Escuela de
Ingeniería y Arquitectura
Universidad Zaragoza

The University of
Nottingham

An approach for distributed rostering by means of Web service technologies

Proyecto Fin de Carrera
Ingeniería Informática

Autor
Miguel López Chaboy

Director
Dario Landa Silva

Optimization and heuristic search
Computer Science
Nottingham University

Ponente
Francisco Javier Fabra Caro

Grupo de Integración de Sistemas Distribuidos y Heterogéneos (GIDHE)
Departamento de Informática e Ingeniería de Sistemas
Escuela de Ingeniería y Arquitectura
Universidad de Zaragoza

Curso 2010/2011
Septiembre 2011

An approach for distributed rostering by means of Web service technologies

RESUMEN

La planificación automática de turnos de trabajo (*rostering* de aquí en adelante) es un proceso fundamental para la mayoría de empresas. Este proceso les permite gestionar sus recursos de forma eficiente. La mayoría de soluciones de planificación automática trabajan en escenarios centralizados.

Este proyecto tiene como meta dar un nuevo enfoque al proceso de planificación automática transformando el entorno centralizado en uno distribuido. La tecnología escogida para esta transformación son los servicios Web. Los servicios Web se han posicionado como una de las tecnologías más importantes y utilizadas, sin embargo la combinación de los procesos de planificación automática con servicios Web no ha sido investigada y estudiada en profundidad. Este nuevo enfoque pretende crear soluciones más ligeras y flexibles para empresas de forma que no tengan que preocuparse de tediosas instalaciones y configuraciones

El enfoque distribuido puede proveer numerosos beneficios a ambas partes; permitiría centrar este área tecnológica en un entorno web 2.0. Las compañías de planificación automática se actualizarían convirtiéndose en empresas con gran capacidad de interacción con el resto de elementos de la web 2.0. Al mismo tiempo, la empresa que consume los servicios de *rostering* no tiene que preocuparse por el mantenimiento y actualizaciones. Los calendarios de turnos de trabajo pueden ser obtenidos en cualquier lugar y en cualquier momento. Esto mejora el valor de la empresa y las relaciones con los empleados.

Para el desarrollo del proyecto se han creado diferentes aplicaciones. Estas aplicaciones han sido configuradas de distintas formas para abarcar distintas posibilidades de desarrollo y poder así obtener una visión más amplia de las capacidades y posibilidades del sistema distribuido. Las configuraciones están centradas en el tipo de servicio Web usado (SOAP, REST) y en la seguridad y sus diferentes posibilidades.

El proyecto es una prueba de concepto de como este nuevo enfoque funcionaría, resaltando los puntos fuertes y deficiencias de éste; de la misma manera pretende sentar una base para futuros proyectos en este área.

El proyecto ha conseguido cumplir con los objetivos propuestos, mostrando los puntos fuertes y débiles del enfoque. De forma sorprendente y contraria a las expectativas los problemas aparecen al obtener los datos necesarios de las diferentes empresas para crear el calendario. En principio, se esperaba que el cuello de botella del sistema se encontrase en el proceso de *rostering*, ya que normalmente es un proceso costoso y pesado. Los resultados del proyecto muestran como el cuello de botella está localizado cuando se intenta obtener la información de empresas externas. Este problema resalta la característica principal de este nuevo enfoque: es necesario un sistema personalizado para cada empresa que quiera consumir los servicios de *rostering*.

Esto significa que a pesar de tener un sistema global funcionando con servicios Web, no es válido para todas las empresas que quieran utilizarlo y consecuentemente hay que adaptarlo a cada una de ellas. La colaboración entre empresas se vuelve esencial en este nuevo enfoque. Esto complica la idea inicial y cambia los objetivos de las empresas de *rostering* que necesitan una profunda transformación de sus sistemas y su funcionamiento.

Este proyecto ha sido desarrollado en la Universidad de Nottingham durante el curso 2010 – 2011 durante el transcurso de una beca Erasmus.

An approach for distributed rostering by means of Web service technologies

SUMMARY

Rostering is a fundamental process for companies. This process allows them to manage their resources efficiently. The most of rostering solutions work in a standalone environment.

This project aims to give a new approach to the rostering process transforming the standalone environment in a distributed one. Web services are the chosen technology to transform the approach. Web services have been positioned as one of the most important and used technologies, however the combination of rostering and web services has not been investigated and deeply studied. This new approach pretends to create new lighter and flexible solutions for companies without complicated installations and configurations

.

The distributed approach could report several benefits to both parts; it will allow focusing this technological area into a web 2.0 approach. The rostering companies will update themselves into new companies full of interaction possibilities with the rest of the 2.0 world. At the same time, the enterprise that uses the rostering service does not have to worry for the maintenance and the updates any longer. Rosters can be obtained everywhere 24/7; this improves the value of the company and the relationships with the employees.

Different applications have been created for the project. These applications have been configured in different ways to give a wide vision and better understanding of the approach's performance. The configurations are focused in the type of web service used (SOAP, REST) and in the security and its different possibilities.

The project is a conceptual test of how this new approach would work. The project aims to highlight the strong and weak points of the approach and settle a foundation for future projects in this area.

The project successes showing the weak points of the approach. Surprisingly and contrary to expectations the problems appear retrieving the external company the necessary data for the roster. The expectations were that the system's bottleneck was into the rostering process which usually is a heavy load for the system. The results of the project show how the bottleneck is located when external enterprise's data is being retrieved. This highlights the fundamental issue of this new approach: a customized system for each company with the same global system for all of them. This means that despite having a global system working with web services, this is not valid for all the companies hence it has to be modified and customized for each company. Collaboration between companies becomes essential with this new approach. This complicates the initial idea and change the focus of the rostering company, which needs to be transformed into a web company.

AGRADECIMIENTOS

A Darío. Javier y Arturo por sus consejos, apoyo y conocimiento

A Carmen, Laura, Miguel, Jesús y Marta por su ayuda en la corrección.

The Straight Edge

INDICE

LISTA DE FIGURAS

GLOSARIO

CORBA: Common Object Request Broker Architecture
DCOM: Distributed Component Object Model
EDI: Electronic Data Interchange
BSE: Bus de servicios de empresa
GUI: Graphical User Interface
HTTPS: HyperText Transfer Protocol Secure (Protocolo de transmision de datos seguro)
IDs: Identificadores
Jpeg: Joint Photographers Experts Groups (Formato de imagen)
JSP: JavaServer Pages. (Tecnología Java para la creación de páginas web dinámicas)
Kerberos: Protocolo de autentificación de redes.
Keystore: Repositorio de certificados de seguridad
OASIS: Organization for the Advancement of Structured Information Standards
REST: Representational State Transfer.
RPC: Remote Procedure Calling.
Rostering: Proceso de asignación automática de recursos sujeto a restricciones.
Roster: Un roster es una lista de personas que muestra donde y cuando van éstas a trabajar. Un roster puede ser creado para diferentes periodos de tiempo: días, semanas o meses.
SaaS: Software as a Service
SAML: Security Assertion MarkUp Language
SCA: Service Component Architecture
SDO: Service Data Objects
Servlet: Tipo de clase de Java usada para trabajar con servidores de aplicaciones.
SOA: Services oriented architecture
SOAP: Simple Object Access Protocol
SSL: Secure Sockets Layer
TLS: Transport Layer Security
Token: Elemento que contiene información para la seguridad del sistema y que es intercambiado durante el proceso de comunicación entre las diferentes partes.
Truststore: Repositorio de certificados externos confiables.
UDDI: Universal Description Discovery and Integration
URL: Uniform Resource Locator
XML: eXtensible Markup Languange
WSDL: *Web Service Descriptor Language*
W3C: World Wide Web Consortium

1. INTRODUCCIÓN

Esta sección introduce el contexto, la motivación y los objetivos de este PFC. Se abordarán los conceptos de SOA, planificación automática y servicios Web. Finalmente se presenta la estructura de esta memoria.

1.1. Contexto del PFC

MidlandHR es una empresa inglesa fundada en 1984 que se dedica al desarrollo de software centrado en el pago automático de nóminas y tareas de gestión de recursos humanos. Esta empresa ha expandido su catálogo de productos y servicios, convirtiéndose rápidamente en un proveedor líder en este sector. El catálogo de soluciones de gestión de recursos humanos de la compañía permite a las organizaciones incorporar todo tipo de actividades -desde el pago de nóminas hasta el control de personal- en un mismo sistema.

En los últimos 5 años, MidlandHR ha trabajado de manera cercana con sus clientes para mejorar su trabajo en la gestión de personal. La gestión de personal se expande hacia áreas como la asignación y planificación de turnos de trabajo. En este momento la empresa está centrada en una de las funcionalidades de sus sistemas: la asignación automática de personal (ASA). MidlandHR ha desarrollado un prototipo para probar esta funcionalidad entre sus clientes.

El prototipo ha sido entregado a los clientes para evaluar su funcionamiento y poder mejorar su desarrollo. Sin embargo, la empresa está teniendo problemas a la hora de obtener datos de los clientes que han usado el prototipo. Esto se debe, principalmente, a la carencia de experiencia a la hora de configurar y usar el prototipo, por lo que normalmente introducen datos incorrectos. Esto hace que la opinión de los clientes sobre el prototipo sea bastante pobre y no puedan contribuir de manera adecuada al desarrollo del producto.

MidlandHR está considerando transformar el actual enfoque de su prototipo, basado en la instalación de este en cada uno de los equipos que lo van a utilizar, hacia un enfoque basado en servicios *(SaaS – Software as a Service*). Este nuevo enfoque permitiría que el prototipo se alojase en los servidores de la compañía de tal manera que los clientes accedan a él a través de un navegador Web. También facilitaría el desarrollo del prototipo ya que la propia empresa se encargaría de la configuración y mantenimiento de éste. Los clientes se verían beneficiados ya que aumentaría su facilidad de uso y manejo del sistema siendo más fácil para ellos introducir los datos. [1]

El hecho de cambiar el enfoque del prototipo hacia un enfoque *SaaS* ha hecho que la empresa considere las arquitecturas orientadas a servicios (SOA) como un posible marco bajo el que desarrollar sus productos. Las arquitecturas orientadas a servicios (SOA) se basan en la interacción entre servicios para crear la lógica necesaria para el desarrollo de aplicaciones orientadas a servicios de manera sencilla. SOA puede ser visto como una forma de diseñar sistemas software para proveer servicios a aplicaciones de usuario o a otros servicios distribuidos en la red a través de interfaces descubribles. Dentro de las arquitecturas SOA aparece la tecnología de los servicios Web. Esta tecnología cuenta con un conjunto de características que cubren los principios SOA y permite de la implementación de este tipo de arquitecturas de manera sencilla y directa [2]

A través de los servicios Web, la transformación del prototipo mencionado anteriormente en una versión distribuida del mismo permitiría a la empresa solucionar los problemas con sus clientes y a su vez estudiar la combinación de estas dos tecnologías (asignación de turnos de trabajo y servicios Web).

1.2. Planificación de turnos de trabajo

En este proyecto los servicios Web son utilizados para obtener calendarios de turnos de trabajo (de aquí en adelante *rosters)* para un centro de ocio. El proyecto no se centra en cómo estos *rosters* son creados y calculados, se centra en la combinación de ambos sistemas. En la actualidad, esta combinación no ha sido estudiada en profundidad y plantea numerosos beneficios. Sin embargo, podría haber sido

utilizada otra tecnología en vez de un servicio obtención de turnos de trabajo (*rostering* de aquí en adelante), ya que lo que también se busca estudiar es la adaptabilidad de los servicios Web a diferentes tipos de problemas.

No obstante, antes de centrarse en el sistema y su diseño es necesario tener cierto grado de entendimiento sobre los sistemas de *rostering* y programación automática para así poder obtener y manejar soluciones óptimas.

Una de las posibles definiciones de programación automática sería:

> "La asignación de recursos, sujeta a limitaciones, a objetos situados en el espacio-tiempo, para lo que el coste total sea minimizado." [3]

Una posible definición de *rostering* podría ser:

> "La colocación de recursos, sujeta a limitaciones, en huecos siguiendo un determinado patrón" [4]

El proceso de *rostering* genera *rosters*, listas de personas que muestran dónde y cuándo van éstas a trabajar. Un *roster* puede ser creado para diferentes periodos de tiempo: días, semanas o meses.

El elemento principal de estos problemas es el turno de trabajo. Un turno de trabajo es el periodo fijo de tiempo en el que una tarea es ejecutada. El término "programación automática" hace referencia al proceso de asignar la realización de tareas a de los empleados, en turnos de trabajo.

Los problemas de programación automática de turnos de trabajo pueden formularse de muy diversas formas: pueden basarse en patrones predefinidos de cargas de trabajo, ser simples problemas de asignación o pueden ser problemas que incluyan objetivos y restricciones complejas.

Los problemas de programación de turnos de trabajo aparecen en un amplio rango de empresas y sectores. Estas empresas incluyen compañías aéreas, aeropuertos, fuerzas armadas, centros de atención telefónica, servicios de emergencias, fábricas, atención sanitaria, ventas al por menor, personal de seguridad, sector del transporte... [5]

Estos problemas suelen involucrar a una organización con un conjunto de tareas que necesitan ser realizadas por un grupo de empleados, con sus propias cualificaciones, restricciones y preferencias. La organización tiene que cumplir ciertas normas globales e intenta alcanzar objetivos tales como: reducir el coste global o divisiones equitativas de la carga de trabajo de sus trabajadores [6].

Existe una extensa literatura sobre programación automática de turnos de trabajo. Teorías relacionadas con este tema han sido estudiadas durante muchos años. Éste es un proceso muy complejo y, a día de hoy, existen numerosos problemas sin resolver. Normalmente la programación automática lleva a problemas de tipo NP.

Esta complejidad hace que el puente entre la teoría y su aplicación sea difícil de superar. Obtener la solución óptima para ciertos problemas de forma que puedan ser comercializados requiere normalmente mucho tiempo.

En el mundo comercial, un alto grado de satisfacción laboral es un factor fundamental para el éxito de una compañía. Esto implica la consideración de restricciones por parte de los trabajadores en el modelo del problema. Añadir más flexibilidad a los empleados genera restricciones en el problema que hacen que este sea más difícil de resolver. Este problema ha atraído la atención de investigadores. Las investigaciones en este ámbito señalan como solución general a métodos de búsqueda locales. Este tipo de métodos han demostrado funcionar adecuadamente incluso en los problemas más complicados.

Las soluciones de software para este tipo de problemas automatizan el proceso de crear y mantener un roster y se han convertido en una parte esencial en el mundo de los negocios. Este software es complejo y permite a la dirección incluir diferentes factores como preferencias de los trabajadores, bajas

por enfermedad, vacaciones o conflictos entre trabajadores. La funcionalidad de este software va más allá de la creación de *rosters* y también permite a la dirección recibir diferentes resultados como respuesta, análisis de la productividad y también pueden ser usados para automatizar el pago de nóminas.

A día de hoy, la mayoría de compañías están buscando una actualización para este tipo de tareas acorde a la evolución Web que está siguiendo el mundo de los negocios. Las empresas saben que es un punto fundamental de su organización estructural y que de ello depende el correcto funcionamiento de la compañía.

1.3. SOA y Servicios Web

Los términos presentados en esta subsección se encuentran ampliamente definidos y detallados en la sección 2 del Anexo I.

1.3.1. SOA

SOA puede ser visto como una forma de diseñar software para proveer servicios a aplicaciones de usuario o a otros servicios distribuidos en la red. [2]

A día de hoy, el desarrollo de un nuevo servicio puede necesitar otros servicios, es difícil desarrollar un sistema global sin ninguna necesidad externa. El uso de servicios probados y evaluados lo hace más fácil y accesible. SOA provee una arquitectura para desarrollar servicios de forma que puedan ser fácilmente enlazados entre ellos

La principal meta de SOA es obtener una interoperabilidad intrínseca a todo tipo de servicio más que modelos específicos para cada comunicación entre servicios. Esta visión global proporciona flexibilidad al modelo que evoluciona bajo una mejora continua. Existen tres conceptos básicos en SOA: visibilidad, interacción y efecto.

La visibilidad permite a los diferentes servicios ser descubiertos por otros servicios y poder así ser consumidos. La interacción entre servicios se realiza a través del intercambio de mensajes. Una vez que el servicio ha sido encontrado y descubierto comienza la interacción entre ellos. SOA busca hacer este proceso lo más simple posible de manera que se adapte al medio y condiciones en las que se da.
El efecto es el resultado que tiene la interacción, puede ser un mensaje de respuesta al servicio o al usuario. Puede cambiar el estado de las entidades participantes o generar nueva información.

SOA ha tenido una gran aceptación. Sin embargo, como todo nuevo paradigma su implementación ha tenido cierta controversia.

Entre las ventajas de esta arquitectura encontramos: interoperabilidad entre aplicaciones, ya sean internas, externas, estén desarrolladas o en proceso de desarrollo, reusabilidad de los servicios existentes, reducción del tiempo de desarrollo y de los costes de mantenimiento, aumento de la visibilidad de los negocios de la empresa.

Por el contrario, SOA ha sido criticado duramente debido a su velocidad de interacción entre servicios y su dependencia del BSE. La implementación de SOA en una empresa requiere un gran esfuerzo que no todas las empresas pueden llevar a cabo por lo que se requiere un estudio exhaustivo sobre la viabilidad de su implementación en el escenario en que se sitúa la empresa.

1.3.2. Servicios Web

Existe un importante número de definiciones para el término "servicios Web" lo que muestra la complejidad de este término. Una posible y simple definición sería referirse a ellos como un conjunto de aplicaciones o tecnologías con la capacidad de operar en la Web. Estas aplicaciones y tecnologías intercambian datos entre ellas para así ofrecer servicios. El proveedor ofrece sus servicios como si fuesen procedimientos remotos y los usuarios realizan llamadas a estos servicios para obtener sus resultados.

Los servicios Web fueron creados debido a la necesidad de estandarizar la comunicación entre diferentes plataformas y lenguajes de programación (PHP, C#, Java, etc.). Antes que los servicios Web diferentes plataformas fueron desarrolladas para conseguir estos objetivos. EDI, DCOM y CORBA son las más importantes. Sin embargo, no consiguieron lograr una interoperabilidad global debido a los diferentes proveedores y tecnologías utilizadas. El protocolo principal para estas plataformas fue RPC, el cual es bastante útil en redes controladas, pero es difícil de controlar y manejar en Internet [7].

En 1999, la necesidad de un nuevo estándar global fue reconocida mundialmente y la W3C y OASIS comenzaron a desarrollar esta nueva tecnología.

Los servicios Web presentan una serie de ventajas sobre otras tecnologías, las principales son: Interoperabilidad entre servicios implementados en diferentes lenguajes o ejecutados en plataformas distintas, usabilidad y adaptabilidad de los servicios a las características concretas de cada compañía, reusabilidad y facilidad para adaptar servicios existentes a otros nuevos.

Por el contrario el uso de estas tecnologías presenta algunas desventajas relacionadas con el rendimiento y su capacidad para realizar transacciones.

Funcionamiento de los servicios Web.

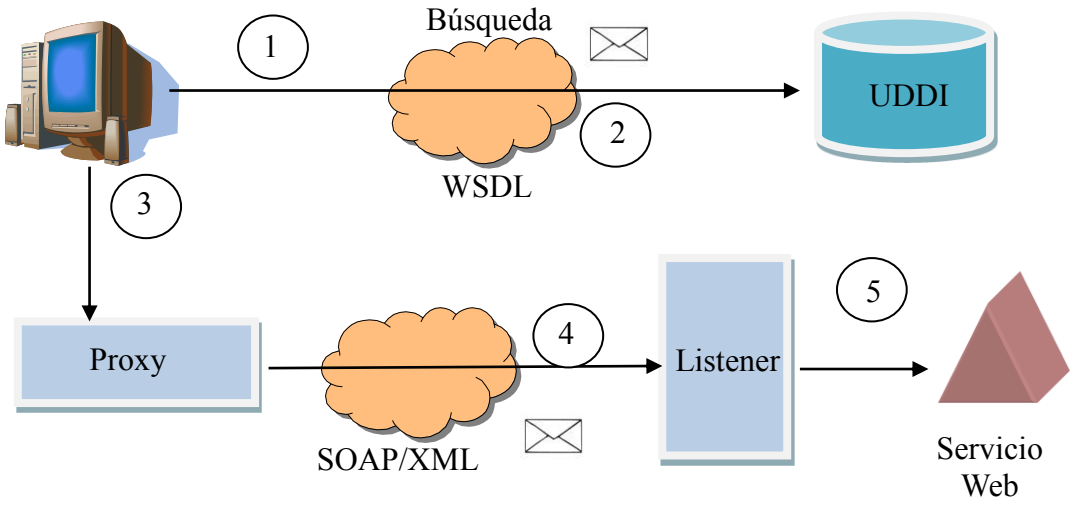


Figura 1 – Esquema de funcionamiento de un servicio Web

El funcionamiento de los servicios Web puede ser resumido en los siguientes pasos.

