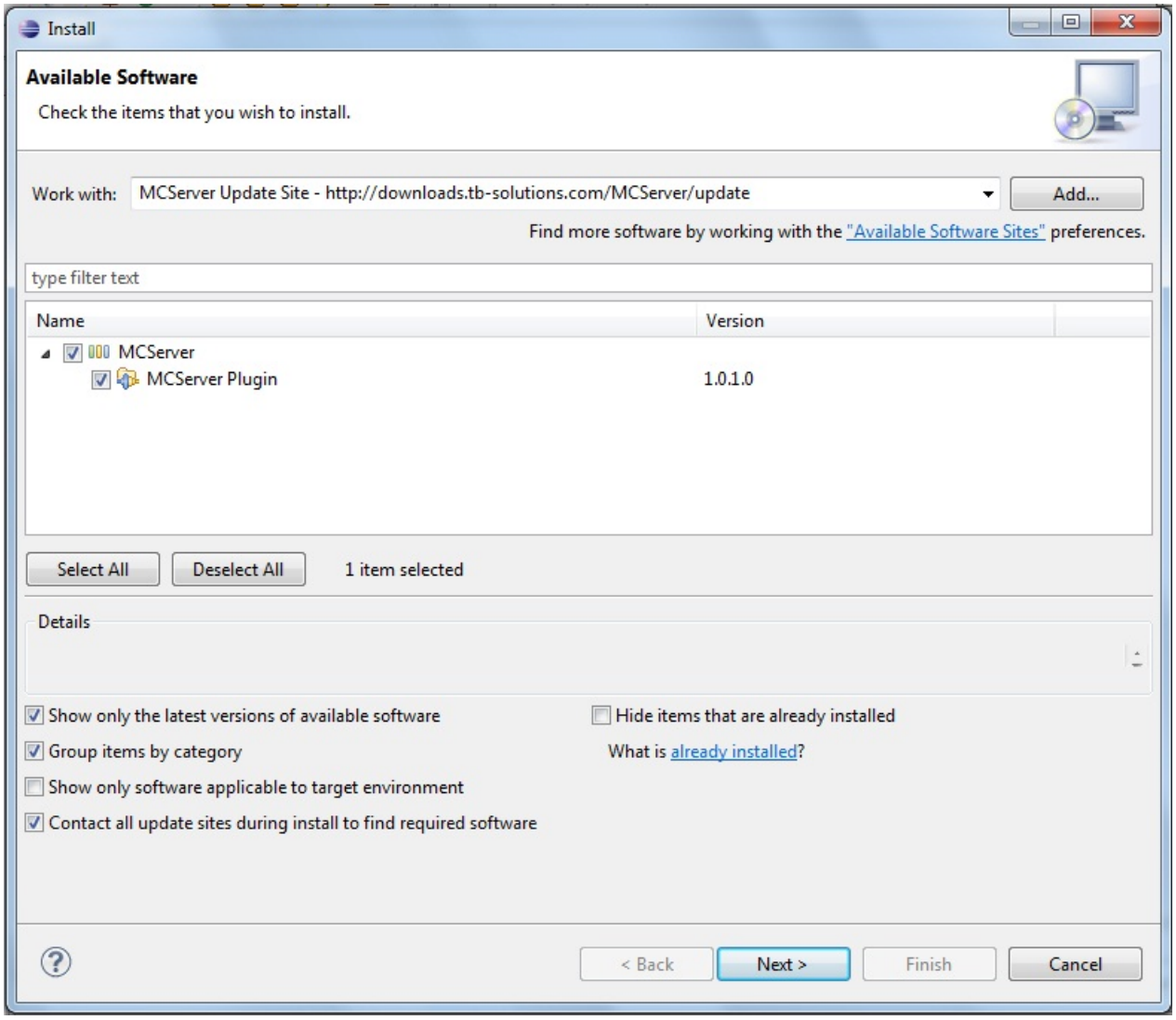


Una vez realizado todos estos pasos, el plugin estará correctamente operativo en nuestro Eclipse. Para verificar que el plugin está correctamente instalado, deberemos ver una modificación en la barra de herramientas, apareciendo ahora la opción MCServer como mostramos en la siguiente imagen.

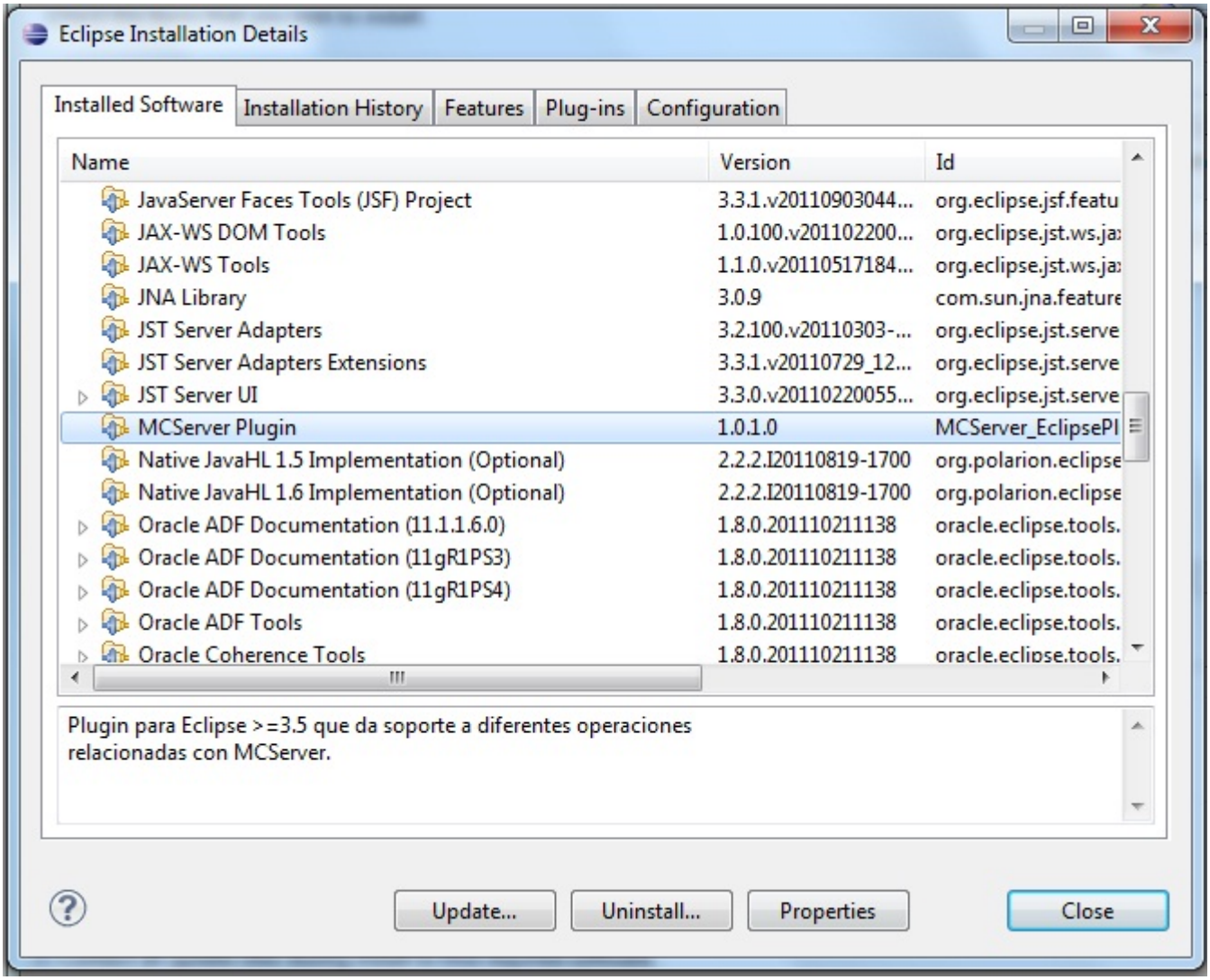


2.-Actualización del plugin

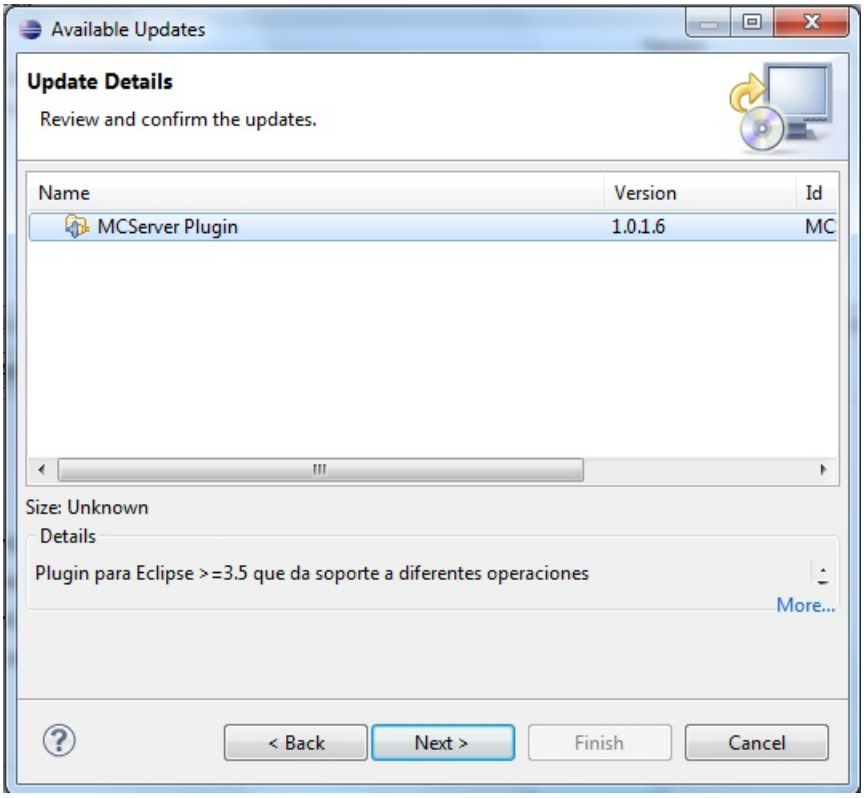
Será necesario tener a punto la última versión del plugin para tener las últimas mejoras realizadas. Para ello, explicamos a continuación el proceso de actualización.
Accedemos a la pantalla de instalación de nuevo software, en el menú *Help* de Eclipse. Nos aparecerá una pantalla como la siguiente, en la que deberemos hacer clic sobre el vínculo de *alredy installed*.

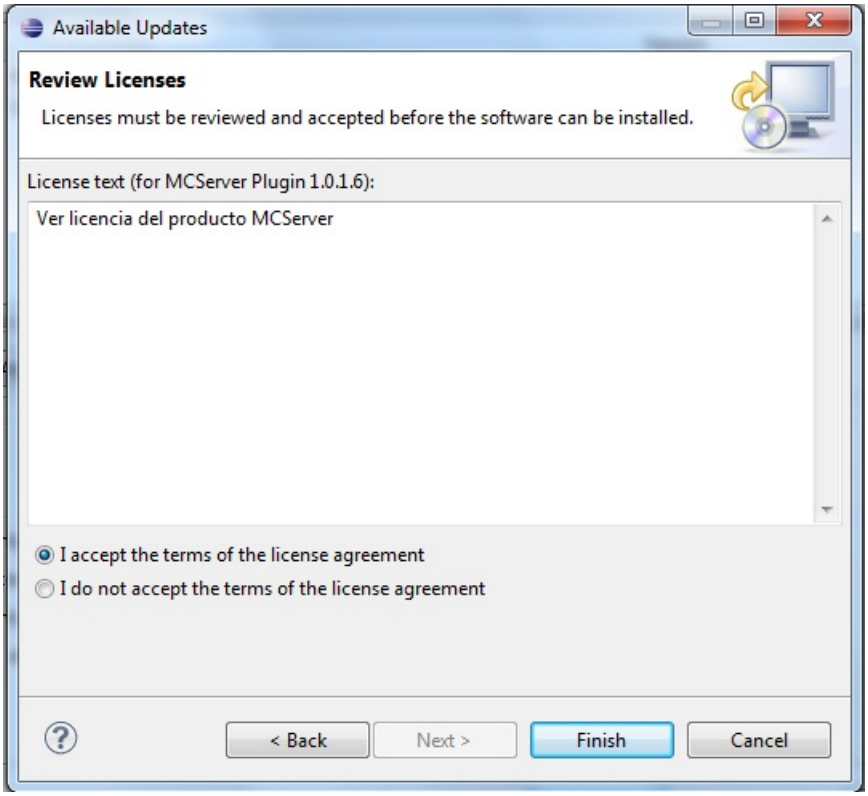


Nos aparecerá una pantalla como la siguiente, en la que deberemos seleccionar la entrada MCServerPlugin y hacer click sobre el botón *Update*.



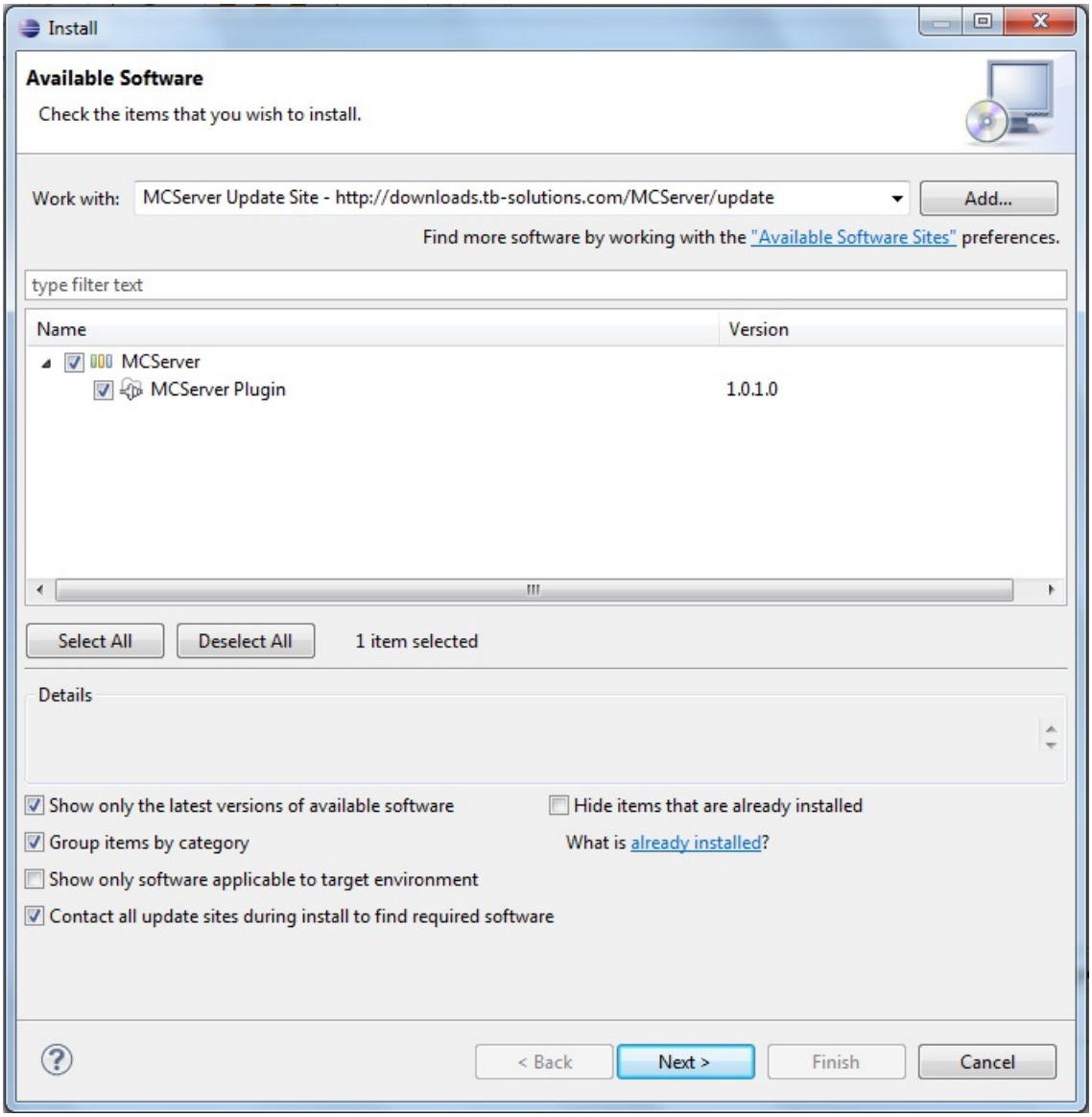
Pasaremos a una pantalla en la que se nos mostrarán los detalles de la nueva actualización y otra pantalla para aceptar la licencia del producto, en el caso de que existan nuevas actualizaciones. Seleccionaremos la que deseemos y presionaremos sobre el botón *Next*. Para que se apliquen los cambios, deberemos reiniciar Eclipse.


Available Updates
Update Details
Review and confirm the updates.
Name
Version
Id
MCServer Plugin
1.0.1.6
MC
Size: Unknown
Details
Plugin para Eclipse >=3.5 que da soporte a diferentes operaciones
More...
< Back
Next >
Finish
Cancel

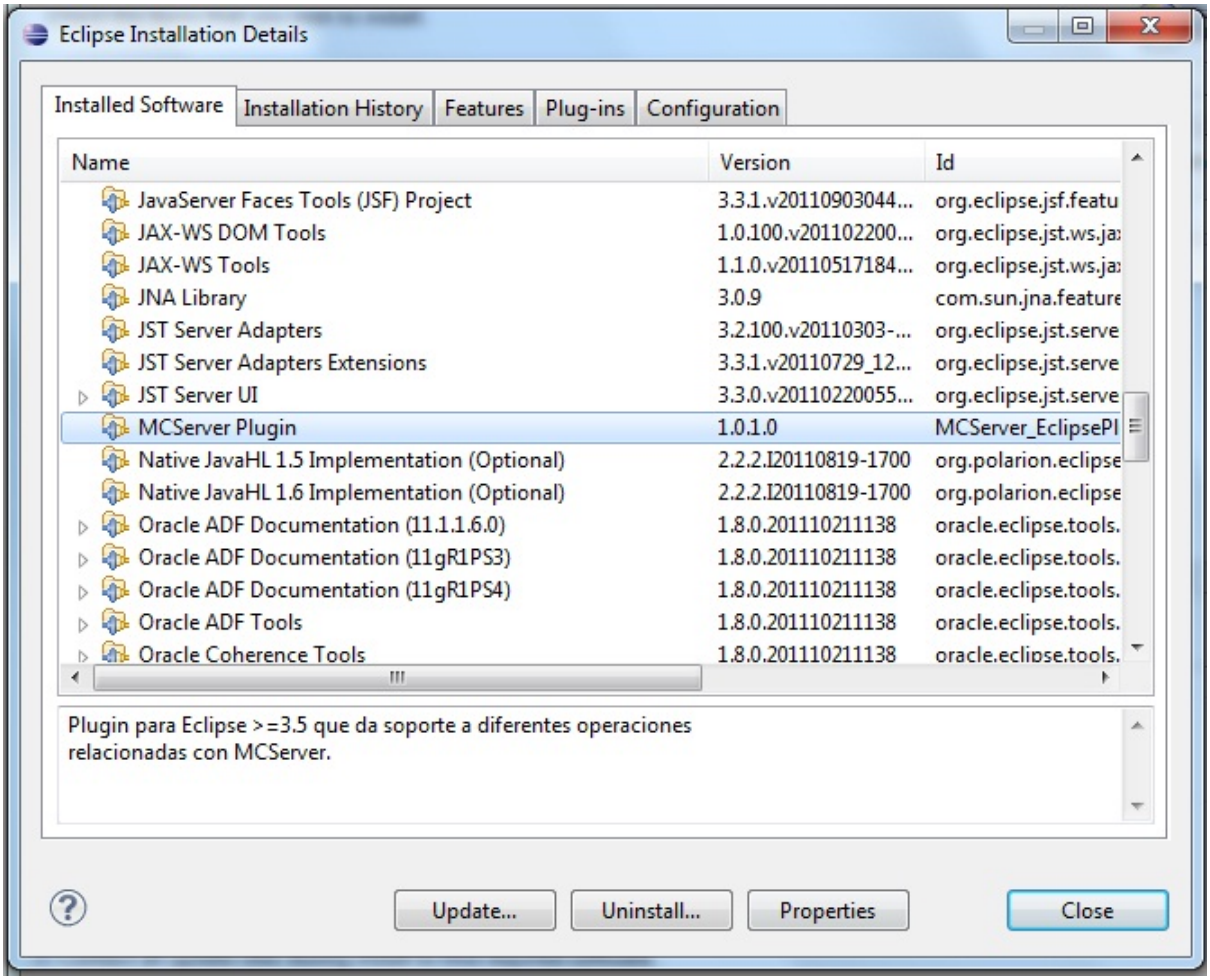

Available Updates
Review Licenses
Licenses must be reviewed and accepted before the software can be installed.
License text (for MCServer Plugin 1.0.1.6):
Ver licencia del producto MCServer
I accept the terms of the license agreement
I do not accept the terms of the license agreement
< Back
Next >
Finish
Cancel

3.-Desinstalación del plugin

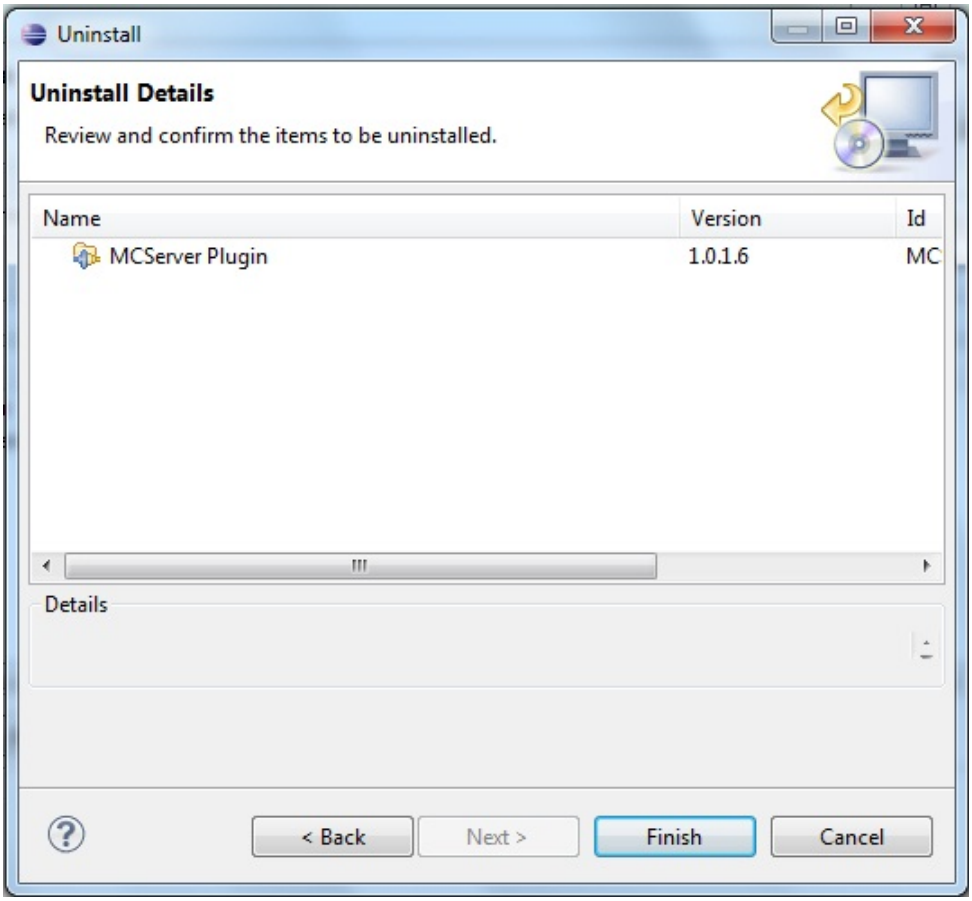
En el momento que deseemos desinstalar el plugin, deberemos realizar unos sencillos pasos que explicamos a continuación.
Accedemos a la pantalla de instalación de nuevo software, en el menú *Help* de Eclipse y hacemos click sobre el link *alredy installed*.



Se mostrará una pantalla como la de a continuación, donde seleccionaremos la entrada del MCServerPlugin y presionaremos sobre el botón *Uninstall*.



Se nos mostrará una pantalla intermedia con los detalles de la desinstalación. Para finalizar la desinstalación, presionaremos sobre el botón *Finish*.



