Portal del Empleado

Marco Antonio Rodríguez Madureira
Grado de Ingeniería Informática
Java EE

Albert Grau Perisé
Santi Caballe Llobet

14 de junio de 2017

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Portal del Empleado*
Nombre del autor:	*Marco Antonio Rodríguez Madureira*
Nombre del consultor/a:	*Albert Grau Perisé*
Nombre del PRA:	*Santi Caballe Llobet*
Fecha de entrega (mm/aaaa):	06/2017
Titulación::	*Grado de Ingeniería Informática*
Área del Trabajo Final:	*Java EE*
Idioma del trabajo:	*Español*
Palabras clave	*TFG, JavaEE, JMS*

Resumen del Trabajo (máximo 250 palabras):

El siguiente proyecto describe el desarrollo de una aplicación a medida para mejorar la disponibilidad del departamento de TI, automatizando algunas tareas que, en general, interrumpen su trabajo diario (como son el alta/baja de cuentas de usuarios, desbloqueo o cambio de contraseñas y la formación básica de los usuarios).

Para lograr este objetivo se crea una aplicación que, mediante agentes, automatizará la gestión de cuentas de usuarios en las otras aplicaciones existentes y, vía un portal web, facilitará documentos para la autoformación de los usuarios en los aspectos mas básicos.

Se aprecia la necesidad de una aplicación como esta en un contexto de crisis económica en la que es imprescindible optimizar el rendimiento de todos los recursos humanos.

La metodología utilizada será de tipo ágil, en concreto una adaptación de SCRUM y como plataforma tecnológica se usará JavaEE (especialmente EJBs, JPA, JMS, JAX-WS y JSF).

Los resultados obtenidos establecen los cimientos de la plataforma sobre la que seguir añadiendo las funcionalidades que formarán la primera fase del proyecto y a posterior aportar nuevas ideas.

Como conclusión, general, se considera que el proyecto ha sido acertado y logrará conseguir mejorar la disponibilidad del departamento de TI y en siguientes fases, ayudar a algún otro departamento (como el de RRHH).

Abstract (in English, 250 words or less):

The following project describes the development of a custom application to improve the availability of the IT department, automating some tasks that, in general, interrupt their daily work (such as creating / deleting user accounts, unblocking or changing passwords, and basic users training).

To achieve this goal, an application is created that, through agents, will automate the management of user accounts in other existing applications and, via a web portal, will provide documents for users' self-training in the most basic aspects.

We appreciate the need for an application like this in a context of economic crisis in which it is imperative to optimize the performance of all human resources.

The methodology used will be of an agile type, specifically an adaptation of SCRUM and as technology platform will be used JavaEE (especially EJBs, JPA, JMS, JAX-WS and JSF).

The results obtained establish the foundation of the platform on which to continue adding the functionalities that will form the first phase of the project and to later contribute new ideas.

As a general conclusion, it is considered that the project has been successful and will manage to improve the availability of the IT department, and in subsequent phases can help some other department (such as HR).

Índice

Lista de figuras

1. Introducción

1.1 Contexto y justificación del Trabajo

Exposición del proyecto

La crisis económica ha forzado un ajuste en las plantillas de las empresas. Este ajuste, en algunos casos, ha dejado al descubierto múltiples carencias de organización y control, antes imperceptibles. Así mismo, la tarea de analizar la situación y establecer protocolos que ayuden a optimizar el tiempo se convierte en una odisea, ya que los recursos humanos restantes se encuentran saturados. ¿ Por donde rompemos el circulo vicioso ?

Si se contase con recursos económicos, lo mas habitual sería contratar una consultora externa pero, precisamente en un crisis, no es posible disponer de recursos económicos. Así que una alternativa puede consistir en empezar por mejorar el departamento de tecnologías de la información (TI en adelante), para que luego éste pueda ayudar en la mejora de otros departamentos, puesto que un buen uso de las tecnologías suele ser de gran ayuda.

En el caso que nos ocupa, se tratará entonces de mejorar la interacción entre el departamento de recursos humanos (RRHH en adelante) y el de TI de una gran empresa. En concreto nos centraremos en el proceso de alta de un nuevo trabajador, desde el punto de vista del departamento de TI .

Situación actual

En un principio, el responsable de RRHH debería avisar con tiempo al responsable de TI de la incorporación, o baja, de un nuevo trabajador. Pero en la realidad, la mayoría de veces, es el responsable del nuevo trabajador, o el propio trabajador, quien avisa al departamento de TI en el mismo momento que ya se encuentra la nueva persona en su puesto de trabajo, esperando poder empezar su primera jornada laboral.

Este hecho, genera una situación de estrés en el departamento de TI, que además de ineficiencia en su trabajo, puede llegar a generar errores.

Etapas del proceso de alta del empleado

En el dibujo, de la página 3, encontramos un diagrama de flujo del proceso de alta de un empleado.

A continuación detallamos los puntos mas importantes:

- Comprobar si en el puesto de trabajo hay ordenador. Si no lo hay habrá que reutilizar alguno de los que se hubiesen retirado anteriormente o pedir la compra de uno nuevo si no hubiera

ninguno en stock. Si se reutiliza uno, se tendrá que realizar, si no se hizo antes, el procedimiento de limpieza[1].

- Una vez comprobado que hay ordenador, se tendrán que instalar las aplicaciones necesarias para ese puesto de trabajo y eliminar las que no se tengan que usar si por seguridad o por cuestiones de licencias es necesario.
- Dar de alta la nueva persona en las distintas aplicaciones y/o sistemas a los que tenga que tener acceso, así como realizar las configuraciones adecuadas.
- Explicar al usuario como acceder a las aplicaciones, que en el punto anterior se le ha dado acceso.

También se pueden observar marcadas con fondo distinto al azul, las tareas donde suelen haber mas problemas, o que se consideran más críticas por la dedicación de tiempo necesario para finalizarlas y que serán las que se intentarán resolver o mejorar con este proyecto (en rojo las mas importantes, naranjas las intermedias y amarillo las menos relevantes).

1 Por limpieza se entiende, principalmente, a la eliminación de todos los datos que pueda contener el ordenador, previa copia si fuese necesario. También en algunos casos puede incluir una limpieza física.

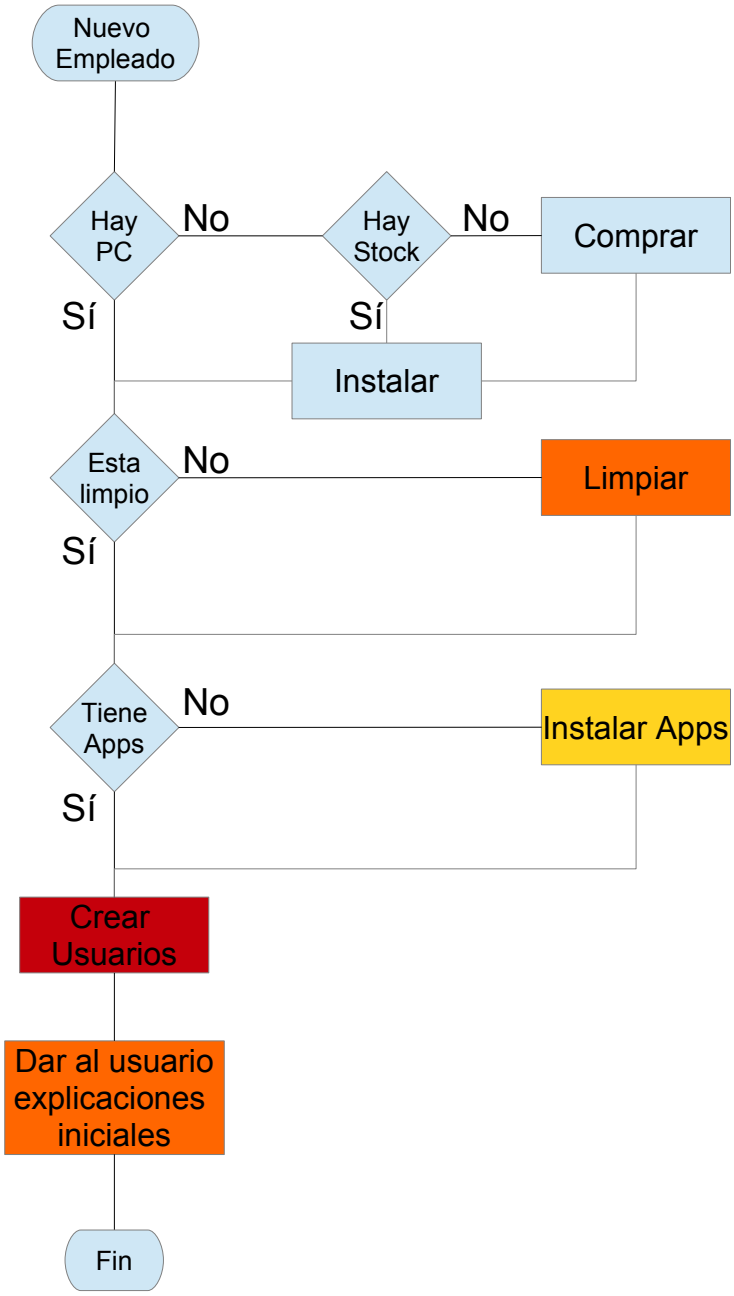


Ilustración 1: Diagrama de flujo: alta empleado

Posibles errores

El tener que acometer los pasos explicados en el punto anterior con prisas genera, en muchas ocasiones, los siguientes errores directos:

- No realizar el procedimiento de limpieza.
- No dar de alta al usuario en todos los aplicativos necesarios, lo que volverá a causar interrupciones cuando se descubra la falta que, como es normal, será cuando se requiera el uso de dicha aplicación y nuevamente habrá que actuar con prisas.
- No configurar correctamente alguna de las aplicaciones. Otra vez se producirán mas interrupciones y/u otros fallos relacionados con la aplicación mal configurada.

- La explicación a todo correr que se le da al nuevo usuario, por lo general no es asimilada por el mismo, obligando al usuario a recurrir al departamento de TI para que se lo vuelvan a explicar y en ocasiones incluso que le restablezcan las credenciales por no recordarlas.

Además de esos problemas directos del proceso, se generan otros que incluso pueden afectar a terceros, como son:

- Al tener que interrumpir el personal de TI lo que estuviera haciendo, esa tarea se verá retrasada, incluso en algunos casos mal terminada.
- Si el puesto no tenía ordenador y no hay ningún ordenador en stock, se puede llegar a utilizar un ordenador que se iba a retirar por bajas prestaciones, lo que volverá a generar más trabajo cuando se tenga que cambiar por el definitivo. En otras ocasiones, se utiliza un ordenador que iba para otro departamento, o que era de otro departamento pero en ese momento no se utilizaba, generando todavía más trabajo, porque cuando llegue el definitivo se tendrán que mover e instalar dos equipos.

En general, podemos ver que un factor común de todas estas situaciones, es que terminará generando más trabajo al departamento de TI, aumentando su colapso y empeorando su rendimiento, eficacia y en general, la imagen y percepción de este departamento que puedan tener las otras áreas de la empresa.

1.2 Objetivos del Trabajo

La propuesta realizada para mejorar el proceso de alta tiene cuatro grandes líneas:

1. Evitar que el nuevo empleado sea quien notifique al departamento TI que ha llegado.
2. Crear unos flujos de trabajo para los técnicos del departamento TI donde podrán ver qué pasos tienen que dar y marcarlos según los vayan haciendo. Así no se olvidarán de ninguno y facilitará el trabajo en equipo, aumentando el rendimiento.
3. Automatizar todas las tareas que sea posible.
4. Facilitar al usuario la auto resolución de cuestiones en las que no es imprescindible el soporte de un técnico (como puede ser qué usuario y contraseña tenía para una cierta aplicación, cómo acceder a la misma, etc).

Otras consideraciones

Aunque el objetivo principal del proyecto es ayudar al departamento de TI, se tendrán en cuenta otros departamentos que puedan participar de alguna manera en el proceso de alta de un empleado para, en futuras versiones, mejorar si es posible su participación.

Por ejemplo, se podrían añadir funcionalidades útiles para los departamentos de recursos humanos, riesgos laborales, jurídico, etc. Incluso opciones prácticas para el propio empleado.

Como el objetivo final del proyecto será servir al trabajador y todos los departamentos que puedan tener una relación, enfocada en el trabajador, se decide bautizar la aplicación como **Portal del empleado**.

1.3 Enfoque y método seguido

Para lograr los objetivos planteados, se propone la implementación de una plataforma sobre JavaEE que contemplará:

- Una aplicación web como interface de usuario.
- EJB como lógica de negocio.
- Web Services y JMS como mecanismo de comunicación con el resto de aplicaciones implementadas en la empresa.
- Agentes encargados de comunicar las aplicaciones existentes con el portal usando los servicios web y las colas. Estos agentes puede ser en formato módulos, en el caso de aplicaciones desarrolladas internamente y aplicaciones de consola o servicios/*daemons* para aplicaciones de terceros.

Como metodología de trabajo, para el desarrollo del software, he optado por una de tipo ágil, porque considero que, estas metodologías, permiten reaccionar muy rápidamente a cambios en los requisitos así como obtener productos utilizables desde las primeras etapas (aunque sea con funcionalidades reducidas, claro).

Como la herramienta que he elegido para el control del proyecto, *OpenProject*, tiene soporte para *SCRUM*, usé una aproximación a esta metodología ágil, aunque no de forma estricta. La adaptación que hago de *SCRUM* constaría de los siguiente pasos:

1. Análisis inicial: reunión con el cliente para recoger los requisitos del proyecto y plantear algunas líneas generales de la solución, así como futuras líneas de evolución. De esta reunión se obtendrá una primera aproximación a la Pila del Producto.
2. Ciclos iterativos, de duración variable entre 1 y 3 semanas:
    1. *Meeting*: reunión con el cliente para ordenar por prioridad las funcionalidades/requisitos que se van a implementar en el ciclo actual. Pueden ser funcionalidades nuevas o provenir de cliclos anteriores porque no dio tiempo a terminarlas o necesitan alguna mejora. De este paso sale la Pila del *Sprint*. Estas reuniones tienen una duración corta, de 1 hora aproximadamente.
    2. *Sprint*: etapa de desarrollo en la que se intentará implementar las funcionalidades recogidas en la pila del *sprint*. Dura de 1 a 3 semanas.

3. Test: realizar pruebas de funcionamiento e integración de las funcionalidades desarrolladas. Se harán en 1 o 2 días como máximo.
4. Paso a producción: las funcionalidades que hayan pasado los test serán puestas en producción, el resto pasarán nuevamente a la pila del producto. Lo normal será que este paso dure menos de un día.

El paradigma de programación será el nativo de Java, es decir, Programación Orientada a Objectos (POO).

Aunque Java soporta algún otro paradigma, como puede ser la Programación Orientada a Aspectos (POA), creo que la primera es idónea para este proyecto, además de ser en la que más experiencia tengo, punto muy importante para que los tiempos de desarrollo no se dilaten en exceso.

Como herramienta de desarrollo, *IDE*, usaré Netbeans y los proyectos estarán gestionados con Maven con un repositorio interno montado con Archiva. Como control de versiones utilizo el servidor de CVS que hay en la empresa, y cuando el proyecto empiece a tener cuerpo, lo agregaré al sistema de integración continua que tenemos sobre Jenkins.