1. Un sistema necesita encontrar un servicio Web para realizar una determinada tarea. Para encontrarlo el sistema busca en el directorio UDDI (*Universal Description Discovery and Integration* [8]) un servicio que encaje con sus necesidades.
2. Una vez que el servicio ha sido localizado y seleccionado, el documento WSDL (*Web Service Description Language* [9]) del servicio es enviado al cliente. Ahora el cliente sabe cómo tiene que tratar con el servicio y como estructurar los datos.
3. El cliente prepara los datos y los envía a través de la red. Los datos son enviados en formato XML en un paquete SOAP.
4. El servicio Web recibe paquete y comprueba si los datos son correctos. Si los datos concuerdan con el documento WSDL, el paquete es procesado.
5. El servicio Web realiza sus acciones y devuelve el resultado (si lo hay) al cliente en otro paquete SOAP. Para realizar las acciones propias del servicio Web, éste podría contactar con otros servicios Web a través de la red.

Un escenario similar podría darse sin el directorio UDDI. Si esto ocurre, el cliente tiene que conocer previamente donde está el servicio Web que quiere consumir y donde está el documento WSDL del servicio. La figura uno muestra los pasos señalados anteriormente para el consumo de un servicio Web.

Seguridad

Los servicios Web SOAP pueden añadir una capa de seguridad a sus mensajes. Esto puede ser realizado con una de las múltiples extensiones que SOAP provee. La extensión para la seguridad se llama WS-Security. Esta extensión contiene especificaciones para garantizar la integridad y la seguridad en los mensajes. Este protocolo incluye soporte para usar SAML y *Kerberos*, de la misma manera puede gestionar certificados X.509.

WS-Security incluye la información relacionada con la seguridad en la cabecera SOAP, trabajando en la capa de aplicación, lo que permite seguridad de extremo a extremo.

La integridad y la confidencialidad de los datos pueden ser garantizadas con el uso de la capa de transporte seguro (TLS), enviando mensajes sobre HTTPS. Esto podría reducir la sobrecarga de los mensajes. Una explicación más detallada de la seguridad en servicios Web y en este proyecto se encuentra en el anexo IX.

1.4. Solución propuesta

La solución propuesta consiste en crear un escenario orientado a servicios donde el rendimiento de los servicios Web de *rostering* pueda ser medido y probado. Este escenario es una transformación directa de la habitual aplicación de *rostering* centralizada a un enfoque orientado a servicios.

Esto permitirá un primer análisis de la utilidad de este enfoque. En el escenario a desarrollar, MidlandHR provee soluciones de *rostering* a través de servicios Web y otra empresa consume estos servicios para obtener calendarios de turnos de trabajo para sus empleados. En este proyecto, esta segunda empresa es un centro de ocio.

En el proyecto se diseñará tanto la infraestructura del centro de ocio como la de la empresa de *rostering* y se procederá comunicarlas a través de servicios Web. Se creará una aplicación para el manejo de datos de la empresa y la obtención de calendarios de trabajo así como una página Web que proporcionará la obtención de estos calendarios a los trabajadores fuera de su entorno de trabajo.

El enfoque distribuido busca probar la viabilidad de las nuevas formas de hacer negocios a través de internet en esta área. La empresa que desea el servicio de *rostering* solo tiene que contactar con la empresa que proporciona estos servicios y proporcionarle los detalles de su personal. En el caso más sencillo, una URL será comunicada a esta empresa, que solo tendrá que consultar esa dirección y seguir unos sencillos pasos para obtener los *rosters*. En este enfoque las actualizaciones y cambios ocurren en la empresa de *rostering*, no en el cliente. Aparece un gran número de posibilidades, las nuevas tecnologías Web permiten crear escenarios interactivos donde la mayoría de las soluciones informáticas usadas por las empresas pueden ser integradas y resueltas. Un portal Web podría reunir todos los servicios contratados por las empresas, centralizando la información y facilitando su acceso. Los usuarios y empleados podrían acceder a esta información en cualquier parte y a cualquier hora.

1.5. Objetivos del PFC

La meta principal de este proyecto es explorar como los servicios Web y las tecnologías distribuidas pueden ayudar en la situación planteada. Hay diferentes formas de conseguir esta meta; este proyecto se centra en realizar una prueba conceptual para comprobar como el paradigma de orientación a servicios encaja con el proceso de *rostering*.

Este nuevo enfoque puede proporcionar numerosos benéficos a las empresas de *rostering*. Hasta ahora, cuando una compañía necesita una solución *rostering*, ésta es en la mayoría de los casos, una suite fija con un elevado precio. Además ésta necesita ser instalada por expertos y el mantenimiento es complicado. El enfoque centralizado limita el potencial de este tipo de aplicaciones. En la actualidad, las empresas no quieren problemas relacionados con la tecnología de la información, solo quieren pagar y obtener un servicio. Las empresas no quieren un proceso de instalación, continuas actualizaciones y un servicio técnico ineficaz.

MidlandHR ha fijado los siguientes objetivos en este proyecto:

- Cambiar la arquitectura del sistema para que su funcionalidad pueda ser accedida a través de Internet.
- Registrar a cada usuario (cliente) en el sistema.
- Crear un fichero de log para cada usuario que al menos debe incluir:
    - Número de ejecuciones
    - Información sobre el escenario solicitado (persona, ausencias...)
    - Información sobre el éxito al obtener la solución.
- Implementar el sistema usando SOAP y REST.
- Al evaluar la solución basada en SOAP explorar los diferentes estándares de seguridad que pueden ser adoptados.
- Obtener medidas relacionas con el rendimiento:
    - Número de usuarios simultáneos
    - Memoria requerida en el lado del servidor.
    - Tiempo requerido en media para obtener una solución.
- Evaluar una posible implementación descentralizada mediante el modelo de coordinación Linda.

A lo largo del proyecto todos estos objetivos han sido alcanzados y su solución se presenta en los diferentes capítulos de este documento. El éxito en la distribución del prototipo ASA sobre servicios Web haría que MidlandHR considerase la implantación de este nuevo sistema entre sus productos.

1.6. Organización de la memoria

La memoria está organizada en diferentes secciones. La segunda sección muestra el estado del arte de las diferentes tecnologías empleadas en el PFC. La tercera sección plantea el diseño del sistema y sus diferentes partes; los detalles más importantes de la implementación del sistema son explicados en la sección cuarta. La evaluación del sistema y sus resultados son presentados en la sección quinta. Finalmente en la sección sexta se explican las conclusiones del proyecto y varias posibilidades de ampliación de éste.

A lo largo de la memoria se hace referencia a distintos anexos que amplían la información señalada en la memoria.

2. ESTADO DEL ARTE

En esta sección se presenta la situación actual de las tecnologías utilizadas en el proyecto. El proyecto se centra en las tecnologías de *rostering* y servicios Web. Sin embargo, la seguridad tiene un papel importante en el desarrollo del mismo por lo que se ha documentado el estado actual de la misma.

2.1. Rostering

En el capítulo anterior se ha visto la importancia del software de *rostering* para las empresas. Pese a la importancia de este software no existe una solución líder en el mercado. Esto se debe a la variabilidad de las características de cada empresa. Es muy difícil crear un software único que se pueda adaptar a un gran número de empresas con pocos cambios.

Existen numerosas empresas y software [10] que proponen una serie de soluciones centrada determinadas características o escenarios. Para empresas donde el *rostering* es un añadido para facilitar su labor este tipo de herramientas son adecuadas sin embargo, para empresas en las que el *rostering* es algo fundamental para la organización es necesaria la personalización de este tipo de herramientas.

Además los sistemas de *rostering* evolucionan y van añadiendo nuevas funcionalidades continuamente. El aumento de funcionalidades del software de *rostering* hace que los diferentes sistemas para dar soluciones a distintos ámbitos del mundo empresarial se estén fusionando en complejas soluciones comerciales.

La mayoría de empresas de este sector se centran en desarrollar algoritmos de *rostering* propios que luego se encargan de adaptar y personalizar a sus clientes, ya sea individualmente o centrándose en un determinado sector.

2.2. Servicios Web

Este proyecto se ha limitado a desarrollar una serie de aplicaciones Web y servicios Web y ejecutarlos en un servidor. Esto es suficiente para una prueba de concepto y para estudiar cómo estas aplicaciones encajarían en la empresa. La meta del proyecto no es desatollar una solución profesional; es estudiar como la programación automática de turnos de trabajo y los servicios Web pueden trabajar juntos.

Por lo tanto, el primer paso para alcanzar nuestra meta es crear un escenario donde los servicios Web funcionen y sea posible probar y medir el rendimiento de estos. Si los test y el sistema son satisfactorios el siguiente paso sería usar una de las plataformas que se presentan a continuación para implementar el sistema en un escenario profesional.

El desarrollo del proyecto sin usar ninguna de estas herramientas profesionales para servicios Web nos dará una buena perspectiva y entendimiento de como ambas tecnologías trabajan juntas. Si los resultados son satisfactorios, el uso de una de las herramientas propuestas siempre será beneficioso para el desarrollo. Hay numerosas que podrían servir para este fin. Estas plataformas pueden dividirse entre soluciones abiertas o soluciones privativas.

Paquetes privativos:

HydraSCA [11]
Es un *framework* centrado en el desarrollo de servicios Web en C++. Este software da especial importancia a XML, SOAP y WSDL, generando automáticamente esqueletos para la descripción de los servicios Web. Su característica principal es la capacidad de incorporar nuevos estándares o requisitos de integración sin alterar el resto de la aplicación.

IBM Pack for SOA [12]
Este software incluye : WebSphere Portal Server, WebSphere Portlet Factory, Lotus ActiveInsight, y Lotus Forms Server. La combinación de estos productos permite al usuario trabajar dentro del paradigma orientado a servicios centrado en los propios servicios y no en su implementación. El paquete permite al usuario separar las diferentes capas de la lógica de negocios SOA y adaptar los servicios existentes.

Oracle SOA Suite [13]
Esta suite integra los productos necesarios para diseñar, desarrollar, ensamblar y gestionar servicios Web. La suite trabaja de forma 100% basada en estándares y puede operar con servicios ya existentes. Esta suite incluye Oracle Service Bus, que da al usuario alta escalabilidad. Las principales características de este suite son: un editor integrado y basado en SCA (pinchar y arrastrar), servicios y eventos unificados y seguimiento de extremo a extremo lo que permite al usuario una total visibilidad de la implementación.

Paquetes Open Source.

Apache Tuscany [14]
La principal característica es simplificar el desarrollo de las soluciones SOA. El paquete provee un modelo para crear lo siguiente: aplicaciones compuestas de manera sencilla, servicios reusables para la capa de negocios, aplicaciones fácilmente adaptables a través de *pluggable bindings* y una arquitectura modular para integrar las diferentes tecnologías. El paquete permite componentes en Java, C++, BPEL, Spring y scripting.

Fabric3 [15]
Esta suite es la primera en cumplir con SCA Assembly 1.1 y ha sido desarrollada por uno de los creadores de SCA. Esta provee al usuario con características que normalmente solo se encuentran en productos comerciales y privativos. La principal característica de Fabric3 es su capacidad para trabajar con servicios reusables generando otros servicios que pueden ser usados y así continuamente. Fabric3 crea servicios modulares que pueden ser ejecutados en diferentes middlewares sin cambios importantes.

SCOrWare [16]
Este proyecto está muy centrado en los estándares y su cumplimiento. Este paquete provee una una herramienta de asistencia en el diseño y desarrollo. Esto soporta un generador de adaptadores para diferentes protocolos, un servicios de transacciones, un servicio de intercambio semántico de componentes SCA. Todo el paquete está integrado en Eclipse.

A la hora de elegir una plataforma las características a las que hay que prestar más atención son:

- Soporte de los estándares SCA (*Service Component Architecture* [17]), SDO (*Service Description Object* [18]) y sus especificaciones.

- Habilidad para tratar con diferentes tipos de adaptadores, componentes y tecnologías

- Bajos tiempos de ejecución.

- Compatibilidad con otras herramientas que pueden ser útiles en la implementación de un servicios Web como Apache Tomcat y Eclipse STP/SCA Composite Designer

2.3. Seguridad

La seguridad es un elemento clave en cualquier comunicación a través de internet, consecuentemente los servicios Web utilizados en el proyecto estarán sujetos a diversas políticas de seguridad.

La seguridad en servicios Web puede ser aplicada de formas diferentes. WS-Security [19] es un protocolo de comunicaciones que proporciona un medio para aplicar seguridad a los servicios Web. Este protocolo ha sido estandarizado por OASIS y se encuentra en su versión 1.1 desde el año 2006.

WS-Security proporciona confidencialidad e integridad a los mensajes intercambiados a través de los servicios. Sobre WS-Security se sitúa WS-Policy [20] que define un modelo genérico y una sintaxis que permite al desarrollador describir y comunicar las políticas de los servicios Web. WS-Policy permite tanto al desarrollador del servicio como al consumidor especificar las políticas relativas a seguridad y calidad del servicio. WS-Policy es una recomendación de la W3C desde el año 2007 y se encuentra en su versión 1.5.

En la actualidad, en un intento de facilitar la composición de servicios Web, OASIS trabaja sobre un *framework* llamado WS-CAF [21] (2006) que busca facilitar esta tarea.WS-CAF busca la creación de formas estándar e interoperables para: coordinación de las actividades de los servicios Web, propagar y coordinar información del contexto, notificar cambios en una actividad y definir las relaciones entre las diferentes partes.

Dentro de este marco de trabajo se encuentra WS-Context [22], que permite a los desarrolladores compartir un contexto común entre las interacciones de servicios Web haciendo más fácil la aplicación de políticas compartidas de seguridad.

WS-Context proporciona un mecanismo a los servicios Web para compartir estados persistentes, lo cual es necesario para soportar la coordinación de transacciones y elementos como *IDs* o *tokens* [23]. WS-Context permite compartir elementos de seguridad (*tokens*) dentro de una misma sesión, de forma que no sea necesario el intercambio de *tokens* con cada nuevo mensaje dentro de la misma sesión.

WS-Context está diseñado para ser compatible con WS-Security. WS-Context proporciona una estructura de contexto que esta típicamente ligada a la cabecera de los mensajes SOAP

Otra posibilidad para proveer seguridad a los servicios Web es la utilización de TLS enviando mensajes sobre HTTPS garantizado así integridad y confidencialidad.

2.4. Conclusiones

Existen numerosas y muy diversas formas de implementar servicios Web en el ámbito de la planificación de turnos de trabajo. Todas las plataformas presentadas, tanto las privativas como las libres, ofrecen distintas maneras de implementarlos. Los servicios Web permiten una nueva forma de acceso a información, por ello lo importante es saber qué tipo de información se está proporcionando y que plataforma permite generar esa información de manera eficiente; son el medio, no el fin.

Factores como accesibilidad y escalabilidad son importantes a la hora de elegir plataforma. En este caso se ha señalado que no existe una solución estándar en aplicaciones de *rostering*, lo cual hace que haya que realizar un análisis más detallado del software de *rostering* que se va a utilizar y de que plataforma beneficia su funcionamiento.

A estos dos factores hay que añadirles el factor de la seguridad que aunque se encuentra bastante estandarizado es necesario adaptarlo a las políticas de seguridad que utilicen las empresas que contraten el servicio de *rostering*.

Todo lo anterior muestra que en este ámbito es necesario realizar un análisis exhaustivo del estado del arte previo a cualquier desarrollo.

3. DISEÑO

Esta sección presenta el diseño del sistema. En primer lugar se va a dar una visión general de éste y a continuación se tratarán cada una de sus partes en detalle.

3.1. Diseño general

El sistema está dividido en tres partes. Cada una de estas partes se corresponde con las diferentes secciones del escenario real donde se desarrolla el sistema.

La figura 2 muestra las diferentes partes del sistema.

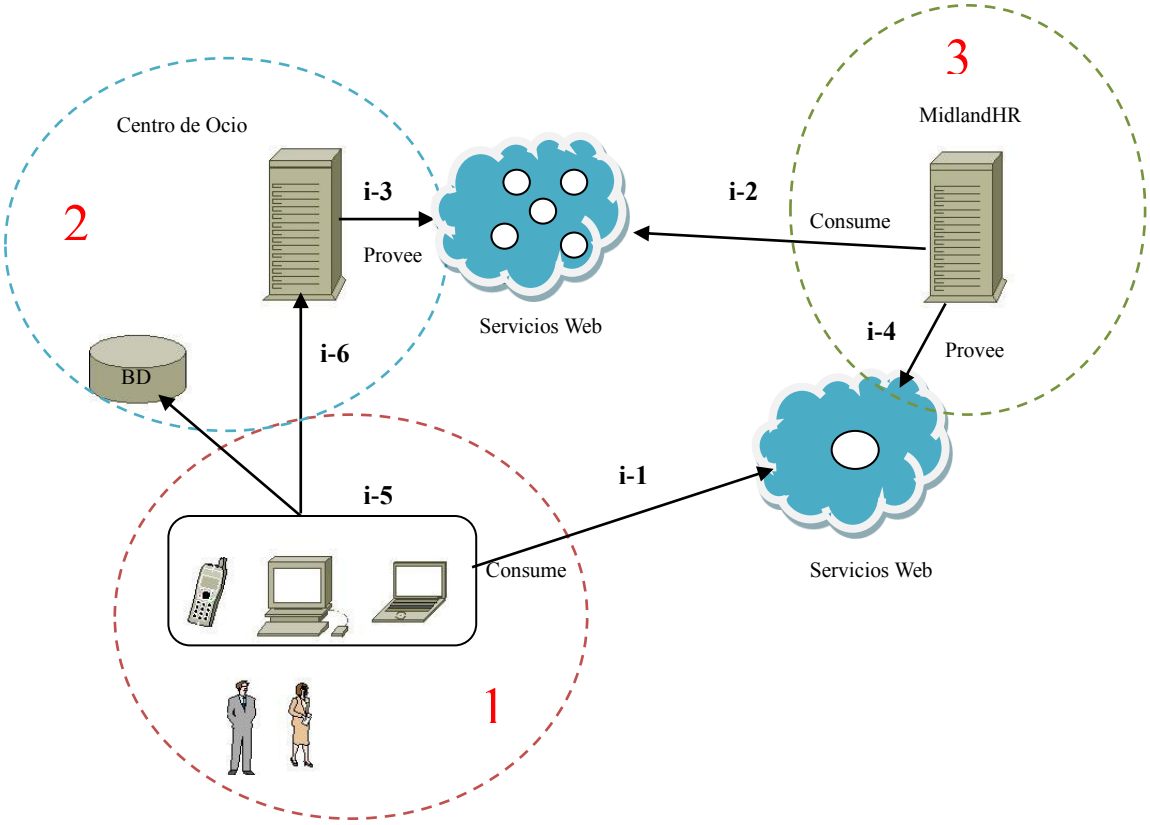


Figura 2 – Esquema General del sistema

La primera parte no tiene una localización fija. El *manager* y los empleados del centro de ocio usaran una aplicación de escritorio para obtener los *rosters*. El manager también podrá usar esta aplicación para añadir, modificar y borrar empleados del sistema y para gestionar la demanda de trabajo del centro. La aplicación puede ejecutarse en cualquier equipo, no necesita estar dentro de los límites del centro de ocio.

La aplicación se llama **RostApp** y se han creado dos versiones, una que trabaja con servicios Web SOAP y otra que trabajar con servicios REST. La versión SOAP tiene a su vez tres configuraciones diferentes dependiendo del nivel de seguridad con el que se trabaje (ver anexo III)

La segunda parte está situada en el centro de ocio. El centro de ocio cuenta con un servidor que almacena la base de datos de la empresa y en el que se ejecutan dos aplicaciones:

- **LeisureCenterWeb:** Esta aplicación Web provee una página Web al manager y empleados de la empresa para la obtención de un *roster*.

- **LeisureCenter:** Esta aplicación Web provee los datos del centro de ocio a diferentes empresas. En este proyecto los datos son provistos a MidlandHR para que esta pueda crear los *roster*. Los

datos son enviados a través de servicios Web. Existen dos versiones de esta aplicación: una con servicios Web SOAP y otra con servicios REST. La versión SOAP cuenta con diferentes configuraciones según el nivel de seguridad.

La tercera parte está situada en el servidor de la empresa MidlandHR. MidlandHR ofrece un servicio Web para la creación de *rosters*. Este servicio permite la posibilidad de seleccionar las fechas en las que está comprendido el *roster*. La aplicación Web que proporciona este servicio se llama MidlandHR y consume los servicios Web provistos por el centro de ocio para poder calcular los *rosters*. Existen también dos versiones una que trabaja con servicios SOAP (con distintas configuraciones de seguridad) y otra que trabaja con REST.

Una interacción normal del sistema sería (Los pasos de la interacción se muestran en la figura 2 a través de las referencias "i-número"):

1. El manager quiere calcular el *roster* de sus empleados para la semana que viene, o uno de los empleados quiere saber cuáles son sus horarios para la semana que viene. El usuario ejecuta la aplicación y elige la opción "Roster", selecciona las fechas del periodo y hace *click* en el botón "Roster" (i-1).