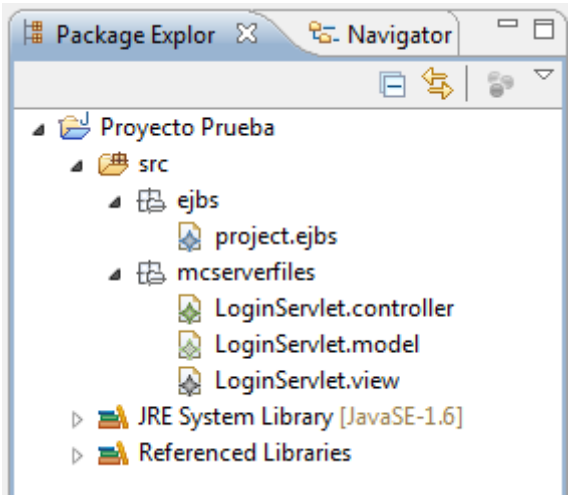
Manual de Instalación

1.- Introducción

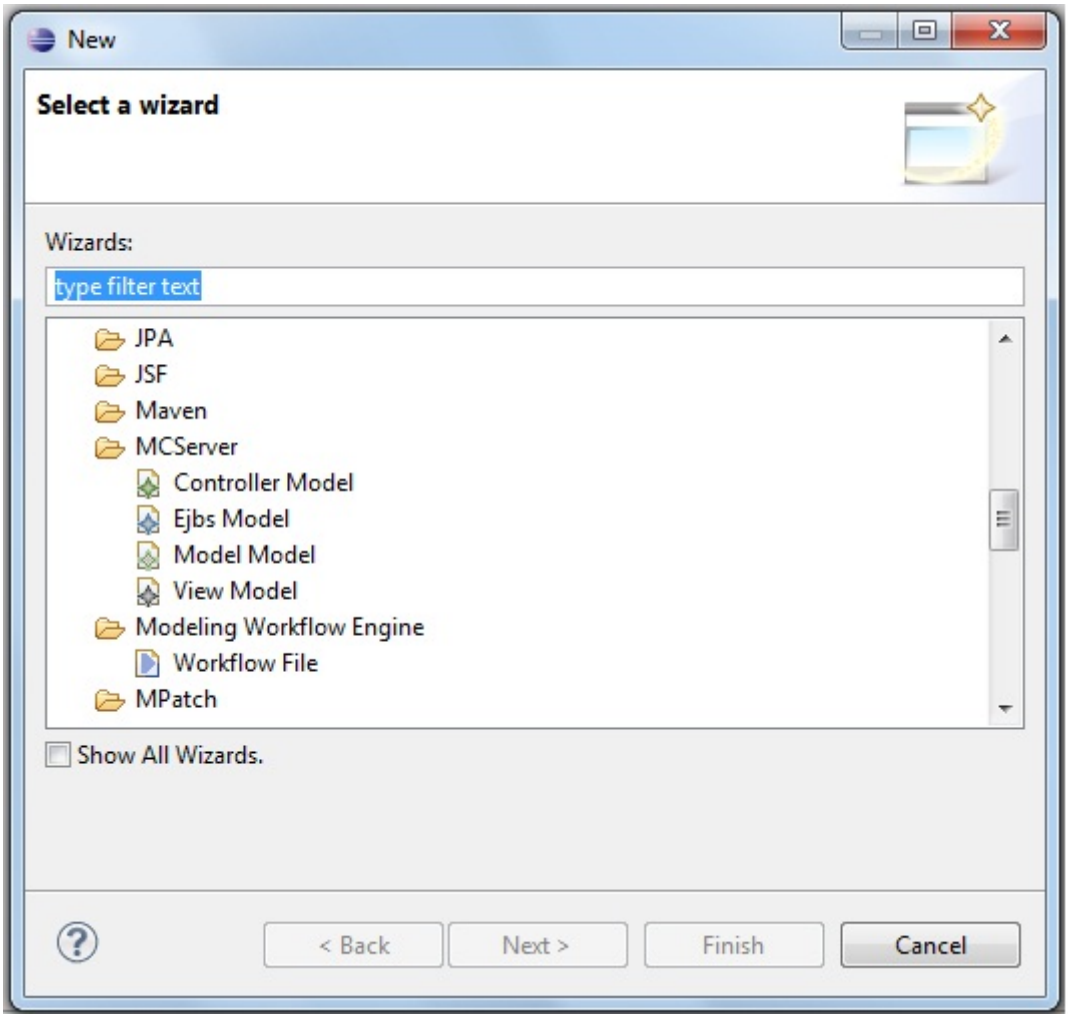
En este manual se explicará cómo realizar cada una de las posibles funcionalidades que posee el plugin para MCServer. Para su utilización en un proyecto real, además, sería necesario añadir al proyecto la librería mcserver.jar para poder reconocer el contenido de los ficheros que generemos con el plugin.

2.- Creación de ficheros

Con este plugin, podremos generar los cuatro ficheros principales que utiliza MCServer. La estructura en un proyecto quedaría como se muestra a continuación.



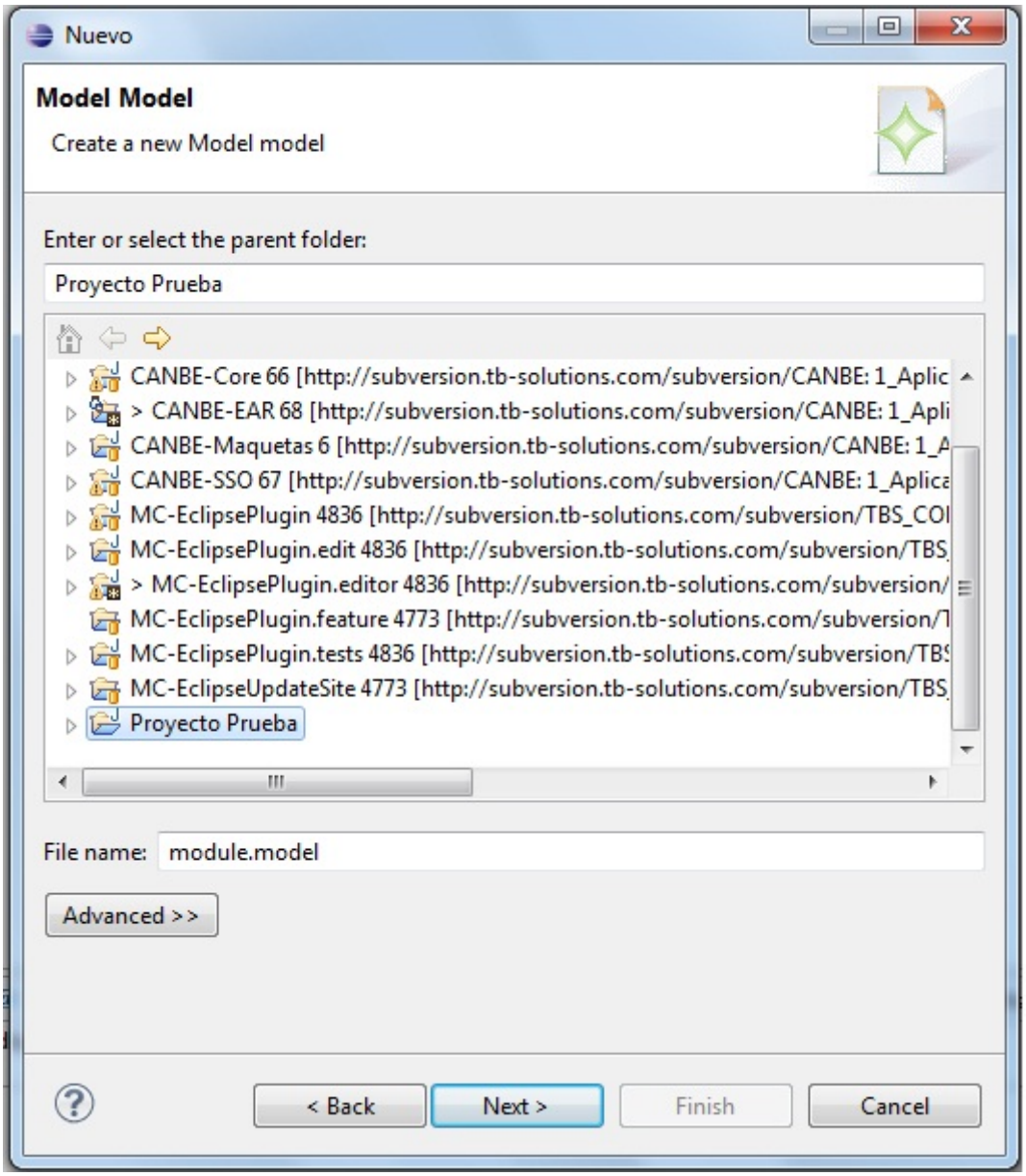
Para la generación de dichos ficheros, presionaremos sobre el proyecto y seleccionaremos la opción *New* y luego *Other* y se nos abrirá una pantalla como la siguiente.



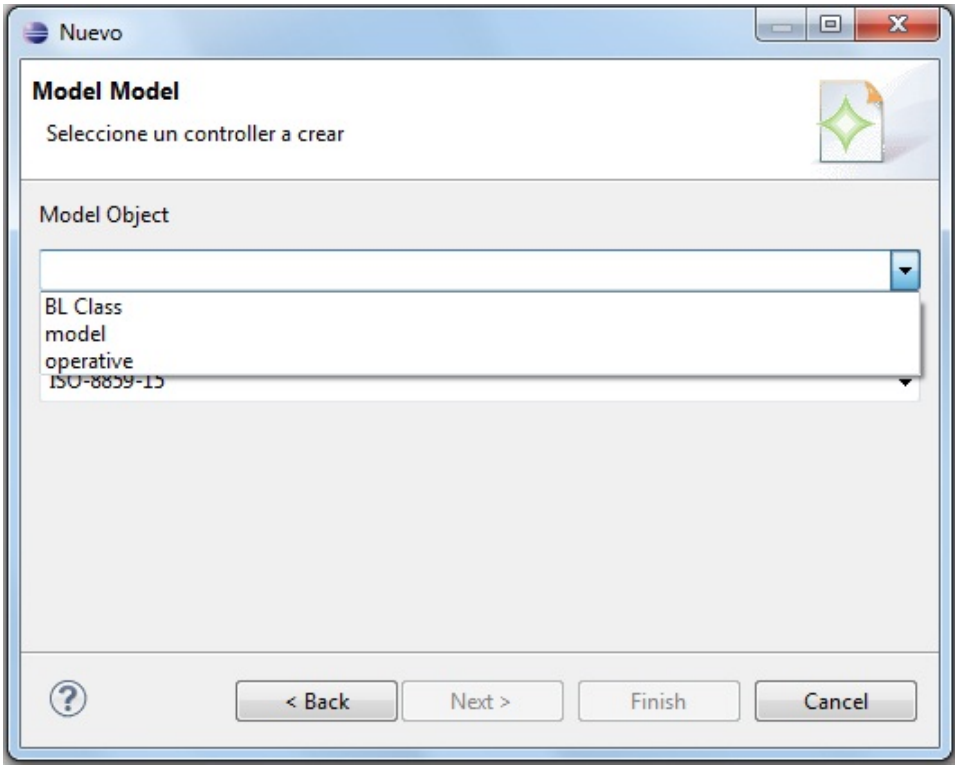
A continuación, explicaremos como generar cada uno de los ficheros con sus propiedades correctas.

2.1.- Creación de ficheros .model

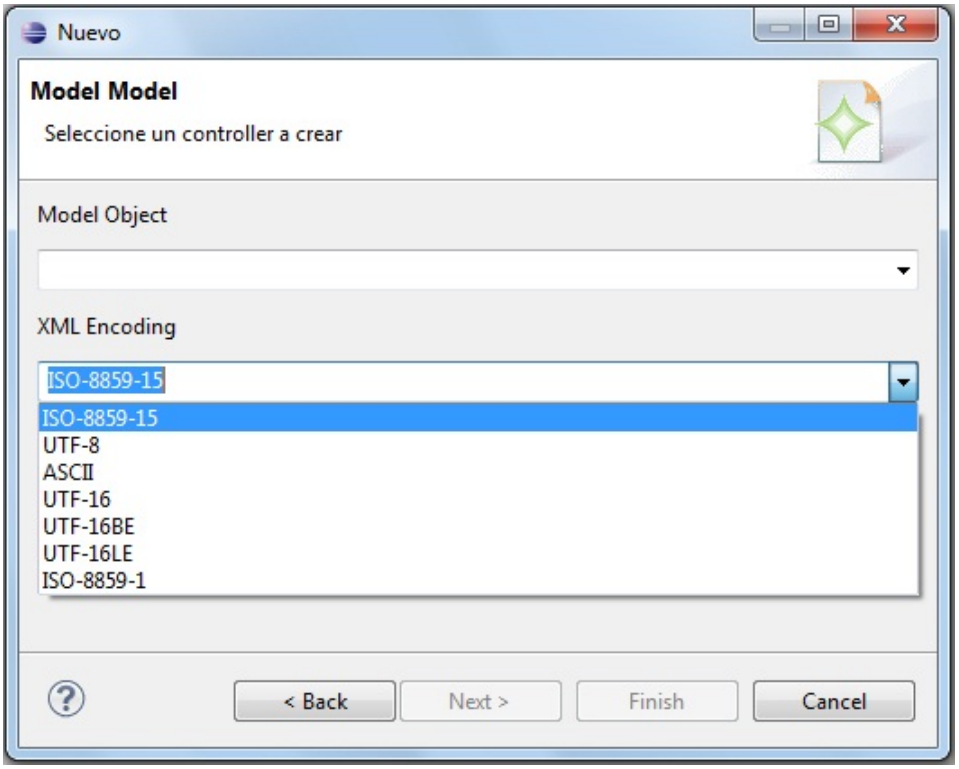
Seleccionamos de la ventana de nuevo fichero, el de tipo *Model* y le damos un nombre como mostramos a continuación.



Seguidamente deberemos de dar las propiedades a la estructura del fichero. Primero seleccionaremos el modelo del objeto, que siempre deberá de ser de tipo *Model*, como mostramos a continuación.

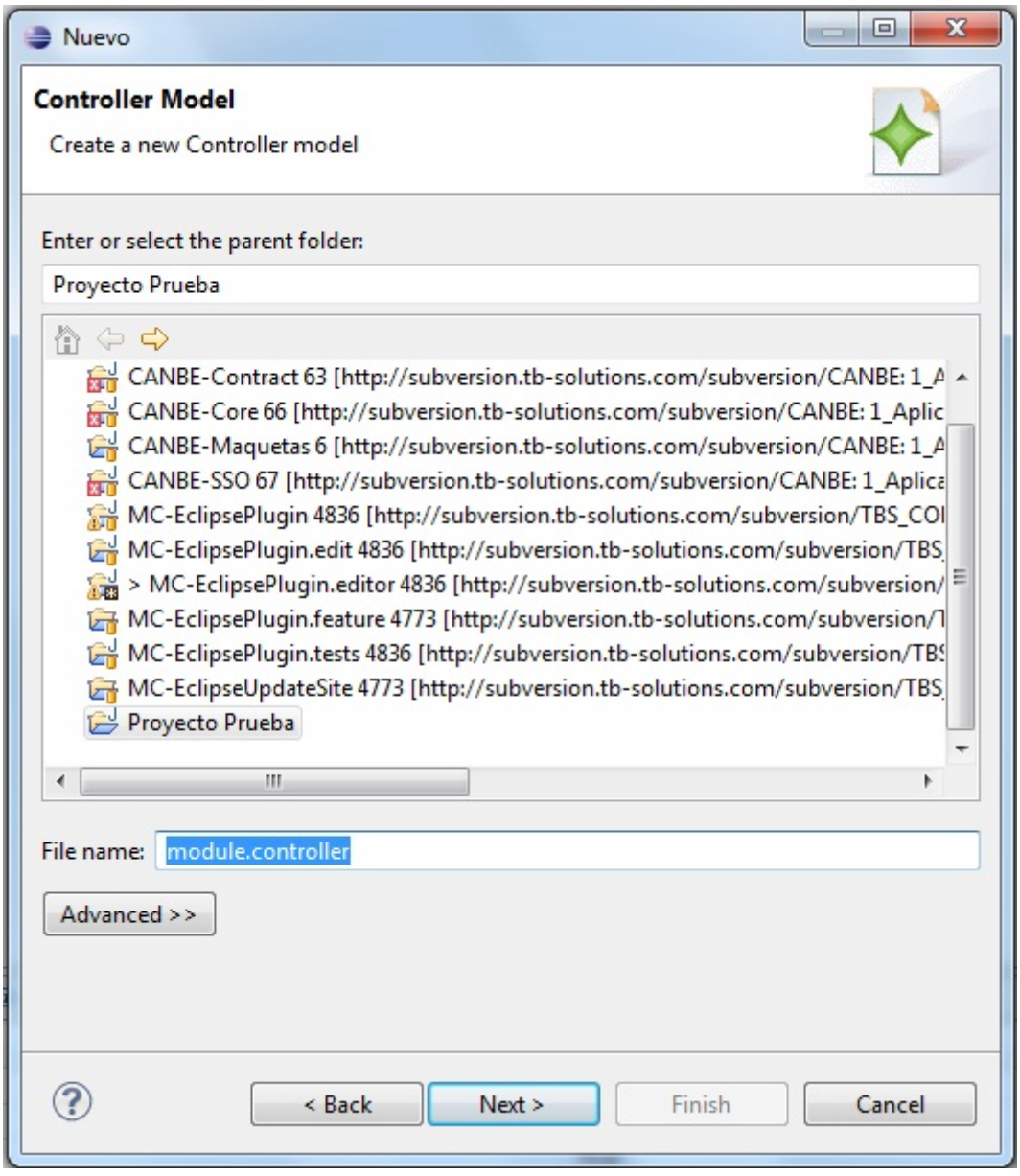


Por último, seleccionaremos el tipo de codificación XML que deseemos, en nuestro caso será la *ISO-8859-15*. Al presionar sobre el botón *Finish* tendremos correctamente creado el fichero de model.

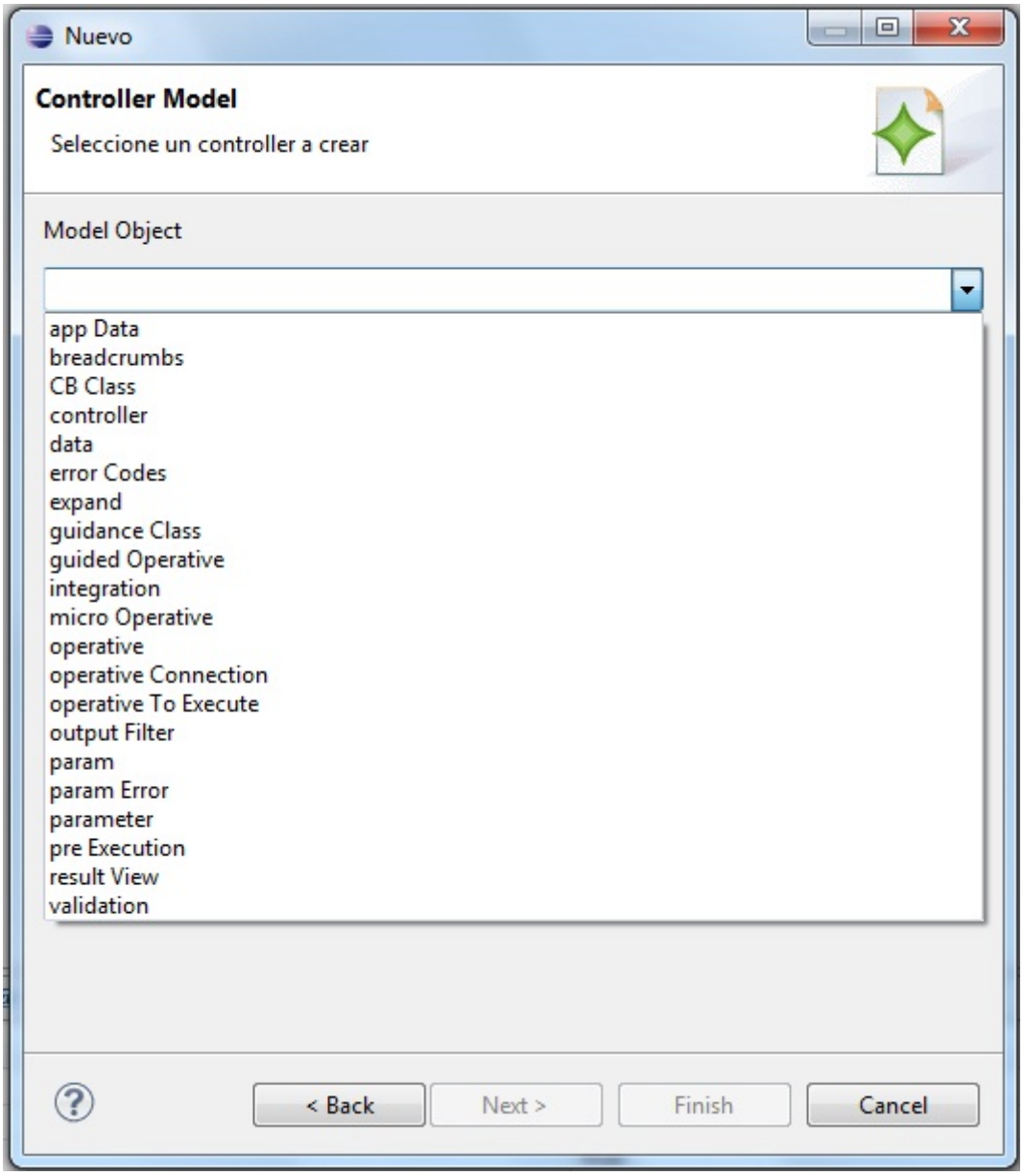


2.2.- Creación de ficheros .controller

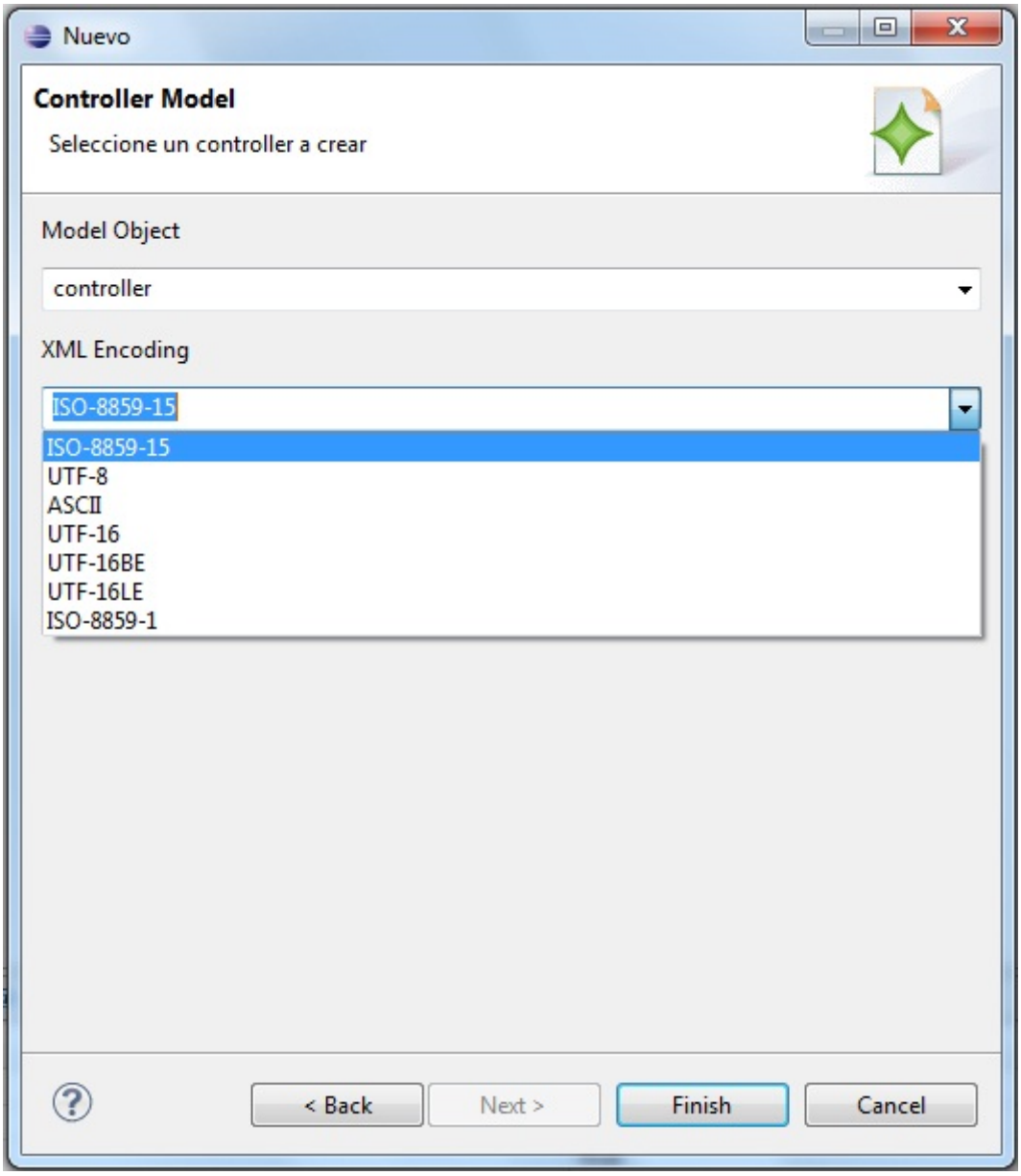
Seleccionamos de la ventana de nuevo fichero, el de tipo *Controller* y le damos un nombre como mostramos a continuación.



Seguidamente deberemos de dar las propiedades a la estructura del fichero. Primero seleccionaremos el modelo del objeto, que siempre deberá de ser de tipo *Controller*, como mostramos a continuación.

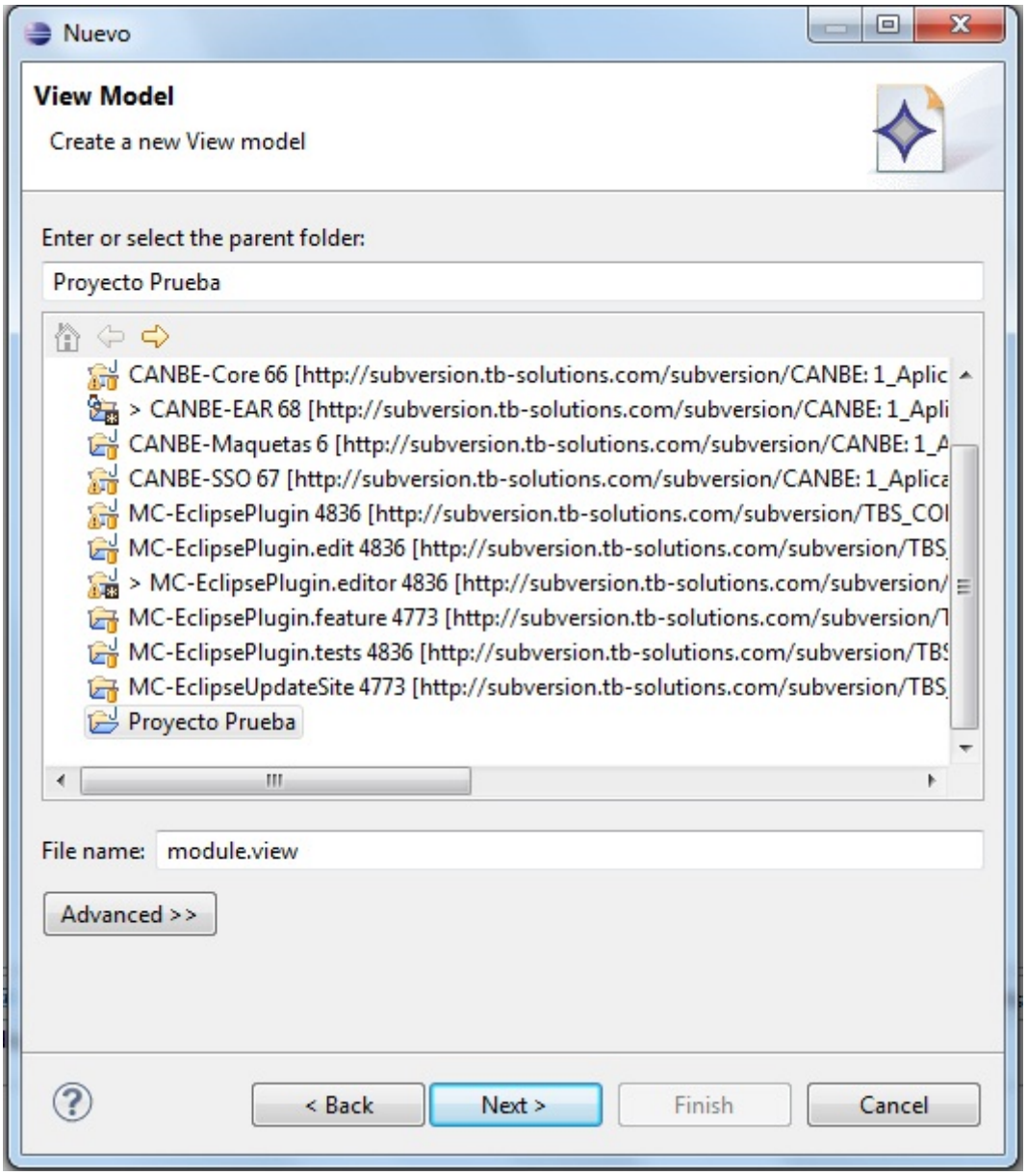


Por último, seleccionaremos el tipo de codificación XML que deseemos, en nuestro caso será la *ISO-8859-15*. Al presionar sobre el botón *Finish* tendremos correctamente creado el fichero de control.