1.4 Planificación del Trabajo

En la siguiente imagen se puede observar la planificación inicial del proyecto.

Hay que tener en cuenta que al elegir una metodología ágil, nunca se realizan planificaciones a tanto tiempo, ya que los módulos en los que se trabajará se van decidiendo al inicio de cada Sprint.

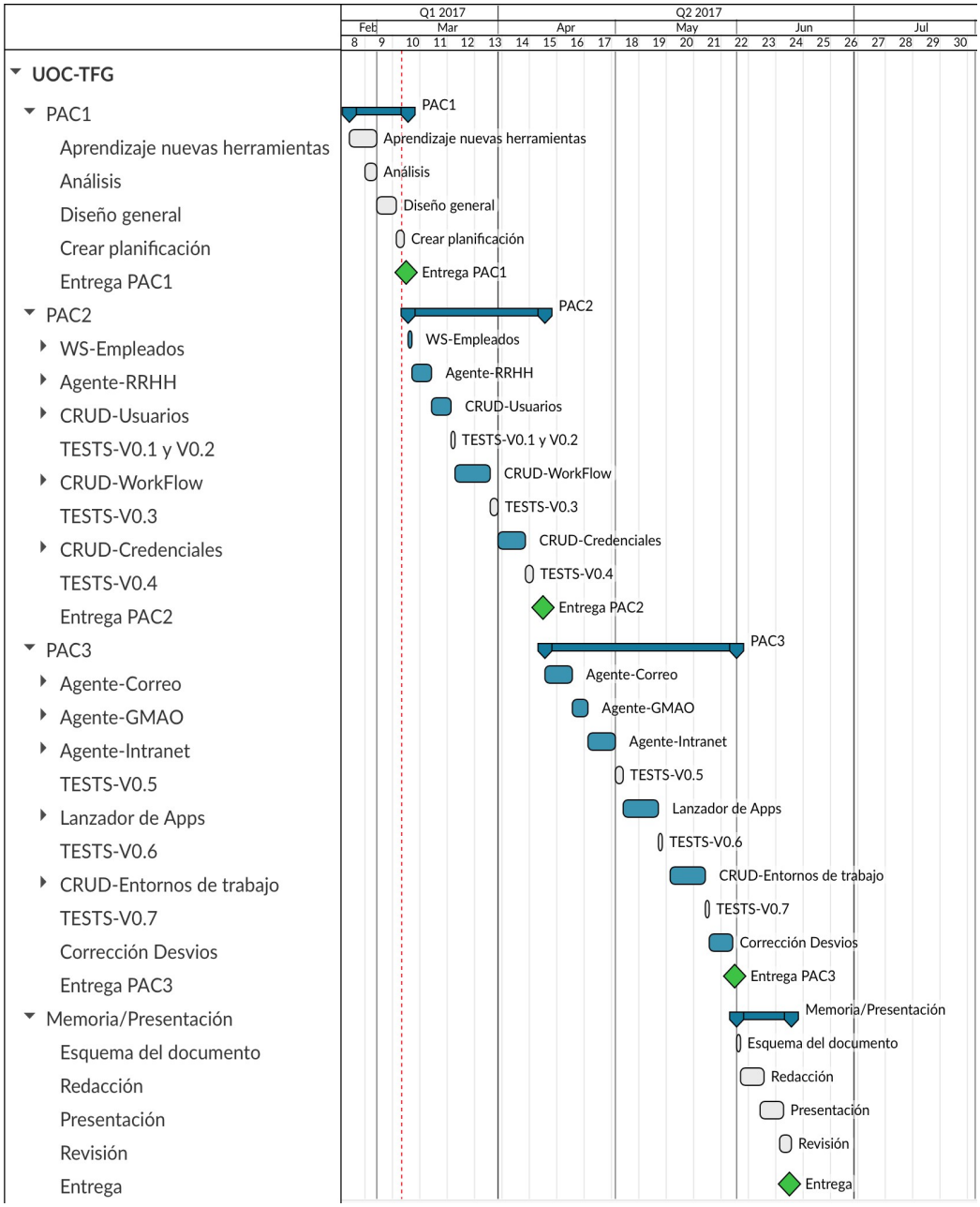


Ilustración 2: Planificación inicial

1.5 Breve sumario de productos obtenidos

Denario Gateway: servicio para windows encargado de notificar las altas y bajas de los trabajadores realizadas en el programa de nóminas.

Workflow Manager: módulo del portal que ejecutará el *workflow* definido para la acción de alta de un empleado.

Portal Web: interface de usuario, en formato web, para que tanto técnicos como usuarios accedan a las funcionalidades prestadas.

1.6 Breve descripción de los otros capítulos de la memoria

En los siguientes capítulos se expondrán, por un lado, la organización del proyecto en *sprints* y por otro el análisis y diseño de las partes mas importantes de la aplicación.

Al utilizar un metodología ágil, las fases de análisis y diseños se van realizando en cada sprint, en muchas ocasiones refinando, incluso modificando los resultados de pasos anteriores. Pero para la memoria, considero que queda mas claro plasmar el diseño final en vez de las distintas versiones/refinamientos que haya ido sufriendo el proyecto.

2. Análisis de requisitos inicial

De la primera reunión mantenida con el jefe de sistemas de la información, se obtienen los siguiente requisitos:

2.1 General

- Obtener la lista de empleados activos desde la aplicación de recursos humanos (pero garantizando que no se vea afectado el funcionamiento del programa de RRHH, ni que el portal del empleado pueda ser utilizado como pasarela para acceder a datos del programa).
- Hay que contemplar la posibilidad de tener usuarios externos, que no están contratados directamente, pero que sí tienen que acceder a aplicaciones de la empresa.
- La aplicación tiene que poder ser utilizada desde cualquier ordenador con conexión a Internet.

2.2 Flujo de trabajo

Cuando se da de alta/baja un usuario del portal se tiene que crear y asignarle un flujo de trabajo, de manera automática, que indicará a los técnicos qué tareas tienen que realizar. También se puede asignar un flujo de trabajo de forma manual.

Los flujos de trabajo tienen las siguientes características:

- Un usuario solo puede tener un flujo de trabajo activo.
- Podrá estar basado en plantilla/s o creado específicamente.
- No será recurrente, el flujo tendrá un único punto de inicio y fin.
- Estará compuesto por una o más tareas.
- Las tareas pueden ejecutarse en paralelo o secuencialmente.
- Las tareas pueden ser de distinto tipo, por ejemplo:
  - Alta de usuario en aplicativo X.
  - Configuración X a realizar en el ordenar del usuario.
  - Asignación de recursos físicos (ordenador, pantalla, etc).
- Aunque haya tareas que tengan un módulo que permita su realización automática, se tendrá que poder desactivar su ejecución automática (ya sea en la definición de la plantilla, o específicamente para un flujo de trabajo).

2.3 Módulos del portal

Un módulo del portal sería como una pequeña aplicación.

Los módulos pueden consultar el perfil que tiene el usuario, para decir qué funciones están disponibles para ese usuario.

Los primero módulos serán:

- Gestor de usuario: tipo CRUD (del Inglés Create, Read, Update and Delete).
- Gestor de flujos de trabajo: tipo CRUD.
- Gestor de entornos de trabajo: tipo CRUD.
- Gestor de módulos del portal: permite hacer las parametrizaciones disponibles en los distintos módulos, además de activar/desactivar un módulo.
- Gestor de credenciales: este módulo guarda las credenciales utilizadas para la creación de las distintas cuentas. Las contraseñas sólo tienen que ser visibles por el usuario al que pertenezcan. Tendrá un generador de contraseñas. Podrá utilizar agentes para la creación automática de las cuentas en los aplicativos y sistemas de la empresa. Algunos agentes podrían ser:
  - Correo electrónico.
  - GMAO.
  - Intranet.
  - Karat.
  - Microsoft Windows (dominio).
  - Microsoft Windows (cuenta local).
  - NovaHis.
- Lanzador de aplicaciones: este módulo contiene enlaces web, accesos directos o instrucciones que le permitan al usuario ejecutar una aplicación.

2.4 Entornos de trabajo

Un entorno de trabajo define un conjunto de módulos del portal a los que tiene acceso un usuario, así como configuraciones de los módulos para adaptarlos.

A un usuario se le pueden asignar uno o más entornos de trabajo, que serán aplicados en el orden indicado.

También se le podrán asignar módulos concretos, o configuraciones sobre un módulo aunque provenga de un entorno asignado.

La herencia de estas configuraciones será jerárquica, prevaleciendo la última aplicada.

2.5 Perfiles de usuario

La aplicación contemplará 3 perfiles de usuario:

- Administrador
- Técnico
- Usuario

2.6 Derechos del perfil Usuario

- Acceder al portal.
- Ver el estado del flujo de trabajo.
- Utilizar los módulos que tenga asignado.

2.7 Derechos del perfil Técnico

- Crear usuarios.
- Bloquear usuarios (menos los que tengan perfil Administrador).
- Crear flujos de trabajo.
- Ejecutar flujos de trabajo.
- Crear entornos de trabajo.
- Asignar entornos de trabajo.

2.8 Derechos del perfil Administrador

Todos los del perfil usuario y técnico más:

- Bloquear usuarios (menos el usuario que se esté utilizando para acceder).
- Eliminar usuarios (menos el usuario que se esté utilizando para acceder).
- Asignar perfil a un usuario.
- Crear plantillas de flujos de trabajo.
- Convertir un flujo de trabajo en un plantilla.
- Cancelar un flujo de trabajo.
- Parametrizar los módulos de la aplicación.

3. Sprints

3.1 Sprint 1 (09-03-2017 al 15-03-2017)

Tras la reunión con el responsable de sistemas de la empresa, la planificación para el primer sprint es la siguiente:

- WS-Empleados: esta funcionalidad consiste en implementar un servicio web que permita actualizar los datos de los usuarios del portal.
- Agente-RRHH: esta aplicación, se encargará de sincronizar los empleados registrados en la aplicación de nóminas con los usuarios del portal del empleado.

En este primer sprint se encuentran los siguientes problemas que aumentaron la duración prevista:

- Localizar las tablas en las que el programa de recursos humanos (Denario) mantiene los datos de los trabajadores.
- Elegir el framework a utilizar para crear el servicio de windows con java.

3.2 Sprint 2 (27-03-2017 al 07-04-2017)

En la reunión para establecer las tareas para este sprint se decide securizar el acceso al servicio web. Por lo tanto el plan de trabajo queda así:

- WS-Empleados-Seguro: la idea será que la comunicación con el servicio web sea por HTTPS y que además se autentique al cliente.
- Gestor-Trabajadores: este módulo tiene que poder persistir los datos de los trabajadores recibidos por el servicio web, así como actualizarlos y recuperarlos.

Nuevamente surgieron varios problemas, principalmente dos:

- El JRE de Java denegaba las conexiones HTTPS del cliente hacia el servicio web, por problemas con el certificado digital.
- Fue imposible hacer que el cliente se autenticase contra el servicio web.

Además, surgen otras necesidades importantes para el departamento que impiden dedicarle mas horas a este proyecto, retrasando el inicio del siguiente sprint.

3.3 Sprint 3 (24-04-2017 al 05-05-2017)

Viendo que estaba atascado en la autenticación con el servicio web, en la reunión se consensúa aparcar esa funcionalidad para seguir avanzando. Se planifican las siguientes opciones:

- Simulador-RRHH: para poder realizar pruebas de funcionamiento del agente con el servicio web, se crea una pequeña aplicación web que permita realizar las típicas acciones CRUD (*Create*, *Read*, *Update* y *Delete*) sobre una tabla con la misma definición que la vista creada en el programa de nóminas.
- Parametrizar-Agente: se tiene que parametrizar el agente para poder hacer que se conecte a distintas bases de datos, sin tener que modificar el código.

Este es el primer sprint sin problemas y que se realiza según la planificación.

3.4 Sprint 4 (08-05-2017 al 19-05-2017)

Ahora que ya tenemos un agente que es capaz de notificar los cambios producidos en una tabla, con unas columnas determinadas, de una base de datos, cualquiera, a un servicio web, se decide planificar:

- Gestor-Usuarios: este módulo será el encargado de persistir la entidad usuario (los trabajadores necesitan un usuario para poder entrar en el portal).
- InterfaceWeb: primera versión del portal, básicamente tiene que contar con la pantalla inicial, la estructura de menús y poder visualizar el listado de usuarios y trabajadores dados de alta.

Fase también sin problemas, solo se podría destacar que otra vez hubo que dedicar horas a otros proyectos, pero en este caso ya se había contado con ello en la planificación.

3.5 Sprint 5 (22-05-2017 al 31-05-2017)

En la reunión mantenida se identifica una nueva necesidad que ha sido solicitada por la persona que se encarga de la LOPD. Cosiste en que la aplicación tiene que poder generar un informe donde se refleje:

- Fecha de alta de un usuario.
- Fecha de baja de un usuario.

Esta funcionalidad se añade a la lista de funcionalidades del proyecto, pero no se planifica para este sprint. La planificación contendrá uno de los módulos principales para esta primera fase:

- Gestor-FlujosDeTrabajo: este módulo tiene que poder crear un flujo de trabajo que responda a algún evento, en este caso al alta de un trabajador y ejecutar las tareas asociadas.
- InerfaceWeb: poder ver un listado de los flujos de trabajo así como una representación gráfica del mismo.

En este Sprint apareció un bug, que me preocupó bastante, y que consistía en que un mensaje enviado a una cola JMS era recibido, aleatoriamente, en otra cola. Pero hasta que me dí cuenta que ese era el fallo, el primer síntoma que noté era que se perdían mensajes.

3.6 Sprint 6 (01-06-2017 al 09-06-2017)

Como ya se tiene un portal web, que contiene datos personales reales cuando se configura el agente de sincronización contra la base de datos real, se decide implementar:

- InterfaceWeb: para poder acceder al portal hay que poner un usuario y contraseña.
- WS-Empleados: volver a intentar que su acceso sea autenticado.

Último sprint finalizado antes de tener que cerrar la memoria. Fue según lo planificado, y tuve la gran satisfacción de lograr que el agente de sincronización se autentique ante el servicio web, aunque para ello tuve que hacer un cambio de diseño, o como me gusta llamar a estas cosas, un *workaround*.

3.7 Sprint 7 (12-06-2017 al 23-06-2017)

Las funcionalidades programadas para esta etapa:

- CRUD-TrabajadoresExternos: añadir al portal las opciones CRUD solo para los trabajadores externos, importante recordar que los trabajadores que están en nómina se gestionan desde el programa de RRHH.
- CRUD-Usuarios: aunque en general los usuario del portal se crearán/borrarán desde los flujos de trabajo, hay usuarios que son independientes, que no tiene ningún trabajador asociado, como pueden ser los utilizados por los modules de sincronización.
- Gestor-Grupos: poder asignar un usuario a uno o mas grupos.