2. MidlandHR recibe la petición de *roster* para el periodo seleccionado y comienza a calcularlo. Para calcularlo necesita datos (empleados, demanda de trabajo) del centro de ocio, por lo tanto precisa consumir los servicios Web del centro de ocio para obtener esos datos (i-2).

3. El centro de ocio recibe la petición para enviar datos y responde con los datos solicitados (i-3).

4. MidlandHR obtiene todos los datos y crea la solución *rostering*. La solución es enviada al cliente (i-4).

5. El cliente recibe la solución y la visualiza en pantalla (i-5).

6. En el primer paso, en vez de haber utilizado la aplicación RostApp, se podría haber usado la página Web que provee el centro de ocio (LeisureCenterServer) (i-6).

3.2. RostApp

Como ya se ha señalado esta aplicación se encarga de la gestión de los datos del centro de ocio. La aplicación tiene dos posibles usuarios: managers y empleados.

La aplicación permite al *manager* gestionar a los empleados de la empresa y su disponibilidad laboral así como gestionar los turnos de trabajo de la empresa.

Los empleados pueden señalar a través de la aplicación su disponibilidad laboral en los turnos que tienen asignados.

La aplicación permite a ambos usuarios la obtención de *rosters*. En el caso del *manager*, éste obtendría un *roster* de toda la plantilla laboral en el periodo seleccionado y en el caso de los empleados, estos obtendrían un *roster* personal del periodo seleccionado.

3.2.1. Diseño Arquitectural

La aplicación está diseñada en varias capas. El modelo de capas es un modelo cerrado que proporciona ventajas como alta portabilidad de los diferentes módulos, ya que estos han sido construidos exclusivamente con los servicios que ofrece la capa inferior. Esto también reduce las dependencias entre las capas y facilita los cambios porque la interfaz de una capa solo afecta a otra. Estos puntos hacen a esta metodología apropiada para este sistema. Estas capas se encuentran esquematizadas en la figura 3.

El sistema está dividido en tres capas: un primer nivel que maneja la base de datos, un nivel intermedio que gestiona los datos obtenidos de la base de datos y de la GUI y un nivel superior

compuesto por la GUI que controla la interacción con el usuario y la presentación de los datos en pantalla.

La "inteligencia" del sistema reside en el nivel intermedio. Todas las funcionalidades están implementadas en este nivel. Este nivel trabaja como puente entre la base de datos y la interfaz gráfica.

El nivel superior gestiona toda la interacción con el usuario. Toma los datos introducidos por el usuario y envía estos datos al nivel intermedio que los procesa y envía a la base de datos.

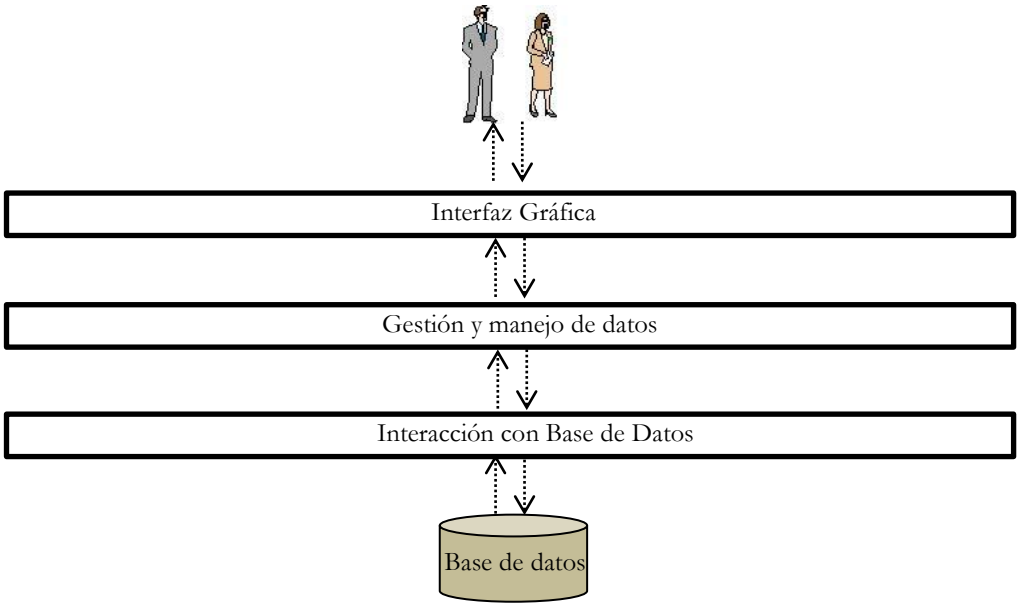


Figura 3 – Esquema de Capas de RostApp

Cada una de estas capas está dividida en paquetes que gestionan las distintas funcionalidades. En el anexo III se realiza un análisis de la aplicación que muestra los requisitos de ésta junto a los modelos de objetos, dinámico y funcional.

El diseño de objetos y operaciones fue realizado para obtener la aplicación más sencilla posible con una separación lógica que facilitase la implementación. Puede parecer que se están creando objetos innecesarios ya que solo almacenan datos intermedios, sin embargo esto nos permite organizar la implementación de manera eficiente.

El diseño de la base de datos se encuentra en el anexo III. En este documento se muestra el diagrama de entidad-relación junto a una descripción del funcionamiento de ésta.

3.2.2. Modelado del sistema

La Figura 4 muestra las dependencias entre paquetes de la aplicación. A continuación se describe cada uno de ellos:

Paquete GUI (GUI)
Contiene todas las clases necesarias para la interfaz gráfica de usuario. Ver Anexo III y Anexo XIII para más información.

Paquete de la base de datos (db)
Contiene todos los elementos (conectores, consultas) requeridos para gestionar la base de datos. Consultar Anexo III para más información.

Paquete de seguridad (password)
Este paquete implementa la clase *password*. Esta clase solo tendrá una instancia en el sistema, este objeto será el que acceda al *password* de la aplicación y valide a los usuarios antes de entrar al sistema.

Paquete Empleados (EmployeeManager)
Este paquete implementa las clases *Employee* y *EmpGestor*. La principal función del paquete es manejar todos los aspectos relacionados con la adición, modificación y eliminación de los empleados.
Consecuentemente, la clase empleado funcionará como un puente entre los datos obtenidos de la GUI y los datos introducidos en la base de datos. Esto nos permitirá abstraernos de la representación de los datos en ambas partes.

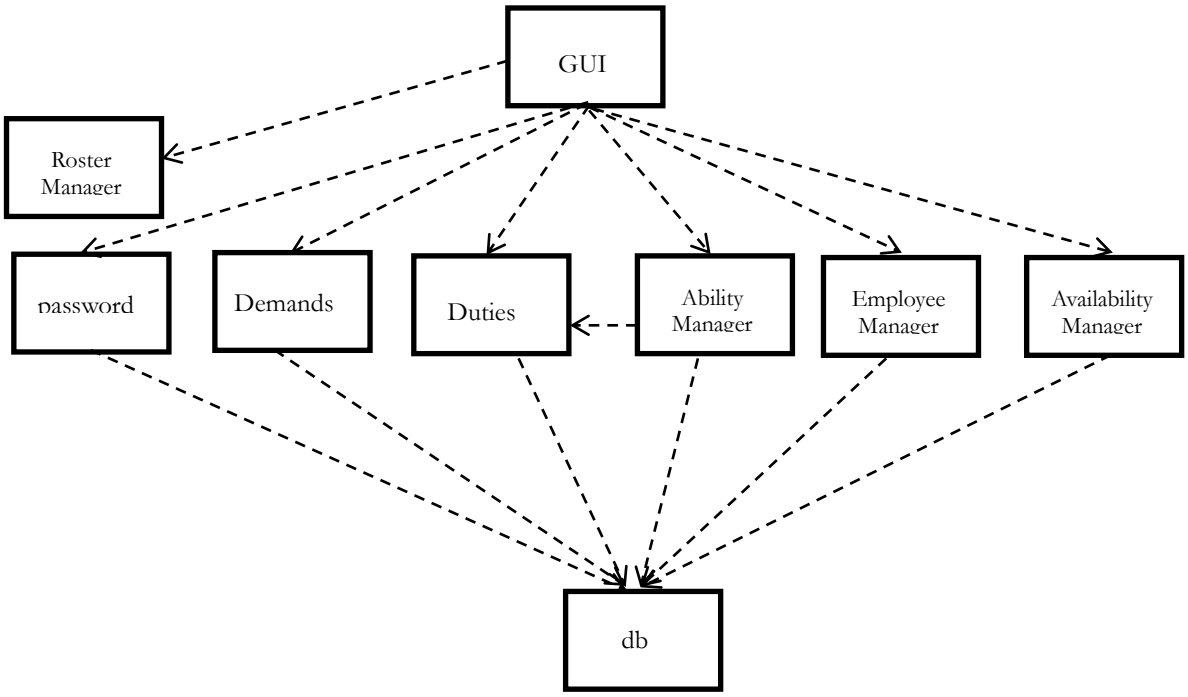


Figura 4 – Dependencias entre paquetes RostApp

Los siguientes paquetes funcionan de manera similar al paquete *EmployeeManager*

Paquete Disponibilidades (AvailabilityManager)
Contiene la clase *Availability* y el gestor de disponibilidades *AvailGestor*.

Paquete Habilidades (AbilityManager)
Contiene la clase *Ability* y el gestor de habilidades *AbiGestor*.

El paquete de habilidades y el de disponibilidades han sido creados para facilitar la implementación. Los datos que estas contienen podrían haber sido añadido a los datos del empleado, siendo el gestor de empleados quien gestionase las operaciones con estos datos. Esta separación nos proporciona una gestión más fácil y un mejor tratamiento de la información.

Paquete demandas (Demands)
Este paquete contiene la clase *Demand* y el gestor de demandas *DemGestor*.

Paquete Tareas (Duties)
Este paquete contiene la clase *Duties* y el gestor de tareas *DuGestor*. La clase *Duties* es una clase muy sencilla que es usada por los paquetes de demandas y habilidades. Sería posible añadir esta

clase a esos paquetes pero con la separación en una clase se obtiene un diseño más modular.

Paquete Rosters (RosterManager)
Este paquete gestiona los *roster*. Por un lado contiene la clase Roster, que se encarga de realizar las peticiones a través de servicios Web y obtener los *rosters*. Por otro, gestiona las diferentes vistas del *roster* que se muestran en pantalla. Este paquete incluye las clases necesarias para dibujar el *roster* en pantalla. Estas clases pertenecer al prototipo cedido por MidlandHR y han sido modificadas para que encajen en la aplicación

La figura 5 ilustra la distribución y composición de la aplicación.

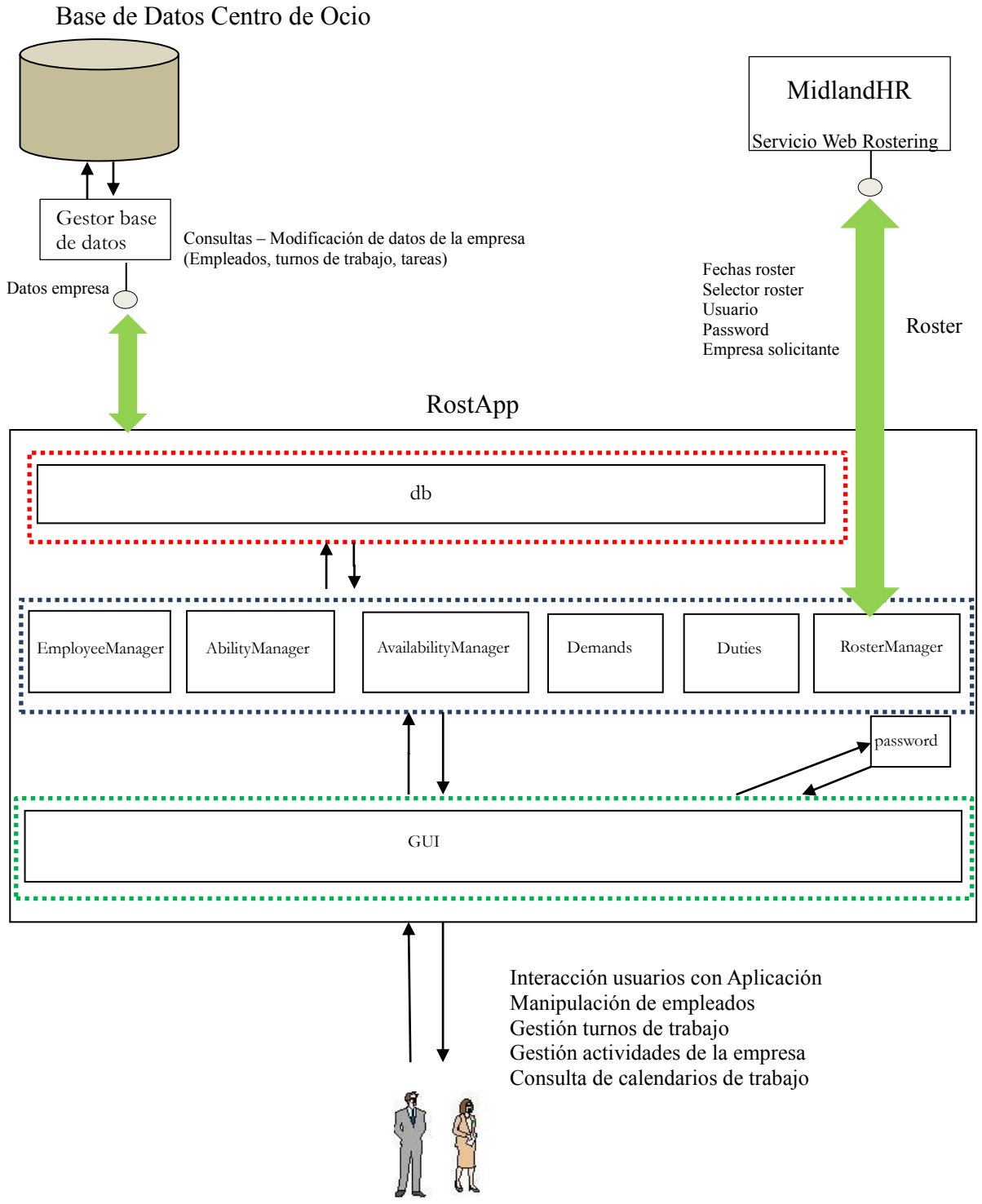


Figura 5 – Diseño RostApp

3.3. MidlandHR

Esta aplicación es la encargada de elaborar los rosters. La aplicación se sitúa en el servidor de MidlandHR y ofrece un servicio Web para la creación de rosters. En el proyecto este servicio es consumido por el centro de ocio.

El proceso de creación del roster necesita información de MidlandHR (trabajadores, cualificaciones, turnos de trabajo, tareas, disponibilidad) y para poder obtenerla la aplicación consume una serie de servicios Web provistos por el centro de ocio.

3.3.1. Diseño Arquitectural

El proceso de *rostering* es un proceso muy complejo. La mezcla de servicios Web y roster puede terminar siendo una aplicación muy compleja. Por esta razón, el objetivo del diseño de esta aplicación ha sido mantener ambas partes lo más independientes posibles. El algoritmo de *rostering* y todas sus dependencias están en paquetes aislados de la aplicación, solo son accedidos a través de una llamada para obtener el roster.

El proceso de *rostering* trabaja como una caja negra: la aplicación obtiene toda la información necesaria y la pasa al algoritmo de *rostering*. Si en vez de un roster hubiésemos querido calcular cualquier otro tipo de solución con los mismos datos, solo habríamos tenido que cambiar los paquetes de *rostering* por los de la nueva solución.

En el diagrama de actividades muestra de forma clara el proceso de funcionamiento de la aplicación. La figura 6 representa el diagrama de actividades.

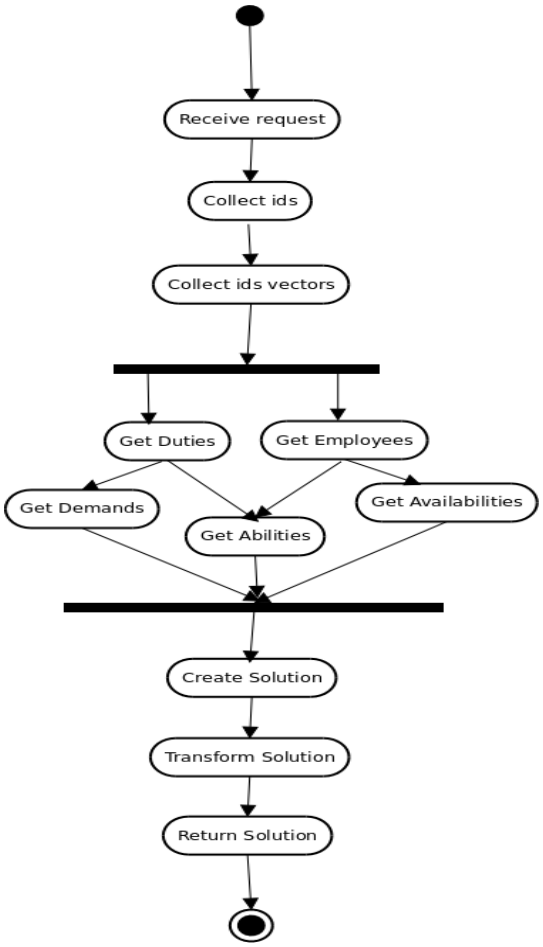


Figura 6 – Diagrama de Actividad MidlandHR

Como se puede ver en la parte de análisis (consultar anexo IV) la aplicación funciona como un sistema transaccional. El diagrama de clases muestra tres partes diferenciadas: una para recibir la petición, otra para procesar la petición y crear la solución y otra para obtener los datos del centro de ocio. La figura 7 ilustra el funcionamiento del sistema

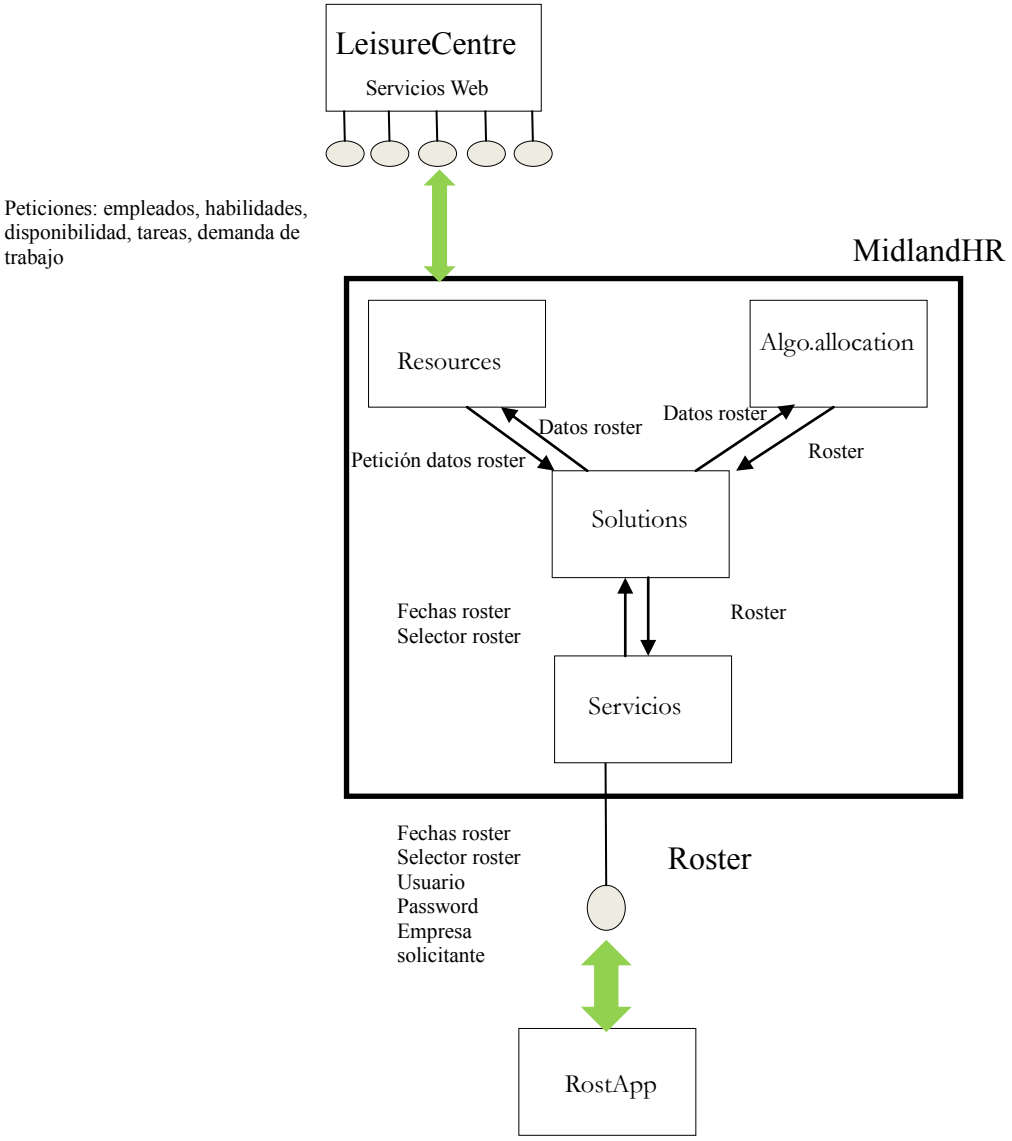


Figura 7 – Diseño MidlandHR

3.3.2. Modelado del sistema

El diseño de los objetos corresponde con las clases creadas en la fase de análisis. Las diferentes partes han sido agrupadas en paquetes. Estos paquetes son:

Servicios
Este paquete contiene la clase roster que es la que implementa el servicio Web que provee la aplicación.