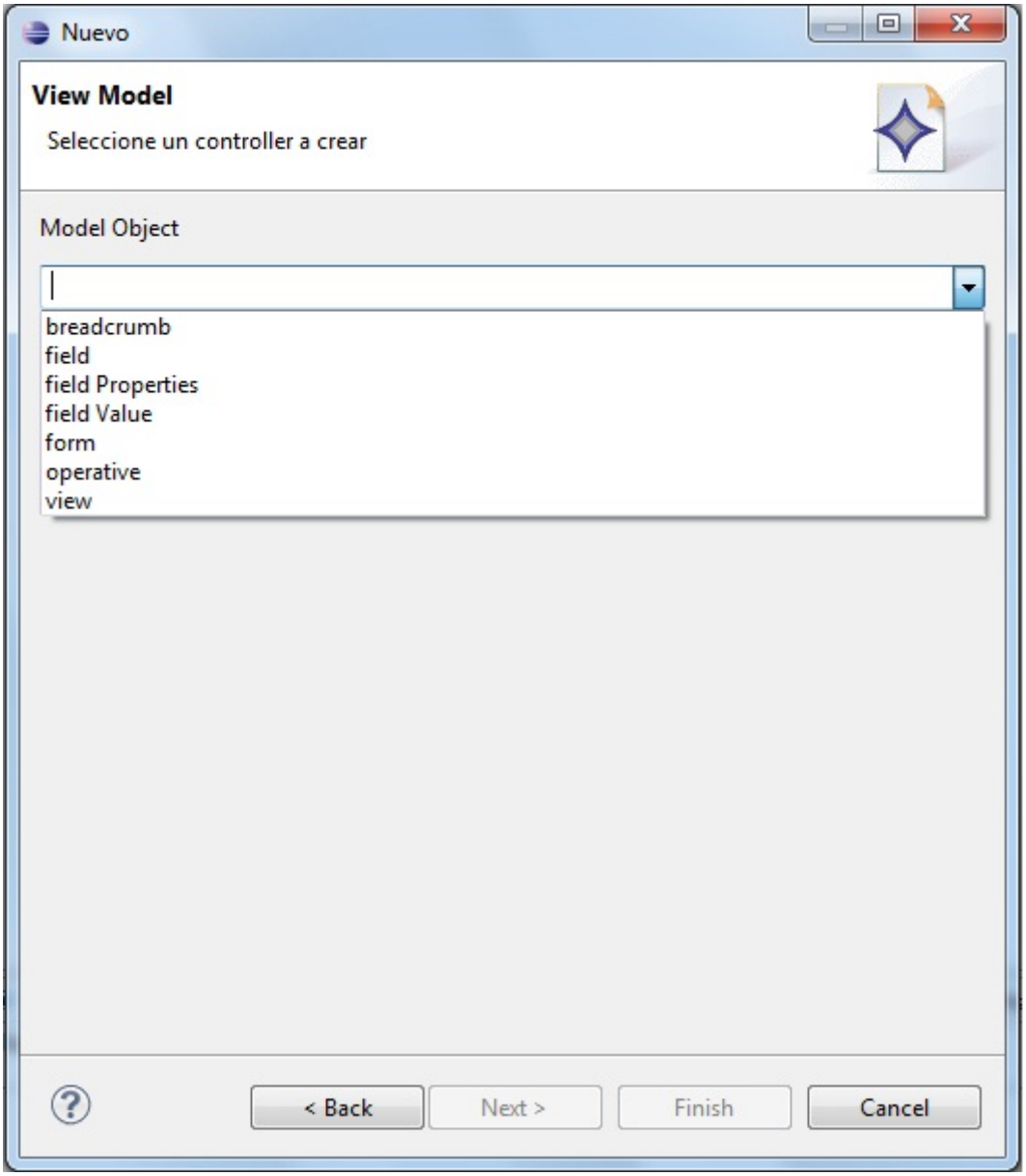

Nuevo
Controller Model
Seleccione un controller a crear
Model Object
controller
XML Encoding
ISO-8859-15
ISO-8859-15
UTF-8
ASCII
UTF-16
UTF-16BE
UTF-16LE
ISO-8859-1
< Back
Next >
Finish
Cancel

2.3.- Creación de ficheros .view

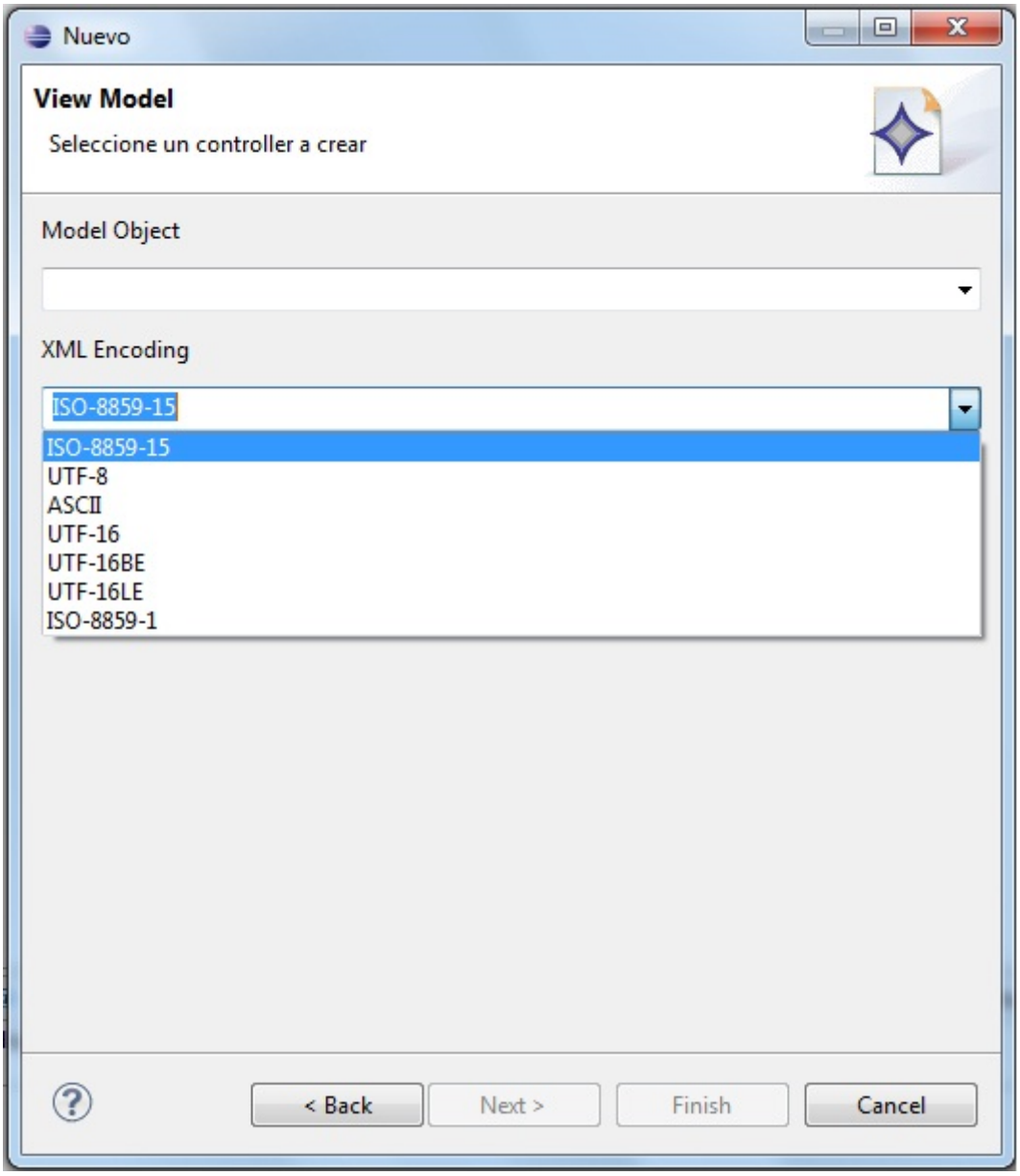
Seleccionamos de la ventana de nuevo fichero, el de tipo *View* y le damos un nombre como mostramos a continuación.



Seguidamente deberemos de dar las propiedades a la estructura del fichero. Primero seleccionaremos el modelo del objeto, que siempre deberá de ser de tipo *View*, como mostramos a continuación.

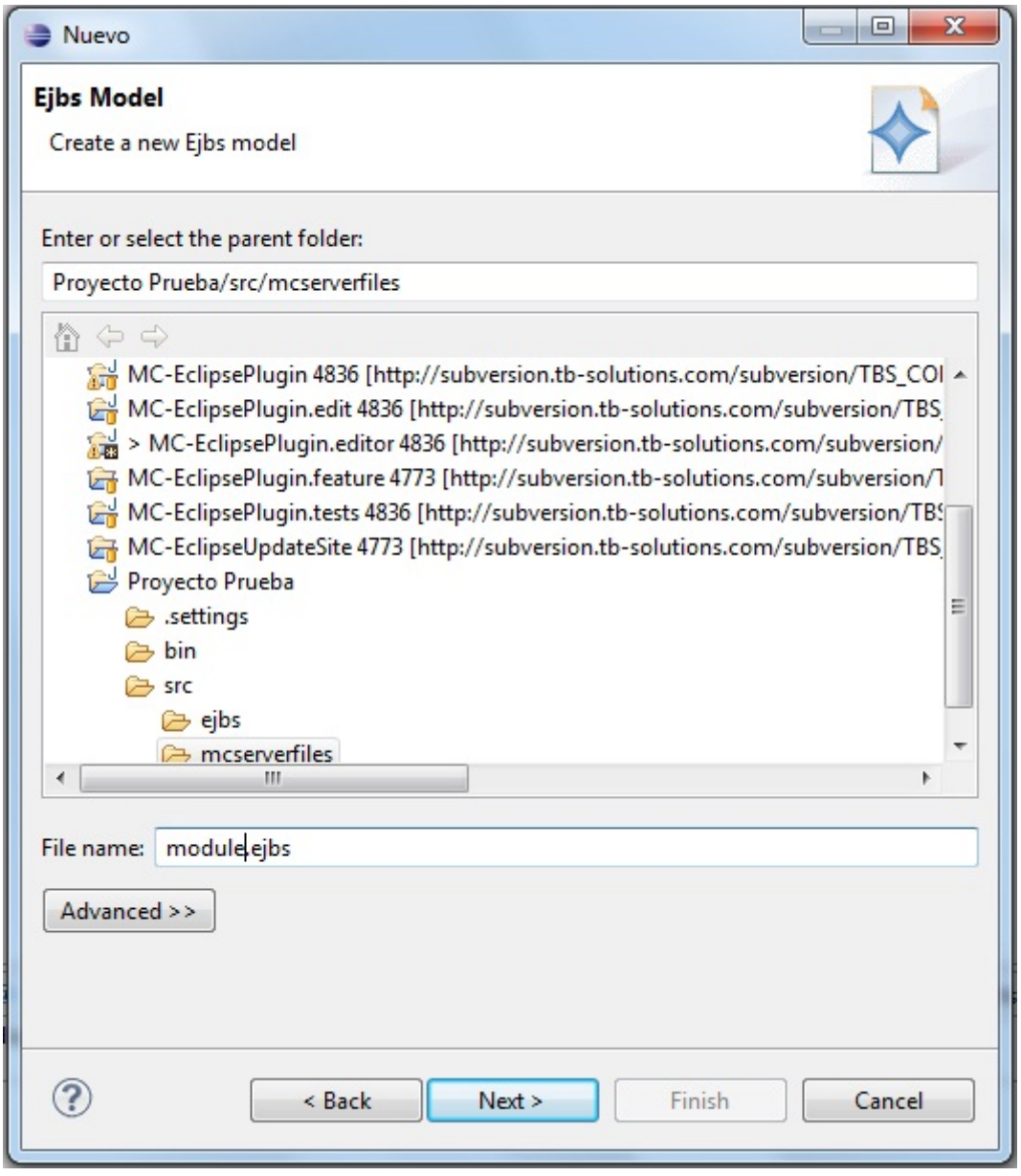


Por último, seleccionaremos el tipo de codificación XML que deseemos, en nuestro caso será la *ISO-8859-15*. Al presionar sobre el botón *Finish* tendremos correctamente creado el fichero de view.

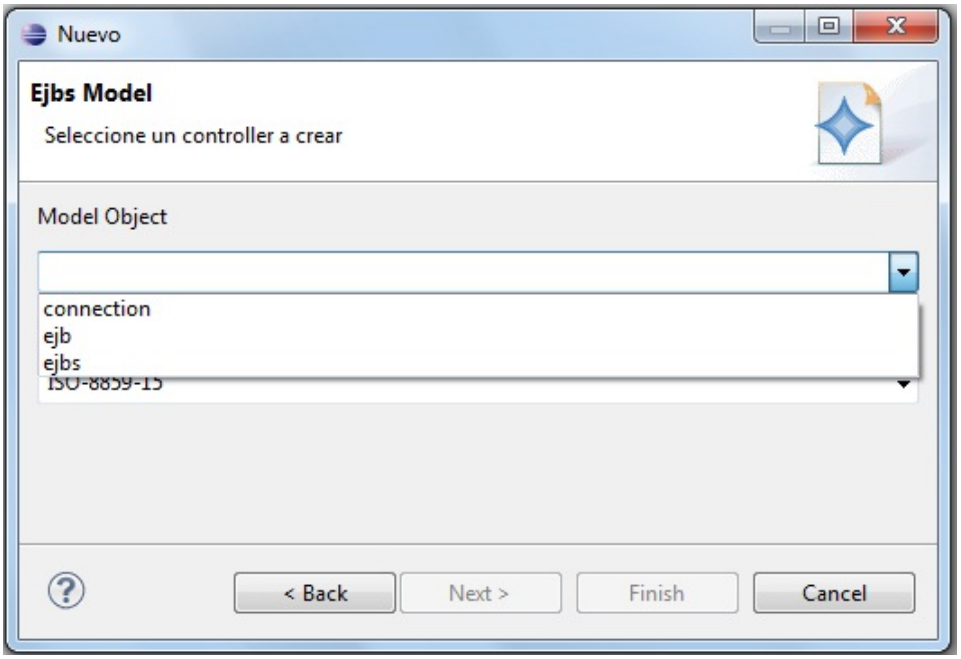

Nuevo
View Model
Seleccione un controller a crear
Model Object
XML Encoding
ISO-8859-15
ISO-8859-15
UTF-8
ASCII
UTF-16
UTF-16BE
UTF-16LE
ISO-8859-1
< Back
Next >
Finish
Cancel

2.4.- Creación de ficheros .ejbs

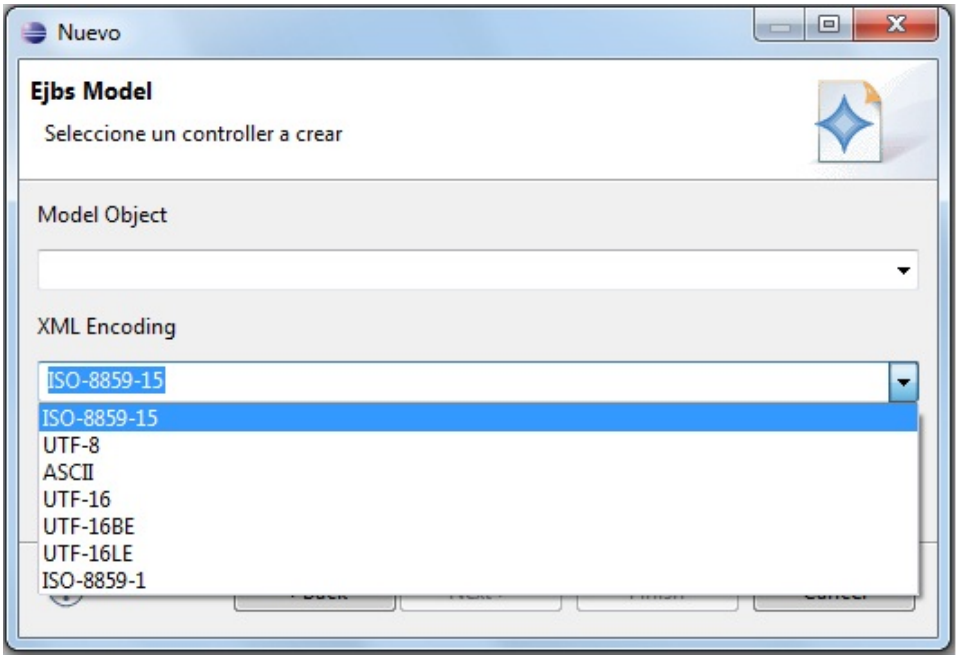
Seleccionamos de la ventana de nuevo fichero, el de tipo *Ejbs* y le damos un nombre como mostramos a continuación.



Seguidamente deberemos de dar las propiedades a la estructura del fichero. Primero seleccionaremos el modelo del objeto, que siempre deberá de ser de tipo *ejbs*, como mostramos a continuación.



Por último, seleccionaremos el tipo de codificación XML que deseemos, en nuestro caso será la *ISO-8859-15*. Al presionar sobre el botón *Finish* tendremos correctamente creado el fichero de ejbs.



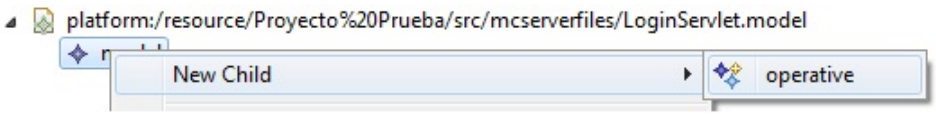
3.- Model

Una vez creado un fichero de tipo *model*, tendremos que poder configurarlo para su ejecución sobre la plataforma MCServer.

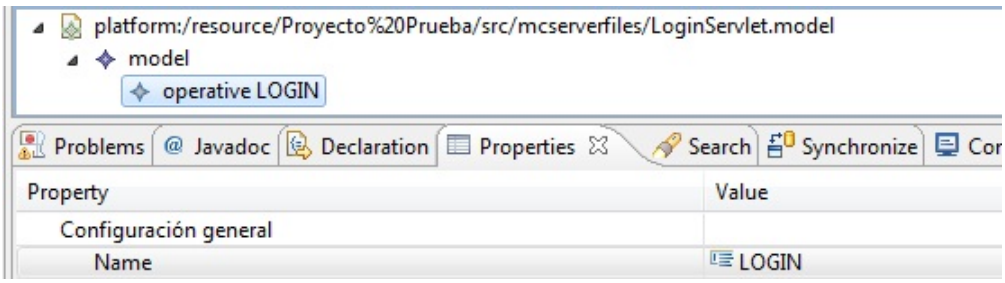
3.1.- Creación de una clase BL

Deberemos abrir el fichero en Eclipse, haciendo doble clic sobre él.

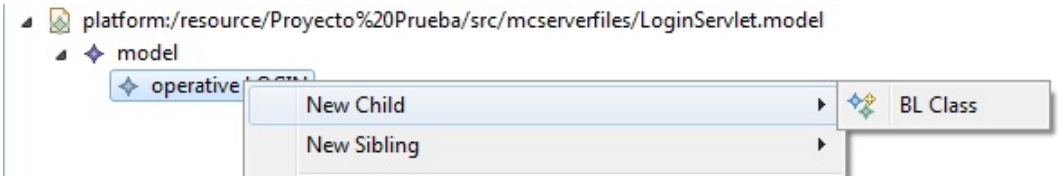
Seguidamente, hacer click con el botón derecho, *New Child* y seleccionar *operative*.



Editamos el nombre de la operativa mediante la pestaña de propiedades de la parte inferior de Eclipse.

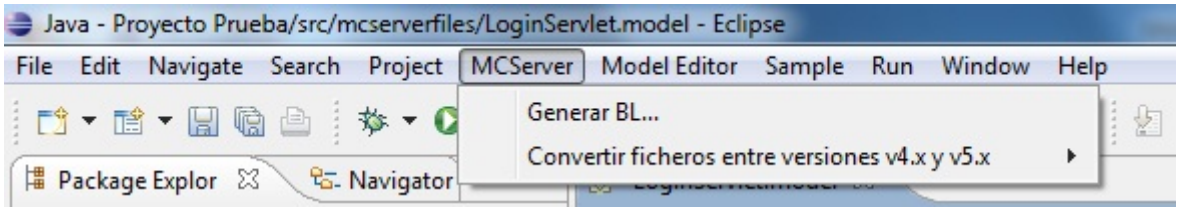


Finalmente, hacemos click con el botón derecho sobre la operativa, *New Child* y seleccionar *BL Class*.

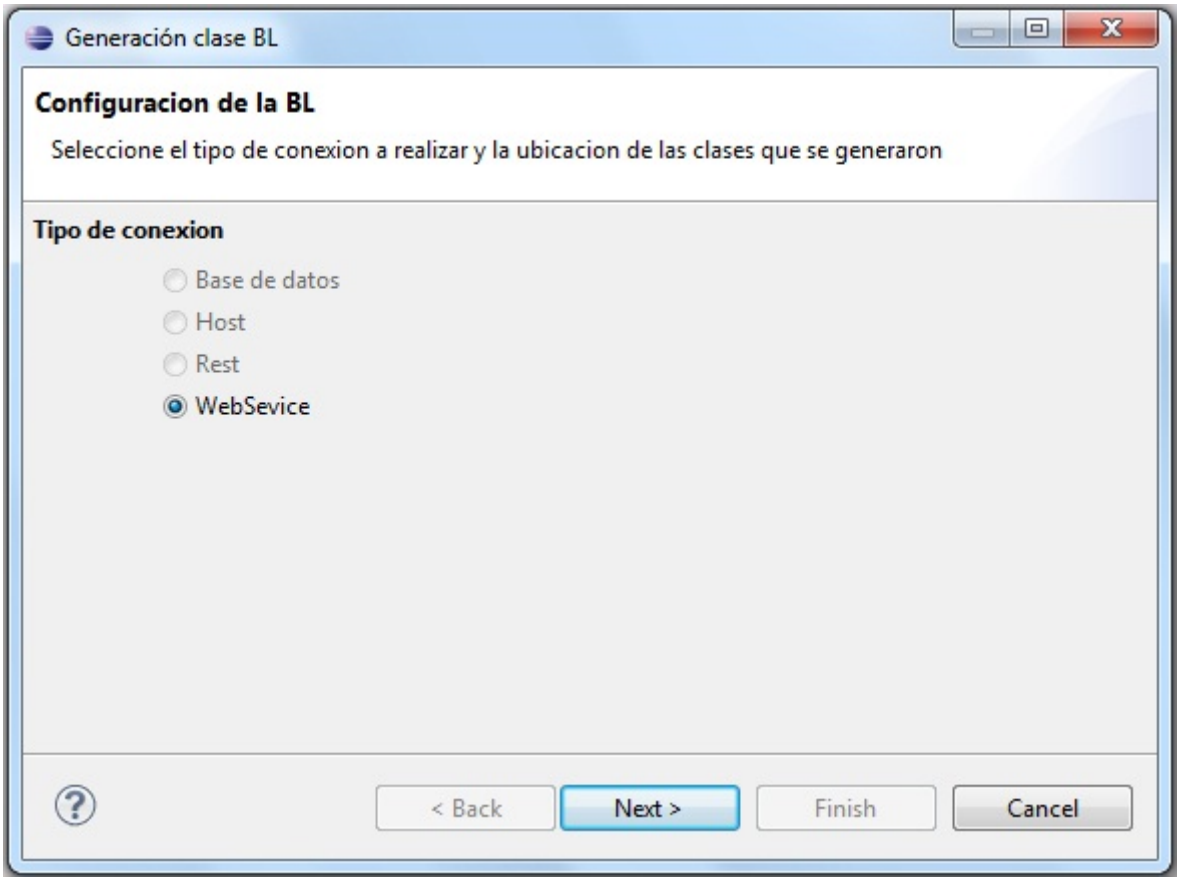


3.2.- Generación de una clase BL

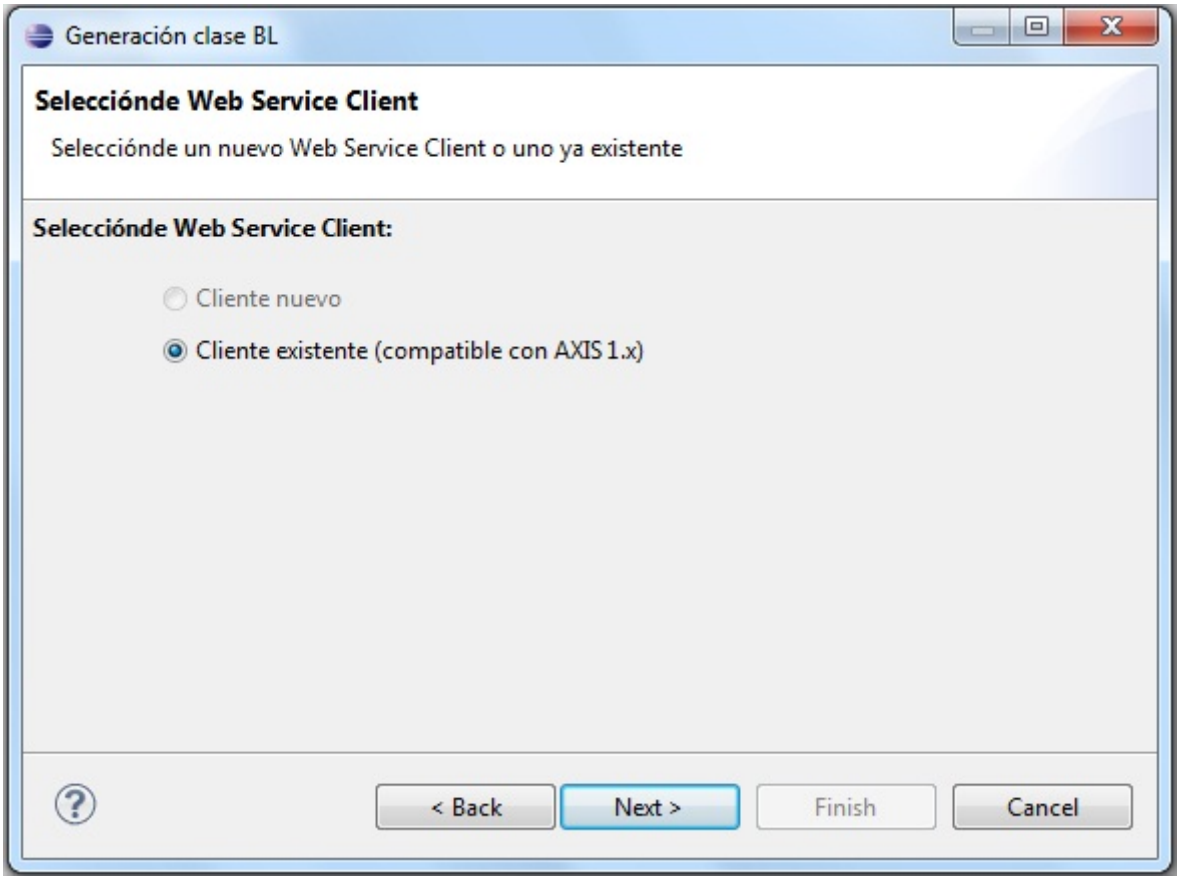
Una vez creada la clase BL en el modelo, procederemos a la generación del código java. Se realizará des de la barra de herramientas, opción MCServer como se muestra en la imagen.