3.8 Calendario de ejecución real.

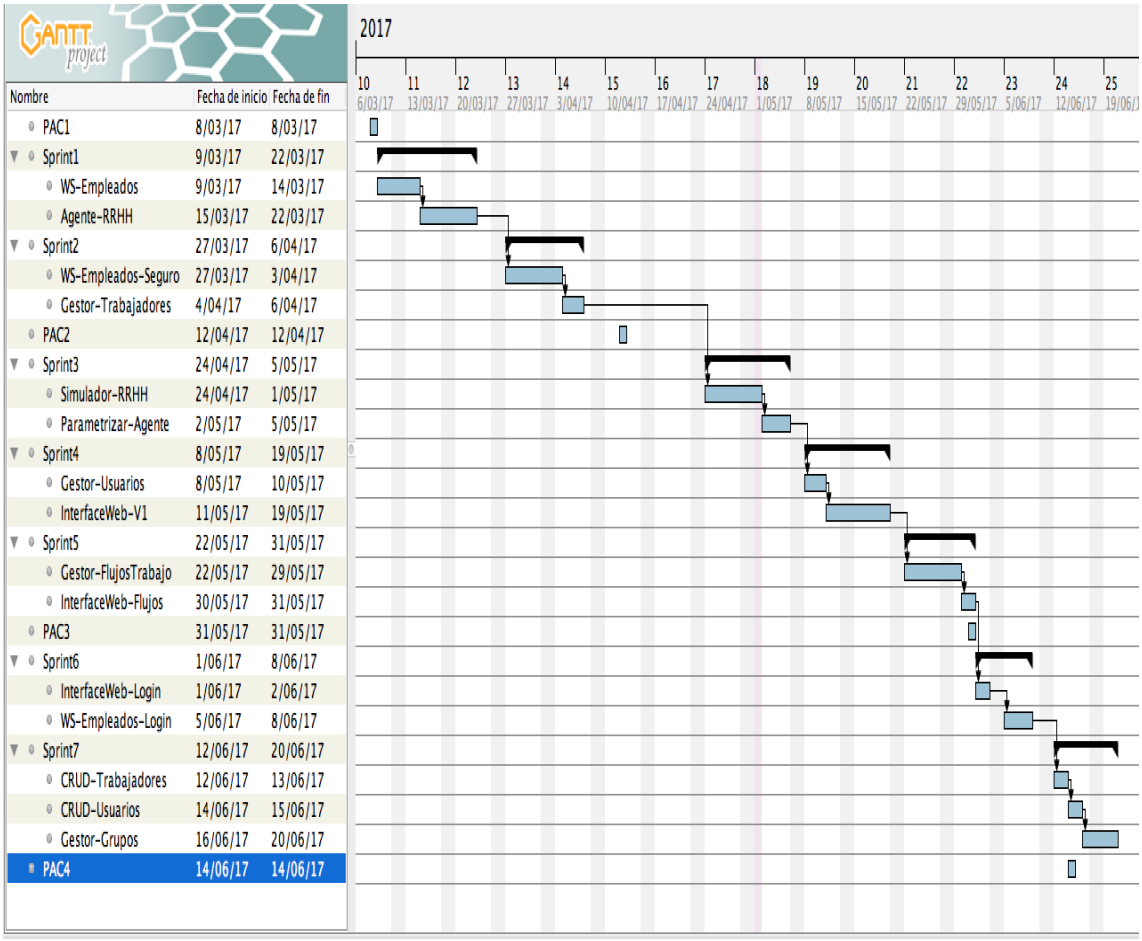


Ilustración 3: Calendario de ejecución real.

4. Diseño

4.1 General

Arquitectura multicapa

El proyecto se basará en una arquitectura típica de tres capas:

- Presentación: esta capa se implementará utilizando JSF y siguiendo un patrón Modelo/Vista/Controlador. Para acelerar el desarrollo y mejorar la parte estética, se usará Primefaces como biblioteca de componentes.
- Negocio: se basará en EJBs. Por norma general, habrá un EJB para cada grupo funcional, por ejemplo para todo lo relacionado con los usuarios existiría el EJB `UserManager`.
- Datos: El acceso a la capa de datos desde la capa de negocio será por medio de la API JPA. Además se utilizará el patrón fachada, para aislar las peculiaridades de JPA de los módulos de negocio. Así un cambio, de API de acceso a datos, no afectará a los módulos de negocio, y solo a los de fachada. Por ejemplo, el EJB `UserManager`, realizará todas las acciones de persistencia utilizando el EJB `UserFacade`.

En el siguiente dibujo podemos ver un esquema de la arquitectura planteada y sus principales componentes:

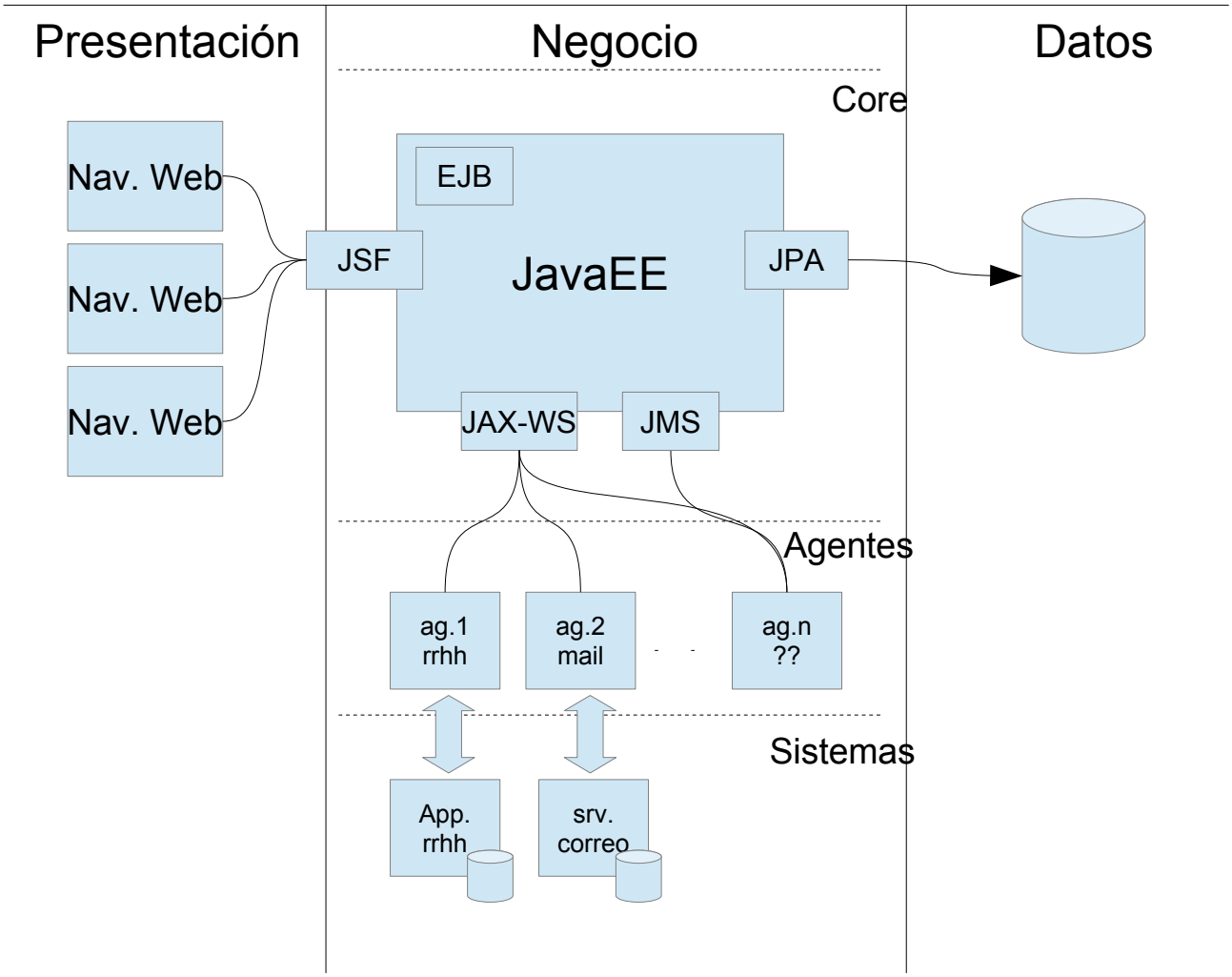


Ilustración 4: Esquema de la arquitectura

Seguridad

Se utilizará el framework de seguridad integrado en Java EE para la autenticación y autorización de los usuarios, y garantizar así que ningún usuario accede a recursos sobre los que no tiene derecho.

La comunicación entre módulos, cuando esta requiera utilizar la red, se hará mediante el protocolo HTTPS.

Infraestructura

Como gestor de base de datos se utilizará MySQL y como servidor de aplicaciones Glassfish.

4.2 Servicio Web

Descripción

Este servicio web tiene que permitir al Agente-RRHH poder dar de alta los nuevos empleados en el portal y dar de baja a los que ya no trabajen, al igual que actualizar los datos si hubiera alguna modificación.

Requisitos

Utilizar una comunicación segura y autenticada.

Modelo de datos

El siguiente diagrama de clases muestra las entidades con las que trabajará este módulo.

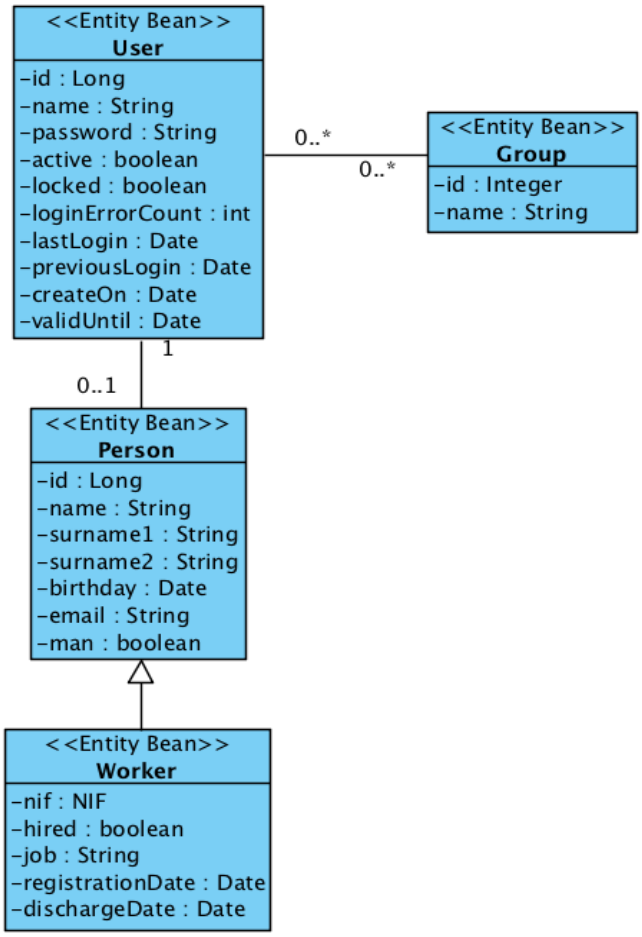


Ilustración 5: Diagrama de clases: entidades servicio web

La clase `User` representa un usuario de la aplicación, y junto con los grupos a los que pertenezcan serán las entidades sobre las que se aplicará la seguridad.

La clase `Person` tiene los datos básicos de una persona concreta y que para poder acceder a la aplicación debe tener un usuario de la misma.

La clase `Worker` es la encargada de representar a un trabajado de la empresa, según el análisis inicial, a la aplicación accederán principalmente empleados directos, pero también tiene que soportar externos. Los externos tendrán la propiedad `hired` establecida a falso.

Casos de Uso

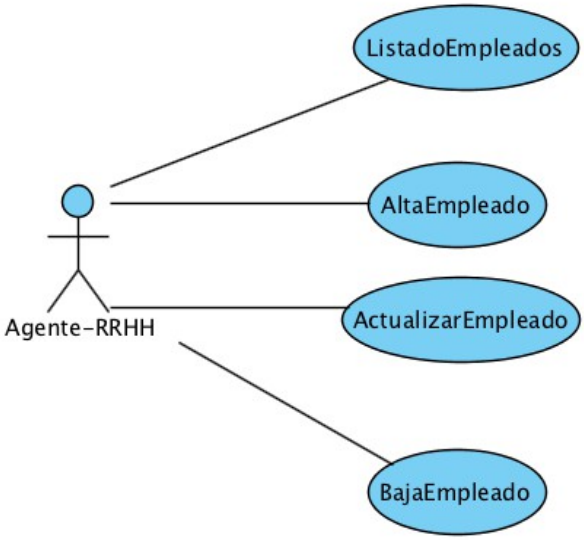


Ilustración 6: Casos de uso: servicio web

El servicio web es realmente sencillo, básicamente permite obtener un listado de todos los empleados actuales, que son aquellos trabajadores con la propiedad `hired` establecida como verdadero y que no han finalizado su contrato, crear un usuario nuevo, modificar uno existente y eliminarlos.

Diagrama de Componentes

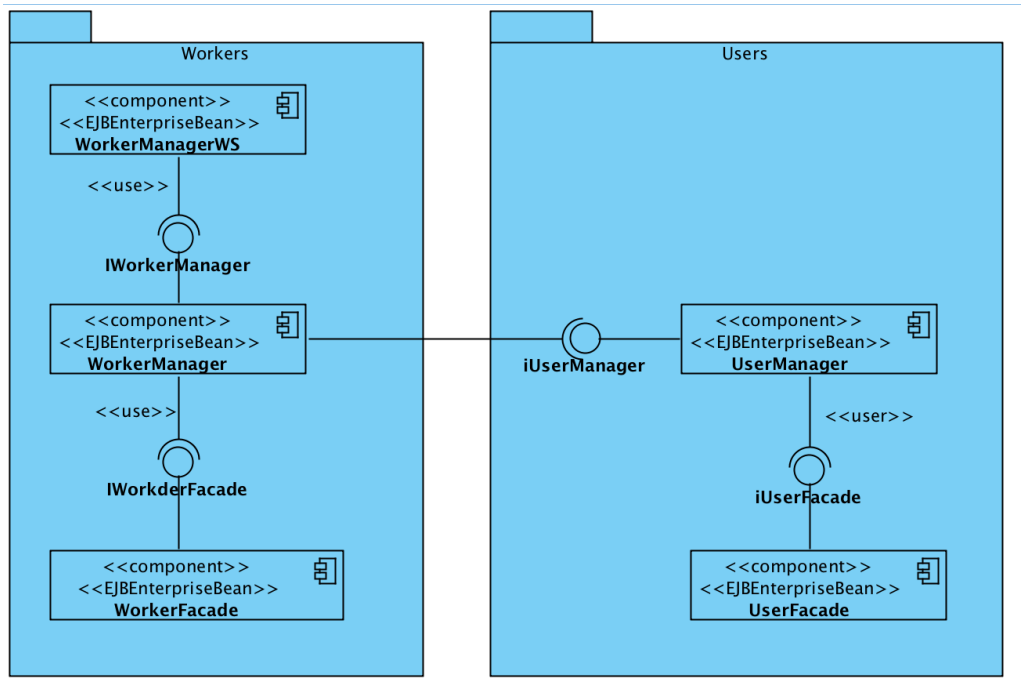


Ilustración 7: Diagrama de componentes: servicio web

Problemas encontrados en el desarrollo

El desarrollo de este módulo, es de los que mas problemas me ha ocasionado, en concreto, lo referente a la autenticación del cliente.