Solutions
Este paquete contiene la clase *LeisureCentre*. Si además del centro de ocio HR estuviera trabajando con más empresas, la clase para cada una de esas empresas estaría situada en este paquete. Es el elemento central de la aplicación, se encarga de gestionar la obtención de datos y la creación del roster.

Resources
Este paquete contiene las clases que obtienen datos del centro de ocio a través de los servicios Web que éste provee.
Algo.allocation
Esta serie de paquetes fueron provistos por Arturo Castillo y son parte del prototipo de MidlandHR.
Ellos contienen el algoritmo para crear el roster. Los métodos de estas clases son invocados desde la clase *LeisureCentre*.
Se han añadido dos clases a estos paquetes: *ElementSolution* y *SolutionList*. Estas clases se encargan de transformar el roster obtenido en una versión viable para ser enviada del mismo.

3.4. LeisureCenter

Esta aplicación se sitúa en el centro de ocio. La misión de la aplicación es proporcionar información del centro de ocio a terceros. Esta información se proporciona a través de servicios Web.

La información esta almacenada en la base de datos de la empresa y la aplicación se encarga de extraerla, adaptarla para poder ser enviada y enviarla.

En el proyecto la aplicación interactúa con MidlandHR, que consume sus servicios Web. La información proporcionada corresponde a: información de los trabajadores (personal, disponibilidades, cualificaciones), tareas a realizar en la empresa y turnos de trabajo.

3.4.1. Diseño Arquitectural

Esta aplicación opera con la base de datos de forma muy similar a como lo hace RostApp. La fase de análisis (ver anexo IV) ha determinado que son necesarias tres capas para un buen diseño. En esta aplicación también se ha intentado aplicar un diseño modular y simple.

Como en HR, los servicios Web entregan el trabajo a otras clases y se limitan a devolver los datos pedidos. La aplicación fue desarrollada tras RostApp y se ha intentado reutilizar el máximo número de componentes. Si los componentes reutilizados son útiles significará que el nivel de modularidad alcanzado en RostApp fue óptimo. Los cambio fueron mínimos y la mayoría se debieron alto nivel de concurrencia de la aplicación.

La base de datos e empleada esta aplicación es la misma que se utiliza en RostApp. La clase de la base de datos interactúa de la misma manera que lo hace la clase de RostApp.

El único cambio se debe a la alta concurrencia de esta aplicación. El consumo de cada uno de los servicios Web necesita una instancia única de la base de datos (un único conector). Esta conexión es creada cuando el servicio Web comienza y se envía a los diferentes métodos hasta que la consulta es ejecutada.

En RostApp solo existía un usuario usando la aplicación en un ordenador a la vez por lo que con una única conexión global era suficiente para ejecutar todas las consultas.

3.4.2. Modelado del sistema

El diseño de los objetos corresponde con las clases creadas en la fase de análisis. Las diferentes partes se han agrupado en los siguientes paquetes:

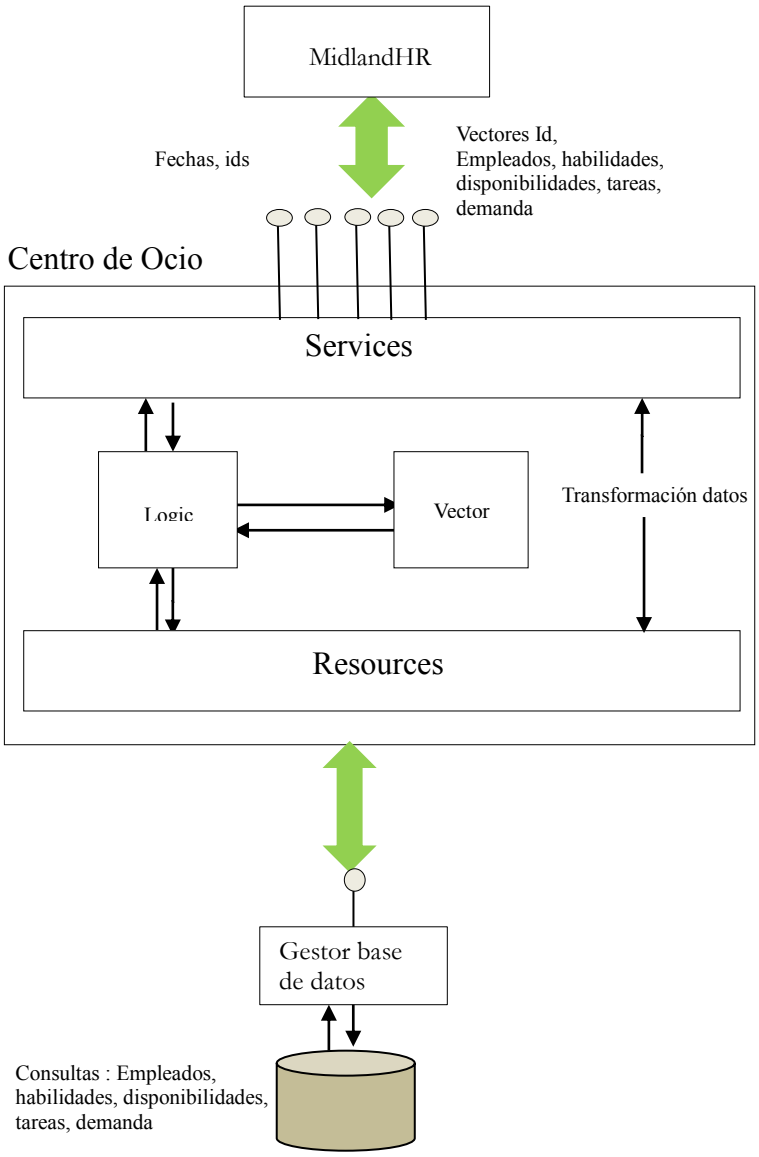
Services
Este paquete contiene todas las clases que implementan los servicios Web.

Database
Este paquete contiene las clases que interaccionan con la base de datos

Logic
Este paquete contiene las clases que representan las entidades del sistema: Empleados, demandas, tareas, disponibilidades y habilidades.

Vectors
Este paquete contiene la clase vector. Esta clase podría haber sido introducida en paquete Logic pero la funcionalidad de esta clase es distinta a las clases de ese paquete ya que esta clase se limita a facilitar el envio de los resultados de un servicio Web.

La figura 8 muestra de forma detallada el funcionamiento de esta parte del sistema:



Base de Datos Centro de Ocio

Figura 8 – Diseño LeisureCentre

3.5. LeisureCenterServer

Como se ha señalado antes, esta aplicación Web se sitúa en el servidor del centro de ocio y provee una página Web a los empleados de éste para que puedan obtener los *roster* en cualquier momento y en

cualquier lugar. La aplicación está compuesta por dos tipos de elementos: páginas Web y *servlets*. Las páginas Web interaccionan con el usuario y los *servlets* gestionan los datos introducidos por el usuario e inician el proceso de obtención del *roster*.

Páginas Web

Las páginas Web han sido escritas en JSP. El diseño visual es simple pero efectivo. A continuación se explica el funcionamiento de las páginas Web y cómo se navega entre ellas.

index.jsp

Esta es la página principal de la aplicación. En ella el usuario se autentifica, la Web toma los datos introducidos por el usuario (usuario y contraseña) y los envía al *servlet validateUser*. Si los datos son validados el *servlet* redirige al usuario a la página *selectDate* o *selectDateEmp* dependiendo si se trata del *manager* o de un empleado común. Si los datos son incorrectos el usuario es redirigido a una página de error.

SelectDate.jsp y SelectDateEmp.jsp

Estas Web permiten introducir las fechas entre las que se solicita el *roster* y la vista que se desea de él. El JSP envía los datos al *servlet validateDate* (*validateDateEmp*) que tras procesar los datos redirige al usuario a la solución.

RosterImage.jpg

Finalmente, si los datos son correctos el usuario obtiene una imagen de la solución *roster*. Esta solución es generada por el paquete *Roster* y servida por los *servlets validateDate*/*validateDateEmp*

Servlets

Hay tres *servlets* en el servidor. Estos *servlets* no trabajan solos y son ayudados por otras clases para realizar su trabajo. Los *servlets* son:

validateUser

Este *servlet* comprueba que el nombre y contraseña del usuario sean correctos. En el caso del *manager* el *servlet* valida el usuario por sí mismo pero si se trata de un usuario los datos deben ser verificados en la base de datos.

El *servlet validateUser* hace uso de la clase *database*. Esta clase tiene el mismo formato que las demás clases de base de datos que se han usado en el proyecto. La clase crea la conexión con la base de datos, ejecuta la consulta y envía el resultado al *servlet*.

ValidateDate and validateDateEmp

Ambos *servlets* trabajan de la misma forma, la única diferencia es el servicio Web que invocan. *validateDate* invoca el servicio *askRoster* para obtener un *roster* completo de la empresa y *validateDateEmp* invoca el servicio *askRosterEmp* que obtiene el *roster* personal de un empleado
Si las fechas son correctas estos *servlets* llaman al servicio Web y obtienen el *roster*. Una vez que han obtenido la solución la transforman en una imagen y crean una Web con ella.

La creación de la imagen es un proceso complejo. Dibujar el *roster* requiere el paquete provisto por MidlandHR. El problema radica en que el paquete dibuja el *roster* un elemento Java Swing, no en un archivo *jpeg*. Por lo tanto, ha sido necesario adaptar el paquete para dibujar la imagen en un archivo *jpeg*.

La figura 9 muestra el funcionamiento de la aplicación Web:

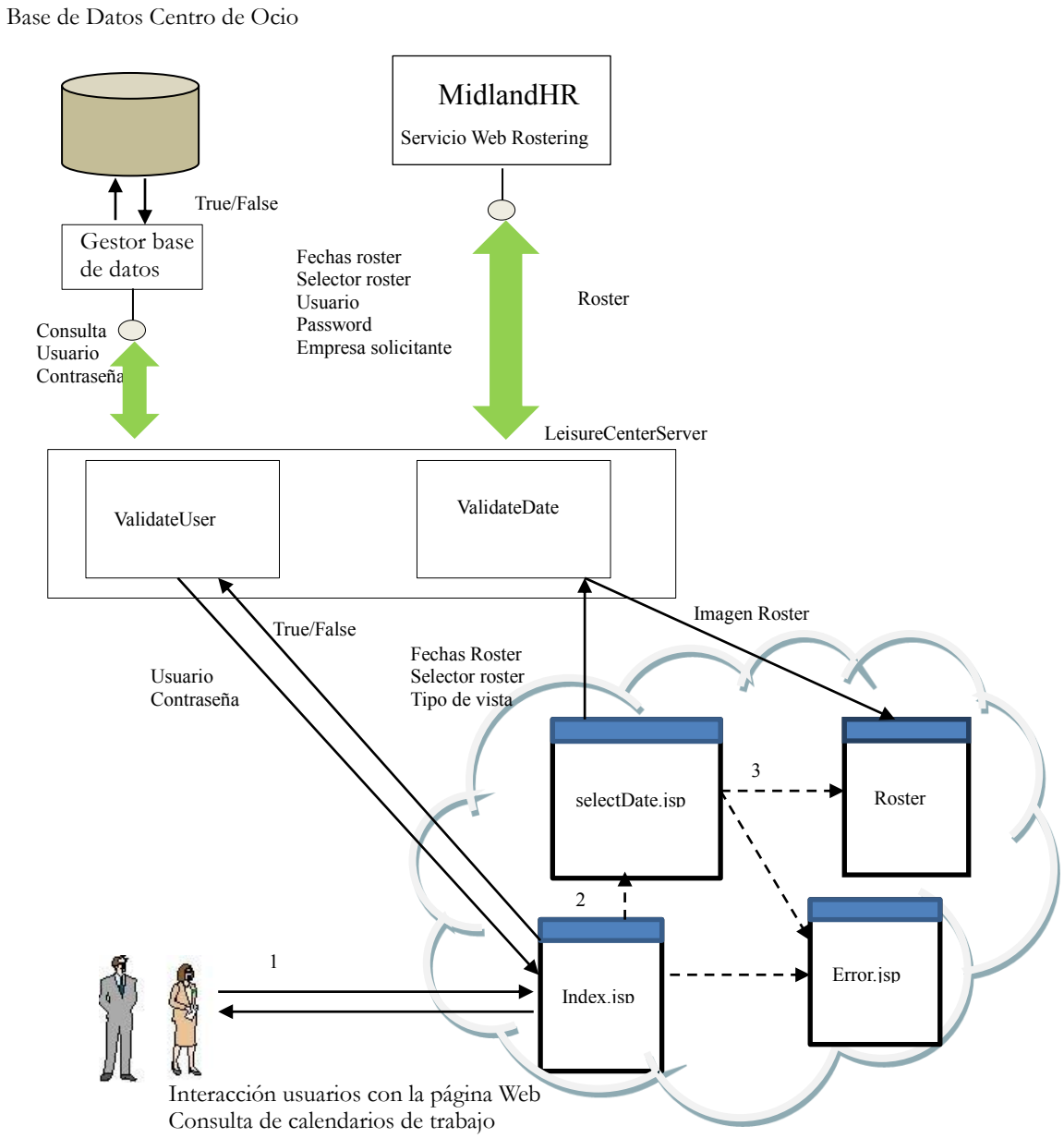


Figura 9 – Diseño LeisureCentreServer

3.6. Seguridad

La seguridad es una parte fundamental de los servicios Web. En este proyecto, datos privados y confidenciales se envían a través de ellos. Por lo tanto, es importante usar las herramientas adecuadas para proveer confidencialidad, integridad y privacidad al sistema.

En los servicios Web, la seguridad está muy relacionada con la eficiencia. Añadir capas de seguridad afecta a la eficiencia del servicio porque hay más datos que ser procesados. Para medir la relación entre seguridad y eficiencia en el proyecto, se han creado y probado tres configuraciones de seguridad distintas. Estas configuraciones han sido creadas con WSIT [24].

WSIT está desarrollado por Sun y Microsoft y es una parte de *Metro Web Services Stack*. WSIT consiste en una serie de especificaciones de servicios Web para soportar características empresariales junto a la optimización de mensajes, mensajería fiable y seguridad.

3.6.1. Analisis de la Seguridad del sistema.

Antes de aplicar las diferentes configuraciones de seguridad es necesario realizar un análisis de la seguridad del sistema y ver donde es necesario aplicarla.
Anteriormente se ha señalado que el sistema está dividido en tres partes. Existen 6 vías de comunicación/conexión entre las diferentes partes del sistema:

RostApp ↔ MidlandHR : Se realiza a través de servicios Web.
LeisureCenter ↔ MidlandHR : Se realiza a través de servicios Web.
LeisureCenterServer ↔ MidlandHR : Se realiza a través de servicios Web.
RostApp ↔ Base de datos: Se realiza a través de la red.
LeisureCenter ↔ Base de datos: Se realiza a través de la red.
Usuario ↔ LeisureCenterServer : Se realiza a través de internet (conexión del usuario a la página Web)

En un escenario real todas estas conexiones deben tener componentes de seguridad. A través de ellas se envía información personal por lo que el hecho de no proteger esta información puede dar lugar a sanciones penales. Dado que el objetivo del proyecto es analizar el funcionamiento de los servicios Web funcionando con tecnologías de *rostering* se han creado diferentes configuraciones de seguridad para las conexiones a través de ellos. De esta forma se podrá ver el impacto que tiene la seguridad en el rendimiento del sistema.
A las conexiones con la base de datos no les ha sido añadido ningún elemento de seguridad en el proyecto. La seguridad de éstas en un escenario real depende de las políticas de seguridad de la empresa y la localización de la base de datos respecto a ambas aplicaciones. El impacto de no añadir este tipo de elementos en las conexiones para el objetivo del proyecto es nulo.

3.6.2. Configuraciones de seguridad.

Las diferentes configuraciones serán aplicadas a todos los servicios Web (MidalndHR + LeisureCentre) Las configuraciones han sido creadas siguiendo el tutorial WSIT [25]

- Sin seguridad.

No se ha tomado ninguna medida de seguridad en esta configuración. Este será el caso base con el que se comparan las otras configuraciones.

- SSL

Secure Sockets Layer (SSL) y *Transport Layer Security (*TLS) son protocolos criptográficos que proveen comunicaciones seguras a través de la red. SSL provee autentificación y privacidad en la información enviada. Normalmente, solo el servidor es autenticado, no el cliente.
Para configurar SSL en nuestro sistema es necesario configurar el servidor Glassfish y la aplicación. En Glassfish es necesario seleccionar el certificado SSL, se han creado certificados X.509 para este propósito (consultar Anexo XIII)

Para configurar la aplicación, el archivo "Web.xml" tiene que ser modificado de la siguiente manera:

```
<security-constraint>
<display-name>Constraint1</display-name>
<Web-resource-collection>
<Web-resource-name>c1</Web-resource-name>
<description/>
<url-pattern>/*</url-pattern>
</Web-resource-collection>
<user-data-constraint>
<description/>
```

```
<transport-guarantee>CONFIDENTIAL</transport-guarantee>
</user-data-constraint>
</security-constraint>
```

"Constraint1" y "c1" son los nombres escogidos por el programador. Este código pide confidencialidad para todas las URLs de la aplicación (/*)

.

- **Mutual Certificates + SSL**

El mecanismo de seguridad con múltiples certificados añade seguridad a través de autentificación y protección de mensajes que asegura integridad y confidencialidad. Cuando se usan certificados es necesario configurar una *truststore* y *keystore*. Ambas deben ser configuradas tanto en el cliente como en el servidor. Este nivel de seguridad junto a SSL provee integridad, confidencialidad, privacidad y autenticación en ambas partes.

Para poder implementar esta política es necesario configurar el servicio Web y sus clientes. Netbeans ofrece un gran soporte para ello y automatiza el proceso de creación del archivo xml del servicio. En el lado del servicio, hay que seleccionar un certificado de la *keystore* (xws-security-server). Una vez seleccionado, se crea un nuevo archivo XML con todas las limitaciones de seguridad. En el archivo hay dos campos que representan las claves seleccionadas.

```
<sc:KeyStore wspp:visibility="private"
location="/home/miguel/glassfish-
3.0.1/glassfish/domains/domain1/config/keystore.jks" type="JKS"
storepass="changeit" alias="xws-security-
server"/>
```

El archivo WSDL del servicio también es modificado automáticamente por el IDE

El primer paso en el lado del cliente es obtener el nuevo WSDL del servicio. Una vez recibido, la seguridad tiene que ser configurada. Se necesitan dos pasos:

1. Proveer la clave primaria del cliente apuntando a un alias en la *keystore*(xws-security-client)
2. Proveer el certificado del servidor apuntando a un alias en la *truststore* del cliente.

Ahora, el XML del servicio tiene un nuevo campo:

```
<wsp:Policy wsu:Id="rosterPortBindingPolicy">
<wsp:ExactlyOne>
<wsp:All>
<sc:KeyStore wspp:visibility="private" alias="xws-security-
client" storepass="changeit" type="JKS"
location="/home/miguel/glassfish-
3.0.1/glassfish/domains/domain1/config/keystore.jks"/>
<sc:TrustStore
wspp:visibility="private"
peeralias="xws-security-server"
storepass="changeit"
type="JKS" location="/home/miguel/glassfish-
3.0.1/glassfish/domains/domain1/config/cacerts.jks"/>
</wsp:All>
</wsp:ExactlyOne>
</wsp:Policy>
```

4. IMPLEMENTACION

En este apartado se señalaran las características más reseñables de la implementación de las distintas aplicaciones.

4.1. RostApp

Lo más reseñable de la implementación de esta aplicación es el proceso de obtención del *roster*. La obtención de la solución es muy sencilla

Ejemplo:

```
sol = askRoster(date1, date2,comparator,"LC","osuosu21");
```

Los parámetros del servicio Web son explicados en el anexo III. En este caso tenemos las fechas entre las que está comprendido el *roster*, un comparador (variable de desarrollo para el prototipo), el nombre de la empresa que solicita el *roster* y el *password* de ésta en el servidor de MidlandHR.