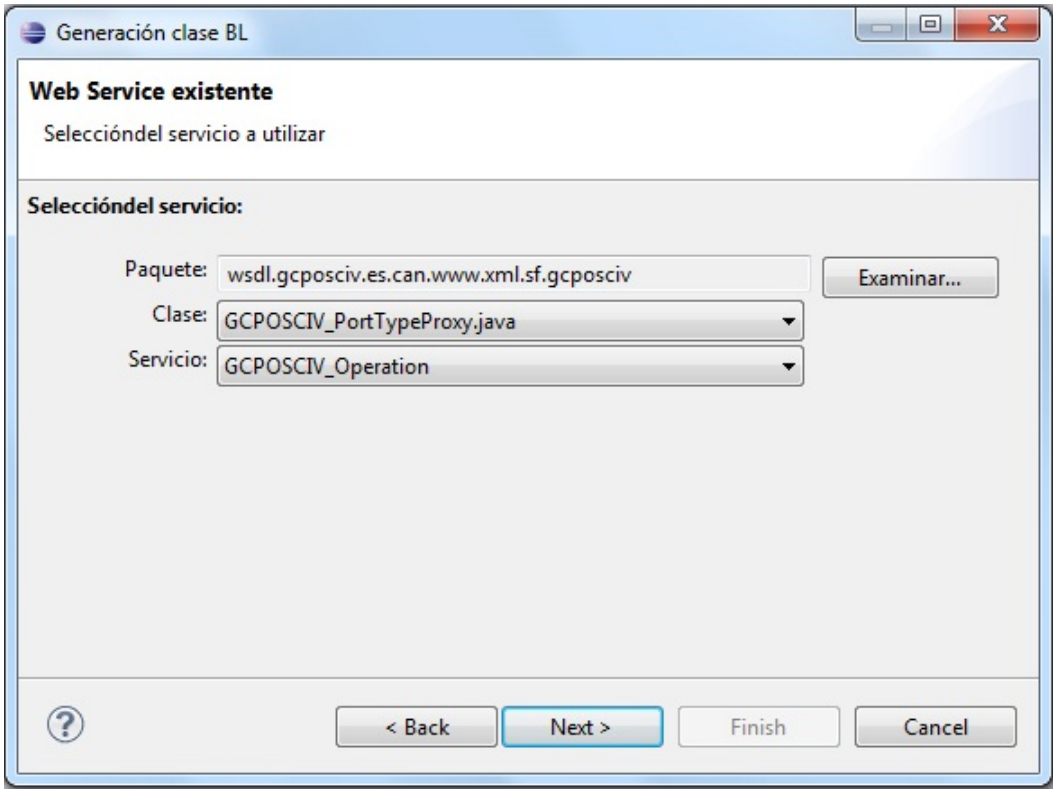
Una vez seleccionada la opción de *Generar BL*, se mostrará una pantalla como la siguiente en la que deberemos de seleccionar el tipo de conexión a generar.



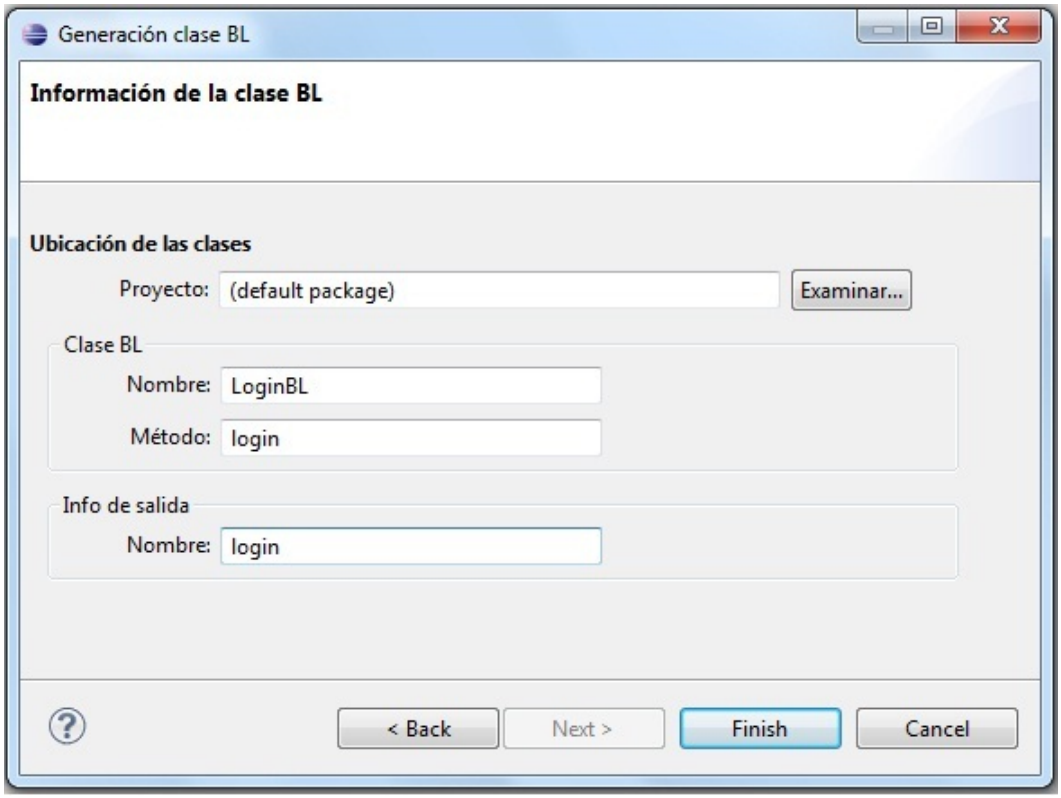
Si seleccionamos la opción de *Web Service*, se mostrará la siguiente pantalla, de la que seleccionaremos la opción de cliente AXIS que es sobre el que están hechos los servicios web con los que trabaja MCServer.



Seguidamente nos llevará a esta pantalla. Tendremos que seleccionar el paquete donde se encuentran los ficheros wsdl del servicio. Seleccionaremos la clase y el servicio deseado y pulsaremos *Next*.



Finalmente, escribiremos el nombre de la clase BL a generar, su ubicación, el nombre del método y el *info* que utilizará como salida.


Generación clase BL
Información de la clase BL
Ubicación de las clases
Proyecto:
(default package)
Examinar...
Clase BL
Nombre:
LoginBL
Método:
login
Info de salida
Nombre:
login
< Back
Next >
Finish
Cancel

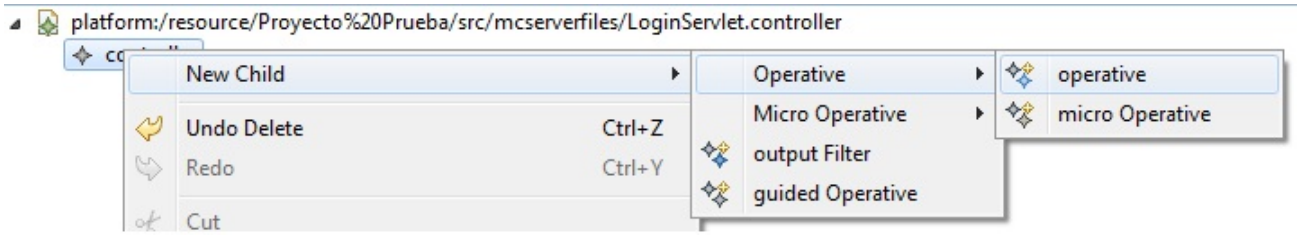
4.- Controller

Una vez creado un fichero de tipo *controller*, tendremos que poder configurarlo para su ejecución sobre la plataforma MCServer.

4.1.- Creación de una operativa

Deberemos abrir el fichero en Eclipse, haciendo doble clic sobre él.

Seguidamente, hacer click con el botón derecho, *New Child* y seleccionar *operative*.



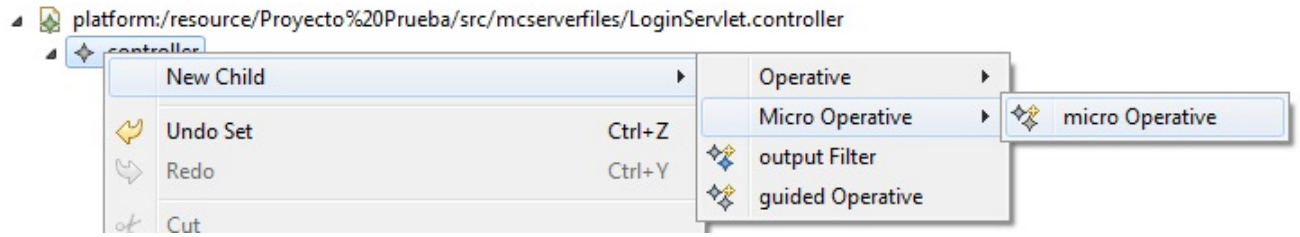
También deberemos de asignarle las propiedades necesarias para su correcto funcionamiento en MCServer.

Property	Value
Configuracion de acceso	
Iden	true
Public	true
Configuración general	
Name	LOGIN
Paginacion	
Paginated	

4.2.- Creación de una micro operativa

Deberemos abrir el fichero en Eclipse, haciendo doble clic sobre él.

Seguidamente, hacer click con el botón derecho, *New Child* y seleccionar *micro operative*.

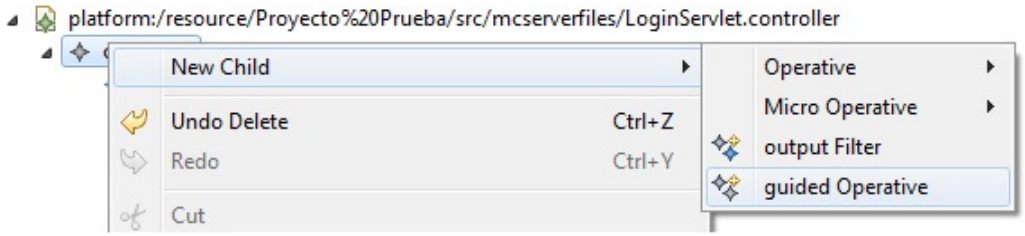


También deberemos de asignarle las propiedades necesarias para su correcto funcionamiento en MCServer.

4.3.- Creación de una operativa guiada

Deberemos abrir el fichero en Eclipse, haciendo doble clic sobre él.

Seguidamente, hacer click con el botón derecho, *New Child* y seleccionar *guided Operative*.

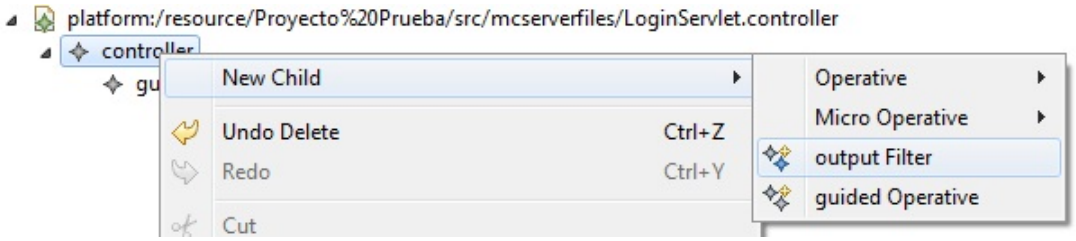


También deberemos de asignarle las propiedades necesarias para su correcto funcionamiento en MCServer como mostramos en la imagen. Además, para este tipo de operativas, se posee un editor gráfico para que resulte más sencillo su modelado.

4.4.- Creación de un outPutFilter

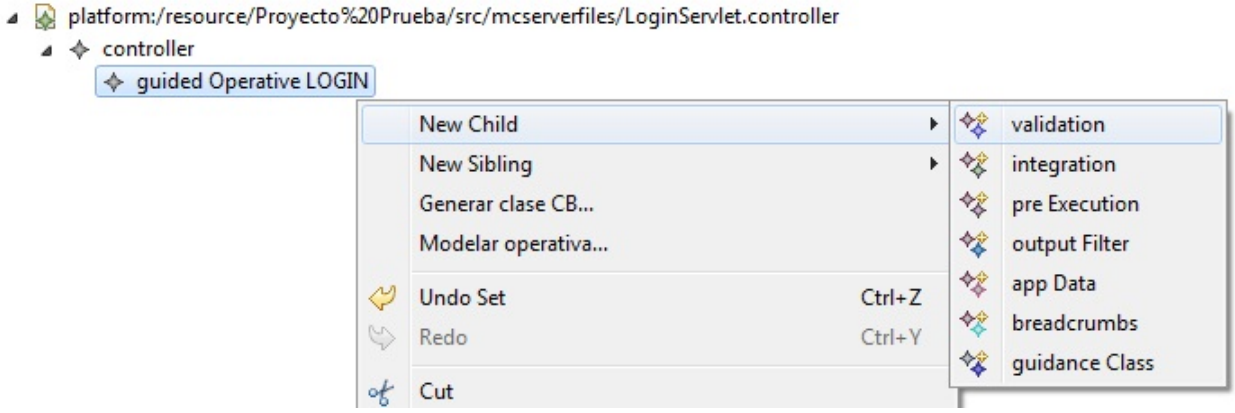
Deberemos abrir el fichero en Eclipse, haciendo doble clic sobre él.

Seguidamente, hacer click con el botón derecho, *New Child* y seleccionar *output Filter*.

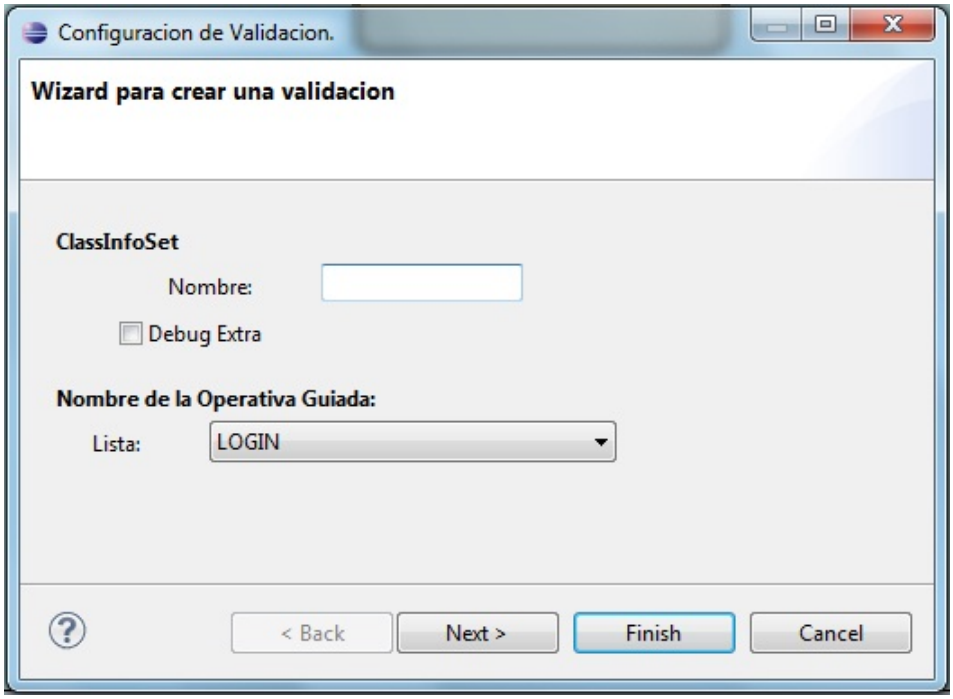


4.5.- Creación de un validation

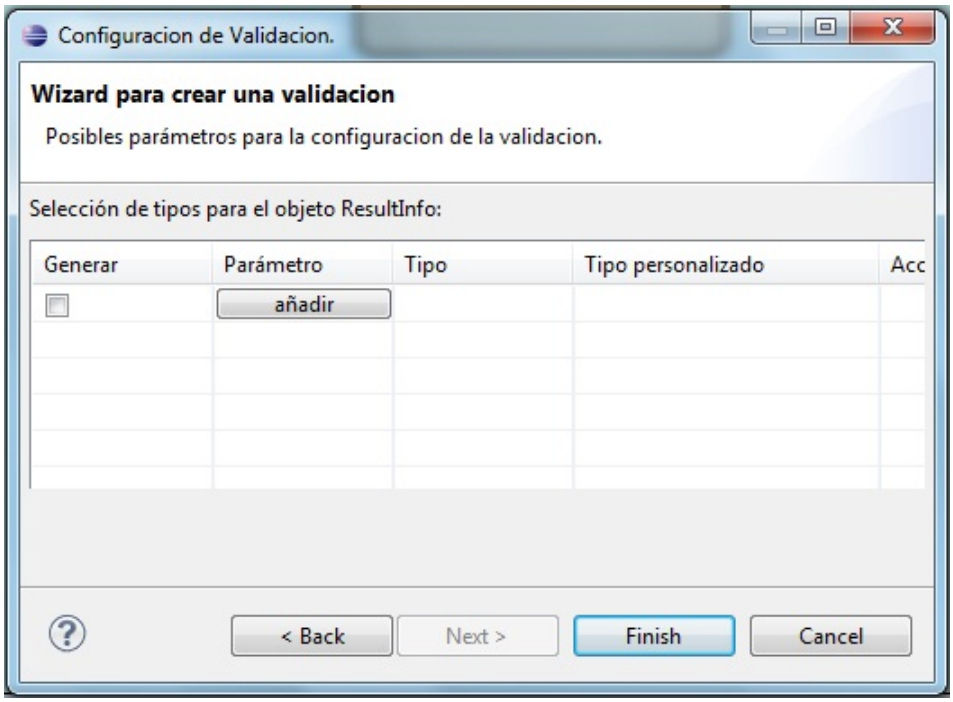
Para crear un elemento de este tipo, haremos click con el botón derecho sobre la operativa, *New Child* y elegiremos *validation*. Para editar sus propiedades lo haremos desde la pestaña inferior *Propiedades*.



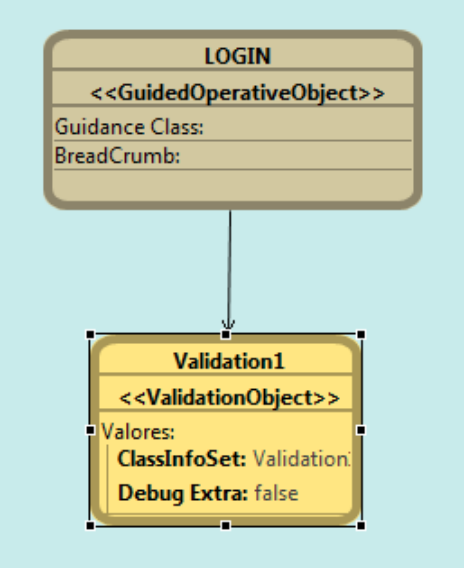
Para el caso de las operativas guiadas, se podrá crear el mismo objeto pero desde un editor gráfico. Seleccionamos en la paleta de la izquierda, el elemento *validation* y al hacer clic en la pantalla del editor, se mostrará un wizard como el que se adjunta a continuación. Asignaremos un nombre a la validación y seleccionaremos la operativa guiada sobre la que deseamos realizarla.



Una vez realizado esto, observaremos la siguiente pantalla donde introduciremos los parámetros a comprobar por el *validation*. Y al pulsar finalizar ya tendremos creada nuestro elemento.



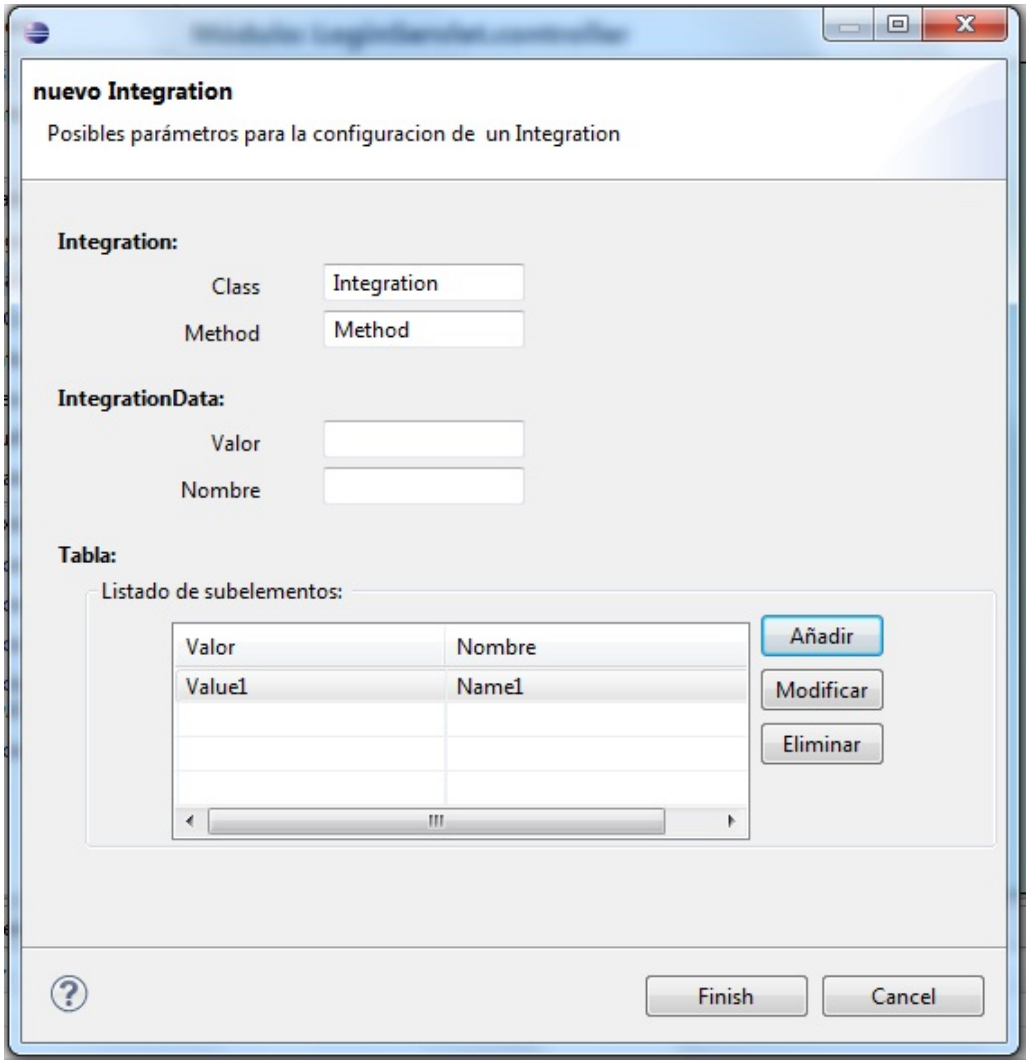
El aspecto final en el editor gráfico será el que mostramos en la imagen. Podremos editar el elemento haciendo click con el botón derecho. También podremos eliminar la conexión con la operativa guiada.



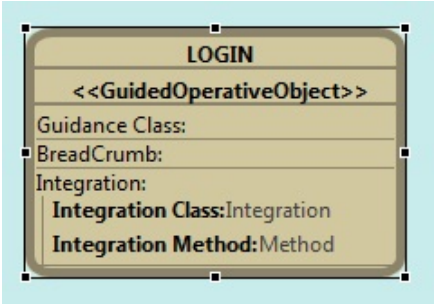
4.6.- Creación de un integration

Para crear un elemento de este tipo, haremos click con el botón derecho sobre la operativa, *New Child* y elegiremos *integration*. Para editar sus propiedades lo haremos desde la pestaña inferior *Propiedades*.

Para el caso de las operativas guiadas, se podrá crear el mismo objeto pero desde un editor gráfico. Seleccionamos en la paleta de la izquierda, el elemento *integration* y al hacer clic sobre la operativa, se mostrará un wizard como el que se adjunta a continuación. Asignaremos un nombre a el integration y un método, y añadiremos tantos integration data queramos.



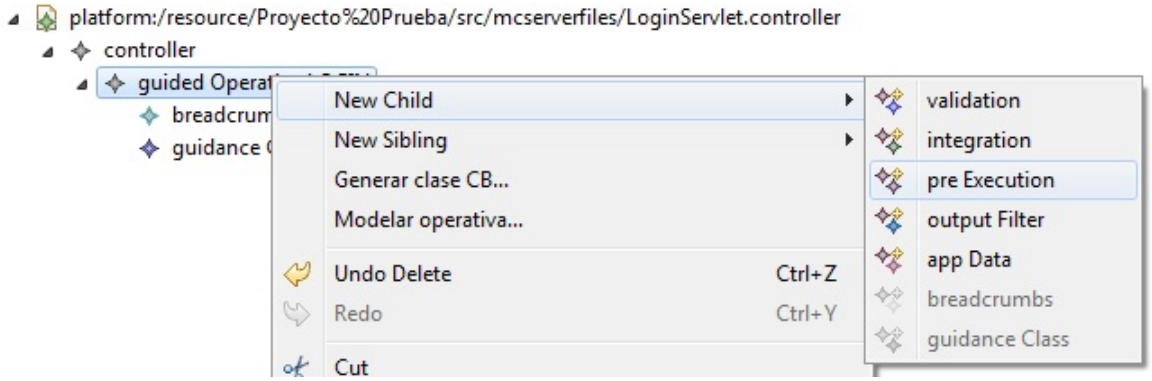
El aspecto final en el editor gráfico será el que mostramos en la imagen. Podremos editar el elemento haciendo click con el botón derecho sobre la operativa guiada.