Según había leído en el tutorial de Java EE, capítulo de seguridad, para que el servidor de aplicaciones se encargue de revisar que el usuario tiene permiso para llamar/ejecutar un método o función de un EJB, basta con indicarlo con la anotación @RolesAllowed(“nombreRol”), ya sea en toda la clase o en métodos concretos.

Funcionar vaya si funcionaba, porque era añadir esas anotaciones y las llamadas al servicio web fallaban por no tener autorización.

Y ahí vinieron los problemas, y lo que me hizo leer la mayoría de referencias que se encuentran en la bibliografía.

La especificación de Java EE, no especifica como se tiene que implementar esos mecanismos, y mucho menos como se tienen que configurar en los servidores de aplicaciones, queda a discreción del implementador.

Como yo había decido utilizar GlassFish, como servidor de aplicaciones, pues miré su documentación así como diferente ejemplos. Para que la seguridad funcionara, había que realizar tres pasos:

1. Decirle al servidor de aplicaciones donde tenía el listado de usuario/contraseñas y grupos. GlassFish a eso lo llama *Realms*.
2. Indicarle a la aplicación que *Realm* tiene que utilizar.
3. Opcionalmente, hacer un mapeo entre los roles que chequea la aplicación y los usuarios/grupos definidos en el *Realm*.

Aparentemente no tendría que ser nada complicado, podríamos decir que es parecido a definir un recurso JDBC en el servidor e indicarle a la aplicación que lo use.

Pero nada mas lejos de la realidad, en la mayoría de ejemplos que encontraba, se hacia uso de un *Realm* sobre fichero, cuando yo quería que los datos los cogiese de la base de datos. Por suerte, sí que veía que GlassFish soporta *Realms* contra un recurso JDBC, pero hasta que no encontré el artículo de Bart Kneepkens, no conseguí configurarlo.

Ya teníamos el punto uno realizado, ahora viene el otro problemático, indicarle a mi aplicación que Realm tiene que usar, o mejor dicho, indicarle al servidor de aplicaciones que Realm tiene que usar con mi aplicación. Nuevamente la documentación que encontraba solo mostraba ejemplos que no me servían, todos indicaban como hacerlo para una aplicación web, y eso no funcionaba porque mi servicio web se encontraba en un paquete de EJBs.