El servicio devuelve un *roster* que es almacenado en la variable "sol". La figura 10 muestra la imagen del *roster* que obtiene el usuario en su pantalla. Para poder enviar las fechas a través del servicio es necesario que sean transformadas al formato *XMLGregorianCalendar*. Los pasos para transformar las fechas son :

```
GregorianCalendar calendar = new GregorianCalendar();
XMLGregorianCalendardate=
DatatypeFactory.newInstance().newXMLGregorianCalendar(calendar);
```

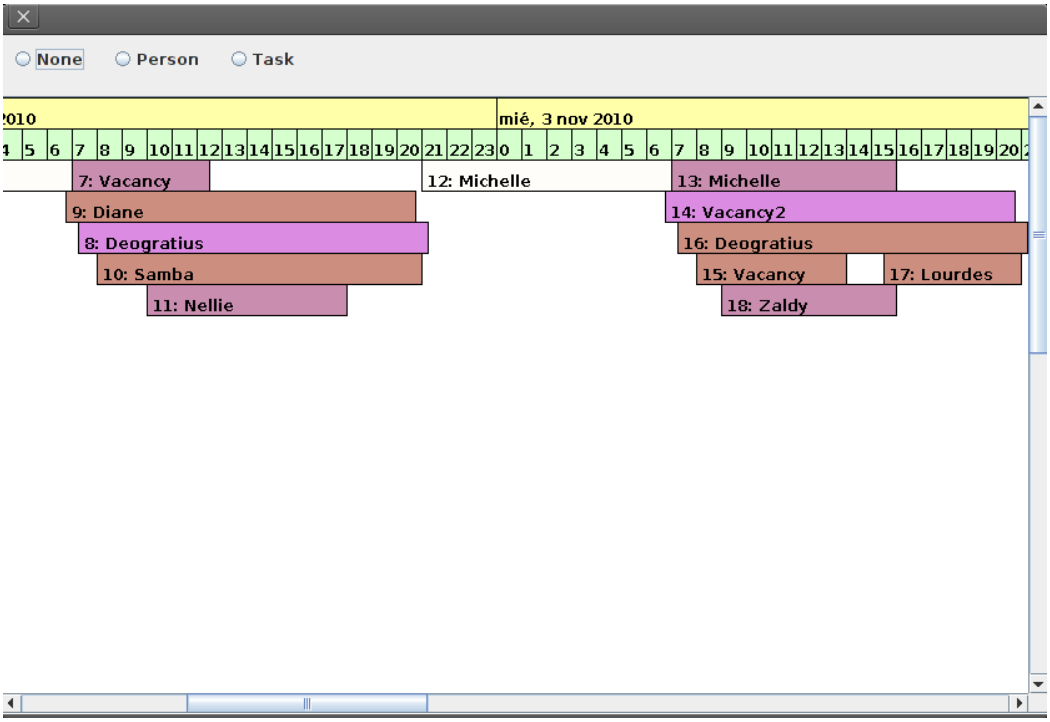


Figura 10 – Roster mostrado en pantalla

4.2. MidlandHR

La implementación de esta aplicación tiene una serie de puntos importantes. Para verlos todos, consultar anexo IV

En primer lugar es importante ver la estructura del servicio Web que MidlandHR provee. Existen dos versiones de este servicio Web: una que crea un *roster* global de la empresa y otra que crea un *roster* personal del empleado que lo solicita. La cabecera del primer servicio es la siguiente:

```
askRoster: Creates a roster.
@WebMethod(operationName = "askRoster")
public List<ElementSolution> askRoster(
@WebParam(name = "parameter")
Date parameter,
@WebParam(name = "parameter1")
Date parameter1,
@WebParam(name = "parameter2")
String parameter2 ,
@WebParam(name = "parameter3")
String parameter3 ,
@WebParam(name = "parameter4")
String parameter4) {
```

Los parámetros *parameter* y *parameter1* son las fechas entre las que se sitúa el *roster*. *Parameter2* es el comparador del *roster* (variable de desarrollo). "parameter3" y "parameter4" son el usuario y contraseña del cliente dentro del sistema de HR.

El algoritmo de estos servicios Web es sencillo y se limita a invocar a otra clase para crear el *roster*:

```
if (checkUser(parameter3,parameter4))
solList = LC.createSolutionEmp(start, finish, parameter2);
}
return solList
```

En primer lugar, la empresa que quiere consumir el servicio Web es validada (*checkUser*) tras lo cual la clase *LeisureCentre* es invocada para crear una solución que es devuelta. Todo el proceso de creación del *roster* podría haberse situado en esta clase, lo cual no hubiese sido bueno para la modularidad y el diseño de la aplicación.

Mantener el servicio simple permite que posibles cambios en el futuro sean más fáciles de lograr. Los mensajes SOAP tienen la siguiente estructura:

Mensaje SOAP:

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ser="http://servicios/">
<soapenv:Header/>
<soapenv:Body>
<ser:askRoster>
<parameter>2010-01-04T00:00:00.000+01:00</parameter>
<parameter1>2011-01-04T00:00:00.000+01:00</parameter1>
<parameter2>Task</parameter2>
<parameter3>LC</parameter3>
<parameter4>osuosu21</parameter4>
</ser:askRoster>
</soapenv:Body>
</soapenv:Envelope>
SOAP message:
<soapenv:Envelope
```

```
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ser="http://servicios/">
<soapenv:Header/>
<soapenv:Body>
<ser:askRosterEmp>
<parameter>?</parameter>
<parameter1>?</parameter1>
<parameter2>?</parameter2>
<parameter3>?</parameter3>
<parameter4>?</parameter4>
</ser:askRosterEmp>
</soapenv:Body>
</soapenv:Envelope>
```

Threads

Como puede ser visto en el diagrama de actividades algunas tareas se ejecutan en paralelo. Alguna de las clases que obtienen datos del centro de ocio pueden ser ejecutadas concurrentemente. Esta es la razón por la que estas clases ha sido implementadas usando hilos de ejecución (*threads*). Los hilos son creados y ejecutados en la clase *LeisureCentre*.

Sin embargo los hilos deben ser ejecutados en un orden determinado debido a las dependencias entre la información que obtiene.

Las dependencias son:

getDemands depende de *getDuties*
getAvailabilities depende de *getEmployees*
getAbilities depende de *getDuties* and *getEmployees*.

Por lo tanto los hilos deben ser sincronizados y tienen que esperarse unos a otros para respetar estas dependencias. Este se consigue con las variables: *end*NameOfTheClass y *check*NameOfTheClass.

La clase *LeisureCenter* controla sus hilos con una espera activa:

```
While (name_of_thread.isAlive());
```

Obtención de información.

Los datos obtenidos a través de los servicios Web del centro de ocio tienen que ser transformados en las estructuras que el algoritmo de *rostering* necesita. En cada categoría (empleado, disponibilidades, habilidades, tareas y demandas) se realiza un consumo del correspondiente servicio Web para obtener cada elemento.

El IDE automatiza el proceso de llamada, solo es necesario escribir:

```
av = askAvailabilites(v.get(i));
```

y el IDE crea el *port* de la llamada del servicio Web

```
private Availabilities askAvailabilites(int parameter) {
askAvailabilities.AskAvailabilitiesWSService service = new
askAvailabilities.AskAvailabilitiesWSService();
askAvailabilities.AskAvailabilitiesWS port =
service.getAskAvailabilitiesWSPort();
return port.askAvailabilites(parameter);}
```

Una vez que el elemento ha sido recibido es transformado en las estructuras del algoritmo de

rostering. En el prototipo cedido por MidlandHR todos los datos eran leídos de un fichero y transformados. Se ha adaptado el código que se encargaba de transformar los datos del fichero a uno que transforma los datos de un servicio Web. Esto muestra la versatilidad de los servicios Web.

Obtención del roster

En el momento en el que toda la información ha sido obtenida el *roster* es creado en tres pasos:

```
sa = new StaffAllocationAlgorithm(demand, staff1, availability);
sa.setSortingComparator();
sa.solve();
```

Tras obtener el *roster*, se genera una nueva entrada en el fichero de log. Esta línea indica si la solución ha sido obtenida o no y las características de esta.

Transformando la solución

La solución provista por el algoritmo de *rostering* es muy compleja y necesita ser modificada para ser enviada a través del servicio Web. Esta transformación es muy simple. Cada elemento de la solución es obtenido y transformado en un objeto del tipo *ElementSolution* que es almacenado en un vector (*SolutionList*) que contiene todo el *roster*.

Cada elemento de la solución original tiene la siguiente estructura:

```
private RosWorkunit workUnit;
private List<RosPair> combined;
private int availableSize;
private List<RosNode> clashes;
private RosPerson person;
private List<AssignationListener> assignationListeners;
```

Sin embargo, toda esta información no es necesaria para mostrar el *roster* en pantalla por lo que es transformada :

```
Integer idWorkUnit Date startTime: start time of the shift
Date endTime: end time of the shift
String Task: Task's name
Integer idTask
String Person: Employee's name
Integer IdPerson
Integer idRoster
```

El proceso de transformación es el siguiente:

```
for (int i=0; i<solutionList.size();i++){
ElementSolution e = new ElementSolution();
wu = solutionList.get(i).getWorkUnit();
person = solutionList.get(i).getPerson();
//IdworkUnit
e.setIdWorkUnit(wu.getIdWorkUnit());
//Start Time
e.setStartTime(wu.getStartTime());
//End Time
e.setEndTime(wu.getEndTime());
//Get Task
e.setTask(wu.getNameTask());
e.setIdTask(wu.getIdTask().getIdTask());
//Person
e.setIdPerson(person.getIdPerson());
e.setPerson(person.getNamePerson());
```

```
//idRoster
e.setIdRoster(wu.getNumberRoster());
l.add(i,e);
}
```

Enviar la solución original hubiera sido también posible pero se hubiesen necesitado métodos para transformar los datos en un esquema XML válido. La solución adoptada es válida para ser transformada directamente en un formato JAX-WS, las herramientas de JAX-WS del IDE codifican la información automáticamente.

Este método tiene otro uso: cuando un empleado pide su *roster*, todo el *roster* tiene que ser creado (dependencias) pero solo su parte ha de ser enviada y mostrada. En este caso, el propio método se encarga de seleccionar las *workunits* que pertenecen al empleado que ha solicitado el *roster*:

```
if (person.getIdPerson() == parameter2)
```

Donde *parameter2* es el id del empleado que ha solicitado el *roster*.

4.3. LeisureCenter

Servicios Web

Los servicios Web *AskAbilitiesWS*, *AskEmployeesWS*, *AskDemandsWs*, *AskDutiesWS*, *AskAvailabilitiesWS* tienen todos la misma estructura. El parámetro "*parameter*" corresponde al identificador del elemento que se está enviando desde la base de datos.
Por similaridad solo se presenta *AskAbilitiesWS*, para consultar los demás ver anexo V.

La cabecera de *AskAbilitesWS*

```
@WebMethod(operationName = "askAbilites")
public Abilities askAbilites(
@WebParam(name = "parameter") int parameter) {
SOAP message:
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ser="http://services/">
<soapenv:Header/>
<soapenv:Body>
<ser:askAbilites>
<parameter>?</parameter>
</ser:askAbilites>
</soapenv:Body>
</soapenv:Envelope>
```

Los siguientes dos servicios Web son distintos de los anteriores. Estos servicios Web se encargan de enviar a HR el número de elementos que se van a enviar junto a vectores con los identificadores de los elementos para que se puedan verificar la recepción de la totalidad de los datos desde el otro lado. Ambos servicios tiene dos parámetros que corresponden a las fechas que delimitan el *roster* solicitado.
Los servicios son:

AskNumbers

```
@WebMethod(operationName = "askNumberElements")
public Vector<Integer> askNumberElements(
@WebParam(name = "parameter") Date parameter,
@WebParam(name = "parameter1") Date parameter1)
```

Mensaje SOAP:

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ser="http://services/">
<soapenv:Header/>
<soapenv:Body>
<ser:askNumberElements>
<!--Optional:-->
<parameter>?</parameter>
<!--Optional:-->
<parameter1>?</parameter1>
</ser:askNumberElements>
</soapenv:Body>
</soapenv:Envelope>
```

AskVectors

```
@WebMethod(operationName = "askVectors")
public vectors askVectors(
@WebParam(name = "parameter") Date parameter,
@WebParam(name = "parameter1") Date parameter1) {
```

Mensaje SOAP:

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ser="http://services/">
<soapenv:Header/>
<soapenv:Body>
<ser:askVectors>
<!--Optional:-->
<parameter>?</parameter>
<!--Optional:-->
<parameter1>?</parameter1>
</ser:askVectors>
</soapenv:Body>
</soapenv:Envelope
```

Obtención de datos.

La manera en la que los datos son obtenidos es igual en 5 de los servicios Web (los correspondientes a entidades). El proceso queda ilustrado con el siguiente ejemplo. En este caso se obtiene una habilidad:

Primer paso:

```
AskAbilitiesWS

database db = new database(); - The instance of the database is
created
message = db.prepare(); - Connection to the database
conn = (Connection) db.getConn(); - Get the connection object.
ab1 = ab.ability(parameter,conn); - Call to obtain the ability.
```

Segundo paso

```
Abilities class.(ability method)
rs = db.ability(parameter,conn); - Execution of the query.
ab = new Abilities( rs.getInt(1), - Creation of the new ability.
n,
rs.getString(3),
```

```
rs.getFloat(4));
return ab;
```

Tercer paso

```
AskAbilitiesWS
db.dissconect(conn); – Disconnection of the database
return ab1; – Return of the requested ability.
```

Los servicios *askNumbers* y *askVectors* trabajan de forma distinta:

askNumbers.

El funcionamiento de este servicio puede ser también explicado en tres pasos. Cada clase en el paquete *Logic* tiene un método para obtener el número de elementos de esa clase en la base de datos.

Primer paso

AskNumbers

```
Vector<Integer> v = new Vector<Integer>(5); – Vector which will be
filled with the numbers.
message = db.prepare();
conn = (Connection) db.getConn(); – Connection to the database
--This adds the different numbers to the vector.
v.add(dt.numberDuties(conn));
v.add(emp.numberEmployees(conn));
v.add(dem.numberDemands(parameter,parameter1,conn));
v.add(av.numberAvailabilities(parameter,parameter1,conn));
v.add(ab.numberAbilities(conn));
db.dissconect(conn);}
```

Segundo paso
(El ejemplo escogido ha sido habilidades, pero podía haber sido cualquier otro)

```
numberAbilties()
rs = db.numberAbilities(conn); – Execute the query
n = rs.getInt(1); – Get the number
return n; – Return the number.
```

Tercer paso

```
--Return the vector
return v;
```

El servicio *askVectors* funciona de la misma manera pero en vez de usar un vector genérico utiliza un objeto "*vectors*"

```
vectors v = new vectors();
v.setAbilities(ab.idsAbilities(conn));
v.setAvailabilities(av.idsAvailabilities(parameter,
parameter1,conn));
v.setDemands(dem.idsDemands(parameter, parameter1,conn));
v.setDuties(dt.idsDuties(conn));
v.setEmployees(emp.idsEmployees(conn));
```

En este caso todas las clases en el paquete *Logic* tienen un método para obtener un vector con los identificadores

4.4. LeisureCenterServer

En la aplicación RostApp, una vez que el *roster* se mostraba el usuario podía cambiar entre las diferentes vistas. En este caso el funcionamiento es distinto. Existían dos posibilidades:

- Crear las tres vistas e introducirlas en un vector que será enviado a otra página Web donde el usuario selecciona la vista y esta se muestra. Esta solución es más interactiva pero mandar las tres imágenes en un vector es una carga muy pesada ya que el tamaño y complejidad del *roster* no se conocen de antemano. Por lo tanto, esta opción con varios clientes al mismo tiempo podría no funcionar de manera adecuada.

- Obtener solo una imagen. Esta solución es ligera pero también tiene sus limitaciones. Si el usuario quiere consultas las tres vistas, tendrá que llamar al servicio Web tres veces.

Ambas soluciones son válidas pero se eligió la segunda ya que es mejor para el servidor trabajar solo con una imagen en vez de con tres a la vez.

Para transformar la solución en una imagen se necesita una clase auxiliar. Esta clase se llama *rosterImage* y se encarga de crear un nuevo lienzo (*allocationPanel*) y devolver la imagen creada.

El proceso es:

```
allocationPanel = new AllocationPanel();
allocationPanel.setView(view);
byteArray = allocationPanel.setSolutionList(sol);
```

La nueva versión de allocationPanel escribe la imagen en un BufferedImage en vez de un swing panel. La *BufferedImage* es escrita en un *ByteArrayOutputStream* que es devuelto. Para este propósito se han creado los métodos *setSolutionList* y *savePanel*.

El proceso para crear la imagen es :

Selección de la vista

```
if (request.getParameter("view").equals("general")){
select = 1;
}
else if (request.getParameter("view").equals("person"))
{
select = 2;
}
else{
select = 3;
}
```

Se crea una nueva instancia de la clase *rosterImage* con el *roster* y la vista como argumentos

```
roster = new rosterImage(sol,select);
```

Creación de la imagen

```
byteArray = roster.createImages();
```

Escritura de la imagen en un buffer al que es redirigido el usuario.

```
ServletOutputStream bufferSalida = response.getOutputStream();
response.setContentType("image/jpeg");
response.setContentLength(byteArray.length);
bufferSalida.write(byteArray);
```

```
bufferSalida.flush();
bufferSalida.close();
```

4.5. REST

Las aplicaciones presentadas anteriormente usan servicios Web SOAP. Este apartado se presenta las mismas aplicaciones pero utilizando servicios REST. Las aplicaciones LeisureCentre, MidlandHR y RostApp han sido transformadas.

HRRest

La estructura de la aplicación es la misma que en MidlandHR y funciona de la misma manera. Solo unos pocos cambios han sido realizados para adaptar la aplicación al enfoque REST.

Se ha creado un nuevo paquete llamado *WebServices*, este paquete contiene las clases que obtienen los recursos REST. Se ha creado una clase para cada servicio.

Estas clases son generadas por el IDE pero es necesario hacer algunos cambios en ellas para que funcionen correctamente. Las clases son clientes Jersey REST.

Ha sido necesario cambiar la forma en la que se consumen los servicios Web. La estructura y parámetros de estos servicios se explican en el siguiente apartado. En HRRest lo único que nos interesa es saber que los recursos que devuelven los servicios REST son cadenas de texto con sus campos separados por espacios. Las cadenas incluyen números que es necesario transformar al formato *Integer* y *Float*. De la misma manera las cadenas incluyen fechas que han de ser transformadas al formato *date*.

Java proporciona un mecanismo llamado *SimpleDateFormat* que permite transformar fechas a *strings* y viceversa. Es un proceso muy simple en ambas partes.

```
SimpleDateFormat formateador = new SimpleDateFormat("HH:mm
dd/MM/yyyy");
date1 = formateador.format(start);
```

En la versión SOAP los servicios Web devolvían objetos. En este caso es necesario crear el objeto y rellenar sus atributos con la cadena recibida. Para consumir un servicio Web son necesarios los siguientes pasos:

1º Crear una instancia del cliente Jersey para consumir el servicio Web:

```
WebServices.Abilities client = new WebServices.Abilities();
```

2º Usar el método del cliente para obtener el recurso. Un *string* con el id de la habilidad es enviado para obtener un *string* con la habilidad completa.

```
String response = client.ability(String.class,v.get(i).toString());
```

3º Ahora, es momento de procesar la cadena . Existen diferentes opciones de hacerlo, en este caso se ha optado por usar un scanner.

```
Scanner s = new Scanner(response);
s.useDelimiter(" ");
4º Obtener los campos de la habilidad.
ab.setNid(Integer.parseInt(s.next()));
ab.setName(s.next());
ab.setDutie(s.next());
ab.setValue(Float.parseFloat(s.next()));
```

5º Una vez que se ha completado el objeto, cerrar la conexión.

```
client.close();
```

El funcionamiento es el mismo con las diferentes entidades del sistema. La división en capas del sistema nos ha permitido modificar el funcionamiento del sistema cambiando solo unas pocas líneas en algunas clases. Esta es otra de las ventajas de la modularidad y de una buena fase de diseño.

LeisureCenterRest

Esta aplicación es más sencilla que su versión SOAP y muestra perfectamente las ventajas de los servicios REST.

En vez de tener una estructura de tres capas solo dos son necesarias: Una capa para los servicios Web y otra para la interacción con la base de datos.

Normalmente los datos devueltos por los servicios REST son datos en formato XML. En nuestro caso es solo una cadena. Este cambio ha sido adoptado para facilitar la implementación de HRRest, de lo contrario procesar el XML para transformarlo en las entidades del sistema habría sido complicado.

En la versión SOAP de LeisureCenter había tres métodos para cada entidad: uno para ella misma, otro para los identificadores y otro para el número de elementos de ese tipo en la base de datos. En esta versión se mantienen los tres métodos pero están estructurados de forma distinta.

Clases:

Cada clase representa un recurso REST

Database
Esta clase es usada para conectar con el sistema de base de datos y ejecutar las consultas que obtienen los datos solicitados.

Abilities, availabilities, employees, demands and duties
Estas clases funcionan de forma similar. Devuelven una cadena con los campos de la entidad solicitada.

IdsAvailabilities, idsAbilties, idsEmployees, idsDemands and idsDuties.
Estas clases funcionan de forma similar. Devuelven un vector de números convertido en una cadena.

Numbers
Esta clase tiene métodos para obtener la cantidad de elementos de cada entidad que hay en la base de datos. Devuelve un vector de números convertido en una cadena.

Los métodos usados en estas clases son los mismos que en la versión SOAP. Para adaptarlos a la versión REST se realizaron tres cambios:

1.Añadir la ruta al comienzo de la clase:

```
@Stateless
@Path("/Employees")
public class Employees {
```

El IDE crea automáticamente toda la infraestructura necesaria para proporcionar el servicio al reconocer este cambio. Para obtener un empleado la ruta sería:
http://host:8080/LeisureCenterWeb/Employees/id

2. Crear un método REST e indicar si es un método GET o POST: Si el método tiene parámetros es necesario indicar cuál será el nombre de los parámetros para que así el cliente pueda obtener la información.