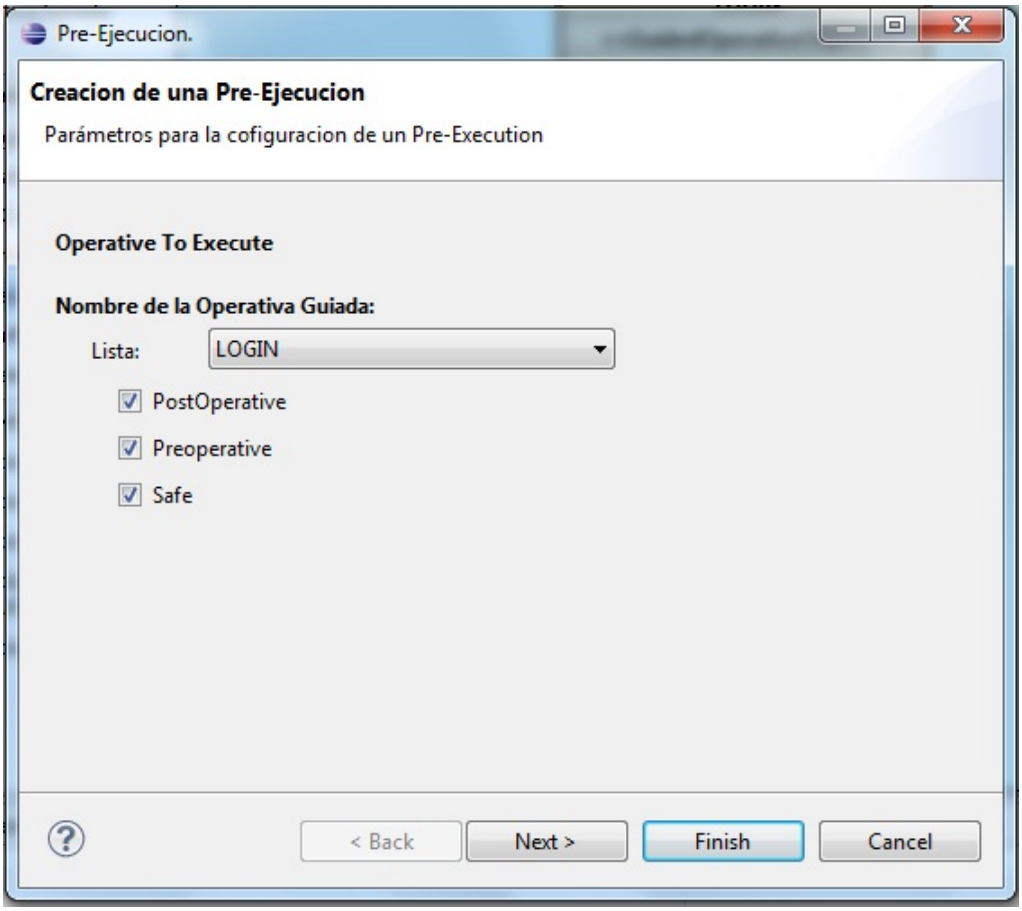
4.7.- Creación de una preExecution

Para crear un elemento de este tipo, haremos click con el botón derecho sobre la

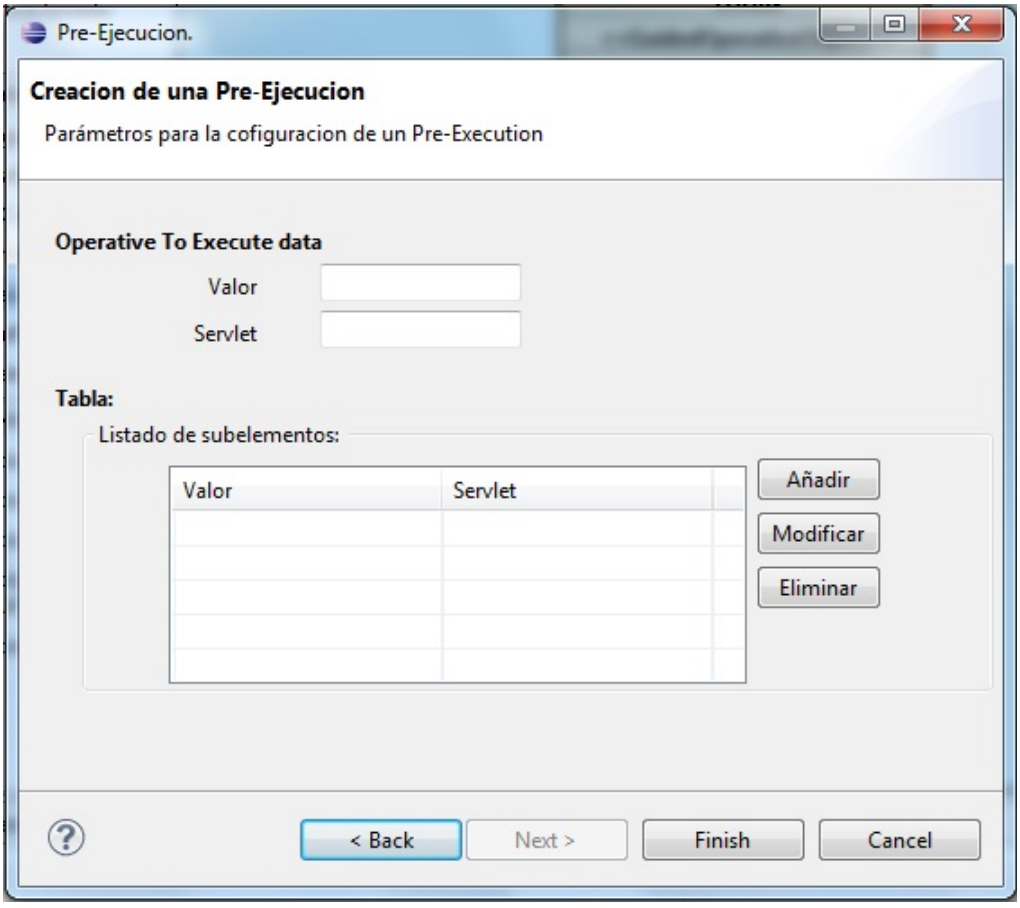
operativa, *New Child* y elegiremos *preExecution*. Para editar sus propiedades lo haremos desde la pestaña inferior *Propiedades*.



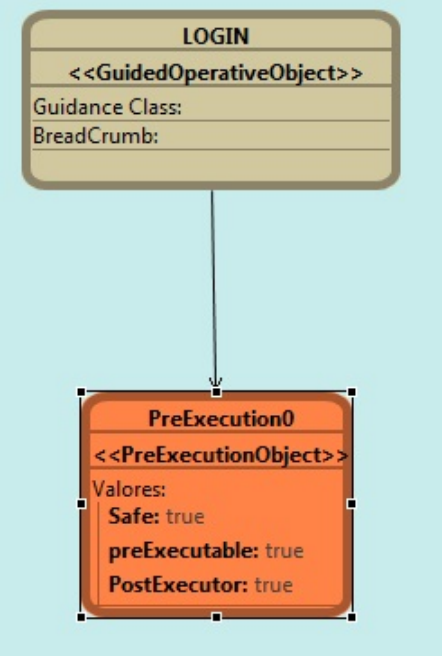
Para el caso de las operativas guiadas, se podrá crear el mismo objeto pero desde un editor gráfico. Seleccionamos en la paleta de la izquierda, el elemento *preExecution* y al hacer clic sobre el editor, se mostrará un wizard como el que se adjunta a continuación. Elegiremos la operativa guiada asociada y seleccionaremos las opciones que nos sean necesarias.



A continuación se mostrarán la pantalla que nos permite añadir una lista de operativas pre-ejecutables, en caso de que el checkBox inicial haya sido seleccionado.



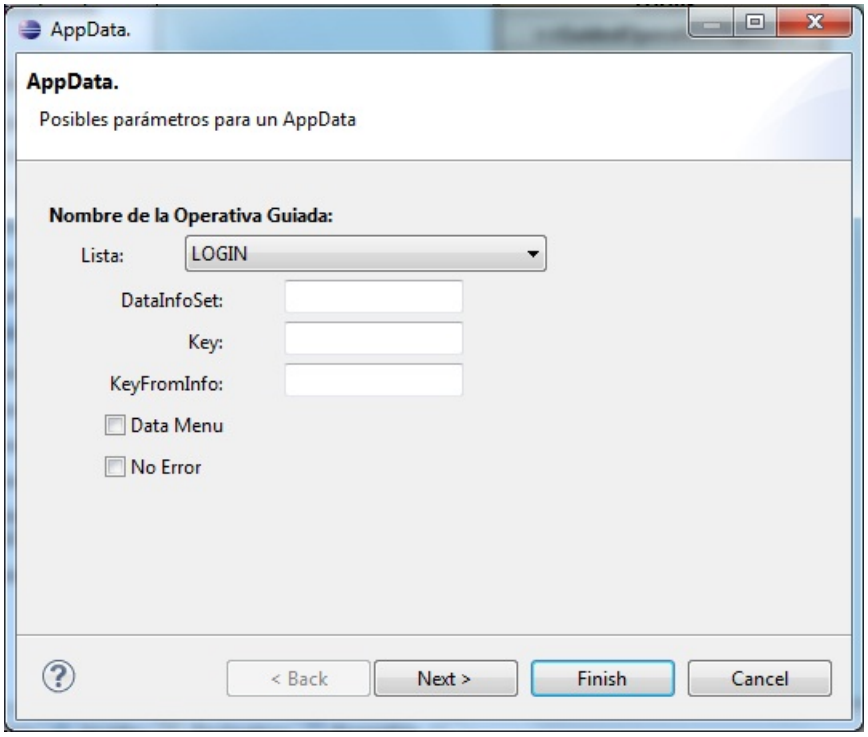
El aspecto final en el editor gráfico será el que mostramos en la imagen. Podremos editar/eliminar el elemento, haciendo click con el botón derecho sobre el propio elemento.



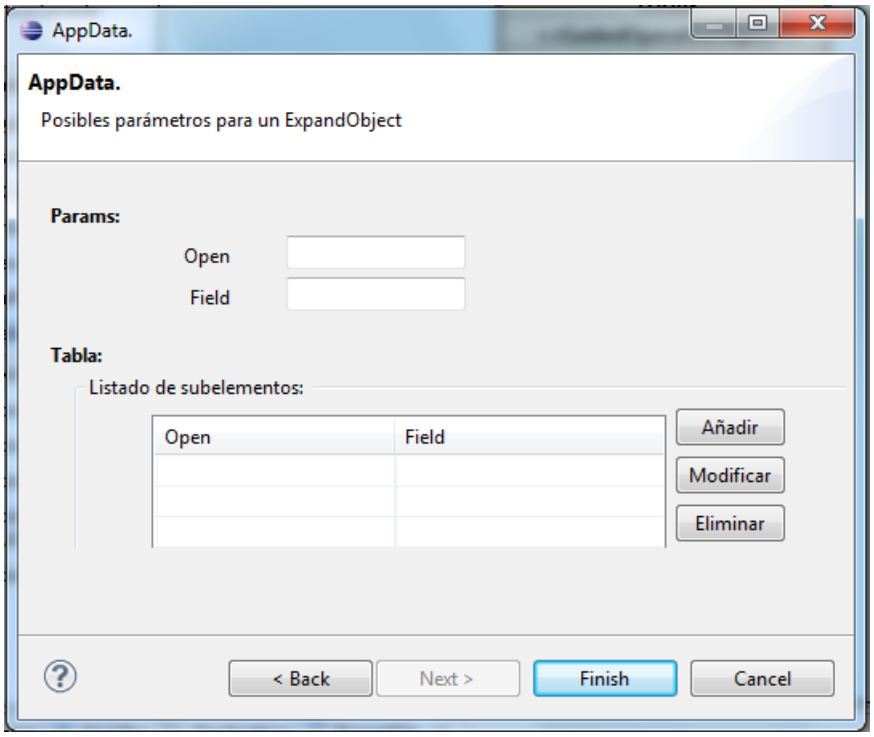
4.8.- Creación de un appData

Para crear un elemento de este tipo, haremos click con el botón derecho sobre la operativa, *New Child* y elegiremos *appData*. Para editar sus propiedades lo haremos desde la pestaña inferior *Propiedades*.

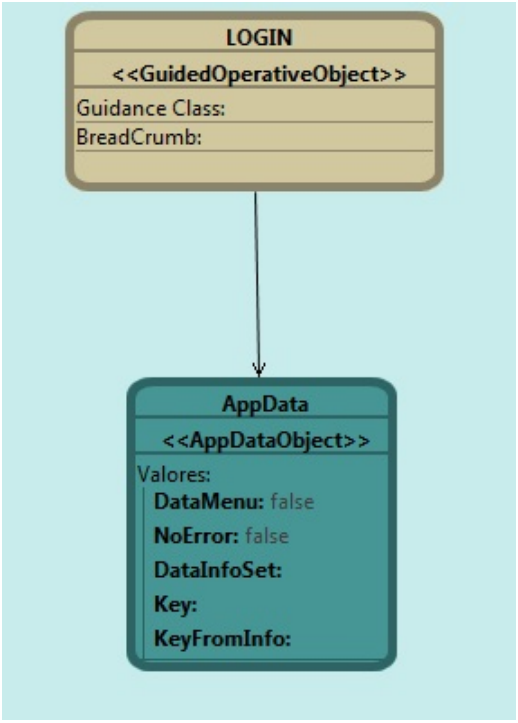
Para el caso de las operativas guiadas, se podrá crear el mismo objeto pero desde un editor gráfico. Seleccionamos en la paleta de la izquierda, el elemento *appData* y al hacer clic sobre el editor, se mostrará un wizard como el que se adjunta a continuación. Seleccionaremos la operativa guiada a la que se le asignará.



En la siguiente pantalla, se podrán añadir parámetros de tipo *ExpandObject*, como se muestra a continuación.

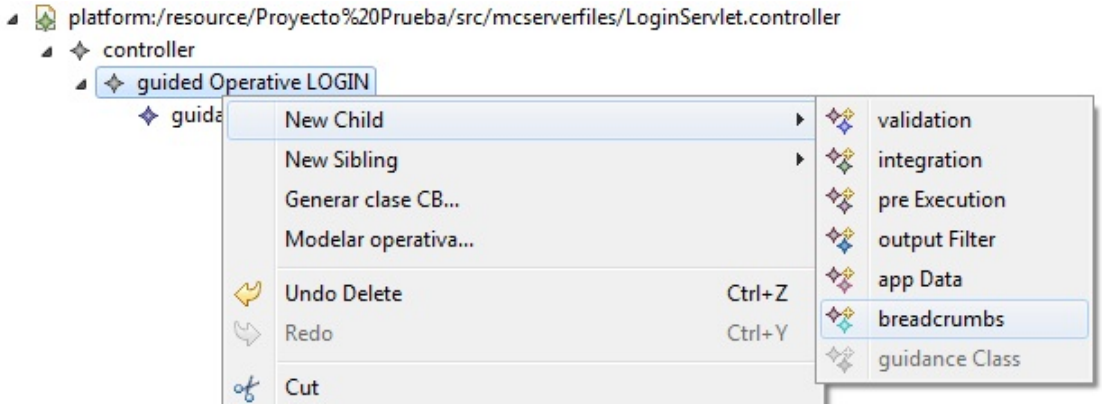


El aspecto final en el editor gráfico será el que mostramos en la imagen. Podremos editar/eliminar el elemento, haciendo click con el botón derecho sobre el propio elemento.



4.9.- Creación de un breadcrumb

Para crear un elemento de este tipo, haremos click con el botón derecho sobre la operativa, *New Child* y elegiremos *breadcrumb*. Para editar sus propiedades lo haremos desde la pestaña inferior *Propiedades*.

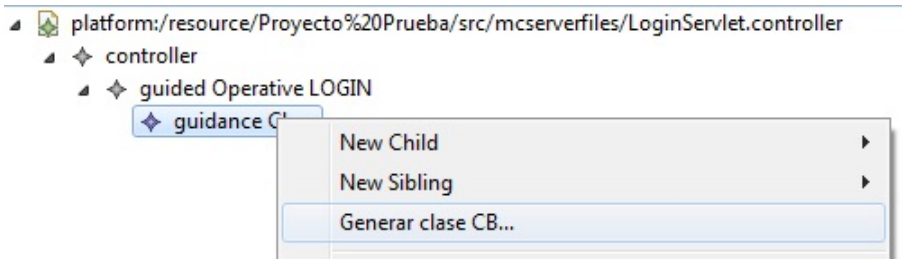


4.10.- Creación de una clase CB

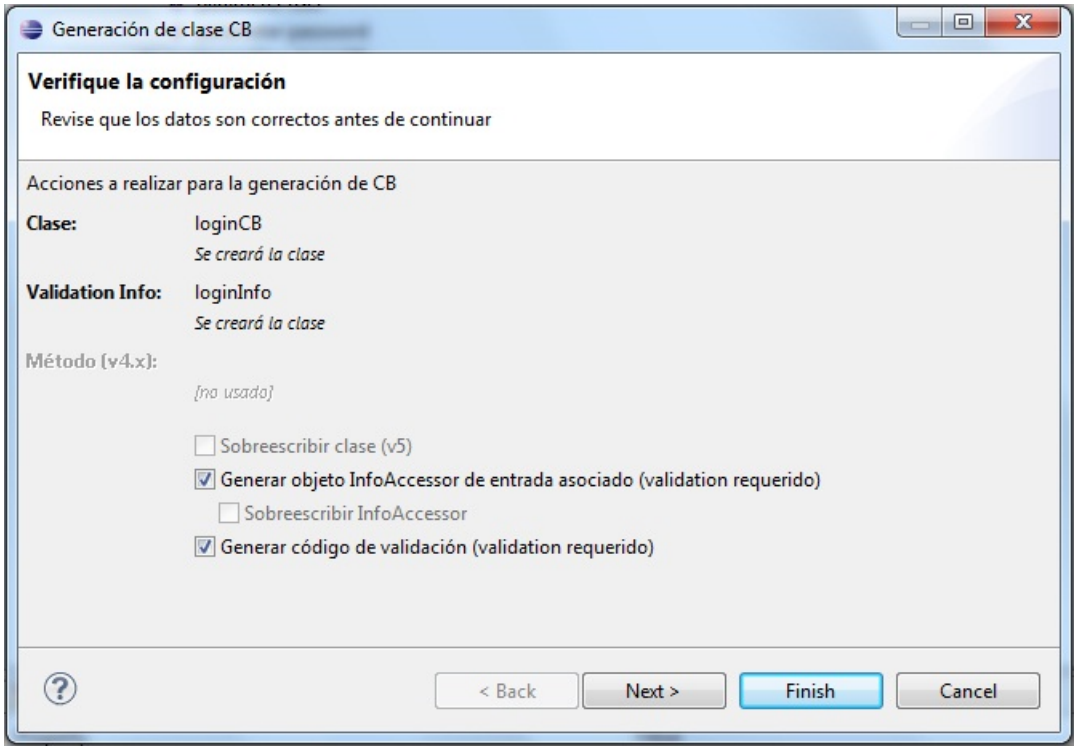
Para crear un elemento de este tipo, haremos click con el botón derecho sobre la operativa, *New Child* y elegiremos *guidance Class*. Para editar sus propiedades lo haremos desde la pestaña inferior *Propiedades*.

4.11.- Generación de código de una clase CB

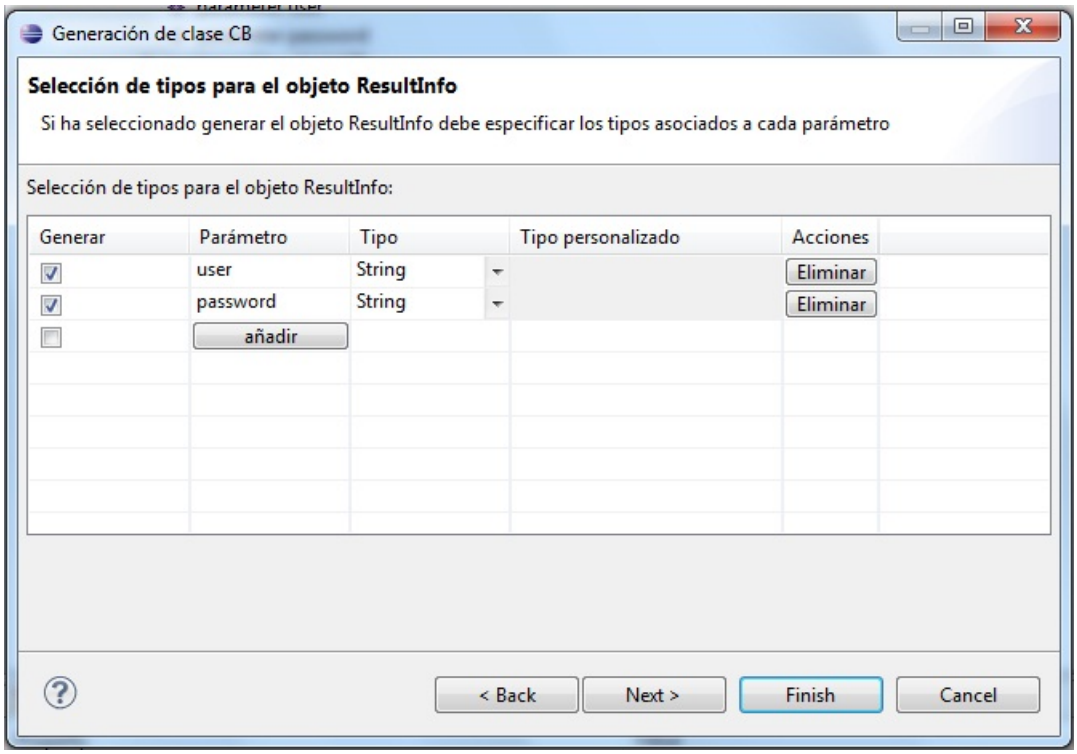
Para la generación del código de la clase CB, haremos click con el botón derecho sobre la guidance class, y elegiremos *Generar clase CB*. Para poder generar el código de la operativa, necesitaremos tener creado un elemento validation con los parámetros que se desee validar.



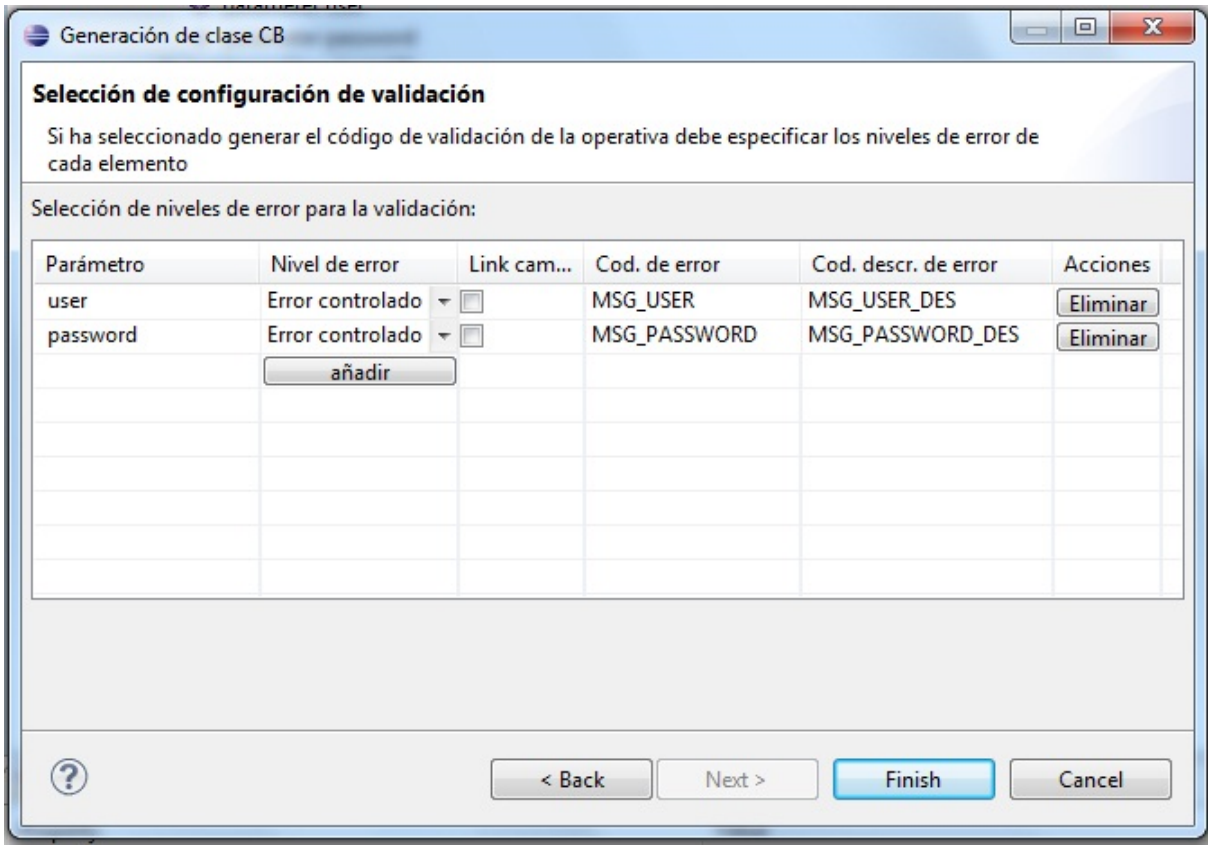
Al seleccionar esa opción del menú, se abrirá una pantalla como la siguiente para configurar la generación de la clase.



La pantalla siguiente permite la edición del resultInfo que devolverá la CB.



Y por último, la pantalla de configuración de la validación de la operativa, que nos permitirá añadir a la clase la generación de las constantes para cada atributo. Al presionar sobre *Finish*, se habrá generado dicha clase Java.