Después de leer mucho, y de hacer pruebas y pruebas, perdí la cuenta, encontré lo que tendría que haber funcionado. Parecido al fichero que había que utilizar para configurar una aplicación web, se podía usar uno para el EJBs. En este caso, añadiendo el fichero `glassfish-ejb-jar.xml` a la carpeta `META-INF`, con el siguiente contenido, tendría que funcionar:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE glassfish-ejb-jar PUBLIC "-//GlassFish.org//DTD GlassFish Application
Server 3.1 EJB 3.1//EN" "http://glassfish.org/dtds/glassfish-ejb-jar_3_1-1.dtd">
<glassfish-ejb-jar>
  <enterprise-beans>
    <ejb>
      <ejb-name>WorkerManagerWS</ejb-name>
      <webservice-endpoint>
        <port-component-name>WorkerManagerPort</port-component-name>
        <endpoint-address-uri>WorkerManager/UserManagerWS</endpoint-address-uri>
        <login-config>
          <auth-method>BASIC</auth-method>
          <realm>WorkersPortalRealm</realm>
        </login-config>
        <transport-guarantee>CONFIDENTIAL</transport-guarantee>
        <debugging-enabled>true</debugging-enabled>
      </webservice-endpoint>
    </ejb>
  </enterprise-beans>
</glassfish-ejb-jar>
```

Pero aunque en la misma documentación del servidor de aplicaciones GlassFish Server Open Source Edition – Application Deployment Guide salen esos parámetros de configuración, cuando arrancaba la aplicación, en el registro del servidor indicaba que varios de esos parámetros no eran reconocidos. Probé con diferentes versiones de GlassFish, incluso con la última beta, pero nada, todas decían lo mismo.

Así que lo único que se me ocurrió fue crear otro proyecto web, que solo contendría el servicio web, ya que por las pruebas que había hecho, la configuración del *Realm* desde ahí sí funcionaba.

Tenía que ser un proyecto separado de la interface web, ya que para la interface web me interesaba la autenticación por formulario y para los servicios web la básica. Y ese es un parámetro que afecta a todo el proyecto web, y no a unas *URL* concretas.

4.3 Agente-RRHH

Descripción

Este agente es el responsable de mantener sincronizados los trabajadores activos de la empresa con los usuarios del portal del empleado. Para ello accederá a la base de datos utilizada por el programa Denario.

Requisitos implementados

1. No interferir en el funcionamiento del programa de RRHH.
2. No representar un problema de seguridad para los datos del programa de RRHH.
3. Si se cambiara el programa de RRHH que afecte lo mínimo posible a este módulo.
4. Funcionar como un servicio de windows.

Para garantizar los dos primeros puntos, se decide crear un *schema* en la base de datos del programa Denario, que tendrá únicamente acceso de lectura a una vista y que contendrá solo las columnas necesarias.

Trabajar con una vista, también ayuda a cumplir el tercer punto, siempre que el nuevo programa mantuviera los datos en una base de datos accesible vía JDBC.

Para poder adaptarse fácilmente a cambios de aplicación, se pondrán los datos de conexión en un fichero de configuración. Así lo único que hay que mantener estable, para no tener que modificar este módulo, es la vista con la que trabaja.

Servicio de windows

Para realizar la implementación del agente como servicio de windows, he analizado diferentes frameworks:

- Java Service Wrapper (https://wrapper.tanukisoftware.com/doc/english/introduction.html)
- Apache Commons Daemon (http://commons.apache.org/proper/commons-daemon/index.html)
- YAJSW (http://yajsw.sourceforge.net)
- WinRun4J (http://winrun4j.sourceforge.net)
- NSSM (http://nssm.cc)

De todas esas opciones elegí Apache Commons Daemon por:

- Soporta sistemas Windows y Linux.
- Fácil integración.
- Baja sobrecarga (la librería general solo ocupa 24KB, y no es necesaria si no se requieren funcionalidades avanzadas).
- Licencia libre y de código fuente abierto.

Modelo de datos

El siguiente diagrama ER muestra la estructura que debe tener la vista de la base de datos del programa de RRHH.

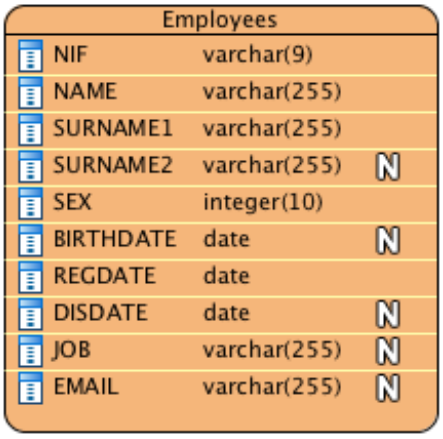


Ilustración 8: Diagrama ER: agente rrhh

Diagrama de Secuencia

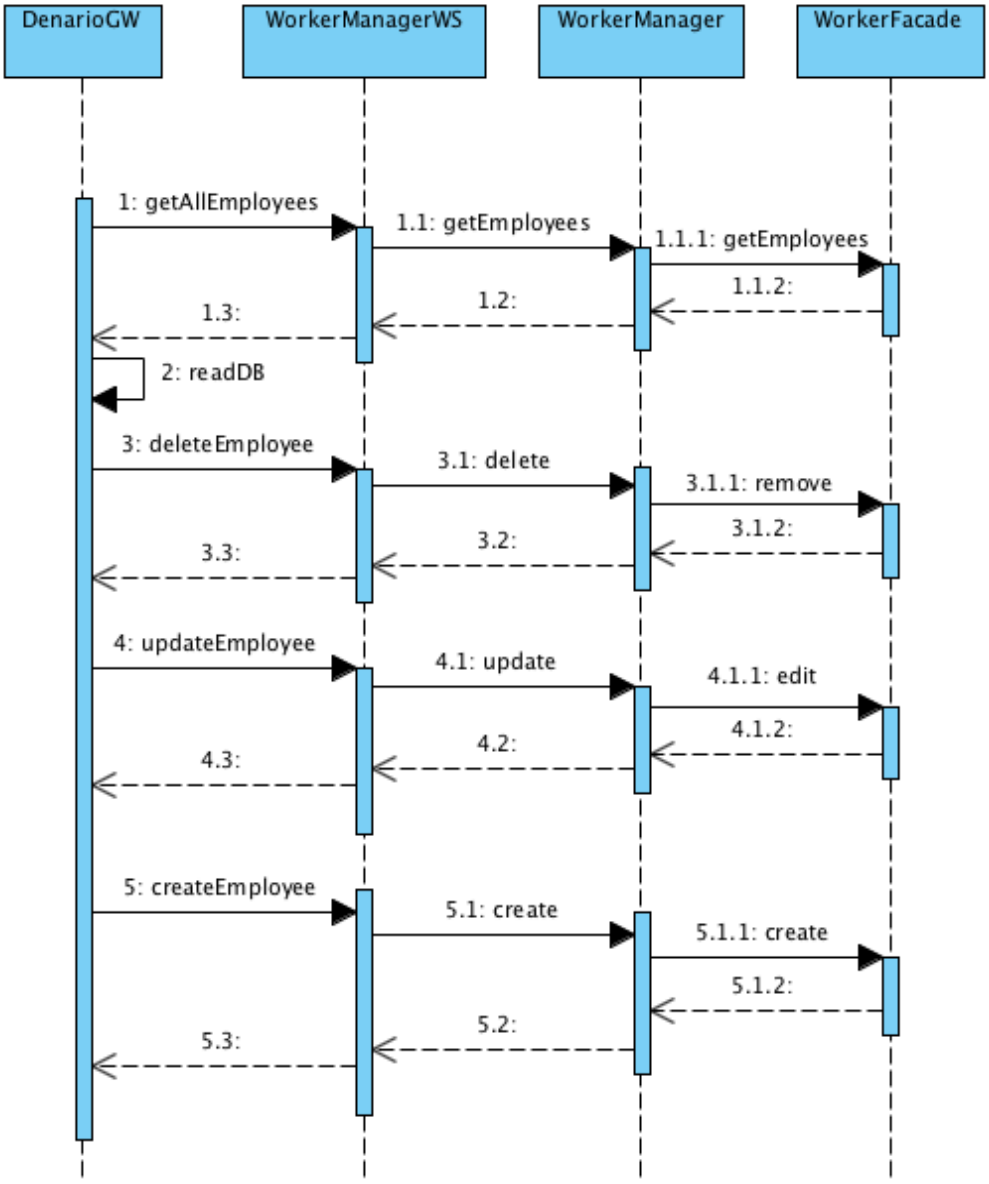


Ilustración 9: Diagrama de secuencia: agente rrhh

El funcionamiento general es sencillo, en un bucle constante, ejecutado cada X segundos, el módulo DenarioGW consulta el servicio web para obtener los empleados actuales (está previsto un parámetro de configuración para que esa consulta la haga solo al arrancar la aplicación, optimizando así las consultas al servio web). Luego leerá la base de datos de la aplicación de RRHH, mediante la vista definida. Comparando ambas listas, se pondrá nuevamente en contacto con el servicio web para borrar, actualizar y añadir los nuevos trabajadores.

A su vez, el servicio web delega toda la operativa en el Manager y este en el Facade correspondiente.

Problemas encontrados en el desarrollo

Este módulo dejó de funcionar cuando quise que se conectara al servicio web utilizando HTTPS. Como el servidor GlassFish tenía un certificado

autofirmado, el JRE se quejaba de que no era confiable y rechazaba la conexión. Probé añadir el certificado a la lista de certificados válidos, pero tampoco funcionaba. En este caso Google, y el estupendo portal stackoverflow, me ayudaron a dar con un mecanismo para, en tiempo de ejecución, desactivar la verificación de los certificados (mecanismo que también se puede configurar desde el fichero de preferencias, por si en la instalación definitiva se utiliza un certificado firmado por una autoridad certificadora).

4.4 Gestor de flujos de trabajo

Descripción

Este módulo engloba las funcionalidades mas importantes, de la primera fase, del proyecto.

Es el encargado de ejecutar las tareas asociadas a un flujo de trabajo. Para este caso inicial, solo existirá un flujo de trabajo que será ejecutado cuando se de alta un nuevo trabajador.

Requisitos implementados

Los flujos de trabajo tienen las siguientes características:

- Un usuario solo puede tener un flujo de trabajo activo.
- No será recurrente, el flujo tendrá un único punto de inicio y fin.
- Estará compuesto por una o más tareas.
- Las tareas pueden ejecutarse en paralelo o secuencialmente.
- Las tareas pueden ser de distinto tipo, por ejemplo:
  - Alta de usuario en aplicativo X.
  - Configuración X a realizar en el ordenar del usuario.
  - Asignación de recursos físicos (ordenador, pantalla, etc).
- Aunque haya tareas que tengan un módulo que permita su realización automática, se tendrá que poder desactivar su ejecución automática (ya sea en la definición de la plantilla, o específicamente para un flujo de trabajo).

Modelo de datos

Aquí encontramos el mayor número de clases. Primero analizaremos las que definen un flujo de trabajo, que será como la receta que indica al `WorkflowManager` que tiene que hacer y como.

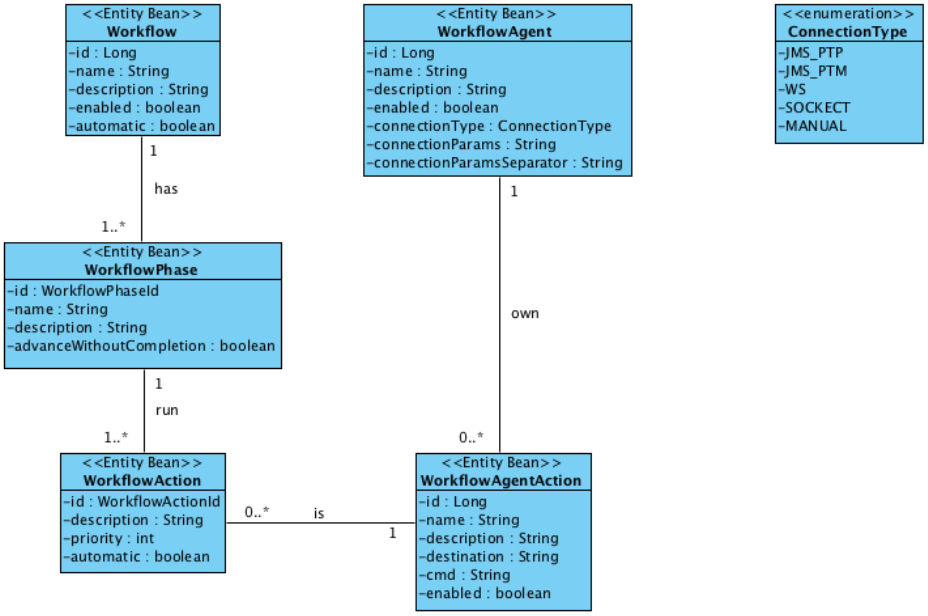


Ilustración 10: Diagrama de clases: entidades flujo de trabajo

La clase `Workflow` es la inicial, desde ella se puede acceder a las fases del flujo de trabajo.

La clase `WorkflowPhase`, sirve para agrupar un conjunto de acciones que se pueden ejecutar en paralelo. El orden en que se ejecutan las fases queda establecido en su id. Si no se indica lo contrario, con la propiedad *`advanceWithoutCompletion`*, hasta que no se han finalizado todas las actividades de una fase no se pasará a la siguiente fase.

Las acciones están recogidas en dos clases (en un diseño estricto, la clase `WorkflowAction` se habría representado como propiedades de la asociación `WorkflowPhase-WorkflowAgentAction`, pero en este proyecto los diagramas de clase he preferido que no sean tan abstractos y tengan una relación directa con las clases Java a implementar).

La clase `WorkflowAction`, actúa de unión entre la fase y la definición de la acción. Nos sirve para indicar el orden de ejecución dentro de la fase y si lo hará de forma automática o requerirá ser ejecutada por un operario.

La clase `WorkflowAgentAction`, junto con la clase `WorkflowAgent` al que pertenece, le da la información necesaria al manager para que pueda ponerse en contacto con un agente y le solicite ejecutar una acción concreta.

Con todo esto, tendríamos la receta lista, pero ahora se necesita lo que he llamado un registro de ejecución, donde saber por que parte de la receta estamos. Para ello el manager creará una copia del *workflow* a ejecutar, utilizando otras clases, parecidas a las anteriores pero que terminan en “Register”, y que añaden atributos para saber cuando se ha ejecutado, si está finalizado, etc. Podemos ver este diseño en el siguiente diagrama:

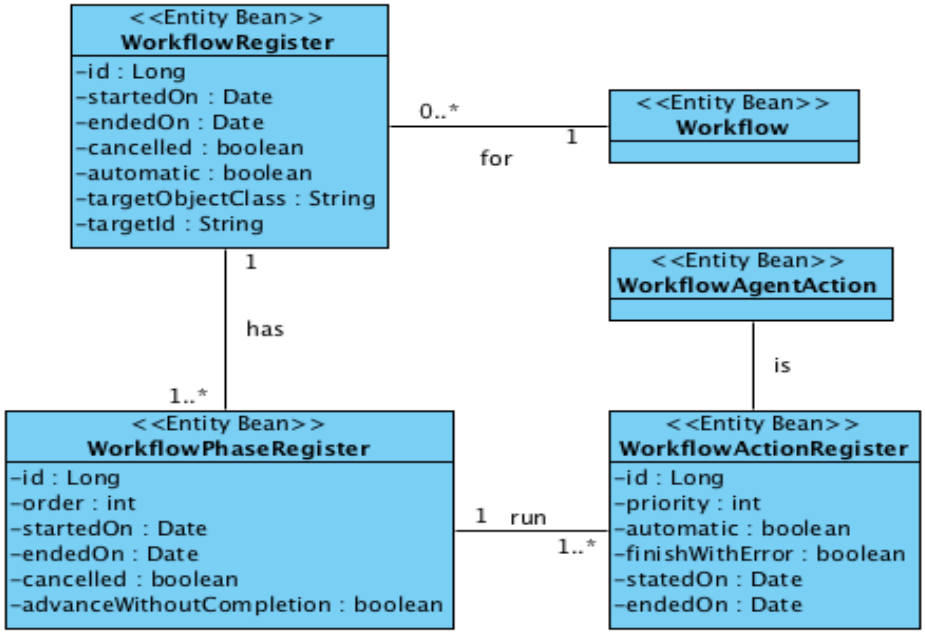


Ilustración 11: Diagrama de clases: entidades registro flujo de trabajo

El concepto es el mismo, el *workflowRegister* tiene fases y las fases acciones que pertenecen a un agente.

Casos de Uso

Tal y como muestra el siguiente diagrama, la activación de los flujos de trabajo no la realiza directamente ningún actor, es un proceso indirecto.

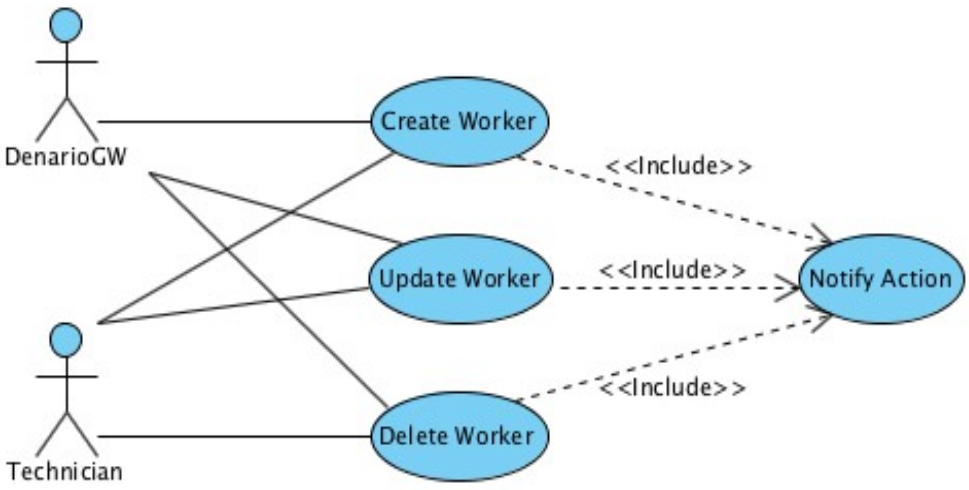


Ilustración 12: Casos de uso: flujos de trabajo

Cuando un técnico o el agente del programa de RRHH crea/modifica o borra un trabajador, se enviará un mensaje, usando una cola JMS, y

será ese mensaje el que activará el flujo de trabajo, como veremos en el diagrama de secuencia.

Diagrama de Secuencia

En este diagrama mostraremos la creación de un trabajador partiendo desde la llamada *`create`* que se hace al `WorkerManager`, ya sea del servicio web que imboca el agente RRHH o desde el `workerController` del interface de usuario.

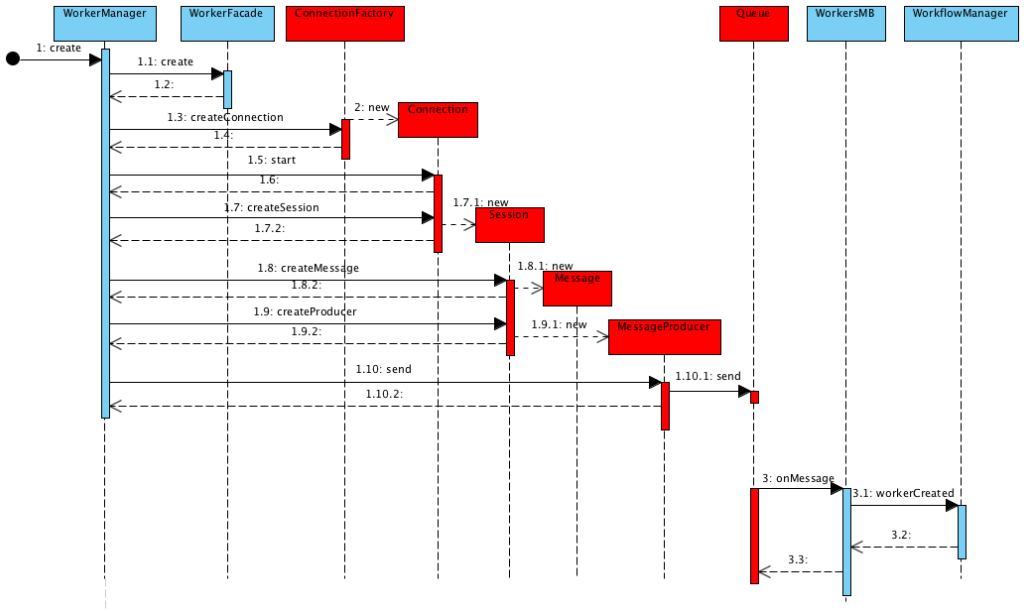


Ilustración 13: Diagrama de secuencia: flujos de trabajo

En rojo están marcadas las partes que pertenecerían al framework JMS, por lo que hay que entenderlas como una representación que puede no ser exacta.

Como se puede observar, después de que el `WorkerManager` haya persistido el trabajador, creará una conexión, con esta conexión una sesión, de la sesión un mensaje y un productor de mensajes que será el encargado de hacerlo llegar a la cola.

La cola, en algún momento, notificará al `MessageDrivenBean WorkersMB` del mensaje nuevo, y este, tras comprobar que el mensaje es del tipo esperado, notificará al `WorkflowManager` para que, así, se pueda iniciar un flujo de trabajo.

Diagrama de Actividad

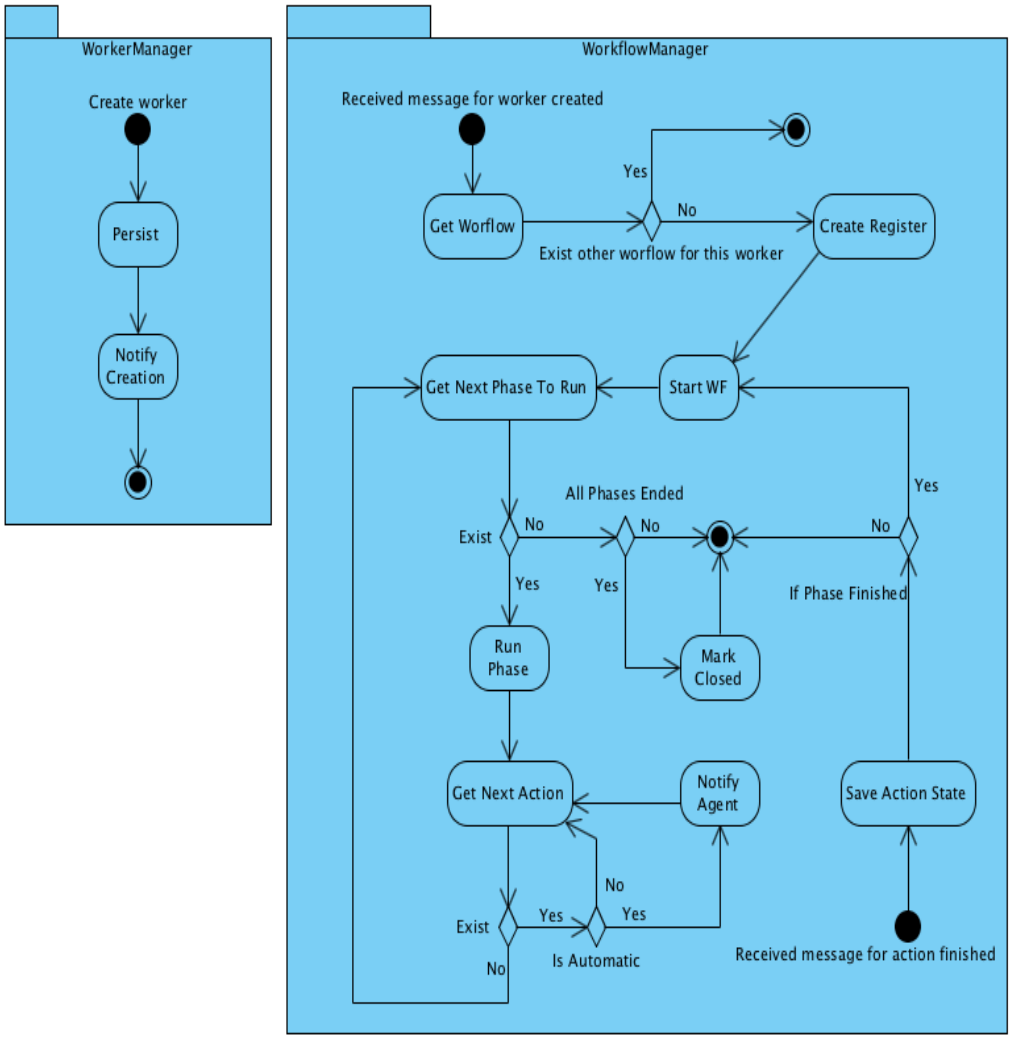


Ilustración 14: Diagrama de actividad: gestor flujos de trabajo

En el diagrama anterior vemos, por una lado lo que hace el `WorkerManager`, que como se había explicado en la sección anterior, básicamente es persistir el trabajador y notificar su creación, por medio de una cola JMS.

Cuando el `WorkflowManager` recibe la notificación de que se ha creado un trabajador nuevo, notificación que lleva el ID del trabajador, cargará el flujo de trabajo que se tiene que ejecutar en estos casos y comprobará que ese trabajador no tiene ya otro flujo asociado pendiente de finalizar. En caso afirmativo, creará un registro para este flujo de trabajo, asignará el ID del trabajador que inicia este flujo y empezará su ejecución que consiste en:

- Obtener la siguiente fase a ejecutar: este proceso recorre por orden las fases del flujo de trabajo. Si la fase que se está analizando no está iniciada, se retorna esa. Si estaba finalizada, se continua con la siguiente fase. Si no estaba finalizada y se permite continuar sin haber finalizado, se continua con la siguiente fase, sino se retorna nulo igual que si no quedan mas

fases. El siguiente diagrama de actividad puede servir para entender mejor lo explicado:

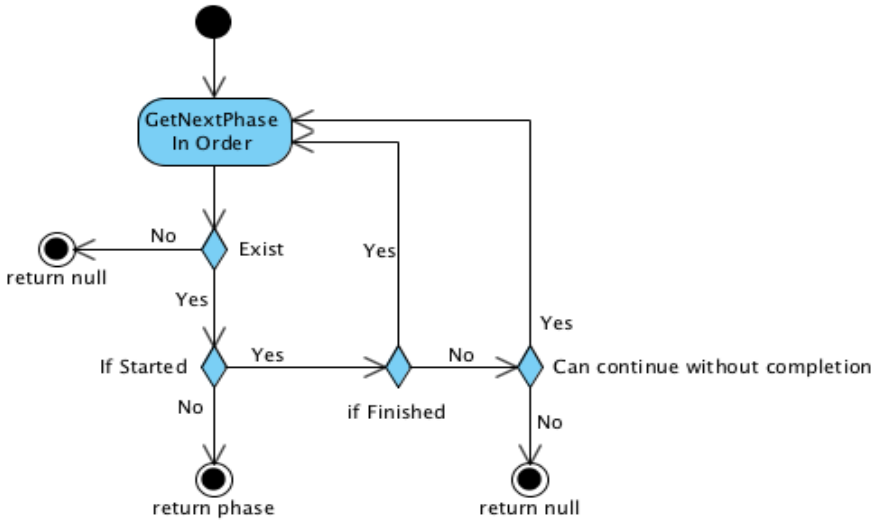


Ilustración 15: Diagrama de actividad: gestor flujos de trabajo, siguiente fase

- Si en el paso anterior se ha obtenido una fase, se inicia dicha fase, que consiste en marcarla como iniciada y recorrer todas sus acciones.
- Por cada acción de la fase, si esta se puede ejecutar automáticamente, se enviará un mensaje a la cola, con los parámetros necesarios para que el agente pueda ejecutar la acción. Esos parámetros se encuentran en los datos definidos en las clases `WorkflowAgentAction`, `WorkflowAgent` y `Workflow`.
- Cuando se han recorrido todas las acciones, se comprueba si se puede ejecutar alguna fase mas.
- Otro punto de entrada al proceso es cuando se recibe un mensaje indicando que una acción a finalizado. En ese caso, después de guardar el estado de la acción, se comprueba si la fase a finalizado, en cuyo caso se vuelve a ejecutar el *workflow* para que se ejecute la siguiente fase que esté pendiente.
- Cuando todas las fases se han ejecutado, y completado, se finaliza el registro del flujo de trabajo.

Colas JMS

El gestor de flujos de trabajo utiliza tres colas JMS para su funcionamiento. Serían las siguientes:

- `jms/WPWorkers`: es de tipo *Queue*, o lo que es lo mismo PTP (punto a punto, del inglés *Point to Point*). A esta cola el `WorkerManager` notifica las acciones que realiza sobre un trabajador. El `MessageDrivenBean WorkerMB` lee esos mensajes y los notifica al `WorkflowManager` para que lance el flujo de trabajo adecuado.

- `jms/WPWorkflowManager`: cola también de tipo *Queue*. Es utilizada por los distintos agentes para notificar la finalización de una acción. El `WorkflowManager` recibirá esas notificaciones por el `MessageDrivenBean` `WorkflowManagerMB`.
- `jms/WPWorkflow`: esta es la única cola tipo *Topic*, o PTM (punto a multipunto, del inglés *Point to Multipoint*). En esta cola publica mensajes el `WorkflowManager`, con los que avisa a los distintos agentes de un flujo, que tienen que realizar una acción. El único agente implementado, el gestor de usuarios, recibe los mensajes con el `MessgeDrivenBean` `UsersMB`.

Problemas encontrados en el desarrollo

Al implementar este módulo, en concreto el uso de las colas JMS, me encontré con un fallo, que llegué a pensar que sería un bug en Glassfish, aunque me costaba aceptarlo. El síntoma era sencillo, no se creaban usuarios para todos los trabajadores.

Primero pensé que se tratase de algún tipo de *race condition*, que hiciera que se perdiese algún mensaje. Luego, después de establecer controles, me di cuenta que el problema venía en que los `MessageDrivenBeans` recibían mensajes que no pertenecían a su cola, fue ahí cuando pensé que el fallo era de Glassfish.

Las colas no las había creado yo directamente en Glassfish, sino que había utilizado un asistente que tiene el propio Netbeans. Este asistente es muy sencillo, pide un nombre para la cola y el tipo de cola, y con esos datos crea un fichero `glassfish-resources.xml` que es utilizado en el momento del despliegue para crear las colas, y cuyo contenido es el siguiente:

```
<?xml version="1.0" encoding="UTF-8"?><!DOCTYPE resources PUBLIC
"-//GlassFish.org//DTD GlassFish Application Server 3.1 Resource Definitions//EN"
"http://glassfish.org/dtds/glassfish-resources_1_5.dtd">
<resources>
    <admin-object-resource enabled="true" jndi-name="jms/WPWorkflow"  res-
type="javax.jms.Topic"  res-adapter="jmsra">
        <property name="Name" value="PhysicalQueue"/>
    </admin-object-resource>
    <connector-connection-pool name="jms/WPWorkflowFactoryPool"  connection-
definition-name="javax.jms.TopicConnectionFactory"  resource-adapter-
name="jmsra"/>
    <connector-resource enabled="true" jndi-name="jms/WPWorkflowFactory" pool-
name="jms/WPWorkflowFactoryPool"  />
    <admin-object-resource enabled="true" jndi-name="jms/WPWorkers"  res-
type="javax.jms.Queue"  res-adapter="jmsra">
        <property name="Name" value="PhysicalQueue"/>
    </admin-object-resource>
    <connector-connection-pool name="jms/WPWorkersFactoryPool"  connection-
definition-name="javax.jms.QueueConnectionFactory"  resource-adapter-
name="jmsra"/>
    <connector-resource enabled="true" jndi-name="jms/WPWorkersFactory" pool-
name="jms/WPWorkersFactoryPool"  />
    <admin-object-resource enabled="true" jndi-name="jms/WPWorkflowManager"  res-
type="javax.jms.Queue"  res-adapter="jmsra">
        <property name="Name" value="PhysicalQueue"/>
    </admin-object-resource>
    <connector-connection-pool name="jms/WPWorkflowManagerFactoryPool"
connection-definition-name="javax.jms.QueueConnectionFactory"  resource-adapter-
name="jmsra"/>
    <connector-resource enabled="true" jndi-name="jms/WPWorkflowManagerFactory"
pool-name="jms/WPWorkflowManagerFactoryPool"  />
</resources>
```

Como se observa, las tres colas tenían una propiedad llamada "`Name`", con el mismo valor. Desde el gestor de Glassfish, se puede ver con mayor claridad, que esa variable es la que indica la cola física, o real, del servidor. Por lo tanto, las tres colas estaban asociadas a la misma cola, y por eso los mensajes podían ser recibidos por cualquiera de ellas indistintamente de por cual hubiesen entrado.

Una vez configurada cada cola hacia una cola física distinta, todo se puso a funcionar correctamente.

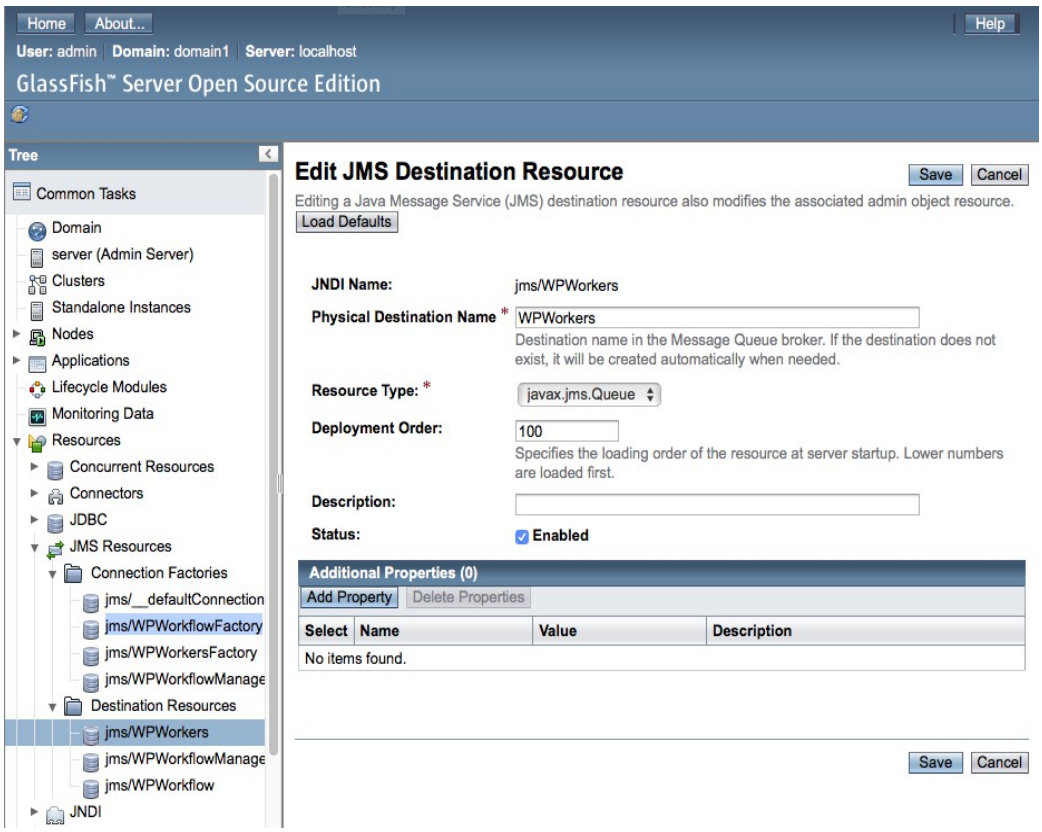


Ilustración 16: Captura: configuración cola JMS

5. Estructura del código fuente

Toda la aplicación, a excepción de los agentes, se empaqueta en un único fichero EAR (*Enterprise Application aRchive*). Este fichero incluirá las distintas librerías, no presentes en Java EE 7, y los archivos propios del proyecto. La capa de negocio aportará dos ficheros JAR (y un fichero WAR para solventar el problema con el servicio web), uno con interfaces, clases de utilidad y modelo y otro con todos los EJBs. Por su parte, la capa de presentación incluirá todos sus archivos en un único fichero WAR.

Así el proyecto quedará organizado en los siguientes subproyectos:

- WorkersPortal-Builder: utilizado para facilitar la compilación de todos los componentes del proyecto.
- WorkersPortal-Master: utilizado como proyecto padre de todos los demás, donde se colocarán configuraciones Maven comunes a todos los proyectos.
- WorkersPortal-Lib: se encarga de generar el fichero JAR, de la capa de negocio, con las interfaces externas, clases de utilidad y de modelo, que pueden ser utilizadas desde otros proyectos.
- WorkersPortal-Ejb: contiene todos los EJBs del proyecto, menos los servicio web.
- WorkersPortal-Web: genera el fichero WAR, que contiene todos los archivos de la capa de presentación, ya sean java beans, jsf, etc.
- WorkersPortal-WS: genera un fichero WAR, que solo contiene los servicios web.
- WorkersPortal-EA: encargado de componer el fichero final del proyecto, utilizando los resultados de los anteriores proyectos.

En el anexo de despliegue se explica como realizar la instalación y configuración en un servidor Glassfish.

Los agentes, que serán aplicaciones java (por lo general de consola) tendrán un fichero JAR principal y, si necesitan alguno mas, como librerías de terceros, se incluirán estos archivos en una carpeta llamada lib.

El agente para sincronizar el portal con la aplicación de recursos humanos, está formado por:

- DenarioGateway: módulo principal del agente.
- DenarioGateway-Lib: contiene clases de utilidad, de modelo, de acceso al servicio web, etc.

- DenarioGateway-Web: es una pequeña aplicación con interface web, para poder crear/modificar y borrar trabajadores y así probar el módulo sin tener que utilizar el programa de recursos humanos.

La instalación y configuración del módulo queda reflejada en el anexo de despliegue.

6. Plataforma de Test

Para facilitar poder probar la aplicación sin tener que montar todo el entorno (base de datos, servidor de aplicaciones) he creado y puesto accesible en internet una plataforma lista para utilizar.

Se puede acceder a la aplicación DenarioGateway-Web (desde donde se podrá simular la creación, modificación y eliminación de trabajadores para que el agente lo notifique al portal) desde la URL:

http://www.mersisl.com/DenarioGateway-Web

Y al portal del empleado (desde donde se podrán ver los usuarios, trabajadores, flujos y registros de flujos activos) desde el siguiente enlace:

http://www.mersisl.com/WorkersPortal-Web

Los datos de acceso al mismo son:

- usuario: uoc
- contraseña: tfg

7. Conclusiones

Llevo ya unos años realizando pequeñas y medianas aplicaciones a medida, por lo que el desarrollo de este proyecto, mas que lecciones nuevas, ha afianzado algunas de las ya adquiridas durante este tiempo.

Por un lado, tengo claro que desarrollar aplicaciones con una sola persona, suele comportar grandes desviaciones en la planificación inicial. Los motivos principales, de estas desviaciones, son:

- No poderse dedicar en exclusiva a un solo proyecto. Si hay que aparcar el programa que se esté realizando, para ponerse en otras tareas (ya sea de otros programas o del departamento de TI), además del tiempo no dedicado en ese instante, suele necesitarse un período de adaptación para retomar el punto donde se había dejado.
- Contar con un solo desarrollador, no permite recuperar tiempo cuando la planificación ha sido demasiado optimista o si se encuentra algún problema por el camino que no se pudo prever (que casi siempre suelen aparecer).
- Cualquier indisponibilidad, total o parcial, de la persona asignada al proyecto (ya sea por cuestiones de salud, personales, etc) se refleja negativamente en la planificación.

Otra cuestión, ya mas personal, es que tengo cada día mas claro que me gustan todas las fases del ciclo de desarrollo (especialmente la de diseño y optimización) y todas las áreas de programación a excepción de las interfaces de usuario, ya que aunque considero que la programación tiene cierto grado/tipo de arte, las interfaces de usuario requieren unas capacidades artísticas que no poseo. Además, me he encontrado muchas veces con problemas de comportamiento extraños, en algunos componentes de interface, que han requerido tiempo y lo que yo llamo *workarounds,* feos, para solventarlos (que en la mayoría, con el tiempo, se han solucionado al actualizar la librería o el componente en cuestión). Como ejemplo, en este proyecto, el componente utilizado para la representación gráfica del *workflow*, en ciertas ocasiones, todavía sin determinar, dibuja mas de una linea, no conectada a ningún nodo, pero que, por suerte, al quedar encima o debajo de otra, solo será visible si se mueve algún nodo.

Por lo dicho hasta ahora, podría parecer que realizar aplicaciones a medida no sea lo mas recomendable, y sobre todo, cuando se cuenta con pocos recursos personales. Pero en mi opinión, no es así. Considero que siempre será mas productivo que una aplicación se adapte a los métodos de trabajo de la empresa, que no adaptar a la empresa a la manera de hacer de aplicaciones, en algunos casos mal llamadas estándares, yo las llamaría genéricas, que para poder llegar a un gran y variado número de usuarios, se quedarán cortas en algunas cuestiones y serán excesivamente exhaustivas en otras.

Estoy diciendo con esto que las aplicaciones genéricas son malas o desaconsejables, tampoco, en algunos casos, por ejemplo en empresas que no tienen establecidos unos buenos procesos internos, puede que tener que adaptarse a alguno les sea beneficioso. O por ejemplo, no tendría sentido querer realizar a medida alguna aplicación que cubra un proceso altamente genérico que ya tiene soluciones en el mercado (como puede ser un procesador de textos).

Como ya he comentado al principio, llevo un tiempo programando aplicaciones a medida y las últimas han sido utilizando Java EE y JSF para la interface de usuario.

Así que para que este proyecto me aportase algo nuevo, propuse los siguientes objetivos.

En el plano tecnológico:

- Crear un servicio de windows en java. Hasta ahora, si había necesitado instalar un servicio en windows, lo había implementado en C#.
- Utilizar JMS. En los anteriores proyectos no había la necesidad de conectar módulos o partes externas, ni siquiera de establecer procesos de estilo creador/consumidor. Vi en el concepto de *workflow* una buena oportunidad para utilizar esta tecnología.
- Comunicación sobre HTTPS. Al ser aplicaciones internas, ejecutadas sobre una red, a priori segura, nunca había tenido como requisito darle una capa mas de seguridad.
- Seguridad en las llamadas a los EJBs, especialmente para los servicios web. Hasta ahora la seguridad que había implementado se basaba en una autenticación del usuario al acceder a la aplicación y en mostrar u ocultar las distintas acciones que podía hacer. Podríamos decir que la seguridad se establecía solo en la interface de usuario. Utilizando los servicios que ofrece la plataforma JaveEE, es muy fácil y cómodo extender esa seguridad a toda la aplicación.

En general estoy contento con los objetivos tecnológicos planteados y su ejecución que ha sido completa.

También me planteé como objetivo realizar una gestión del proyecto mas profesional con alguna herramienta existente y aprovechar para aplicar una metodología ágil.

La herramienta elegida (OpenProject), me ha gustado, y seguramente la siga utilizando en nuevos proyectos. Ayuda a plasmar el estado, realizar un seguimiento de la ejecución, permitiría o facilitaría que colaborase algún desarrollar adicional, aunque fuese en tareas concretas.

Como metodología elegí SCRUM, por gustarme su filosofía y por estar soportada de forma nativa en OpenProject.

Estos dos objetivos, la utilización de una herramienta de gestión de proyectos y el uso de la metodología SCRUM, serían los que peor sabor me han dejado. No porque crea que hayan sido elecciones equivocadas, todo lo contrario, sino porque no le he podido dedicar el tiempo necesario para sacarle partido. Al retrasarse el proyecto, por los motivos que se han comentado en el capítulo 3, tuve que recortar tareas para poder alcanzar el mayor número de funcionalidades. Las tareas sacrificadas son las que yo llamo de gestión:

- Control del proyecto,