```
@GET
public String employee(@QueryParam("id")String parameter)
```

3. Devolver una cadena. Java tiene métodos para convertir la mayoría de tipos predefinidos a strings.

```
return id.toString() + " " + name + " " + contract.toString() +
+ Float.toString(ranking) +
" " + Float.toString(capacity);
" " + Float.toString(wage) + " "
```

Todos los métodos REST se han creado aplicando estos cambios a los métodos existentes. Como se puede observar se ha creado una clase para cada tipo de identificador pero solo se ha creado una clase para el numero de recursos. Esto se debe a que si solo se crea una clase para los identificadores se obtendría como resultado un vector de vectores el cual es más complicado de transformar en una cadena y procesar en HRRest.

5. EVALUACIÓN

Dos tipos de test han sido realizados: test SOAP y test de carga. Los test SOAP han comprobado la conformidad de los mensajes de los servicios Web con los estándares y los test de carga han estresado al sistema para obtener conclusiones de la idoneidad de la combinación de servicios Web y *rostering*.

Los requisitos de calidad del servicio para servicios Web son: disponibilidad, accesibilidad, integridad, fiabilidad, *throughput*, latencia, conformidad con los estándares y seguridad.

En este apartado se muestran una serie de graficas que resumen de forma concisa el funcionamiento del sistema. En el anexo X se pueden encontrar gráficas y tablas que proporcionan más información del proceso de evaluación.

5.1. Evaluación SOAP

Estos test han sido realizados automáticamente por SoapUI [26]. SoapUI es una aplicación que cuenta con numerosas herramientas para verificar la corrección de los mensajes SOAP. Así mismo permite, la creación de test de carga y esfuerzo para servicios Web. Todos los mensajes SOAP que se utilizan en el proyecto han sido validados por este programa. Todos ellos cumplen el estándar de mensajes SOAP.

Para realizar los test, el programa utiliza el WSDL propio de cada servicio para recrear uno de sus mensajes, comprobar su formato y verificar el mensaje de respuesta.

5.2. Evaluación de carga

Estos test simulan un escenario real de trabajo. Diferentes casos son simulados y ejecutados para medir tiempo, CPU, memoria...

El escenario en esta evaluación tiene tres ordenadores que comparten una red WiFi privada. Cada parte del sistema ha sido instalado en un equipo. Las características de los equipos son:

Equipo 1 – RostApp
Ubuntu 10.10 Maverick
Linux Core 2.6-35.25.
Gnome 2.32.0
Intel(R) Core(TM)2 Duo CPU
2GB DDR2
Server Glassfish 3.0.1
T7300 @ 2.00GHz

Equipo 2 – MidlandHR
Ubuntu 10.10 Maverick
Linux Core 2.6-35.25.
Gnome 2.32.0
Intel Pentium Dual Core Processor T2370 @ 1,73GHz
Enhanced Intel SpeedStep technology
2GB DDR2 Go SDRAM
Server Glassfish 3.0.1

Equipo 3 – LeisureCenter
Ubuntu 10.10 Maverick
Linux Core 2.6-35.25.
Gnome 2.32.0
Intel(R) Core(TM)2 Duo CPU T7300 @ 2.00GHz
2GB DDR2
Server Glassfish 3.0.1

Hay diferentes test de carga. Es importante explorar todas las posibilidades para obtener un conocimiento global de cómo responde el sistema a diferentes situaciones. El sistema espera picos de acceso de hasta 15 usuarios concurrentes pidiendo diferentes tipos de rosters: individuales, globales, 1 semana, 2 semanas....

Los test han sido ejecutados utilizando SoapUI para simular los diferentes usuarios (hilos) solicitando *rosters*. VisualVM [27] ha sido usado para controlar la CPU y la memoria de los equipos.

Los siguientes escenarios han sido probados:

- Escenario 1 – Sin seguridad
  Este escenario no tiene ninguna configuración de seguridad.

- Escenario 2 – SSL
  En este escenario los servicios Web usan SSL

- Escenario 3 – Certificados mutuos + SSL
  En este escenario los servicios Web usan SSL junto a certificados mutuos para autenticarse entre ellos.

- Escenario 4– Servicios REST + SSL
  El servicio Web para obtener el *roster* funciona sobre SSL y SOAP. El resto de los servicios Web (LeisureCenter) funcionan utilizando REST;

Para todos estos escenarios se han realizado los siguientes tests:

5.3. Resultados

Para resultados más detallados consultar anexo X. A continuación se presentan brevemente los resultados más importantes, así como una explicación de los mismos.

Tamaños and BPS

Los tamaños de los *rosters* son:
1 semana: 10511 bytes
2 semana: 21543 bytes
3 semana: 32363 bytes

Los bps en los diferentes escenarios han sido:
Escenario 1: 2659 bps
Escenario 2: 3598 bps
Escenario 3: 4256 bps
Escenario 4: 4309 bps

Escenario 1 - Sin seguridad

La figura 11 muestra los tiempos de respuesta del escenario1.

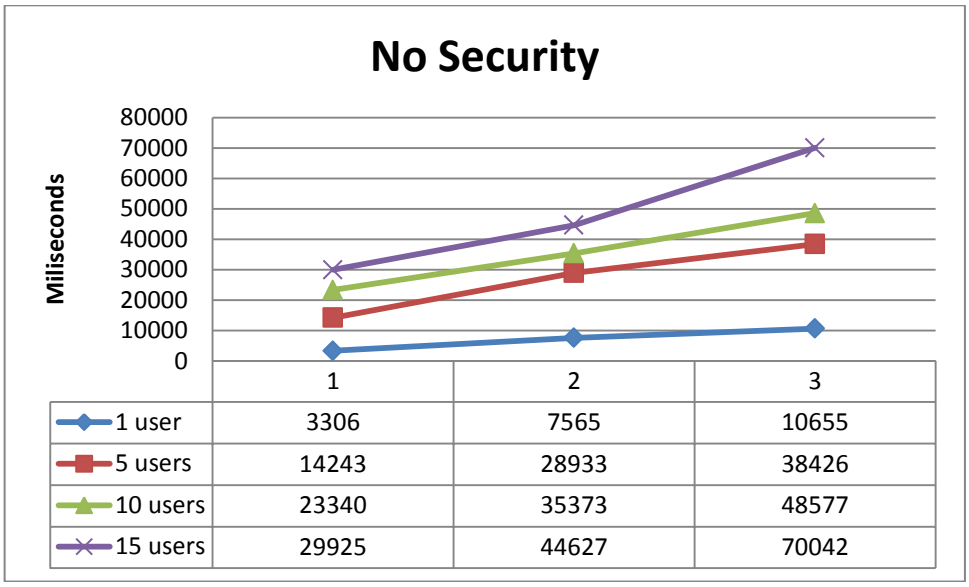


Figura 11 – Resultados Escenario 1

Escenario 2 - SSL

La figura 12 muestra los tiempos de respuesta del escenario 2.

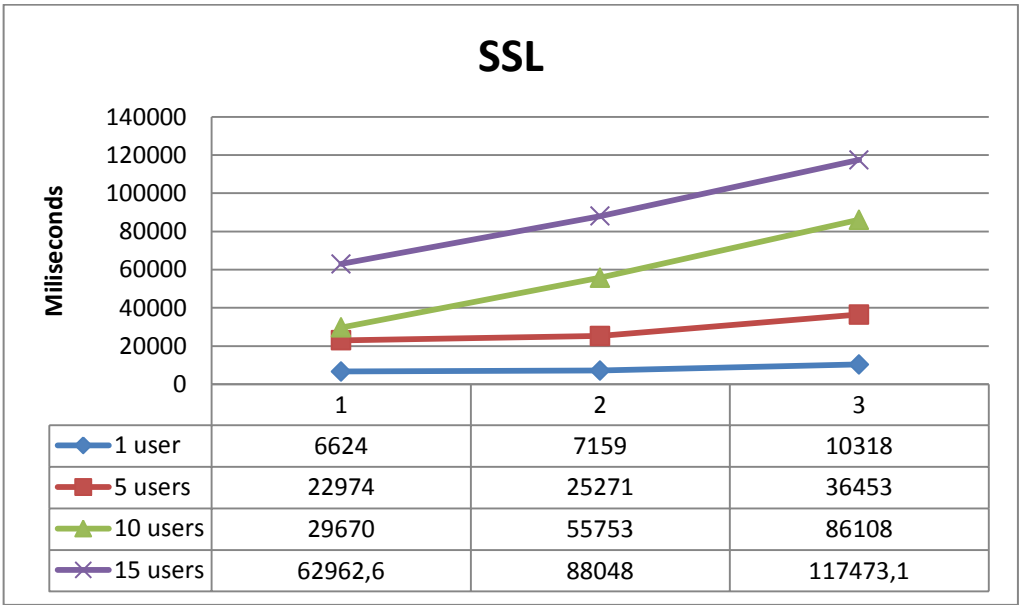
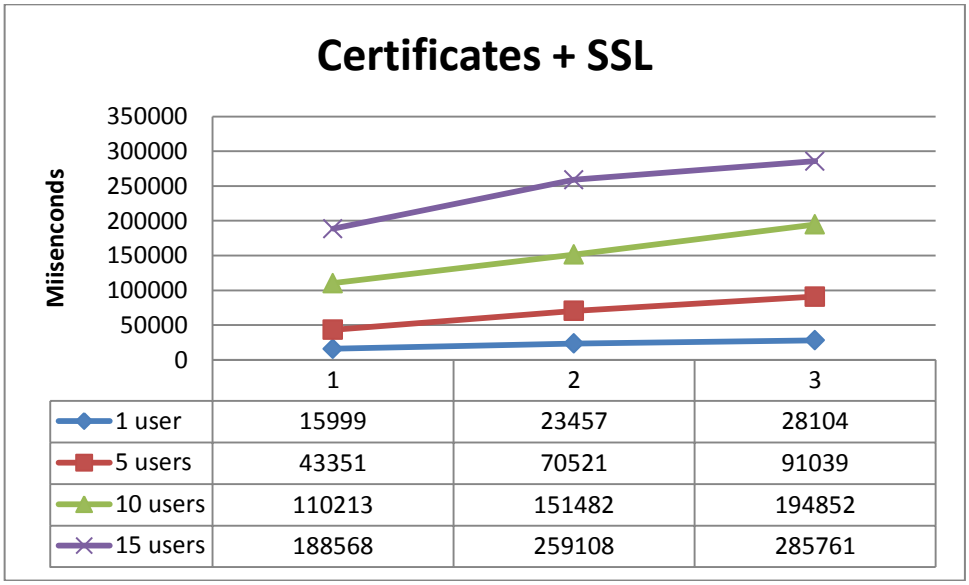


Figura 12 – Resultados Escenario 2

Escenario 3 - Certificados mutuos + SSL

La figura 13 muestra los tiempos de respuesta del escenario3.



	1	2	3
1 user	15999	23457	28104
5 users	43351	70521	91039
10 users	110213	151482	194852
15 users	188568	259108	285761

Figura 13 – Resultados Escenario 3

Los resultados de estos test siguen el patrón esperado. Los tiempos de respuesta aumentan gradualmente con el número de usuarios y tamaño del *roster*. Solo el escenario sin seguridad está bajo tiempos de respuesta aceptables (60 segundos). SSL da tiempos aceptables para 10 usuarios o menos y el tercer escenario es solo útil para un usuario.

El uso de certificados dobla los tiempos de respuesta del escenario SSL, lo que hace que esta opción no sea recomendada. Los beneficios no merecen el coste. Sin embargo, SSL es una opción recomendable ya que los tiempos de respuesta son muy similares a los del escenario sin seguridad y solo con más 10 usuarios los tiempos son demasiado elevados.

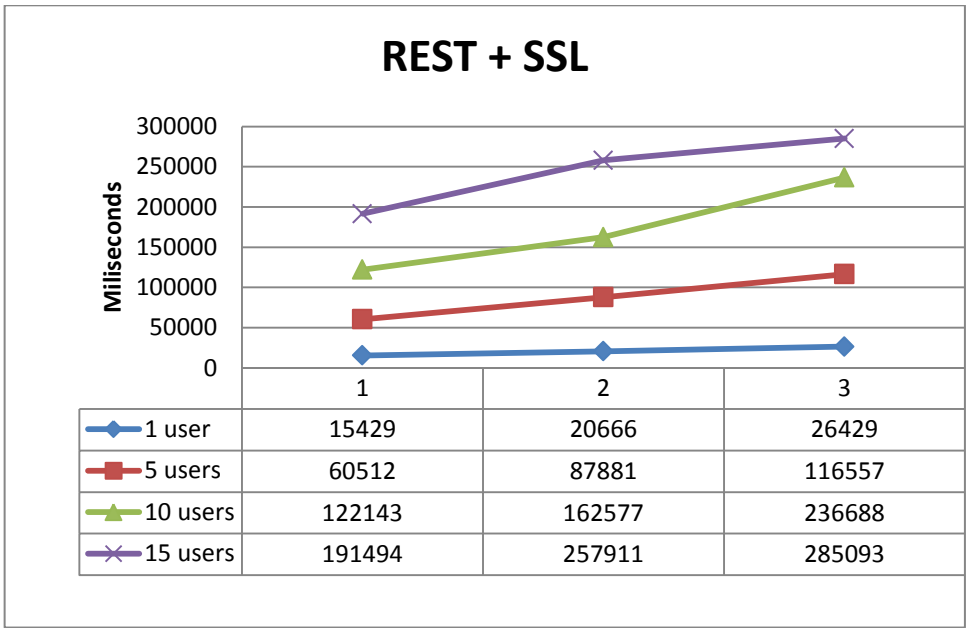
Los tiempos de respuesta aumentan según aumenta el tamaño del *roster*. Esto no tiene que cumplirse siempre ya que lo que hace que aumenten los tiempos de respuesta es la complejidad del *roster*. En este escenario concreto en el que se han realizado los test, la complejidad aumenta junto el número de semanas por lo que los tiempos aumentan a la vez.

El uso de la CPU y de la memoria son muy similares en todos los escenarios. Como se puede observar el proceso de *rostering* requiere casi toda la capacidad del procesador. Los ordenadores usados son ordenadores portátiles y no son válidos para trabajar como servidores reales. Sin embargo, estos test muestran la importancia de adquirir un servidor de alto rendimiento para este tipo de sistemas (ver anexo X)

En la parte del centro de ocio, el uso de la CPU incrementa con SSL y los certificados. El uso en estos escenarios triplica el uso del escenario sin seguridad. En ambos casos, el uso es constante y no hay grandes diferencias entre casos.
El consumo de memoria aumenta bajo el patrón esperado y es similar en MidlandHR y en el centro de ocio. El consumo de memoria no es tan crítico como el uso de CPU.

Escenario 4 - Servicios REST + SSL

La figura 14 muestra los tiempos de respuesta del escenario 4.



	1	2	3
1 user	15429	20666	26429
5 users	60512	87881	116557
10 users	122143	162577	236688
15 users	191494	257911	285093

Figura 14– Resultados Escenario 4

Este escenario incluye REST y SSL. Un escenario sin SSL fue también probado pero los resultados obtenidos fueron muy similares a este. Esto se debe a que en este escenario SSL solo se aplica a un servicio Web (servicio *roster*).

Los resultados en este escenario son sorprendentes. Teóricamente los servicios REST son más rápidos y ligeros que los servicios SOAP pero los resultados de este escenario son similares al escenario SSL+Certificados.

En este punto se puede ver que la discusión entre REST y SOAP (Anexo I) no es válida. Ambos enfoques son válidos solo si se usan en los escenarios correctos y de la manera correcta. La clave está en la implementación, no el tipo de servicio.

La forma de operar de los servicios Web del centro de ocio corresponde a un enfoque REST más que a uno SOAP. El problema está en que la implementación de los servicios REST se ha basado en las directrices de la implementación SOAP realizada anteriormente.

En la implementación REST, los datos son obtenidos de la base de datos para seguidamente ser procesados, preparados y enviados. Sin embargo existen herramientas para obtener información de la base de datos fácil y rápidamente en formato XML. El uso de estas herramientas (EJB Beans, NetBeans las provee y automatiza el proceso) hubiese complicado la implementación del sistema de HR pero hubiese reducido los tiempos de respuesta.

Los resultados anteriores muestran que el uso de la CPU en el servidor MidlandHR está alrededor del 90%. Si se hubiese añadido más trabajo al sistema (procesar y preparar los datos proporcionados en XML) , el sistema podría haber resultado insostenible.

En el sistema probado hay 214 llamadas a servicios Web (218 en la versión REST) entre los servidores de MidlandHR y del centro de ocio. Hay demasiadas llamadas y estas pueden ser incluso más si hay más ausencias o demandas de trabajo. Un alto número de llamadas pueden congestionar la red y degradar el rendimiento de los servidores. Los datos deberían ser cacheados en el cliente siempre que sea posible para evitar peticiones al servidor. En este caso, el cacheo de datos no se puede realizar de una manera tradicional debido a la naturaleza del sistema.

Otro de los puntos que hay que decidir es el uso de muchos paquetes de pequeño tamaño o de paquetes grandes. Los paquetes grandes requieren menos llamadas (solo 10 en nuestro sistema) pero procesar toda la información e introducirla en el sistema puede resultar difícil y añadiría más trabajo que sobrecargaría el servidor de MidlandHR.

En resumen, los "extraños" resultados del escenario REST muestran la importancia de elegir la tecnología correcta. Ellos también muestran la parte crítica del sistema: el intercambio de datos entre MidlandHR y las diferentes empresas que contraten sus servicios. Obtener los datos es fácil pero obtenerlos de manera eficiente es más complicado, aparte de que los requisitos de las diferentes empresas son diferentes. Los requisitos de las empresas deben ser estudiados en profundidad para obtener sistemas de calidad. El proceso de *rostering* es muy pesado y añadir más carga al servidor supondría serios problemas pero de la misma manera podemos observar que altos tiempos de respuesta también suponen problemas.

El principal problema de este aspecto para MidlandHR es que cada compañía con la que ellos trabajan tiene su propio sistema, servidor, base de datos, políticas de seguridad...Por lo tanto, configurar todo ello de manera idónea para MidlandHR puede ser difícil ya que puede que existan más empresas trabajando a la vez y que necesiten esos datos.

Para aclarar este punto se ha creado un nuevo test. Este test pretende medir la proporción de tiempo que el servidor de MidlandHR pasa obteniendo datos comparado con el tiempo que pasa creando el *roster*. Los resultados son:

(Tiempo total/ Tiempo obteniendo datos/ Tiempo calculando la solución)

	1 semana	2 semana	3 semanas
1 usuario- No seguridad	3396/3316/28	5261/5120/61	6531/6348/92
1 usuario- REST	11823/11761/35	18194/18056/79	22676/22516/104

El cuello de botella del sistema está al obtener los datos del centro de ocio. Al principio, se esperaba que pasase la mayor parte del tiempo calculando el *roster*, pero como se puede ver no es así.

6. CONCLUSIONES

Este Proyecto Final de Carrera ha logrado la consecución de los objetivos planteados inicialmente, resolviendo los nuevos requisitos y necesidades que han aparecido durante su análisis y desarrollo. El sistema propuesto ha sido implementado y se han mostrado las ventajas, desventajas, puntos clave y débiles de la combinación de la programación automática y los servicios Web.

El proyecto ha demostrado como esta combinación es posible. La interoperabilidad de los servicios Web facilita la implementación del proceso de *rostering* en un escenario distribuido. La seguridad es una parte importante de los servicios Web y como se ha podido comprobar, implementarla no es algo difícil. El problema reside en que hacerlo puede ser costoso ya que en la mayoría de casos va a depender de las políticas de seguridad que emplean las compañías que contratan los servicios de *rostering* y esto puede complicar el sistema.

Los test han mostrado resultados poco favorables pero han resaltado el cuello de botella del sistema: el intercambio de datos entre la compañía de *rostering* y el resto de compañías. Sin embargo, la importancia no está en el cuello de botella, sino en lo que esperan las compañías que del servicio que contratan.

Internet es democrático, todo el mundo y todas las empresas tienen las mismas oportunidades en él. Internet también promociona una alta competitividad donde el mejor producto no siempre triunfa. Esto hace que la empresa de *rostering* necesite una nueva estrategia de negocios. El hecho de pagar para obtener un servicio en vez de pagar para obtener una solución software podría crear conflictos entre empresas: quizás una empresa no esté dispuesta a pagar lo mismo por el servicio que por el software cuando el coste del servicio podría ser incluso mayor. Esta combinación transforma la empresa de *rostering* en una empresa Web.

Mencionar las palabras "servicios Web" se traduce en velocidad e interoperabilidad para la mayor parte de empresas. Por lo tanto, todos los esfuerzos deberían centrarse en obtener estas prestaciones. Esta tarea no es sencilla, si la empresa de *rostering* trabaja con diferentes empresas y cada una de ellas tiene su propio servidor y tecnología, obtener los mejores resultados para cada una requiere una gran cantidad de esfuerzo y dedicación.

Cada cliente tiene que ser estudiado con detalle y la colaboración entre ambas partes es esencial. Sin embargo, para el cliente, colaborar con la empresa de *rostering* no puede suponer un esfuerzo extra ya que en ese caso puede que no quieran el sistema porque no les compensa. Es importante recordar que ellos sólo quieren obtener un servicio y pagar por él. La empresa de *roster* tiene que ser cuidadosa y mantener la distancia adecuada con los clientes.

El propósito del sistema requiere obtener un nuevo *roster* con cada petición, pero varias peticiones pueden pedir el mismo *roster* . Crear el mismo *roster* varias veces es malgastar tiempo y recursos. Sin embargo, el hecho de que los datos del *roster* puedan ser cambiados en cualquier momento y de forma instantánea fuerza a la empresa a crear un nuevo *roster* con cada petición.

Por las razones mencionadas anteriormente, el cacheo de datos realizado de forma tradicional no es posible. Consecuentemente habría que desarrollar nuevos métodos, técnicas y estrategias para reusar los datos y evitar malgastar recursos. Para desarrollar estas técnicas se necesita:

- Estudios del uso real del sistema, que podrían llevar a generar automáticamente los *rosters* más solicitados para evitar ser calculados con cada petición.

- Métodos para notificar si los datos del cliente han sido modificados. Esto lleva otra vez a establecer una mayor colaboración con el cliente.

Este proyecto y el sistema desarrollado han demostrado como los servicios Web y la planificación automática son adecuados para trabajar juntos y, al mismo tiempo, cuál es la característica más importante de la combinación: personalización. Normalmente, un servicio Web da un servicio global a todo el mundo, pero en esta área ese enfoque no es válido. Cada servicio Web ha de ser personalizado para cada cliente.

Esta característica debería centrar el enfoque de los siguientes pasos en este tipo de escenarios.

El proyecto ha logrado los objetivos previstos. Se ha transformado la arquitectura del sistema a una distribuida a través de servicios Web y utilizando tecnologías SOAP y REST. Se ha evaluado la implementación SOAP viendo su funcionamiento con diferentes niveles de seguridad. Esta evaluación nos ha permitido estudiar los tiempos de respuesta del sistema así como el uso de memoria y CPU viendo las ventajas y desventajas de la combinación de los sistemas de planificación automática con servicios Web. Finalmente se ha realizado un estudio de la posible utilización de un *bróker* de mensajes en el sistema.

6.1. Conclusiones personales

Personalmente estoy contento con el trabajo realizado. Ha sido un proyecto con numerosos retos, es un primer acercamiento valido al área, abre la puerta y señala el camino para futuros proyectos en este ámbito. Tengo curiosidad por saber cómo esta solución evolucionará y se implementará en la industria. En este punto creo que es un área muy poco explorada y llena de posibilidades. Sin embargo, creo que el desarrollo de esta idea podría ser complicada. Además del problema de obtener los datos de las empresas, el hecho de transformar el enfoque centralizado en uno distribuido cambia el enfoque de la empresa de *rostering*. Hasta ahora las empresas de este tipo estaban preocupadas y centradas en su software pero con el cambio de enfoque esto desaparece ya que aparecen nuevos focos a los que prestar atención .La empresa puede centrarse en funcionar de manera "2.0" lo que podría actuar en detrimento de la calidad de su software.

6.2. Trabajo Futuro

Este proyecto representa un punto de partida para explorar la combinación de ambas tecnologías en proyectos futuros. Este proyecto me ha dado un entendimiento global y completo de ambas aéreas. Numerosas mejoras pueden ser hechas. Creo que las mejoras relacionadas con el *rostering* y sus técnicas son importantes pero las relacionadas con elementos Web son más importantes. Estas van a proveer a la empresa un valor extra frente a sus competidores. Algunas mejoras podrían ser:

- Desarrollar un interfaz Web accesible y fácil de usar. El uso de tecnologías web específicas para mostrar información por pantalla permitiría una mayor interacción del usuario con los resultados recibidos así como una mejor calidad de éstos.
- Lanzamiento de código abierto para una potencial comunidad de desarrollo. La creación de grupos de desarrollo en este ámbito de trabajo potenciaría estos primeros desarrollos y crearía un nexo en común que fomentaría la compatibilidad de futuros desarrollos.
- Adaptación en un *plugin*/componente para ser mostrado junto a otros contenidos. De esta manera podría incluirse este tipo de sistemas en portales web privados y elementos de comunicación empresarial.
- Sistemas de gestión (CRMs y gestores colaborativos de contenido). Esto permitiría un fácil acceso al software de *rostering* desde distintas plataformas.

Cabe destacar la posibilidad de mejorar el proceso para la obtención de datos donde un *pool* de conexiones u otras técnicas similares podrían cambiar significativamente los resultados.

Otra de las posibles mejoras del sistema es la utilización de un *bróker* de mensajes que se encargue de gestionar las comunicaciones entre la MidlandHR y sus clientes. Existen distintos tipos de *brokers* y *middleware* para este propósito. Entre ellos encontramos RLinda [28] y su versión distribuida DRLinda [29]. Este software permite gestionar de forma más eficiente el tratamiento de las peticiones de *roster* recibidas mejorando así la calidad del servicio proporcionado a los clientes. En el anexo VIII se muestran las características y detalles de una posible implementación de este sistema utilizando este software.

ANEXO I – BACKGROUND INFORMATION

1. INTRODUCTION

This project aims to develop a web services solution for automated scheduling. Web services and automated scheduled are important areas in the computer world. However, the combination of them has not been deeply studied.

Nowadays, this combination has a great amount of opportunities in rostering industry. The project aims to explore the different possibilities of implementation with different technologies (SOAP, REST) and to study which are the advantages and disadvantages of this approach.

Overall, the aim of this project is to explore how Web Services and distributed technologies could help in this particular situation. There are different possibilities to achieve this aim; this project focuses on performing a proof of concept of how the serviced oriented paradigm fits with the rostering process.

The project works in a real scenario, the enterprise MidlandHR is interested in this new approach and all the project has been developed from the resources supplied by this company. These resources are based in a real environment where MidlandHR works.

MidlandHR is an independent company founded in 1984 to meet the needs of payroll and people management, MidlandHR has rapidly expanded its product and service portfolio to become a leading provider of human resource management software and services. The company's range of HR Management solutions allows organisations to incorporate all human resource activities - from payroll to personnel - into a single system.

MidlandHR a software provider for people management solutions is looking to explore the advantages of web services when used to deliver staff allocation functionality. In order to achieve this objective a re-engineering of the prototype module would be necessary as well as the test of several features involved when using web services.