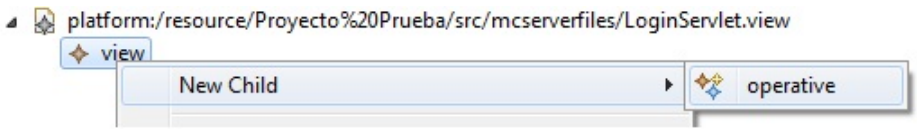
5.- View

Una vez creado un fichero de tipo *view*, tendremos que poder configurarlo para su ejecución sobre la plataforma MCServer.

5.1.- Creación de una operativa

Deberemos abrir el fichero en Eclipse, haciendo doble clic sobre él.

Seguidamente, hacer click con el botón derecho, *New Child* y seleccionar *operative*.



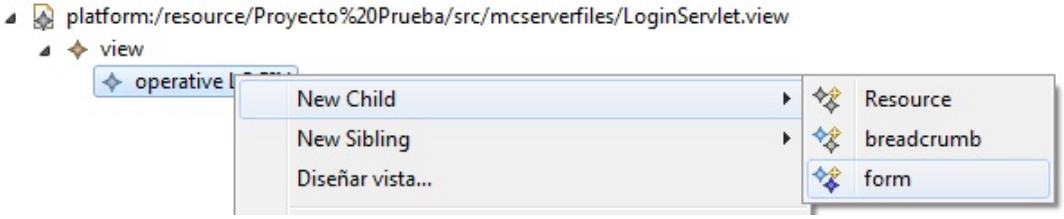
También deberemos de asignarle las propiedades necesarias para su correcto funcionamiento en MCServer.

Property	Value
Configuración general	
Debug Extra	false
Transformer	DEFAULT
Misc	
Name	LOGIN
Redireccion	
External Resource	
Forward	false
Redirect	false

5.2.- Creación de un form

Deberemos abrir el fichero en Eclipse, haciendo doble clic sobre él.

Seguidamente, hacer click con el botón derecho, *New Child* y seleccionar *form*.



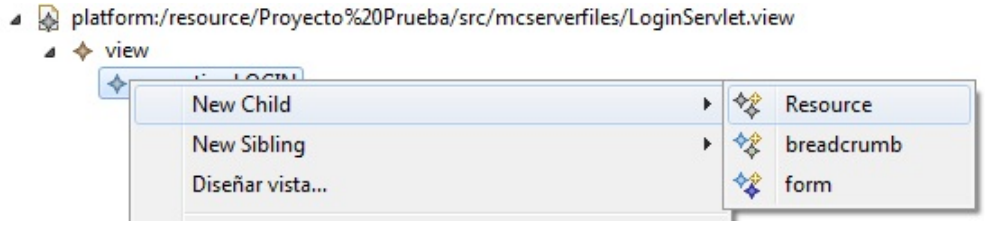
También se puede crear el form automáticamente, presionando sobre la opción *Diseñar vista*.

5.3.- Creación de un source

Deberemos abrir el fichero en Eclipse, haciendo doble clic sobre él.

Seguidamente, hacer click con el botón derecho, *New Child* y seleccionar *Resource*.

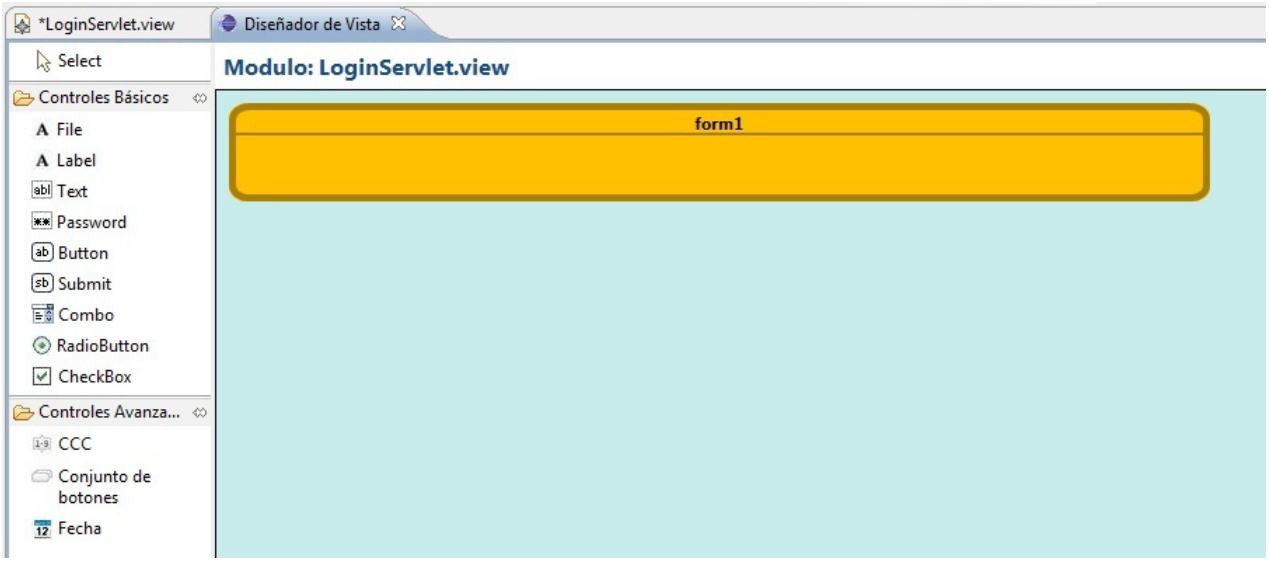
En este parámetro, deberemos de indicar la ruta del fichero .xsl a mostrar por la aplicación.



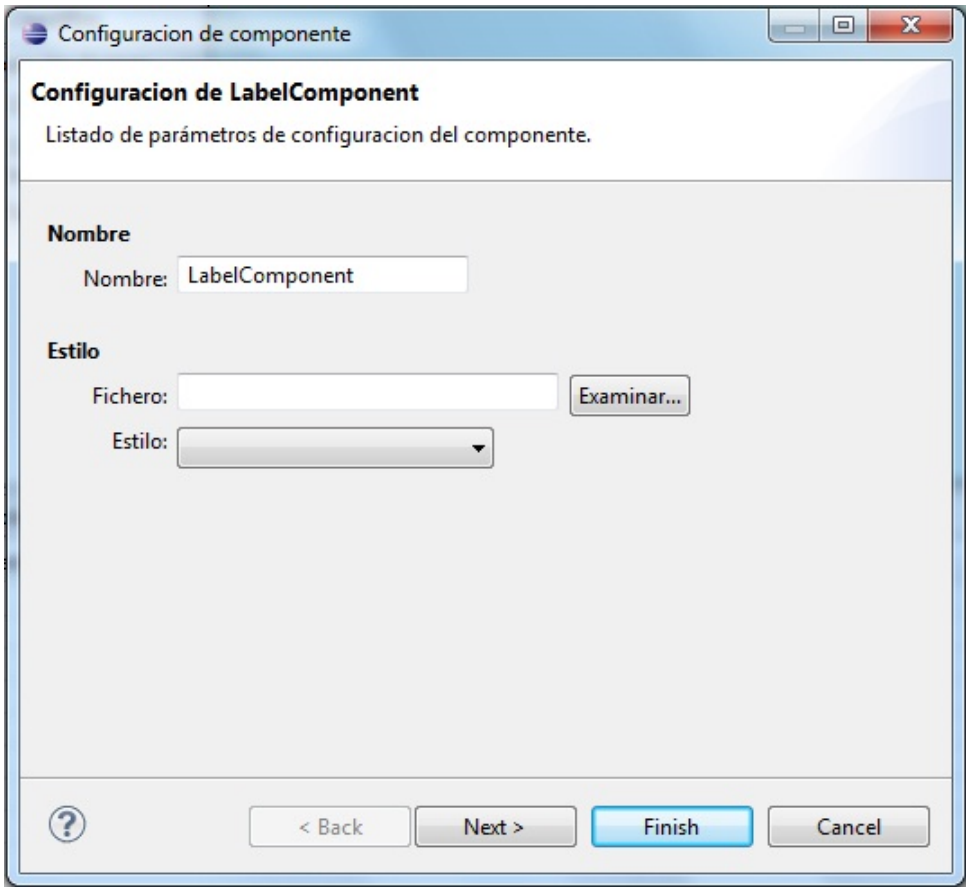
5.4.- Creación de fields

Para la creación de fields, se utilizará un editor gráfico muy intuitivo que permitirá editar las propiedades de todos los componentes del formulario.

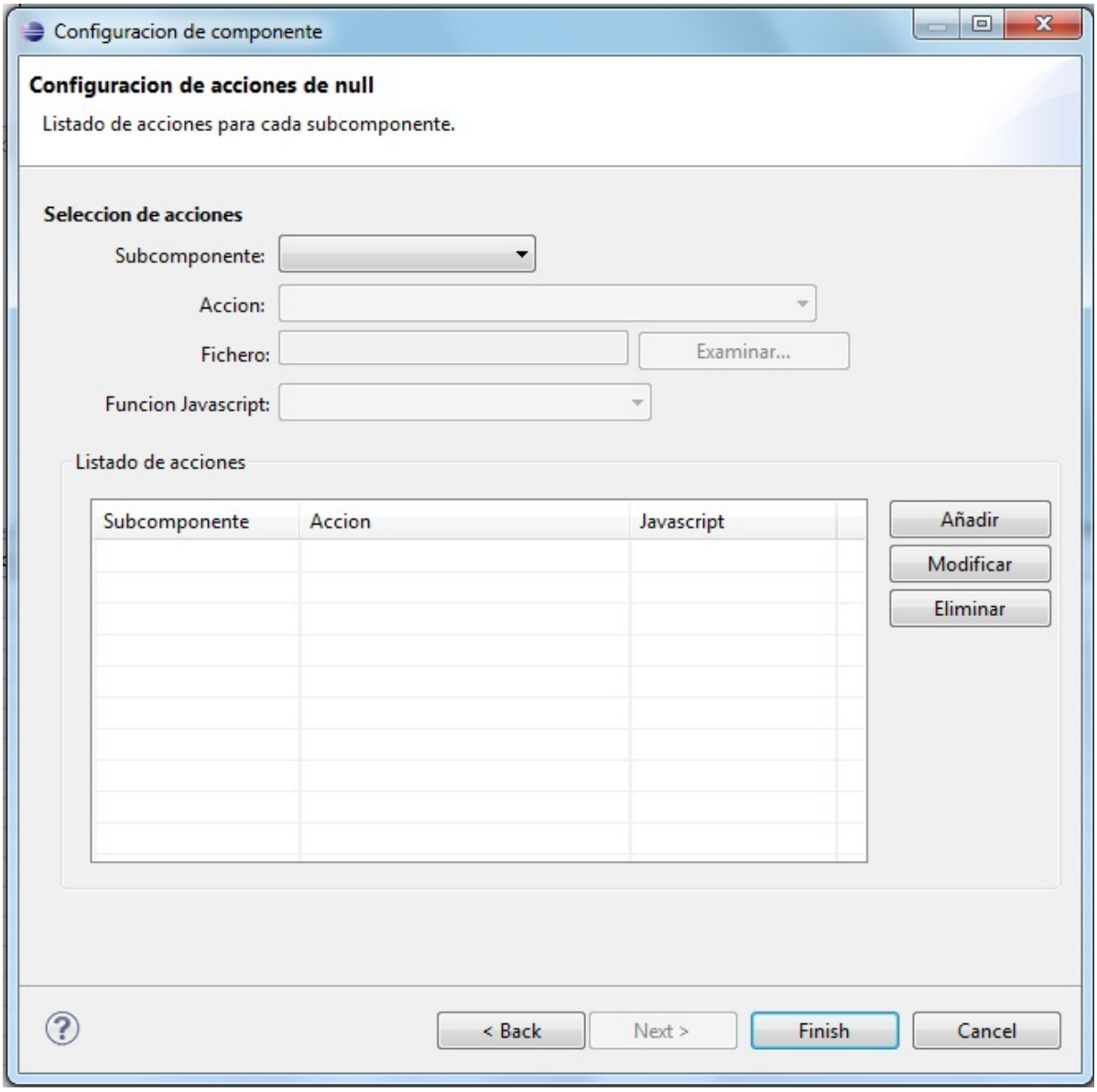
Para acceder a dicho editor, seleccionaremos la opción *Diseñar vista*, como se muestra en la imagen a continuación.



Para la creación de nuevas componentes del formulario, se deberá seleccionar el que se desee de la paleta de la izquierda. Seguidamente se abrirá una ventana emergente de configuración general del componente como la que mostramos a continuación, y que será común para todos ellos.



También existirá una pantalla común para todos los componentes, que servirá para asignar acciones Javascript a cada subcomponente, como la que mostramos a continuación.

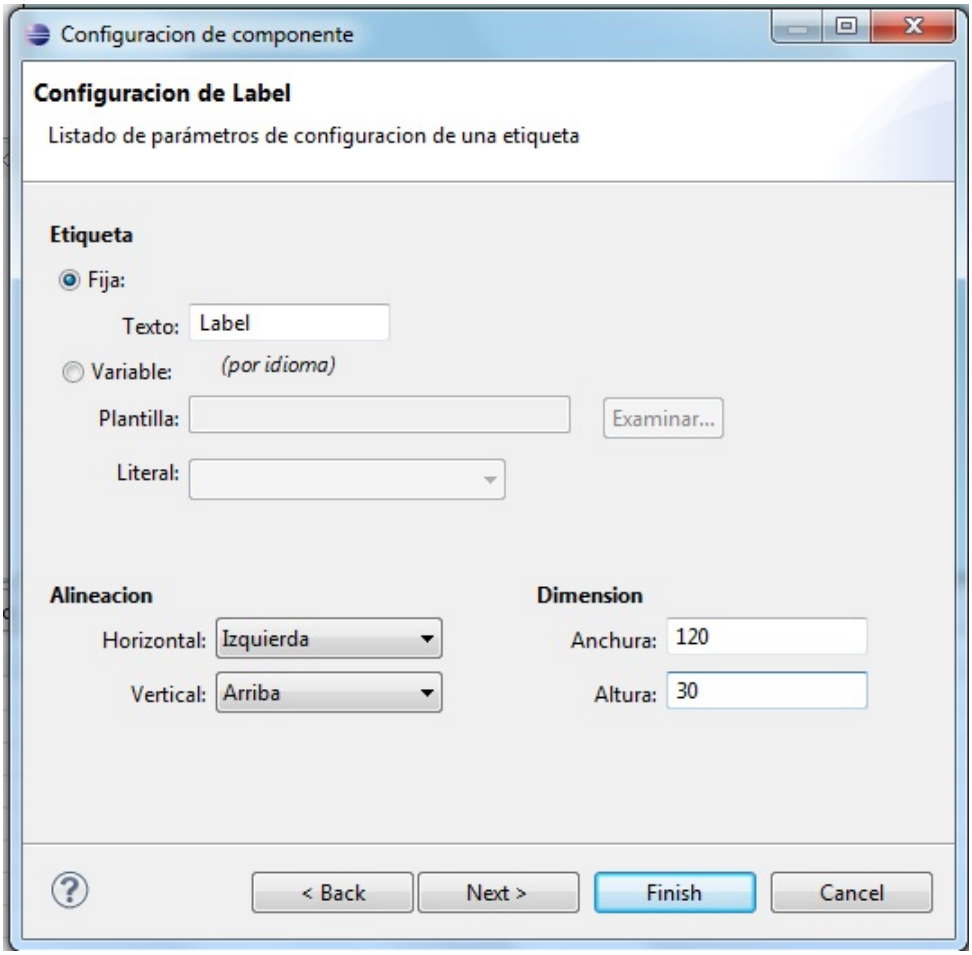


Y para la edición individual de las propiedades de cada subcomponente, presionando con el botón derecho sobre uno de ellos y seleccionando la opción *Editar*, se nos mostrará una ventana emergente similar a las de creación de un nuevo elemento.

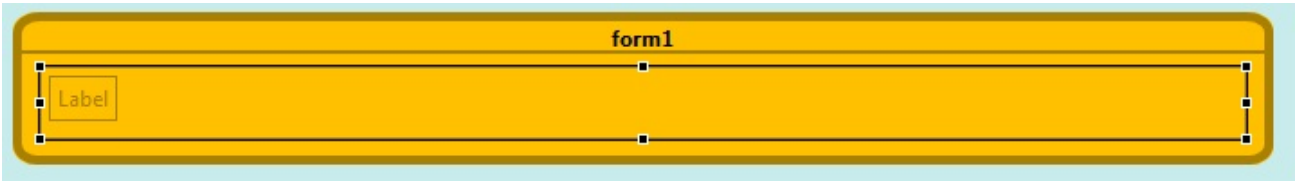
A continuación, explicamos brevemente el proceso de creación de cada uno de los elementos del formulario.

5.4.1.- Label

Para la creación de este componente, deberemos presionar en la paleta de la izquierda sobre el elemento *Label*. Seguidamente se nos abrirá una ventana emergente de configuración general del componente como se muestra en la introducción de este apartado. Después se mostrará una pantalla de configuración propia del componente como la que adjuntamos a continuación.

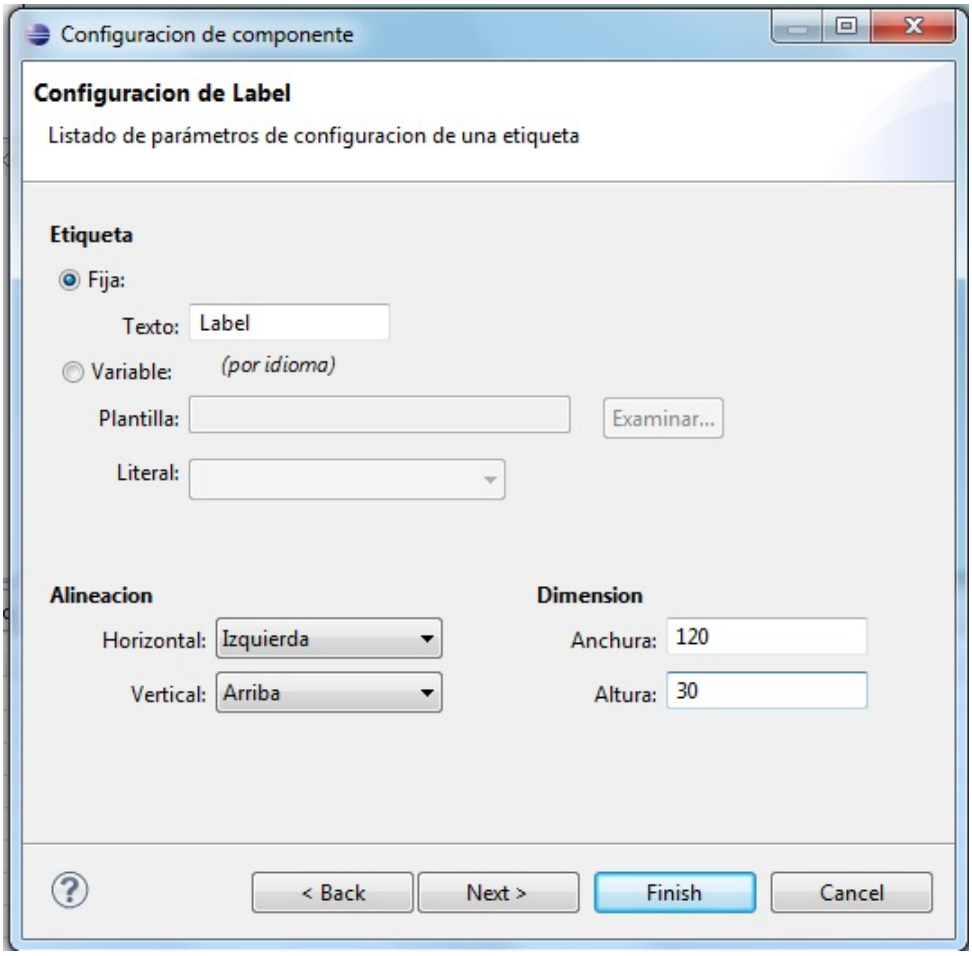


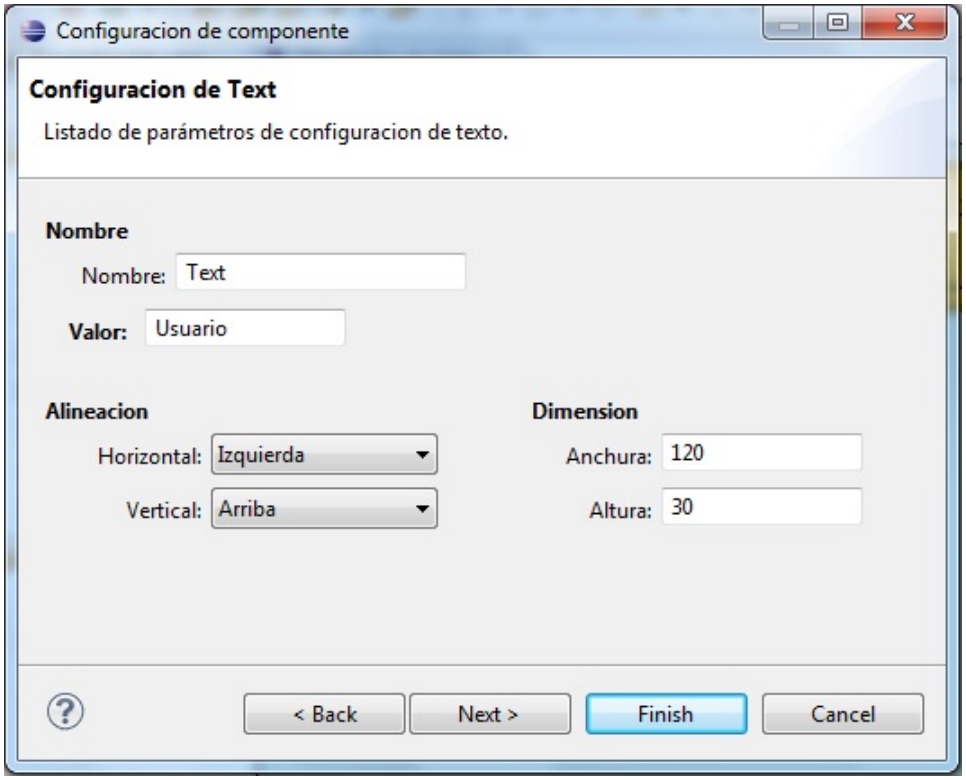
Finalmente, el aspecto del componente en el formulario será similar al que mostramos a continuación.



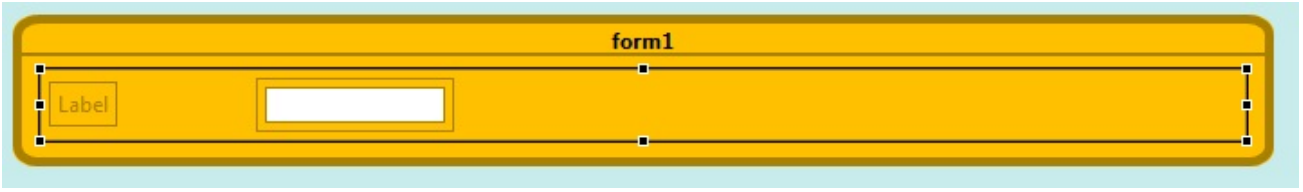
5.4.2.- Text

Para la creación de este componente, deberemos presionar en la paleta de la izquierda sobre el elemento *Text*. Seguidamente se nos abrirá una ventana emergente de configuración general del componente como se muestra en la introducción de este apartado. Después se mostrará una pantalla de configuración propia del componente como la que adjuntamos a continuación.



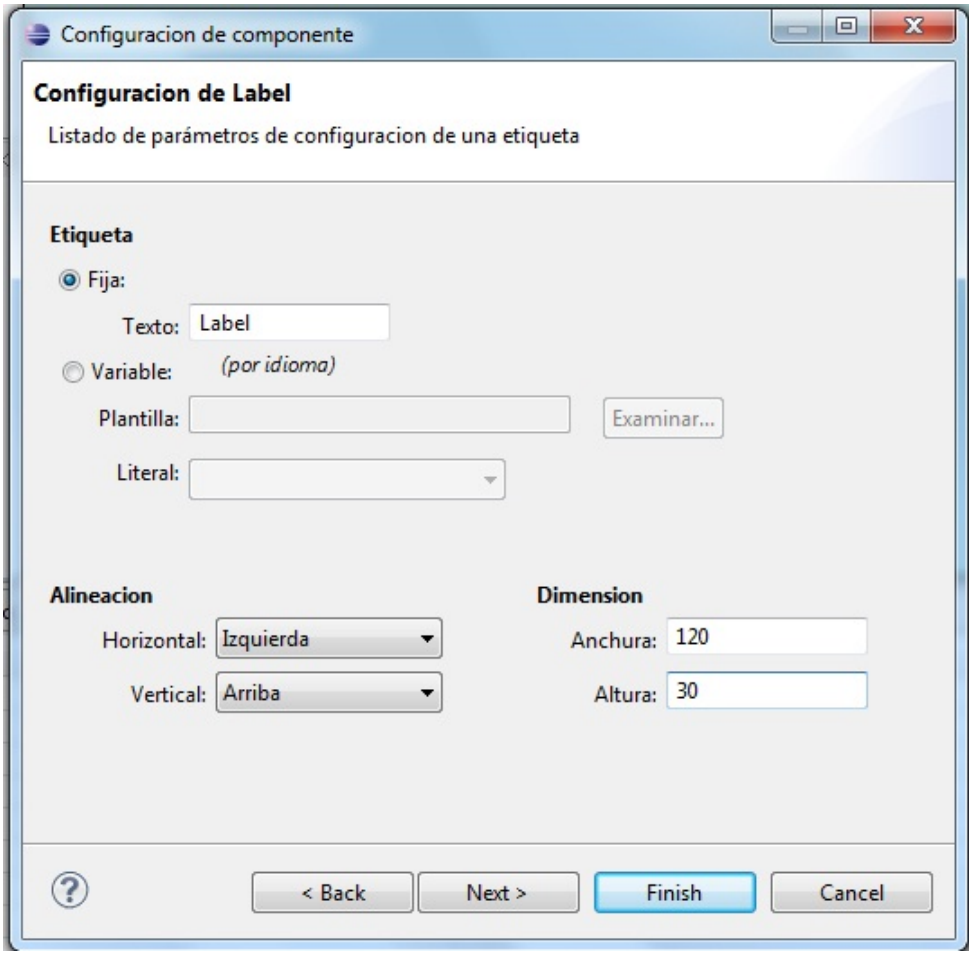


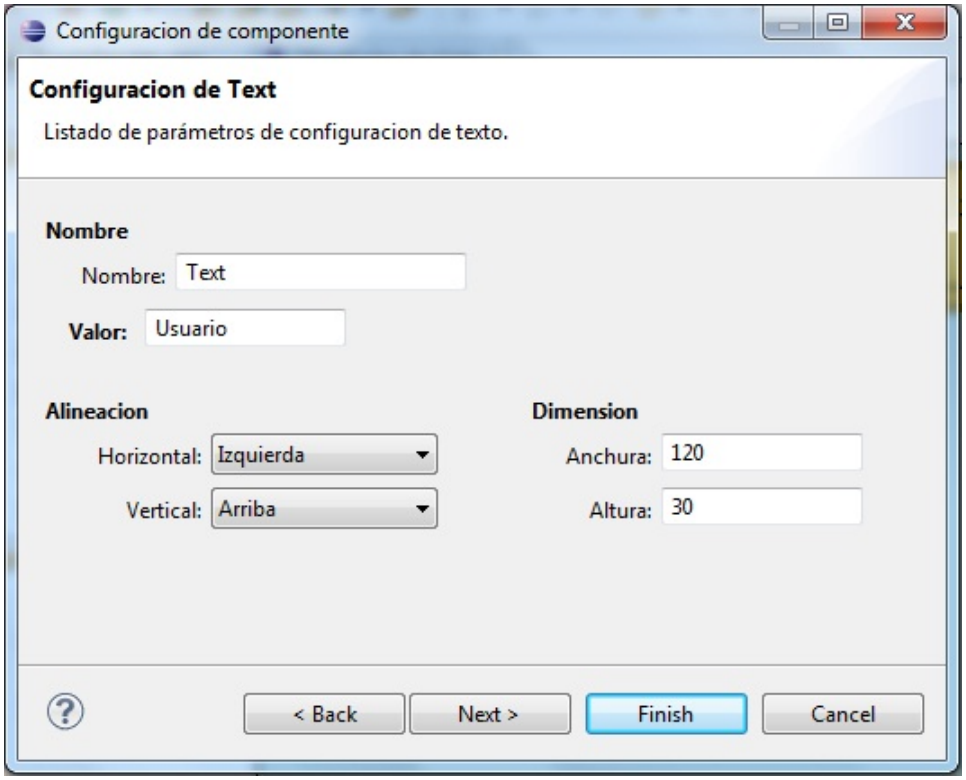
Finalmente, el aspecto del componente en el formulario será similar al que mostramos a continuación.