- Documentación, desde los propios comentarios dentro del código, a manuales, diseños, etc.

La planificación inicial apenas se pudo mantener las primeras semanas. Como ya se ha comentado, es muy difícil mantenerla cuando solo tienes un recurso humano, y no lo tienes disponible todo el tiempo, porque hay que realizar otras tareas, que además son de larga duración. También hubo algunos fallos en la planificación, que fue muy optimista en los tiempo que se tardarían en algunas tareas, y no se pudo prever problemas al implementar ciertas funcionalidades (sobre todo la seguridad de los servicios web), además de alguna bajada de rendimiento debido al estrés por exceso de trabajo acumulado.

Esta experiencia me hace pensar que para proyectos muy pequeños, el sobre coste en horas por llevar una buena gestión, puede no ser del todo beneficiosa, y se hace mas indispensable en grandes proyectos y, sobre todo, cuando colaboran varias personas.

Me habría gustado terminar el trabajo de fin de grado con toda la primera fase de la aplicación implementada, pero no ha sido así. Pero estoy contento porque, teniendo en cuenta los hechos, he podido realizar las bases sobre las que se ejecutarán el resto de módulos. Teniendo un agente, los demás serán, casi, copias de este adaptadas al caso. Teniendo un gestor CRUD, el resto serán similares, pero con otras entidades. Por lo tanto, creo que se ha hecho un buen trabajo.

Obviamente, esta aplicación no se quedará aquí, en los próximos meses habrá que seguir implementando agentes para varias de las aplicaciones internas, los gestores que han quedado pendientes de la primera fase y luego, se podrán añadir otras funcionalidades como:

- Pequeño gestor documental para:
  - Documentación de interés.
  - Documentación de riesgos laborales.
  - Plan de seguridad.
  - Documentos y formularios relacionados con la LOPD.
- Formularios comunicación para:
  - Solicitar vacaciones.
  - Envío parte de baja.

    - Contacto con RRHH.
- Apartado de noticias, como:
    - Ofertas para los trabajadores.
    - Circulares.
- Portal adaptado a dispositivos móviles.
- Aplicación nativa iPhone para que los técnicos reciban las alertas de alta/baja instantáneamente.

Otra funcionalidad que podría ser muy interesante, sobre todo desde el punto de vista del desarrollo, sería la posibilidad de acceder al portal mediante el DNI electrónico, claro que esto requeriría que todos los ordenadores tuvieran ese lector.

8. Glosario

CRUD: crear/leer/actualizar y borrar, del ingles *Create/Read/Update/Delete*. Operaciones típicas a realizar sobre un conjunto de datos.

CVS: del inglés *Concurrent Versions System,* o sistema de control de versiones.

EAR: *Enterprise Application aRchive*, archivo comprimido que contiene EJB.

EJB: *Enterpise Java Beans*. Clases javas diseñadas para ejecutarse en un contenedor específico.

HTTP: protocolo de comunicación por internet, del inglés *Hypertext Transfer Protocol*.

HTTPS: versión securizada extremo a extremo del protocolo HTTP.

IDE: *Integrated Development Environment*. Software que soporta todo el ciclo de desarrollo.

JAR: *Java ARchive*, fichero comprimido que contiene clases Java y recursos.

JavaEE: *Java Enterprise Edition*. Plataforma de desarrollo con funcionalidades específicas para implementar programas empresariales.

JAX-WS: *Java Web Services*. Especificación Java para soportar servicios web.

JMS: *Java Message System*. Especificación Java para la implementación de un sistema de mensajes entre productores y consumidores o subscriptores.

JPA: *Java Persistent API*. Especificación Java para la persistencia de entidades Java en bases de datos relacionales.

JRE: *Java Runtime Enviroment*. Plataforma encargada de la ejecución de aplicaciones Java.

JSF: *Java Server Faces*. Especificación java específica para el desarrollo de entornos web de la plataforma JavaEE.

LOPD: Ley Orgánica de Protección de Datos de carácter personal.

NetBeans: IDE que soporta varios lenguajes de programación como Java, PHP y C/C++

OpenProject: aplicación web focalizada en la gestión de proyectos de software.

POO: Programación Orientada a Objetos**.** Paradigma de programación.

RRHH: Recursos Humanos.

SCRUM: metodología ágil de desarrollo.

Servicio Web: tecnología de comunicación entre aplicaciones sobre el protocolo HTTP/S.

TI: Tecnologías de la información.

WAR: *Web Application aRchive.*

Workaround: superar un fallo sin corregirlo, mas bien, rodeándolo.

WS: *Web Service*, servio web.

9. Bibliografía

Libros:

- [Eric Jendrock, Ricardo Cervera-Navarro, Ian Evans, Kim Haase, Willian Markito] [Java Platform, Enterprise Edition - The Java EE Tutorial - Release 7] [Oracle] [2014]
- [Ian Evans] [Java Platform, Enterprise Edition – Your First Cup: An Introduction to the Java EE Platform – Release 7] [Oracle] [2014]
- [Linda DeMichiel, Bill Shannon] [JSR-342 Java Platform, Enterprise Edition 7 Specificaction] [Oracle] [2015]
- [Marina Vatkina] [JSR-345 Enterprise JavaBeans, Version 3.2, EJB Core Contracts and Requirements] [Oracle] [2013]
- [Ron Monzillo] [JSR-115 Java Authorization Contract for Containers] [Oracle] [2013]
- [Ron Monzillo] [JSR-196 Java Authentication SPI for Containers] [Oracle] [2013]
- [Oracle] [GlassFish Server Open Source Edition - Application Deployment Guide - Release 4.0] [Oracle] [2013]
- [Oracle] [GlassFish Server Open Source Edition - Application Development Guide - Release 4.0] [Oracle] [2013]
- [Oracle] [GlassFish Server Open Source Edition - Security Guide - Release 4.0] [Oracle] [2013]

Webs:

- [Using a Glassfish 4 JDBCRealm for basic user authorization] [Bart Kneepkens] [29/3/2017] [http://bart-kneepkens.github.io/using-a-glassfish-4-jdbcrealm-to-secure-your-web-application/]
- [How to fix the “java.security.cert.CertificateException”] [6/4/2017] [http://stackoverflow.com/questions/19540289/how-to-fix-the-java-security-cert-certificateexception-no-subject-alternative]
- [Número de identificación fiscal] [26/4/2017] [https://es.wikipedia.org/wiki/Número_de_identificación_fiscal]

10. Anexos

10.1 Despliegue

Portal

El paso previo, y que no detallaré, es instalar un motor de base de datos, en el ejemplo uso MySQL, y el servidor de aplicaciones Glassfish, versión 4.1.2 (aunque la aplicación tendría que funcionar en cualquier versión 4.x, ha sido probada solo en la 4.1.1 y 4.1.2).

Una vez instaladas esas aplicaciones, lo siguiente será crear las bases de datos y un usuario para acceder a ella. Se podría utilizar algún interface gráfico, como MySQLWorkbench, o acceder desde la consola. Las instrucciones necesarias para crear la base de datos del portal serían:

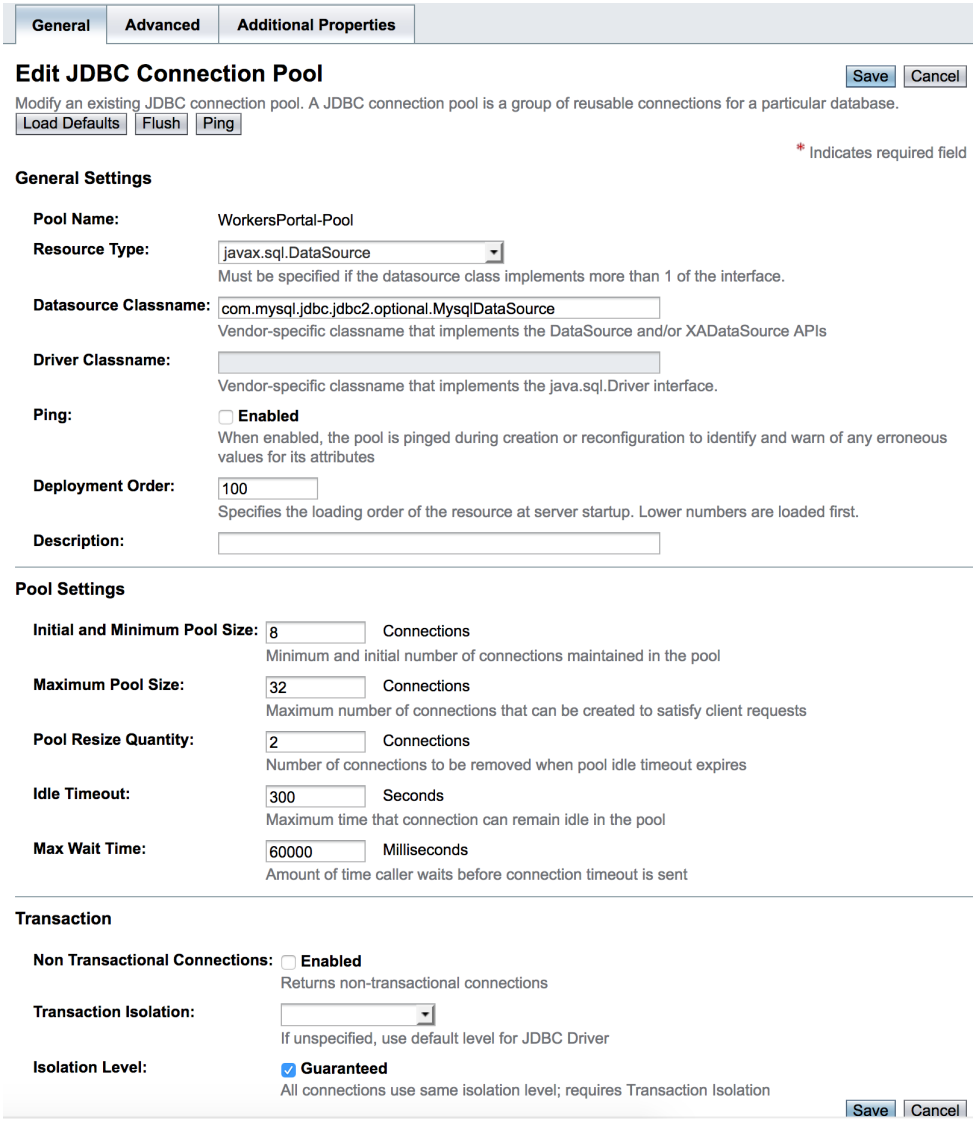
```
create schema WorkersPortal;
grant all privileges on WorkerPortal.* to wpuser identified by 'wppass';
flush privileges;
```

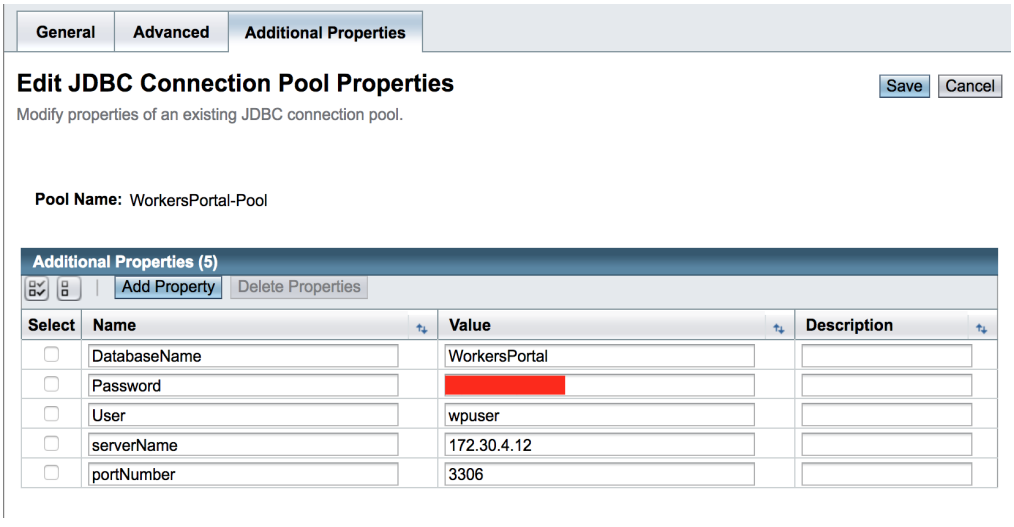
Donde el nombre de la base de datos es `WorkersPortal` y el usuario `wpuser` con contraseña `wppass`. Estos datos los tenemos que apuntar para configurarlos después en Glassfish.

Ahora nos conectamos a la consola de administracion de Glassfish, por defecto en http://ip-servidor:4848/.

En el menú de la izquierda accedemos a Resources->JDBC->JDBC Connection Pools y hacemos click en el botón New.

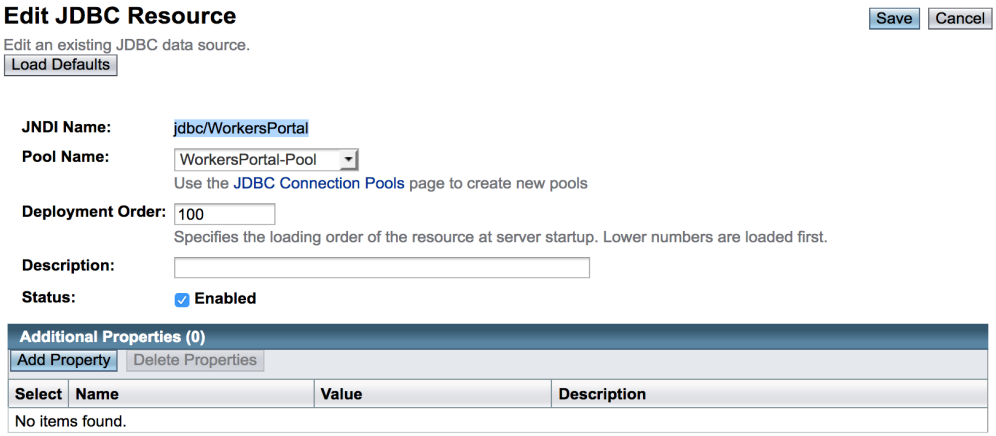
Aquí configuraremos la conexión a la base de datos, según el gestor de base de datos que hayamos elegido. Por ejemplo para un MySQL, quería algo así:


General
Advanced
Additional Properties
Edit JDBC Connection Pool
Save
Cancel
Modify an existing JDBC connection pool. A JDBC connection pool is a group of reusable connections for a particular database.
Load Defaults
Flush
Ping
* Indicates required field
General Settings
Pool Name:
WorkersPortal-Pool
Resource Type:
javax.sql.DataSource
Must be specified if the datasource class implements more than 1 of the interface.
Datasource Classname:
com.mysql.jdbc.jdbc2.optional.MysqlDataSource
Vendor-specific classname that implements the DataSource and/or XADataSource APIs
Driver Classname:
Vendor-specific classname that implements the java.sql.Driver interface.
Ping:
Enabled
When enabled, the pool is pinged during creation or reconfiguration to identify and warn of any erroneous values for its attributes
Deployment Order:
100
Specifies the loading order of the resource at server startup. Lower numbers are loaded first.
Description:
Pool Settings
Initial and Minimum Pool Size:
8
Connections
Minimum and initial number of connections maintained in the pool
Maximum Pool Size:
32
Connections
Maximum number of connections that can be created to satisfy client requests
Pool Resize Quantity:
2
Connections
Number of connections to be removed when pool idle timeout expires
Idle Timeout:
300
Seconds
Maximum time that connection can remain idle in the pool
Max Wait Time:
60000
Milliseconds
Amount of time caller waits before connection timeout is sent
Transaction
Non Transactional Connections:
Enabled
Returns non-transactional connections
Transaction Isolation:
If unspecified, use default level for JDBC Driver
Isolation Level:
Guaranteed
All connections use same isolation level; requires Transaction Isolation
Save
Cancel


General
Advanced
Additional Properties
Edit JDBC Connection Pool Properties
Save
Cancel
Modify properties of an existing JDBC connection pool.
Pool Name: WorkersPortal-Pool
Additional Properties (5)
Add Property
Delete Properties
Select
Name
Value
Description
DatabaseName
WorkersPortal
Password
User
wpuser
serverName
172.30.4.12
portNumber
3306

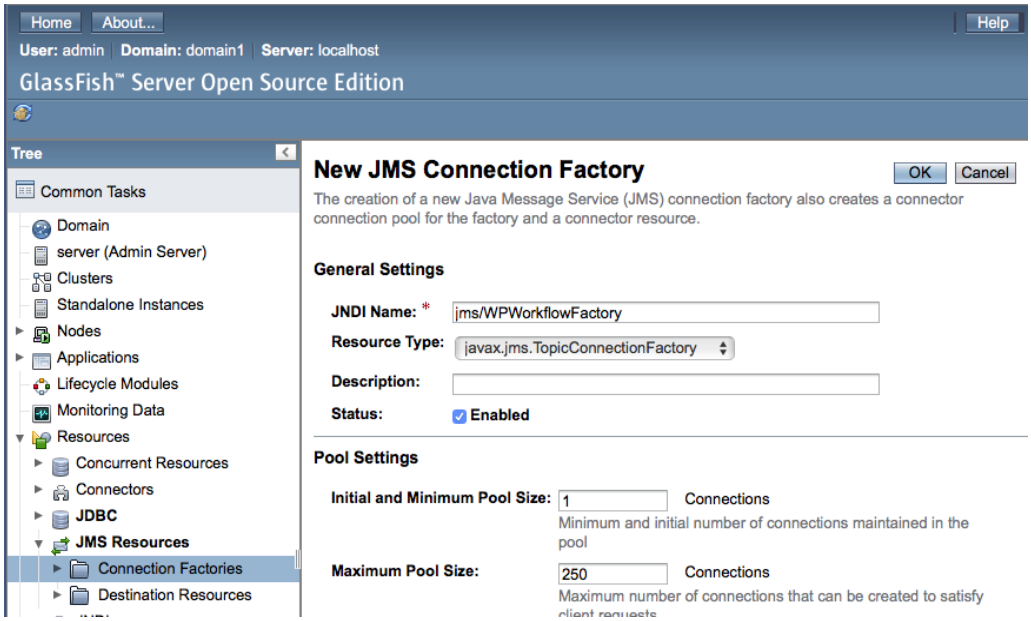
Luego vamos a Resources->JDBC->JDBC Resources y nuevamente hacemos click en New.

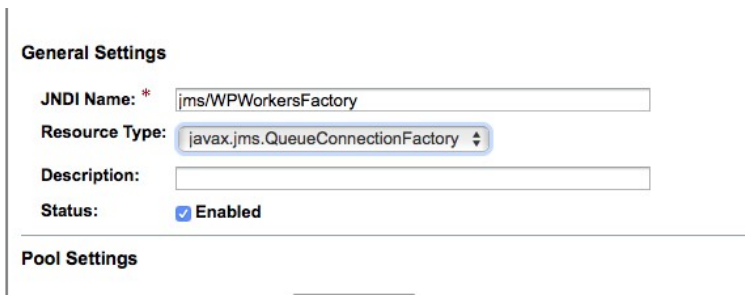
Como nombre JNDI hay que ponerle jdbc/WorkersPortal y asociarlo al *pool* anterior. Quedaría así:

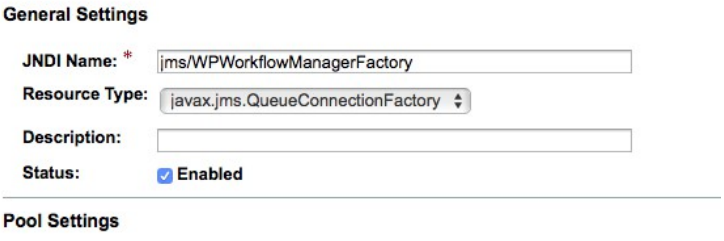


El siguiente paso es configurar las colas JMS. Vamos a

Resources->JMS Resources->Connection Factories New

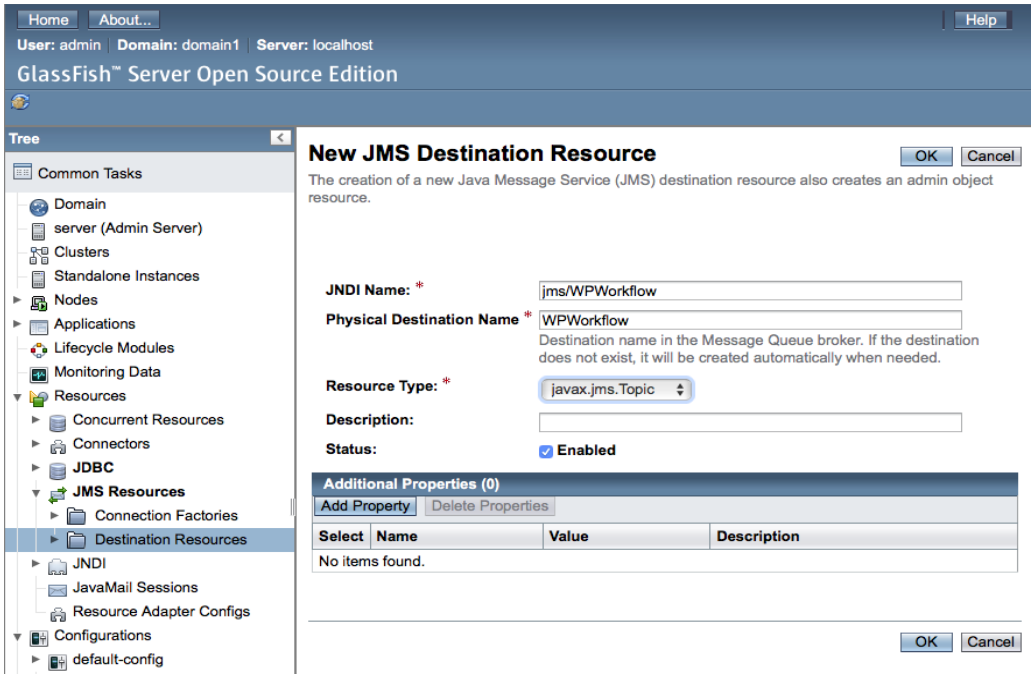


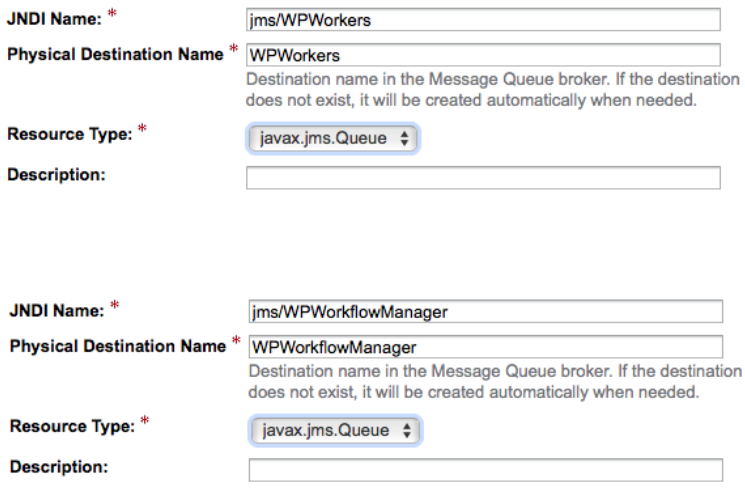




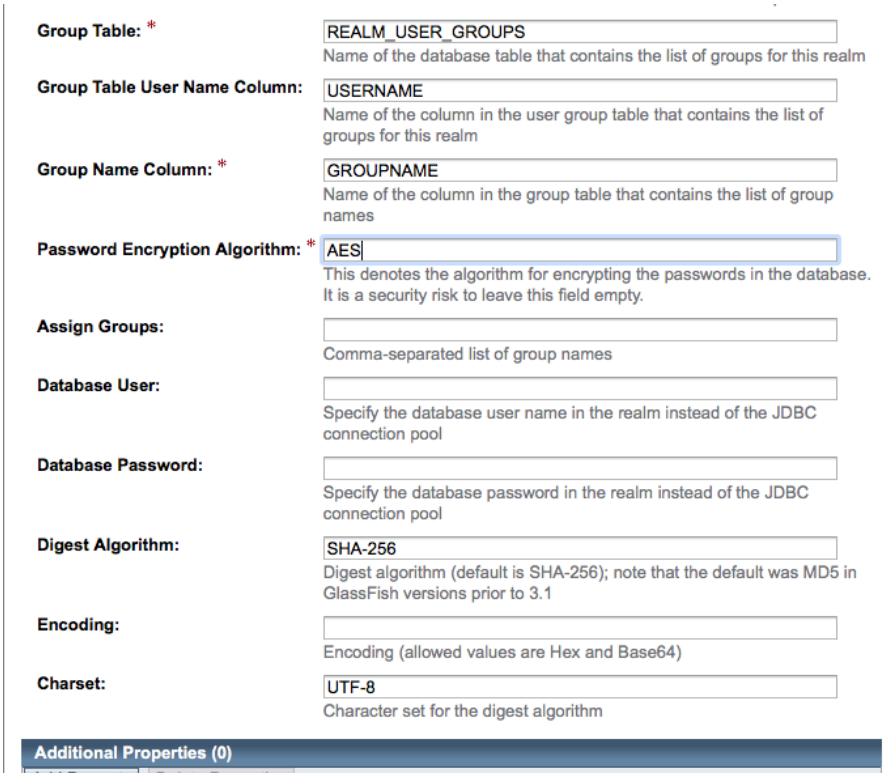
Y ahora las colas en sí, desde

Resources->JMS Resources->Destination Resources





Ya solo falta el Realm, Configurations->server-config->Security->Realms



Ahora ya podemos desplegar la aplicación desde Applications->Deploy y subimos el fichero .ear

Dejamos todos los datos por defecto y damos a ok.

En la primera ejecución, la propia capa de persistencia creará las tablas necesarias. Pero antes de poder validarnos, tendremos que crear la siguiente vista:

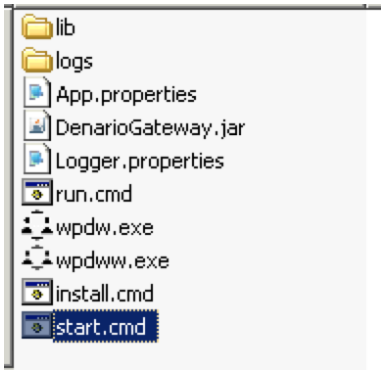
```
CREATE VIEW `REALM_USER_GROUPS` AS
SELECT `u`.`NAME` AS `USERNAME`, `g`.`NAME` AS `GROUPNAME`
FROM `GROUPS_HAS_USERS` AS `ghu`
LEFT JOIN `USERS` AS `u` ON `ghu`.`USER_ID` = `u`.`ID`
LEFT JOIN `GROUPS` AS `g` ON `ghu`.`GROUP_ID` = `g`.`ID`
```

Agente de RRHH

Lo primero será crear la vista de la base de datos que utiliza el programa Denario. Como en estos momentos está en un Ms. SQL Server, se tendría que ejecutar las siguientes instrucciones:

```
USE [Denario]
GO
CREATE SCHEMA wpgw
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE VIEW [wpgw].[Employees]
AS
SELECT TOP (100) PERCENT { fn CONCAT(e.DAIDENIDENTIFICADOR,
e.DADCIDENTIFICADOR) } AS NIF,
        c.FECONALEMP AS REGDATE,
        c.FECONBAEMP AS DISDATE,
        e.DCIDENNOMBRE AS NAME,
        e.DCIDENAPELLIDO1 AS SURNAME1,
        e.DCIDENAPELLIDO2 AS SURNAME2,
        e.FEIDENNACIMIENTO AS BIRTHDATE,
        e.CLIDENSEXO AS SEX,
        e.NUIDENTELEFONO3 AS EMAIL,
        p.DCPUESTOTRABAJO AS JOB
FROM c2p.TIDENCONTRATOS AS c
LEFT OUTER JOIN c2p.TIDENTIFICATIVOS AS e
        ON e.COIDENNUMERO = c.COIDENNUMERO AND e.COEMPRESA = c.COEMPRESA
LEFT OUTER JOIN c2p.TPUESTOTRABAJO AS p
        ON p.COPUESTOTRABAJO = c.COPUESTOTRABAJO AND p.COEMPRESA = c.COEMPRESA
WHERE (c.FECONBAEMP IS NULL) AND (c.COEMPRESA = 'CMD')
ORDER BY REGDATE
GO
```

Ahora copiamos los archivos que conforman el agente a una carpeta, por ejemplo c:\denariogw, quedando así:



Para instalar el agente como servicio se tendría que ejecutar la siguiente orden desde la consola de comandos, o ejecutar el script install.cmd:

```
wpdw.exe install  --DisplayName "Workers Portal Gateway" --Description "Denario
to workers portal gateway" --Startup manual --Classpath=%CLASSPATH
%;DenarioGateway.jar --StartMode jvm --StartClass com.delfos.denariogateway.Main
--StartMethod windowsService --StartParams start --StopMode jvm --StopClass
com.delfos.denariogateway.Main --StopMethod windowsService --StopParams stop
--LogPath C:\denariogw\log --LogPrefix wpdw --LogLevel Debug --StdOutput auto
--StdError auto
```

Y para iniciar el servicio se puede ejecutar, por ejemplo, desde el administrador de servicios de windows o utilizando el script start.cmd. Si se prefiere ejecutar como una aplicación de consola, para observar mas fácilmente los mensajes de salida, hay que ejecutar el script run.cmd.

10.2 Ejemplo de ejecución

En el video de presentación, se encuentra un ejemplo de la ejecución del programa.

Se pueden descargar los videos desde los siguientes enlaces:

- Presentación [1080p]
- Presentación [540p]