The proposed solution consists in creating a service oriented scenario where the performance of the rostering web services can be measured and tested. This scenario is a direct translation of the traditional stand-alone scheduling application to a service oriented approach. This will provide a first analysis of how useful this approach could be. In this scenario, MidlandsHR provides rostering solutions through web services and one enterprise consume that web services to obtain rosters for its workforce. In this project the enterprise which requests the services of MidlandsHR is a leisure centre.

The project also focuses in small and medium enterprises (SMEs) and how they can use the web services technology. Nowadays it is easy for SMEs to use new technologies; however some of them see all these new technologies as a strange and complicated world with more cons than pros. The project aims to show SMEs how web services can ease their labour on a straightforward way, without spending big amounts of money.

Web services, automated scheduling and SMEs are wide areas; a little knowledge about them is given in the second point of this annex.

The project details are given in the annexes I. An overview of the system is explained in annex II and the annexes III, IV, V, VI and VII explain all the system implementation in detail. Security details are explained in annex IX. Annex XIII add documentation about the implementation of the different parts of the system. The created system has been tested in order to provide feedback of the project and of how it has been developed. Annex X reflects all the results and their interpretation. Finally, annex XI resumes the conclusions and results of the project.

2. BACKGROUND INFORMATION

There are four key concepts in this project: SOA, Web services, automated scheduling and SMEs. This annex will give a brief but complete description of the main features of these concepts in order to provide a good understanding of them and the project.

2.1. Service Oriented Architecture

Definition

"Service Oriented Architecture (SOA) is a paradigm for organizing and utilizing distributed systems" [5]

SOA could be seen as a way to design software systems to provide services to user end-applications or to other services distributed in the network via discoverable interfaces.

Nowadays, the deployment of a new service may need from other services. It is hard to develop a global system without any external need. Using developed and tested services will be easier and more affordable than creating a new service starting from scratch. SOA provides an architecture to develop services so that they could be easily linked between them.

SOA is focused on the business value united to the strategic goals. The main aim of SOA is an intrinsic interoperability rather than specific interoperability models for every communication among services. This global vision provides flexibility to the model which evolves under a continuous improvement.

There are three key concepts in the SOA paradigm: visibility, interaction and effect [30]. Visibility allows different services to be discovered for other services. Services are self-described (wsdl, xml schema) and if a service has some need; it just has to discover services and perform a search to find which is the one who best fits its needs. Syntax and semantics has to be widely accessible and understandable.

The second one is interaction. Once than a service has been discovered and chosen to be used, the interaction begins. Interaction between services is performed through the interchange of messages. The key point in this process is to keep the simplicity. Every interaction between services is unique due to the conditions and policies of the medium. It is necessary to create a simple way of communication among services in order it can be adapted to the medium in an efficient way.

The third one is effect. Effect is the result of interaction; it could be a response message to the service or the user. Interaction could change the state of entities or generate new information.

Principles

SOA has a series of guiding principles to be developed and implemented. These principles suffer slightly variations from one author to another. However, there are a few than could be considered as fixed:

- Standardized Service Contract- The Most fundamental part of SOA design. The service contract has to be standardized and understandable for worldwide services.
- Service Loose Coupling - Minimum dependencies. Services should be developed independently achieving for the maximum interoperability.
- Service Abstraction - Services only need to know the service contract of other service in order to use it. Implementation details should not be revealed.
- Service Reusability - It is a core aspect in the designing process.
- Service Autonomy - Services have control over the logic they encapsulate.

- Service Statelessness - No state is kept in most of interactions in order to reduce memory consumption.

- Service Discoverability - Services need to be effectively and easily discovered and understood.

- Service Composability - Services are effective composition participants, regardless of the size and complexity of the composition.

These and other principles has been summarized in some web pages like the SOA-Manifesto web [30] and the SOA-principles web [31]

Benefits and criticism

SOA has had a great acceptance in the IT world. However, as every new paradigm it has had some controversy. The more important benefits of this paradigm could be summarized in:

- Interoperability increase among external, internal, developed and undeveloped applications.

- Reusability increase in the enterprise's IT services.

- Reduction of development time. Changes could be implemented easily without stopping the service.

- Business visibility increase. Enterprise's services can be exposed what leads to a better integration and optimization.

- Reduction of the maintenance cost due to the removal of duplicated and unnecessary applications.

Globally, this enables the enterprise to perform a cost reduction and a revenue enhancement. SOA has been successfully introduced in management processes such as Lean, six-sigma and TQM. [32]

On the other hand, SOA has been criticized. Serious criticisms refer to the interaction speed among services and the dependency on the BES (Service bus). Implementing SOA in an enterprise requires a considerable effort, which some enterprises are not able to achieve.

In another way SOA has also been criticized. Some people think that it does not add anything new to the existent methodologies, it is said to be a new name to create a product which could be sold easily to the enterprises as the "new way of doing business".

The fact that SOA is an architecture widely studied to bring great benefits does not imply that it is advisable to use it in all scenarios. The implementation of a SOA structure is often a slow process due to the large impact on systems that are in production. Ultimately, the advantages gained often compensate the development and implementation efforts. Therefore, they are being increasingly installed in more businesses and institutions.[30]

SOA and Web Services.

Commonly, the SOA and web services concepts are confused. Web services are a way of implementing SOA. It is not necessary to use web services to implement SOA, there are another technologies such as XML-RPC, COM or CORBA. The usage of web services may not provide enough benefits to justify their costs in performance. The SOA's implementation through web services is recommended when the services must operate over the Internet and the different systems use different products, languages or technologies. [33]

2.2 Web services.

Definition

The W3C web service definition is:

"A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards" [34]

There are a significant number of definitions for web services, which shows the term's complexity. A possible and simple definition would be referring to them as a set of applications or technologies with capability to operate in the Web. These applications and technologies interchange data among them in order to offer services. The providers offer their service as remote procedures and the users ask for a service calling these procedures through the Web,

Origin

Web services were created due to the need of standardize the communication between different platforms and programming languages (PHP, C#, Java, etc.).

Before web services different platforms were developed to achieve the objective. EDI, DCOM and CORBA are the most important ones. However, they did not achieve a global interoperability due to the different vendors and technologies used. The main protocol for these platforms was RPC, which is quite useful in controlled networks but it is difficult to control and manage on the Internet.[35]

In 1999, the need for a new a global standard was recognized worldwide and the W3C and OASIS started to develop this new technology.

Web service technologies

Web services are constituted by a series of different layers and technologies. The interaction of these technologies is what creates the web services. The technologies and layers can be seen in the figure 15.

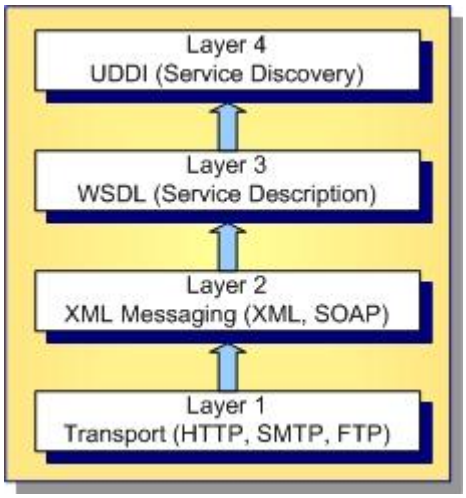


Figure 15. Web services protocol stack

Layer 1 is the transport layer and it uses the common Internet's protocols, especially HTTP. The rest of layers are exclusive to web services. The technologies used are:

XML (eXtensible Markup Language)

"It describes a class of data objects called XML documents and partially describes the behaviour of computer programs which process them. XML is an application profile or restricted form of SGML, the Standard Generalized Markup Language [ISO 8879]." [36]

This language has a very important function in the web services messages and how they are interchanged, structured and sent. The language describes the data in order to structure it using non-predefined tags which create interoperability in the data.

SOAP (Simple Object Access Protocol)

"SOAP is a lightweight protocol intended for exchanging structured information in a decentralized, distributed environment. It uses XML technologies to define an extensible messaging framework providing a message construct that can be exchanged over a variety of underlying protocols. The framework has been designed to be independent of any particular programming model and other implementation specific semantics." [37]

SOAP is the core of web services. It allows a standard mechanism to package packages. SOAP has been supported by the whole industry. A SOAP message is composed by an envelope which contains the body of the message (the information) and the header which describes the message and adds extra information to manipulate it. Figure 16 shows the format of a SOAP message.

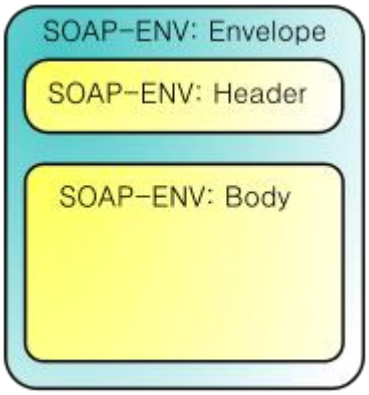


Figure 16. SOAP message

WSDL (Web service description language)

"WSDL describes a Web service in two fundamental stages: one abstract and one concrete. Within each stage, the description uses a number of constructs to promote reusability of the description and separate independent design concerns.

At an abstract level, WSDL describes a Web service in terms of the messages it sends and receives; messages are described independently of a specific wire format using a type system, typically XML Schema.

At a concrete level, a binding specifies transport and wire format details for one or more interfaces. An endpoint associates a network address with a binding. And finally, a service groups together endpoints that implement a common interface." [7]

Web service definitions can be mapped to any implementation language, platform, object model, or messaging system. WSDL allows to define which a web service does according to the functionality that it offers. This language represents the usage interface of the service. This interface represents which the other services will have to take into account to access the service's functionality.

It allows the client and the service to establish an agreement about the transporting of the messages and its content. This is achieved through a document that both parts understand. WSDL

represents a contract between provider and client; it specifies the syntax and the mechanisms for the exchange of messages.

UDDI (Universal Description Discovery and Integration)

"Universal Description, Discovery and Integration is a platform-independent framework for describing services, discovering businesses, and integrating business services by using the Internet. " [8]

UDDI is a protocol which allows to describe web services, products, companies, transactions, etc. UDDI provides a framework in order the clients find dynamically other web services, creating a standard interoperable platform. This allows the companies to use web services in a fast, easy and dynamic way. Using the UDDI's interface, the enterprises can be connected with services provided by external partners. All this requires the register in the UDDI platform.

How web services work

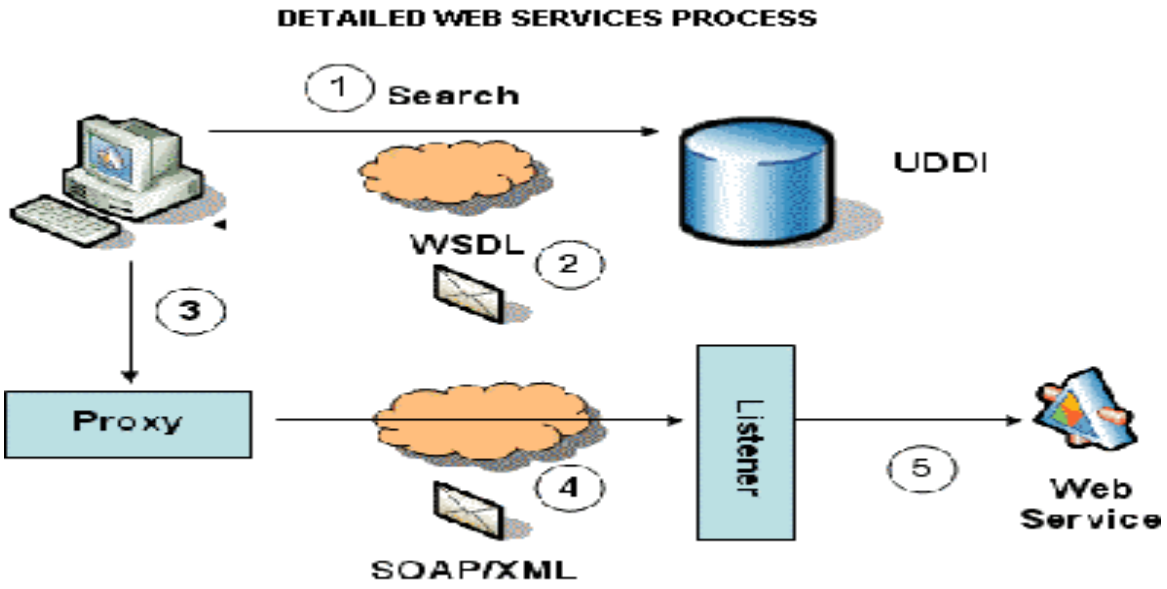


Figure 17. The process flow of a web service

The performance of web services can be summarized in the following steps:

1. A system needs to find a web service to perform a determined task. In order to find it, the system searches in the UDDI directory for a web service which fits its needs.
2. Once that the web service has been located and selected, the wsdl document of the service is sent to the client. Now the client knows how it has to deal with the service and how it has to structure the data.
3. The client prepares the data and sends it through the network. The data is sent in XML format in a SOAP package.
4. The web service receives the package and checks if the data is correct. If the data agrees with the WSDL document the package is processed.
5. The web service performs its actions and returns the result (if any) to the client in another SOAP package. In order to perform the actions, the web service could also contact with other web services through the web.

A similar scenario could happen without the UDDI directory. If it happens, the client has to previously know where the web service that it wants to consume is and where the service's wsdl document is. All this steps are depicted in figure 17.

Security

SOAP web services can add a layer of security to their messages. This can be done with one of the multiple extensions that SOAP provides. The extension for security is called WS-Security. This extension contains specifications to guarantee the integrity and the security on the messages. This protocol includes support to use SAML and Kerberos. It also manages X.509 certificates.

WS-Security provides the security information in the SOAP's header, working in the application layer. This allows an end-to-end security.

The data integrity and the confidentiality could be guaranteed with the usage of the transport layer security (TLS), sending messages over HTTPS. This could reduce the overhead in a significant amount. Issues related to security and performance in web services will be explained in annex IX.

REST (Representational State Transfer)

Rest web services are based on Roy Fielding's PhD thesis:

"REST enables intermediate processing by constraining messages to be self-descriptive: interaction is stateless between requests, standard methods and media types are used to indicate semantics and exchange information, and responses explicitly indicate cacheability"[38]

Rest is not a standard and it does not define any format: it is a way to develop web services. Rest is based in HTTP and it works with the HTTP's methods: PUT, GET, POST and DELETE.

In SOAP the client communicates with a service to obtain information, there is only one service for all the information (they have to use parameters in the call to specify the desired information). In Rest you directly access to the information; every resource equals to "one" web service. In addition, it is faster and conceptually simpler than SOAP and it does not require any special tools.[39]

In Rest, every object (resource) is denoted by a URI. The URI contains a description of an object. Therefore, the URI has to be descriptive and meaningful. Every request from client to server has to contain all the information necessary for the request understanding, there is no state info held in the server.

Nowadays Rest has the support of the main web companies like Google, Facebook, Ebay, Yahoo or Microsoft. Most of Web 2.0 developments are based on Rest. However, Rest is relatively new and it has to improve aspects like Security.

Web Service advantages

- Interoperability: Web services are platform and language independent. This allows services deployed by different companies in different ways interact together. This feature has been the reason of the web services' success.[40]

- Usability: Companies can choose what service they want to use. There is a wide variety of services with different implementations of the same web service. Companies can evaluate the different services and choose what service fits their needs. This is very useful in the today's business world. New demands and needs appear every day and web services allow to change your company's way of work quickly and easily.

- Reusability: Software and services from companies located in different parts of the globe can be easily combined and adapted.[41]
- Standardization: Web services encourage protocols and standards based in text. This makes more straightforward the access to its content and the understanding its behaviour.

Disadvantages of web services

- Web services technology is still not good enough to perform transactions. Standards like CORBA are more developed in this area.

- Performance. The performance of web services is not optimal. The reason is the adoption of standards based on text. These standards (XML) do not look for efficiency and speed. Adding security reduces the performance even more.

- HTTP based: They are based on HTTP with the consequences that it implies. Some security issues related with firewalls can affect the proper performance of this technology.

2.3 Automated Scheduling

In this project, web services are used to obtain a rostering solution for a leisure centre. The project is not focused in how the roster is calculated and created. Another topic could have been used instead of a rostering service. The adaptability of web services to different problems is what is wanted to demonstrate.

However, it is important to have significant understanding on scheduling and rostering in order to obtain and manipulate an optimal solution.

A possible definition of Scheduling could be:

"*The allocation, subject to constraints, of resources to objects being placed in space-time, so that the total cost is minimised*" *[1]*

A definition of rostering could be:

"The placing, subject to constraints, of resources into slots in a pattern." *[2]*

The result of the rostering process is a roster. A roster is a list of people which shows where and when they are going to work. A roster can created for different periods of time: days, weeks or months.

Shift scheduling problems are found in a wide range of industries and settings. These include airlines, airports, armed forces, call/contact centres, emergency services (police, fire and ambulance crews), factories, healthcare (physicians and nurses), hospitality, retail, security personnel, transportation sector (train and bus drivers)[3].

Automated scheduling problems usually involve an organisation with a set of tasks that need to be fulfilled by a set of employees, with their own qualifications, constraints and preferences. The organization enforces some overall regulations and attempts to achieve some global objectives such as lowering the overall cost, or an equitable division of work among certain employees [4].

Automated scheduling problems have many diverse formulations: they can be based on predefined workload patterns or can be simple assignment problems or they can be problems that include many forms of complex constraints and objectives.

The main object in these problems is the shift. It is the time period where a task is performed and it is fixed in time. The term automated scheduling refers to the process of assigning employee to task in shifts.

In the commercial world of today, higher degree of job satisfaction is a fundamental factor for the success of a company which implies the consideration of personal constraints in the problem model. Adding more flexibility for the staff generates personal constraints and results in problems even harder to

solve. This problem has attracted the attention of researchers that have mainly pointed out that local search methods perform well to solve even the hardest instances.

There is an extensive literature about scheduling. Scheduling theory has been studied for many years. Scheduling is very complex and nowadays there are several scheduling problems without an answer. Normally scheduling leads to non-deterministic polynomial problems (NP problems).

This complexity makes that the bridge between the theory and its application to be difficult. Obtaining the optimal solution to certain problems so that they can be commercialized requires a long time.

Scheduling problems are classified based on machine environment (α), job characteristics (β), and optimality criteria (γ). A scheduling problem is formulated with different values for these three parameters. Each parameter takes values to represent the problem's characteristics. There are well-known problems with optimal solutions, when a company desires to schedule its workforce it has to model all its characteristics with these three parameters. Searching for similar working models is useful because it saves time and resources.