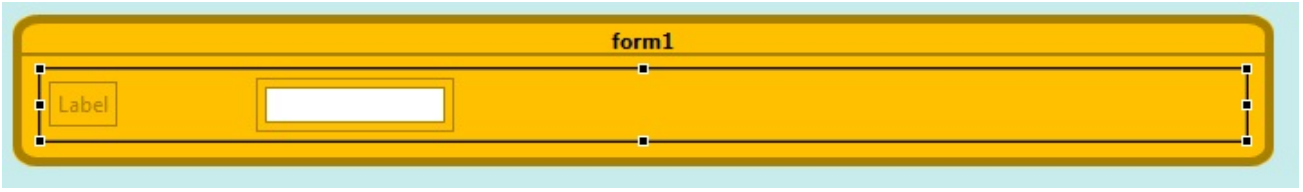
5.4.3.- Password

Para la creación de este componente, deberemos presionar en la paleta de la izquierda sobre el elemento *Password*. Seguidamente se nos abrirá una ventana emergente de configuración general del componente como se muestra en la introducción de este apartado. Después se mostrará una pantalla de configuración propia del componente como la que adjuntamos a continuación.



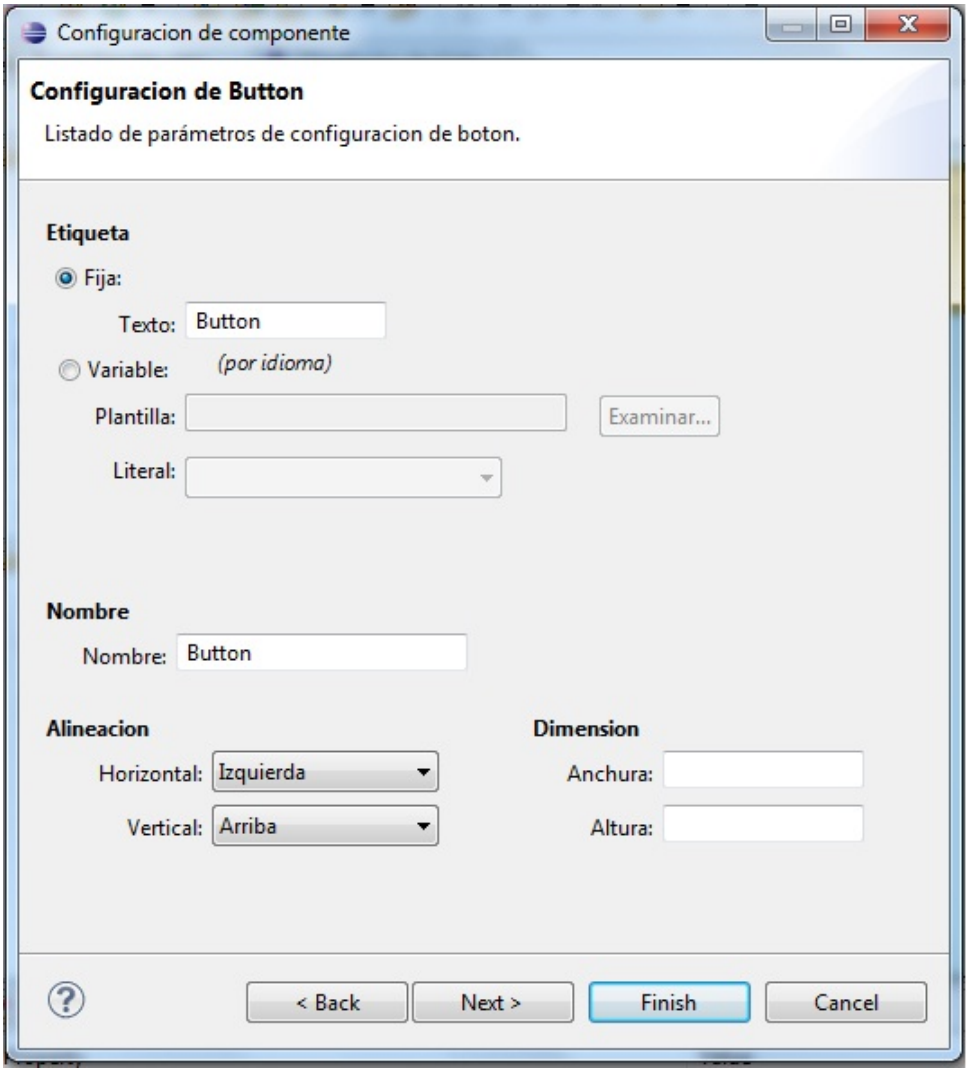


Finalmente, el aspecto del componente en el formulario será similar al que mostramos a continuación.

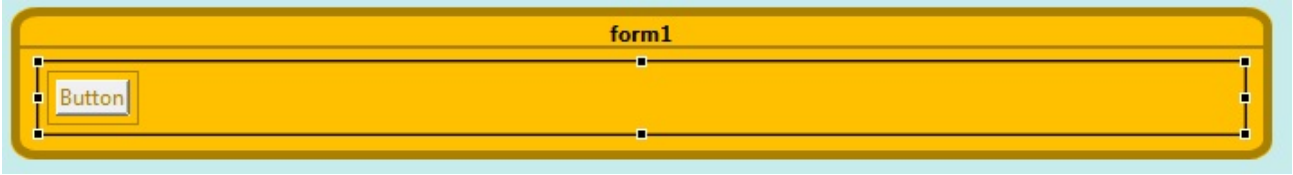


5.4.4.- Button

Para la creación de este componente, deberemos presionar en la paleta de la izquierda sobre el elemento *Button*. Seguidamente se nos abrirá una ventana emergente de configuración general del componente como se muestra en la introducción de este apartado. Después se mostrará una pantalla de configuración propia del componente como la que adjuntamos a continuación.

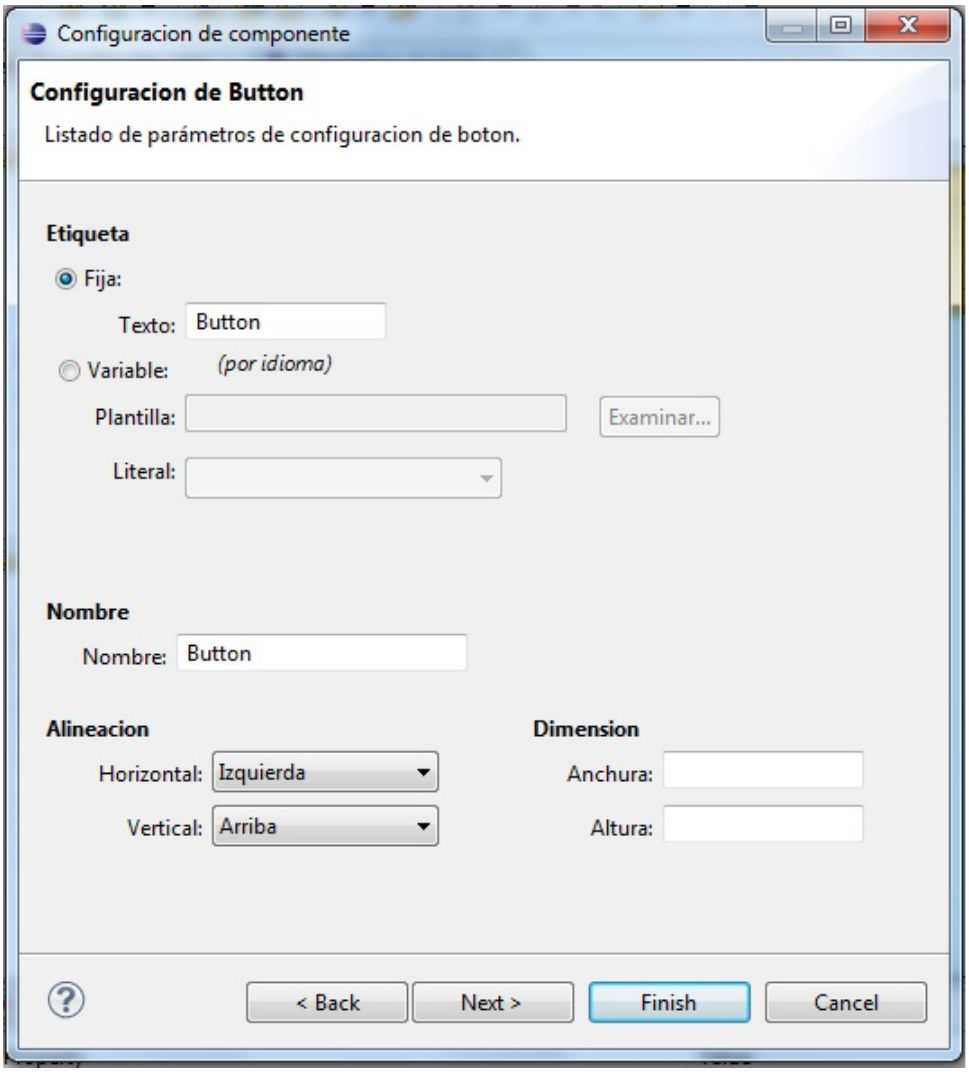


Finalmente, el aspecto del componente en el formulario será similar al que mostramos a continuación.

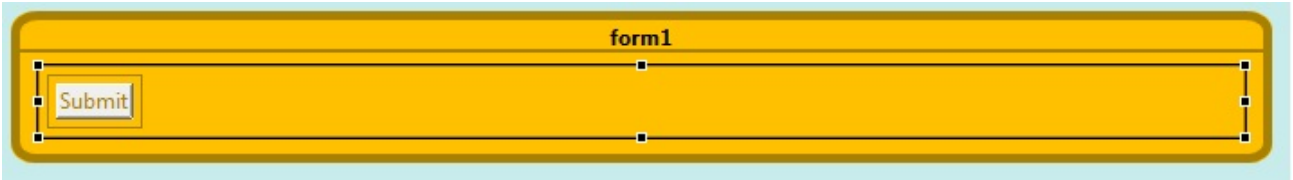


5.4.5.- Submit Button

Para la creación de este componente, deberemos presionar en la paleta de la izquierda sobre el elemento *Submit Button*. Seguidamente se nos abrirá una ventana emergente de configuración general del componente como se muestra en la introducción de este apartado. Después se mostrará una pantalla de configuración propia del componente como la que adjuntamos a continuación.

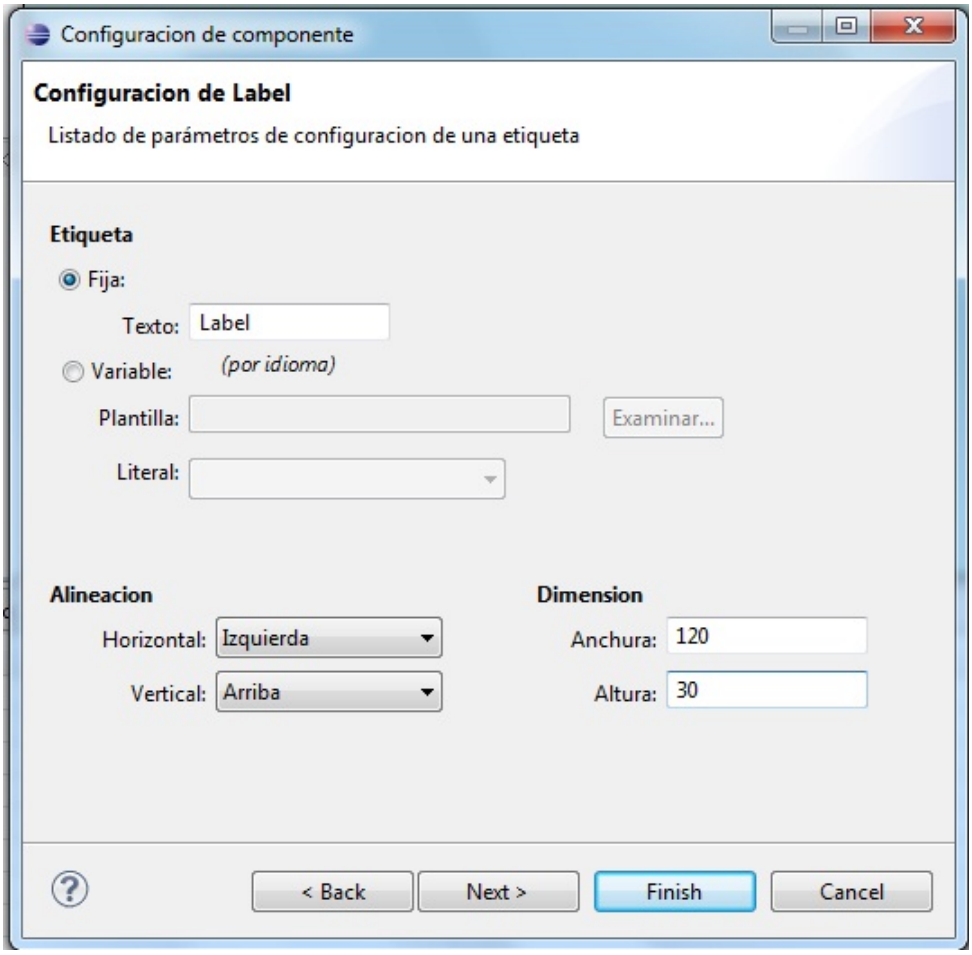


Finalmente, el aspecto del componente en el formulario será similar al que mostramos a continuación.

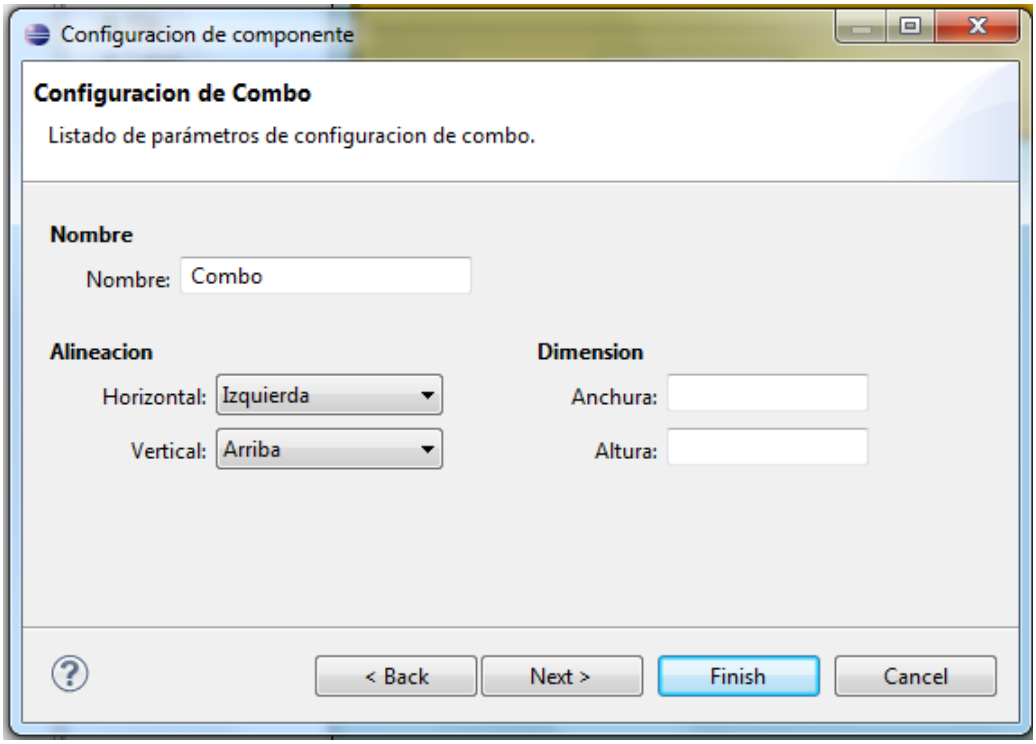


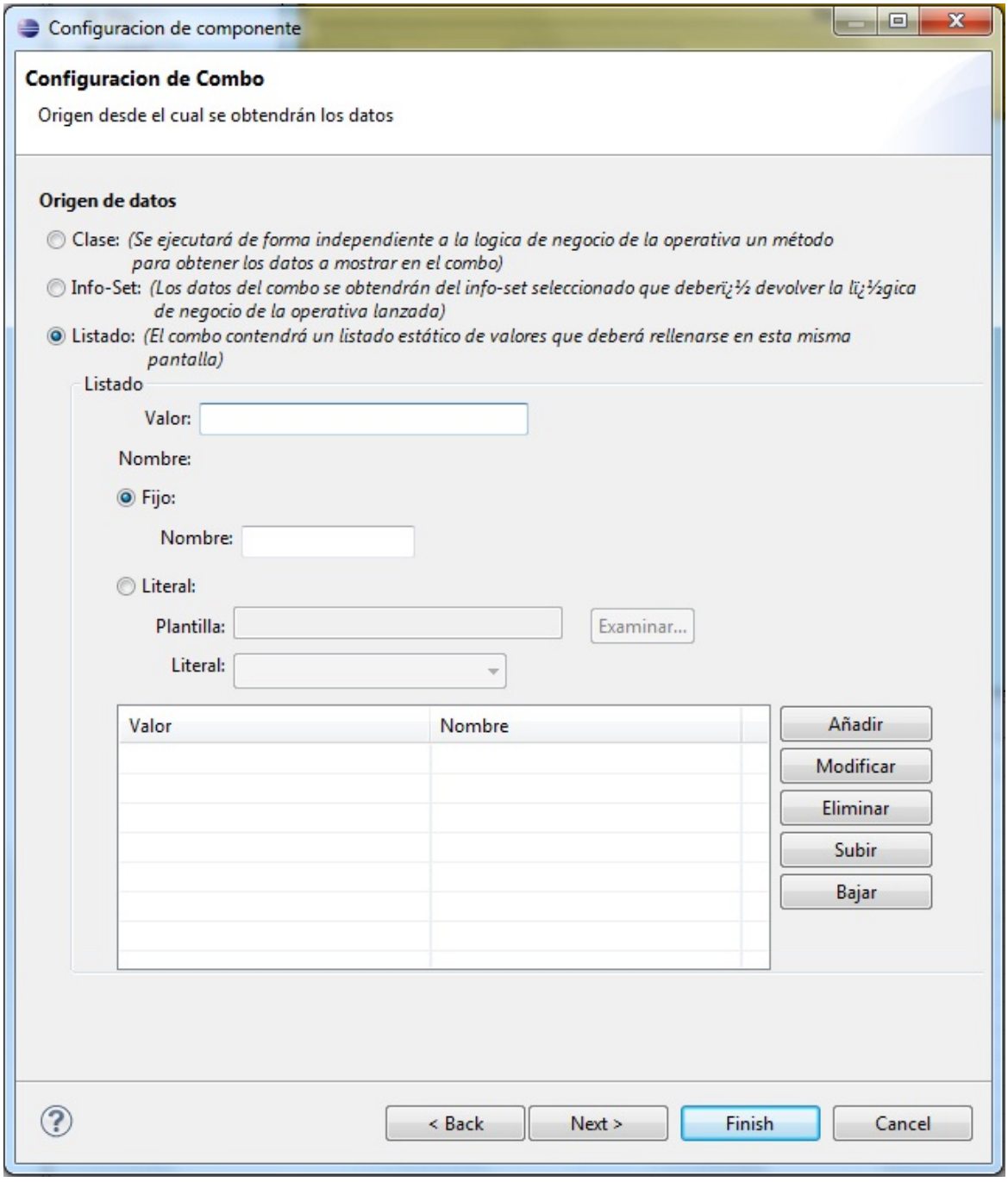
5.4.6.- Combo Box

Para la creación de este componente, deberemos presionar en la paleta de la izquierda sobre el elemento *ComboBox*. Seguidamente se nos abrirá una ventana emergente de configuración general del componente como se muestra en la introducción de este apartado. Después se mostrará una pantalla de configuración propia del componente como la que adjuntamos a continuación.

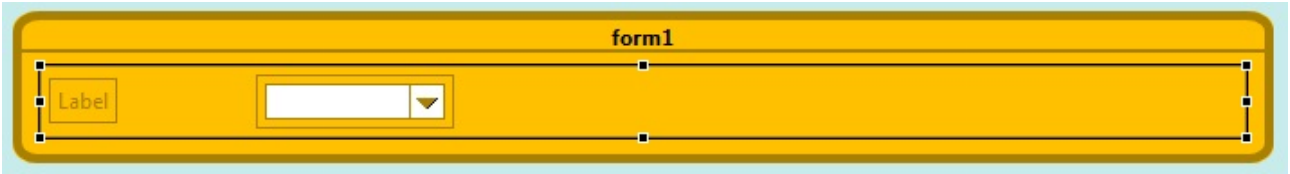


La pantalla siguiente nos permitirá añadir la lista de valores al combo desplegable.



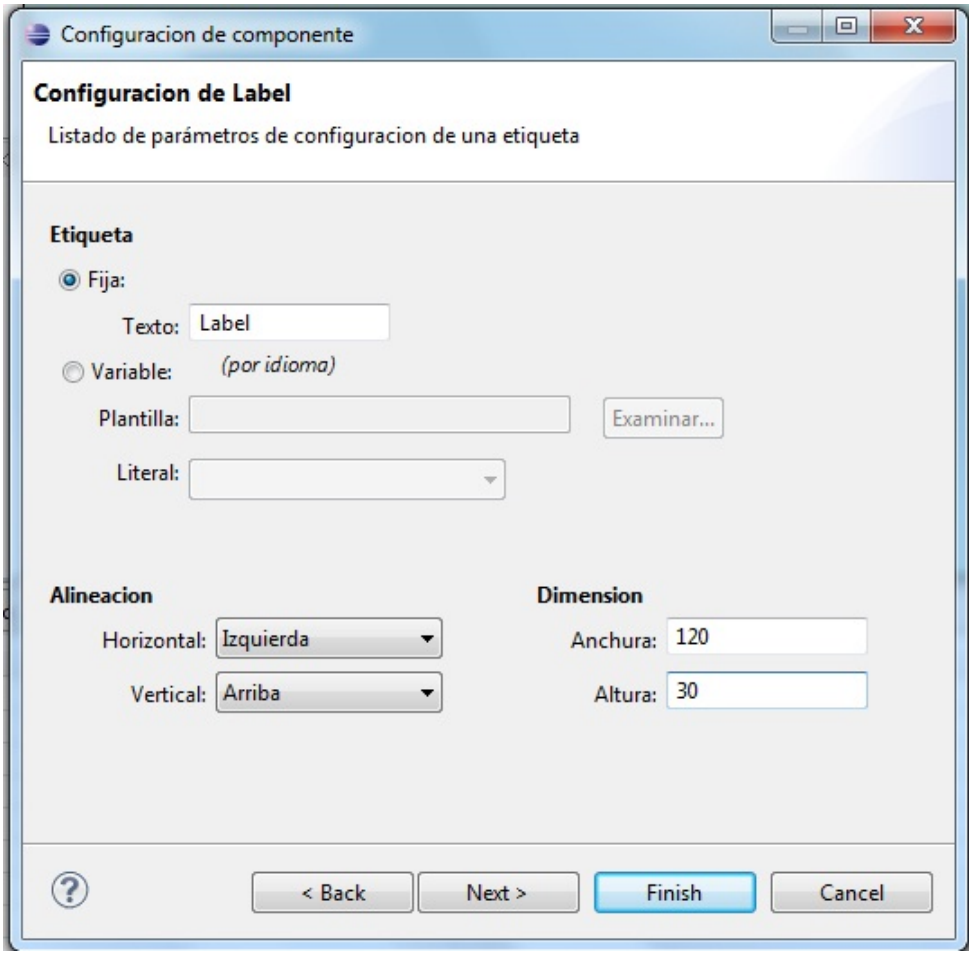


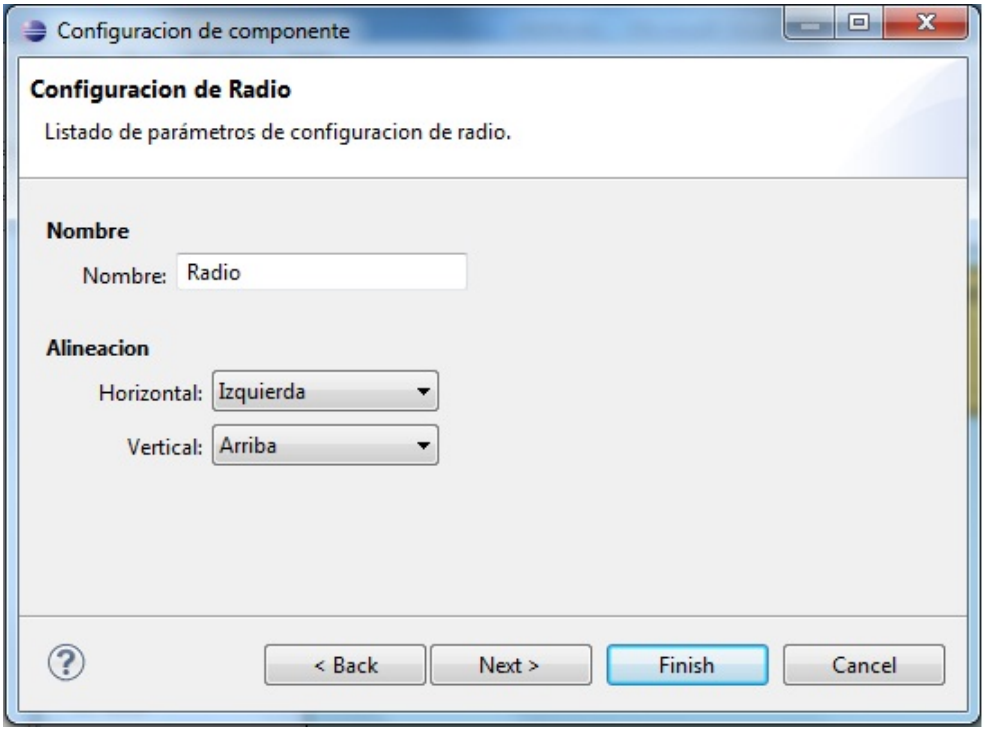
Finalmente, el aspecto del componente en el formulario será similar al que mostramos a continuación.



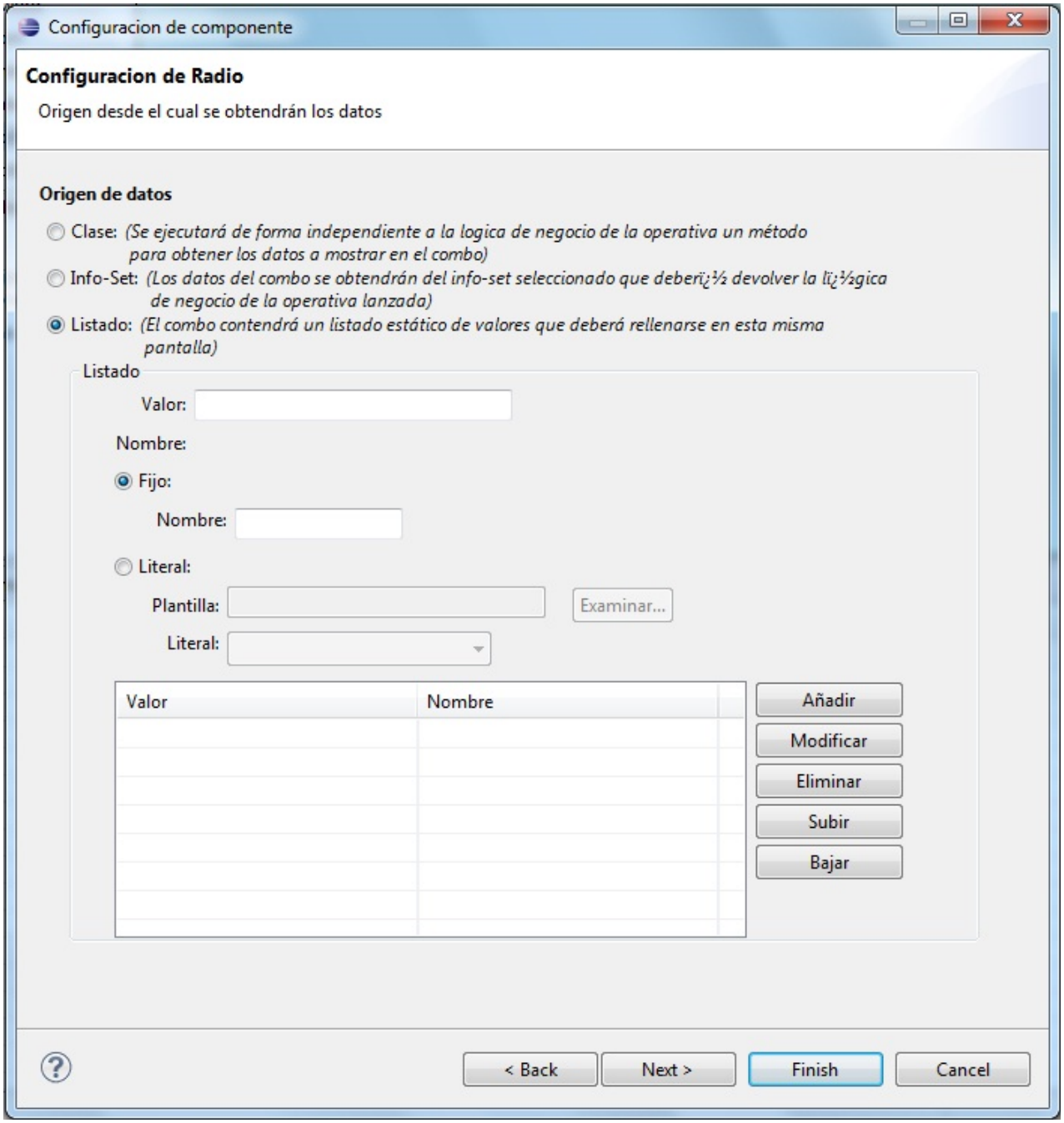
5.4.7.- Radio Button

Para la creación de este componente, deberemos presionar en la paleta de la izquierda sobre el elemento *Radio Button*. Seguidamente se nos abrirá una ventana emergente de configuración general del componente como se muestra en la introducción de este apartado. Después se mostrará una pantalla de configuración propia del componente como la que adjuntamos a continuación.

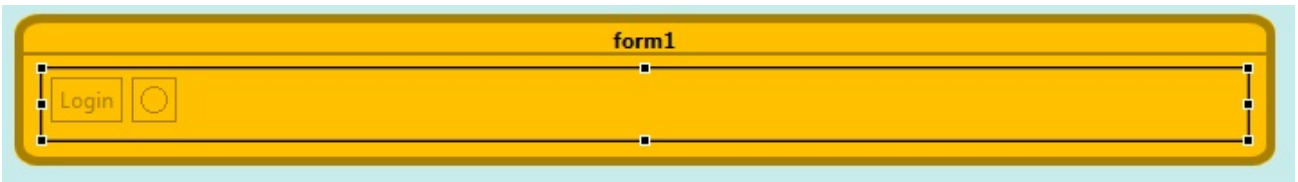




La pantalla siguiente nos permitirá añadir la lista de valores al radio button.

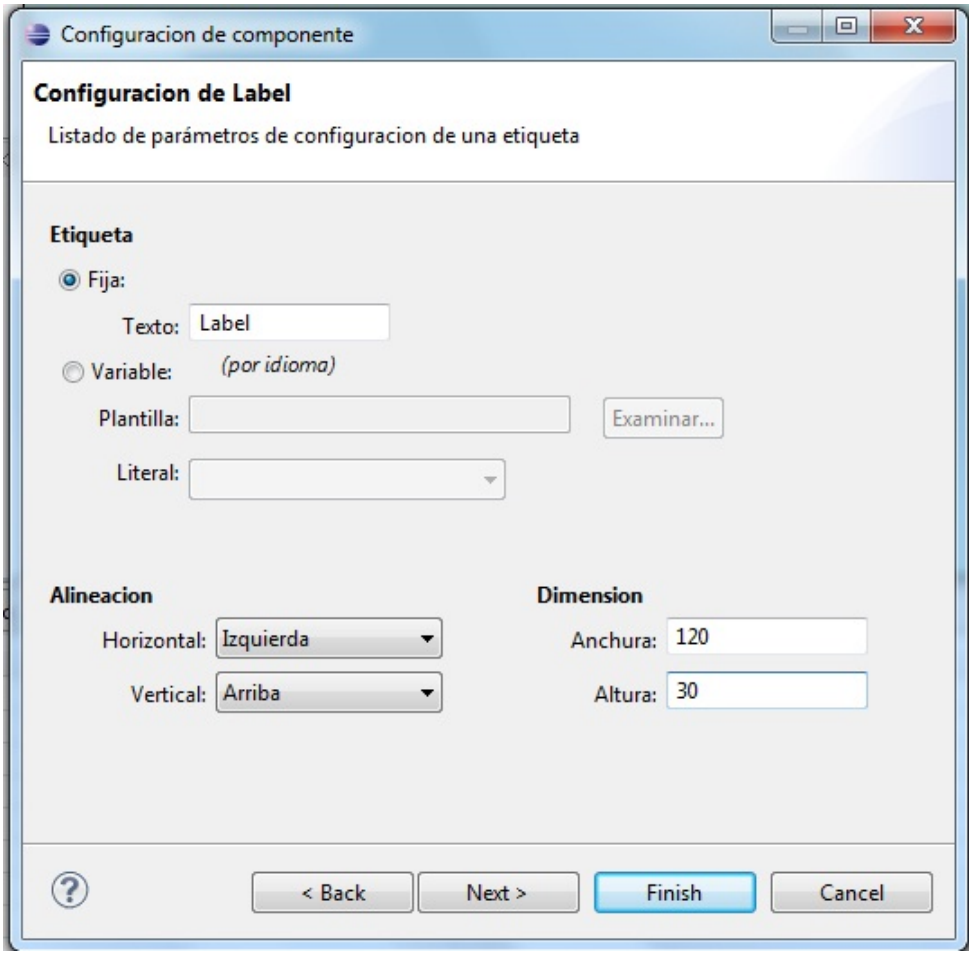


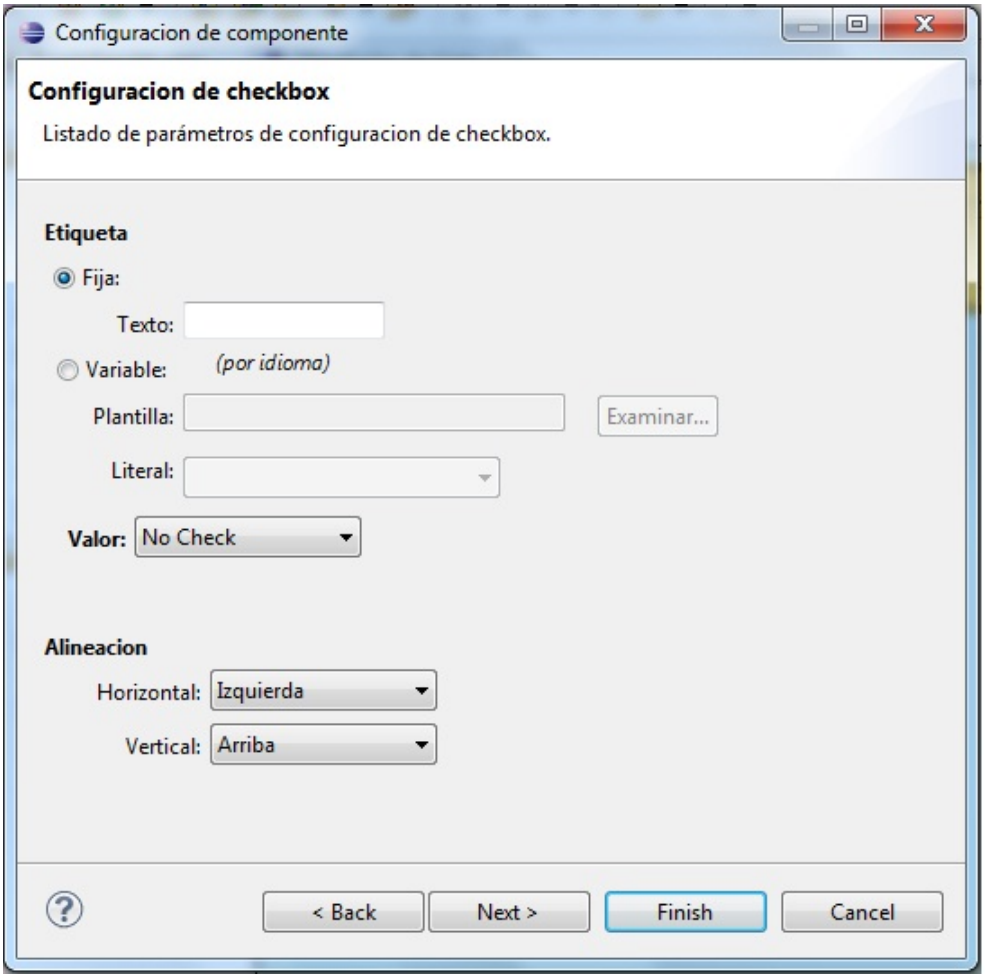
Finalmente, el aspecto del componente en el formulario será similar al que mostramos a continuación.



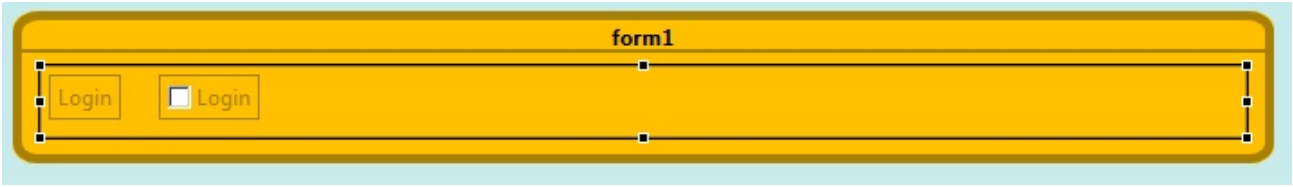
5.4.8.- Check Box

Para la creación de este componente, deberemos presionar en la paleta de la izquierda sobre el elemento *Check Box*. Seguidamente se nos abrirá una ventana emergente de configuración general del componente como se muestra en la introducción de este apartado. Después se mostrará una pantalla de configuración propia del componente como la que adjuntamos a continuación.



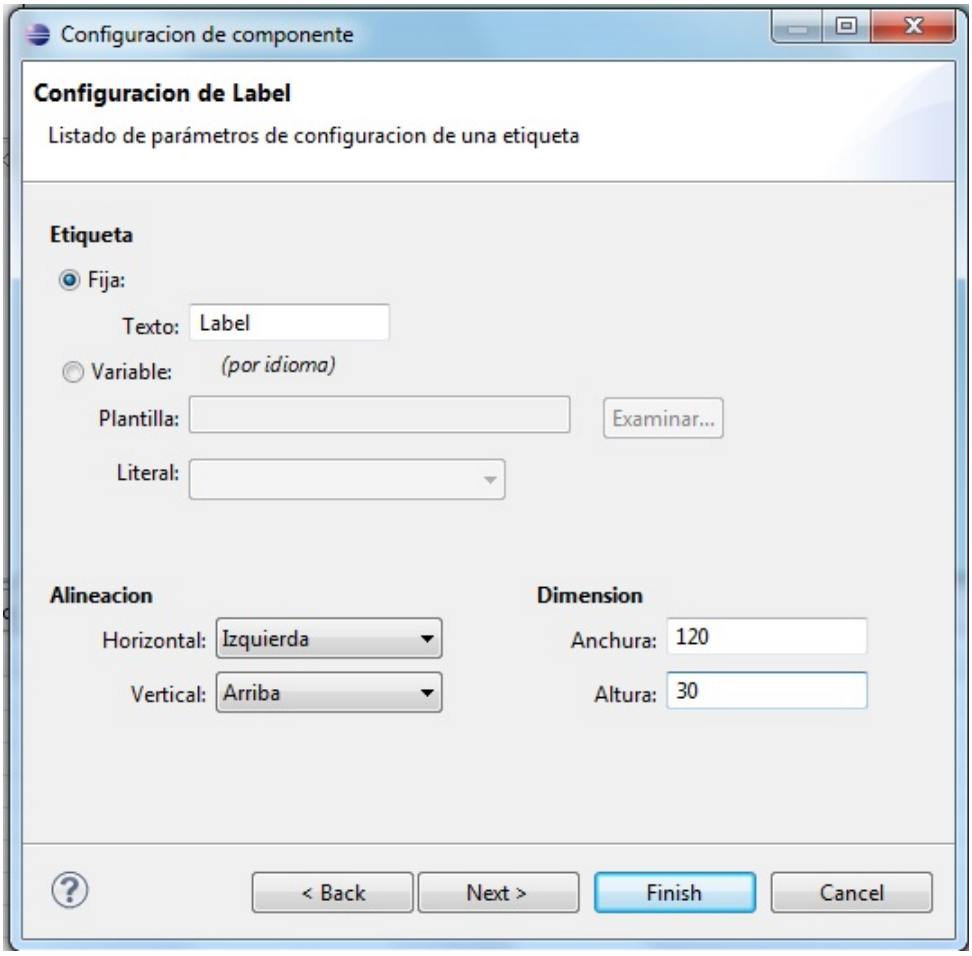


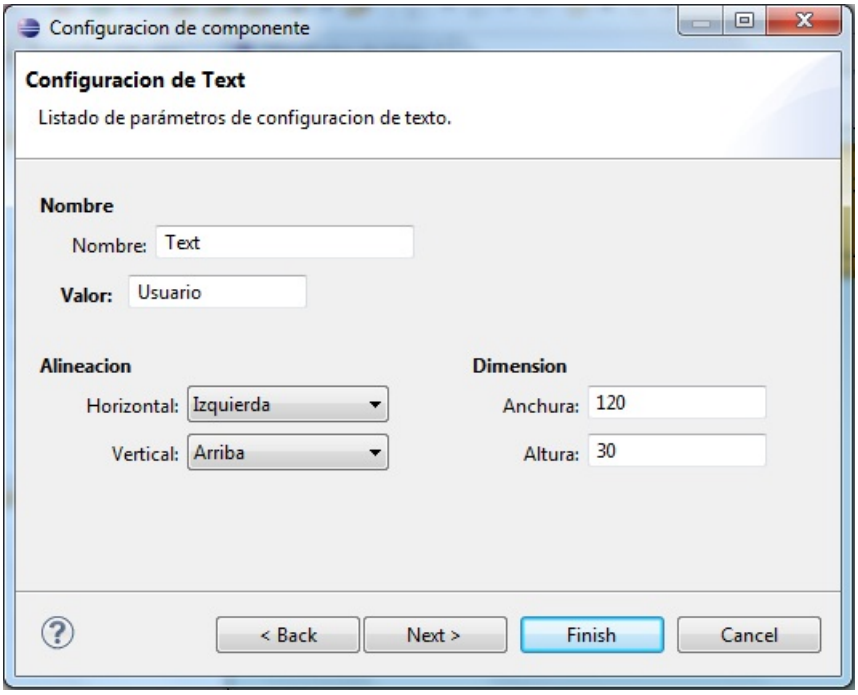
Finalmente, el aspecto del componente en el formulario será similar al que mostramos a continuación.

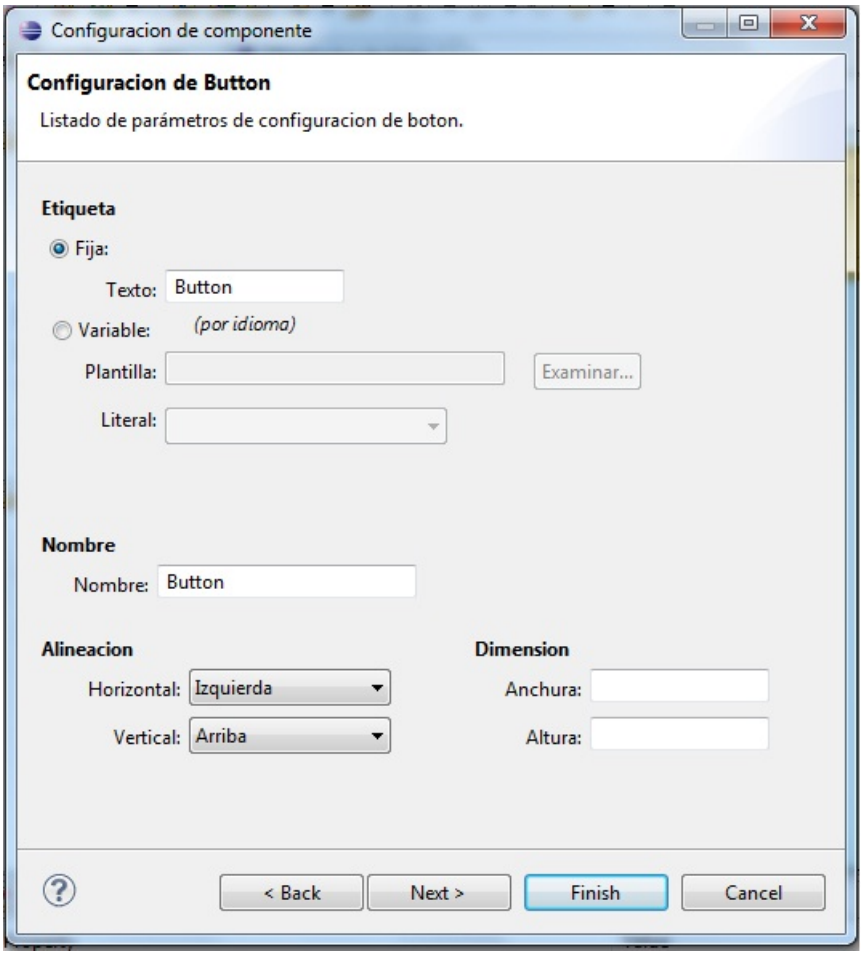


5.4.9.- File Component

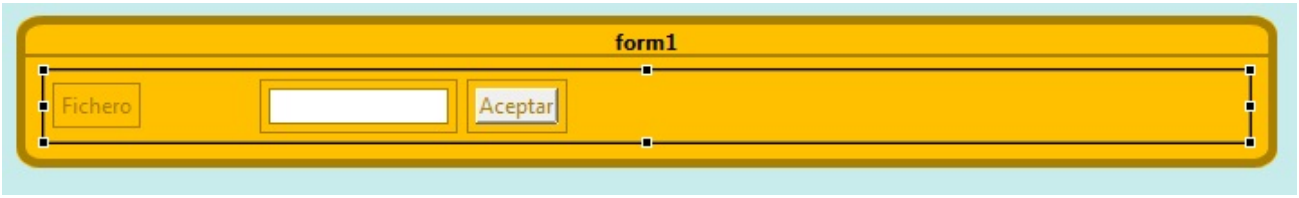
Para la creación de este componente, deberemos presionar en la paleta de la izquierda sobre el elemento *File Component*. Seguidamente se nos abrirá una ventana emergente de configuración general del componente como se muestra en la introducción de este apartado. Después se mostrará una pantalla de configuración propia del componente como la que adjuntamos a continuación.




Configuracion de componente
Configuracion de Text
Listado de parámetros de configuracion de texto.
Nombre
Nombre: Text
Valor: Usuario
Alineacion
Horizontal: Izquierda
Vertical: Arriba
Dimension
Anchura: 120
Altura: 30
< Back
Next >
Finish
Cancel


Configuracion de componente
Configuracion de Button
Listado de parámetros de configuracion de boton.
Etiqueta
Fija:
Texto: Button
Variable: (por idioma)
Plantilla:
Examinar...
Literal:
Nombre
Nombre: Button
Alineacion
Horizontal: Izquierda
Vertical: Arriba
Dimension
Anchura:
Altura:
< Back
Next >
Finish
Cancel

Finalmente, el aspecto del componente en el formulario será similar al que mostramos a continuación.



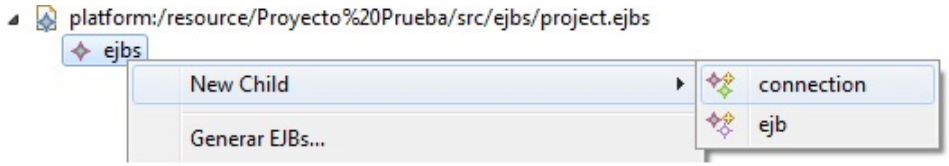
6.- Ejbs

Una vez creado un fichero de tipo *ejbs*, tendremos que poder configurarlo para su ejecución sobre la plataforma MCServer.

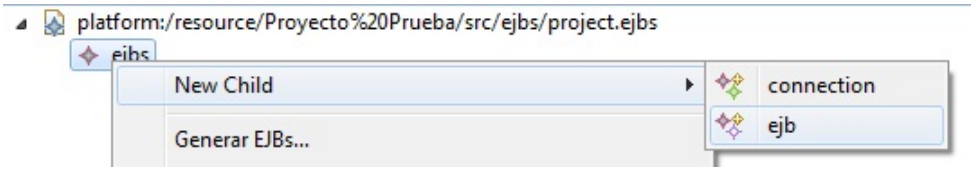
6.1.- Creación de una conexión

Deberemos abrir el fichero en Eclipse, haciendo doble clic sobre él.

Seguidamente, hacer click con el botón derecho, *New Child* y seleccionar *connection*.

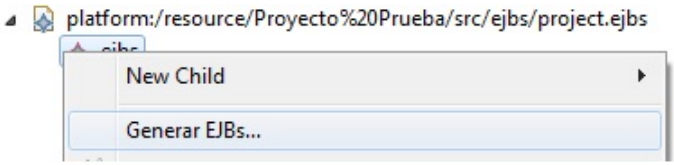


También deberemos de asignarle las propiedades necesarias para su correcto funcionamiento en MCServer.



6.2.- Creación de un EJB

Deberemos abrir el fichero en Eclipse, haciendo doble clic sobre él.
Seguidamente, hacer click con el botón derecho, *New Child* y seleccionar *ejb*.



También deberemos de asignarle las propiedades necesarias para su correcto funcionamiento en MCServer.

Property	Value
Configuración general	
Model Path	/mcserverfiles/ContractServlet.model
Name	ejb/ContractEJB
Transaction Time Out	180

6.3.- Generación de los ficheros JAR

Una vez generada una lista de ejbs con una conexión asociada, podremos generar el fichero JAR para su interconexión con la BBDD.

Property	Value
Configuración general	
Authorization	Container
Jndi Name	jdbc/CANBEPool
Resource Name	jdbc/CANBEPool
Type	javax.sql.DataSource

Se mostrará una pantalla como la siguiente, con la información de los EJBS a generar y una ruta donde se generarán, así como la versión del EJB (J2EE1.3 o J2EE5). Al presionar el botón *Finish*, se generarán los ficheros seleccionados.