Rostering is a complex process that has to look at different types of factors. Traditionally, rostering has been a management's task. This increasing complexity has led to software solutions with which the manager can obtain a roster with only a few mouse clicks.

These software solutions automate the process of creating and maintaining a roster and it has become an essential part of business. This software is also very complex and allows the management to consider different factors such as preferences, sick times, vacation times or conflicts between employees. The functionality of this software goes beyond rostering and it also allows the management receive different results as feedback, analysis of productivity and it also can be used to automate the payroll.

Nowadays, most companies are looking for upgrade their labor-scheduling capabilities. They know this is a key point in the organization's structure and on it depends the proper functioning of the company.

The IBM's paper "Good timing: Realizing value from investments in labor-scheduling" [42] shows two main reasons for this upgrade. The first of them is the changing business conditions. New ways of doing business are appearing and they require a change in how managing is performed in the enterprise. The paper mentions different forces which are changing the managing. The more relevant for the rostering process are:

- Shift to a services-oriented economy: demand fluctuates and new needs appear every day. Customer is now the crucial objective of many companies and they need a special treatment which increases the complexity of rostering. New non-traditional services are created around the customer and they add an extra level of difficulty to the process.

- Changing workforce demographics: The way of working has changed during the last years and in the recent past it has been affected by the economic instability. This has generated new workforce demographics with a wide range of ages. This new workforce has different preferences for shifts and part time job is more common than 5 years ago. This leads to more complex rosters.

The other reason that the paper mentions is the boom of the new technologies. New technologies have democratized the information's access and this has allowed that even small enterprises with few resources can take advantage of rostering systems.

The usage of these systems provides to the company with a significant cost reduction. At the same time it benefits the employee, who sees how his/her preferences are taken into account. A good relationship between management and employees is necessary because it has an important impact in the customer satisfaction. The customer will have a better service and his/her needs will be achieved due to the good organization in the enterprise. In summary, a revenue enhancement will be accomplished with this kind of systems.

For an SME with more than 5 employees and different duties, the rostering process could be a nightmare if it is not automated. These systems could set the competitive edge between two companies. SMEs pay special attention to their resources and this kind of software allows them to reach optimum performance while they maximize their resources.

2.4. Small and Medium Enterprises

SME is the recognised abbreviation for Small and Medium sized Enterprises. In UK, sections 382 and 465 of the Companies Act 2006 define a SME for the purpose of accounting requirements. According to this, a small company is one that has a turnover of not more than $6.5 million, a total balance sheet under $3.26 million and less than 50 employees. A medium enterprise has less than 250 employees, a maximum turnover of $25.9 million and a total balance sheet less than $12.9 million. [43]

99% UK's business network is composed by SMEs. 59.8% of them belong to the private sector employment and 49% to the private sector turnover. In addition, SMEs employs 59.8% of the UK's workforce and they represent 35.7% of the turnover [44]. These percentages are depicts in figure 18.

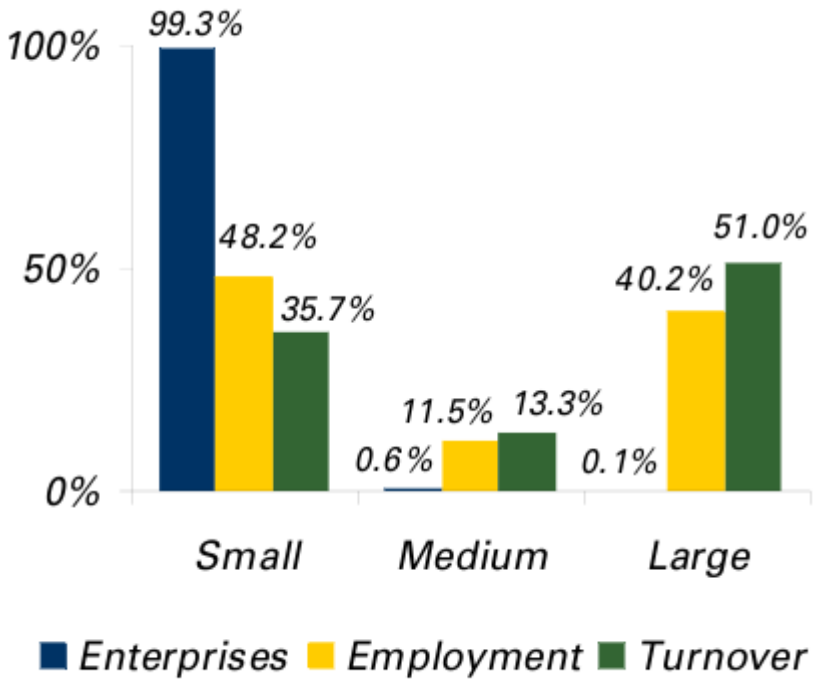


Figure 18 Share of enterprises, employment and turnover by size of enterprise UK private sector, start of 2009

Nowadays all the enterprises have to deal with technology. Technology and IT are crucial for business and they could set the difference between success and failure. A few years ago, SMEs had significant problems with technology but now, with the Internet's and the cloud's boom is easier than ever to seize all the possibilities of the network.

Cloud computing is low-cost, easy to implement, flexible and scalable. SMEs can get the benefit of all these qualities. SMEs can improve their productivity and quality with them; it is estimated that they could build applications 5 times faster and at half of the cost than using traditional software platforms [45].

Developing in the cloud allows the new SMEs to reduce start-up costs as they do not have to buy complex software and hardware solutions.

Instead of closed software and proprietary systems is easy to use open systems and open solutions written in widely used languages which makes easier to update them and create new features. Flexibility is becoming important as well, it has to be possible to access the applications from different devices and environments (Mobiles, Pda, Laptops...).

The key point in this approach is keeping everything simple and focusing on your business. Cloud computing allows the enterprise to get that point and to obtain an easy configuration for its software. Let the complex problems of configuration and deployment to the specialized company and focus the enterprise's efforts in doing its best at its business field.

Among all the cloud's technologies there is one that is increasing its position and is worldwide used by different types of systems and business: web services. As it has been discussed in the previous section, they incorporate open internet standards-based APIs using SOAP and REST, which makes them worldwide accessible and allows the SMEs to use systems of different enterprises very easily

However, research shows a lack of understanding over IT issues amongst SMEs [46]. The survey was made to 1000 IT decision-makers in small enterprises. The results show that while 96% of UK SMEs think IT to be essential for the business, a 25% of them recognised that they do not know enough about IT and how to implement it in their activities.

One sensible point to consider in this scenario is the security of the managed information. The Information Commissioner's Office (ICO) is the responsible to monitor and investigate this kind of problems. If some irregularity is detected, an investigation will be opened and monetary penalty fine could be imposed to the enterprise (up to $500,000 [47]). It is important to declare a policy about data security and it is even more important if the data travels through the network. Normally, the use of web services and cloud technologies implies confidential data travelling, so a clear and safe policy must be implemented.

3. DESCRIPTION OF THE PROBLEM

3.1. SMEs and SOA

SOA is not easy to implement. SOA is an architecture and it implies the application of design methodologies, new tools. Taking decisions thinking in the medium and long term is also necessary. The way of working has to be formalized. For new SMEs, this could not be a problem due to the high flexibility that they have. However, for an established SME the change could be a rough task.

With SOA, the IT department has to start to thinking in business. Up to now, the IT department was focused on developing new solutions for the enterprise but it maintained a distance with which they developed. They had a good understanding of what they were doing and how it affected to the enterprise. Now, it is completely different: SOA makes the IT department to become the main department in the enterprise. Therefore, the profile of the employees in this department changes and new qualifications are required.

Unfortunately, sometimes, the cost exceeds the benefits. It is hard to quantify the benefits and most of times they are intangibles. The reason to change to SOA model has to be a strategy with defined objectives and a serious evaluation of all the possible problems.

At this point, an SME could refuse the SOA idea. Does this mean that they cannot use web services? The answer is NO. They can add web services to their business. Once, they have differentiated between SOA and web services, the next steps are much simpler. [48,49]

Developing web services could be an easy task but it could become in a serious problem. The difference between both cases is how the SME has analyzed the problem and its multiple questions (SOAP/Rest). If the study is serious and realistic, no problem should appear.

The words "Web services" sound attractive and a SME could be influenced by new fashions and business trends and try to add this technology without any clear reason. Eventually this will be a problem for the SME and it will make the SME lose money, time and hours of work.

SME should seize all the possible tools to achieve their goals. In this case, tools like NetBeans and Eclipse offer plugins and tools to create web services easily. This open-source environment is very useful for experimentation and learning before acquiring expensive and professional solutions. [50]

3.2 Description of the problem

Arturo Castillo, MidlanHR's collaborator describes the company and the problem they have with the following words:

"MidlandHR a software provider for people management solutions is looking to explore the advantages of web services when used to deliver staff allocation functionality. In order to achieve this objective a re-engineering of the prototype module would be necessary as well as the test of several features involved when using web services.

MidlandHR is an independent company founded in 1984 to meet the needs of payroll and people management, MidlandHR has rapidly expanded its product and service portfolio to become a leading provider of human resource management software and services. The company's range of HR Management solutions allows organisations to incorporate all human resource activities - from payroll to personnel - into a single system. MidlandHR's iTrent, is a truly integrated HR and Payroll solution and can be implemented in a number of ways: in-house software, full application outsourcing, varying degrees of bureau services, and fully-managed payroll service with HR applications

For the last 5 years MidlandHR has work closely with customers to enhance its module on personnel management. Personnel Management spans across areas such as staff scheduling and allocation. The functionality required to explore at the moment is the one related to automatic staff allocation (ASA). MidlandHR has developed a prototype system to test ASA among several customers.

Currently, MidlandHR is facing several problems when trying to retrieve data from customers that have used the prototype. This is because, customers are not fully trained to use it and therefore they often input incorrect data. When asked for their feedback, customer's response has been poor. The company is keen to put the prototype on the web so that customers don't have to install it on their infrastructure. By using the web, MidlandHR will remove administrative tasks for its customers in the hope they could test the prototype. Lastly, it will help incorporate other customers that are willing to participate in the development of the system, since on the web no further installation or maintenance is required on their side".

3.3 Proposed Solution

The proposed solution consists in creating a service oriented scenario where the performance of the rostering web services can be measured and tested. This scenario is a direct translation of the traditional stand-alone scheduling application to a service oriented approach. This will provide a first analysis of how useful this approach could be. In this scenario, MidlandsHR provides rostering solutions through web services and one enterprise consume that web services to obtain rosters for its workforce. In this project the enterprise which requests the services of MidlandsHR is a leisure centre.

Two infrastructures will be created: one for the MidlandHR's side and another for the Leisure centre's side. The MidlandsHR's infrastructure will create the rosters and will provide web services in order to obtain a roster.

In the other side, the leisure centre's infrastructure will consist in one web application to provide internal data to the Internet and one application to manage the employees and working demands of the

enterprise and to obtaining working schedules. The data provided by this enterprise will be used by MidlandHR to create the roster

3.4 Aim and objectives of the solution

Overall, the aim of this project is to explore how Web Services and distributed technologies could help in this particular situation. There are different possibilities to achieve this aim; this project focuses on performing a proof of concept of how the serviced oriented paradigm fits with the rostering process.

This new approach can report several benefits to rostering enterprises. Up to now, when a company needs a rostering solution it is in the most of cases a stand-alone and expensive suite. Besides it needs to be installed by experts and the maintenance is complicated. This centralized approach limits the potential of this kind of applications.

Nowadays, the companies do not want problems with IT related issues; they just want to pay and obtain a service. They do not want installation processes, regular updates and an inefficient technical service.

The distributed approach aims to prove that these new ways to doing business through the Internet are also possible in this area. The company only has to contact with the rostering enterprise and provide the details of its workforce. In the easiest case a web address will be returned to the company. They only have to check the address and follow the steps to obtain a roster. In this approach all the upgrades and changes happen in the rostering enterprise. A great number of possibilities appear; the new web technologies allow to create an interactive scenario where most of the companies' rostering issues could be resolved. In addition, a web scenario could collect all the services consumed by the company, centralizing the information and easing their access.

The approach also provides a new role to the employees, now they can obtain their roster everywhere 24/7, this improves the communication flow between the companies and their employee

MidlandHR has set the following objectives in this project:

- Change the architecture of the system so that its functionality could be accessed through the web
- Register every user (customer) in the system.
- Create a log of user history, this should at least include:
  - Number of runs,
  - information about the scenario (people, absences, etc)
  - And, if a solution was achieved or not.
- Explore at least two different kinds of web services delivery: REST and SOAP,
- When evaluating SOAP based services explore the different security standards that could be adopted.
- Finally, performance related issues
  - Number of simultaneous users
  - Memory required on the server side
  - Average time required for finding a solution

Throughout the project all these objectives are achieved and their resolution will be explained in the different annexes of this dissertation.

Success in the deployment of the ASA prototype over distributed technologies such as web services will also allow MidlandHR to consider developing the final ASA module in the same way.

3.5 Limitations of the proposed system.

The developed system has tried to be as complete as possible. However, in some parts the completeness has been limited for several reasons:

- Design. The design of the web service page and the client's application. The design will be something artless but very easy to understand and guided in order to give chances to the user.

- Roster presentation. The way the roster is presented corresponds to the prototype given for the enterprise. Some improvements could have been done, but these improvements had not changed the performance of the system.

- Security. The certificates used by the web services are provided by the Glassfish server and they are limited to a develop environment. The possibility of create a new certificates was considered but discarded. Creating new certificates and adding them to the server is not complicated but it implies a configuration process that could led to problems with the IDE. Some certificates were created and signed by Verisign [51], however the life of these certificates is short (1 month) and load them in the server had implied to change them monthly. Consequently, the configuration process would have had to be repeated monthly, losing time for the development of the process. Finally in order to test the whole system a X.509 authority was created to sign our own certificates.

- Configuration: The submitted version is configured to work in *localhost*. All the web applications and all the web services will be working in the same computer. The database will be in the same computer as well. The application has been tested in a real scenario with different computers.

3.6 Requirements

The system is divided in different applications (Annex II). Each of these applications has their own requirements and they will be showed the each application's annexes. All these requirements are dictated by MidlandHR.

However, before starting the development of the system, a series of global requirements were presented. These requirements need to be taken into account when designing and implementing the proposed system and its different parts:

- Obtaining a complete work schedule from the data entered by the user.

- Detecting inconsistent data entered by the used.

- Two different users: Manager and Employees.

- Employees can change their availability in the system.

- Two different applications to obtain a roster: Stand alone application and web page application.

- Performance through web services (SOAP and Rest)

- Multiple users accessing the system at the same time.

4. POSIBLE SOLUTIONS

Web Technologies' world is full of different tools. All of them have their advantages and disadvantages. This point aims to explain why the chosen technologies have been chosen and to explain their main differences against the similar ones.

4.1. SOAP vs REST

In this project both approaches have been implemented. The comparison about their performance is located in annex X. However, time and resources are limited and it is not always possible develop and implement both approaches. Therefore it is necessary to choose one.

The election of a suitable solution is crucial; otherwise the consequences for the SME could be disastrous. In addition to having a general knowledge of each one, it is important to know the common aspects, the differences and what makes each of them better than the other.

RESTful web services have shown a high capability to response to the requirements of publication and syndication of media and content. On the other hand, SOAP fits better in major solutions with large requirements: more services, more applications, complex messages, etc; it is suitable for business solutions.

As has been noted before, REST has simpler and lighter messages and it has better response times than SOAP. However, REST is limited to HTTP while SOAP could use another protocols like JMS. SOAP also has a greater level of security due to the WS-Security features. These features are located in the WS-* stack [52], which provides specifications to the SOAP messaging. REST does not have this kind of specifications, but new projects to develop them are appearing.

In order to choose one solution, this list with a comparison of the main advantages and disadvantages has been compiled [53, 54]:

SOAP	REST
Advantages • W3C Standard: Compatibility and interoperability with another W3C standards • Different transport protocols: Reliability, Security and availability • Easy development and usage of XML • WSDL service description. Global solution to describe web services. • Extensibility • Development tools • Rigid, type checking, adheres to a contract.	**Advantages** • Response times • Scalability • No discovery needed • Authentication through HTTP • Easy to build, no toolkits required • Human readable and configurable results. • Thinner clients
Disadvantages • Efficiency • Information cannot be cached.	**Disadvantages** • No suitable for B2B applications • Lack of error messages • No support for attachments

In both cases and before choosing one, it is important to think about where the complexity of our system will be: Server or client.

4.2. JAX-WS vs AXIS2

In this project, JAX-WS has been used. JAX-WS (Java API for XML Web Services)[55] is a Java API for creating web services. It is part of the Java Web Services Development Pack. The Reference Implementation of JAX-WS is developed as an open source project and is part of project GlassFish, an open source Java EE application server. It is also part of the Metro Stack [56] and WSIT[24].

In the other side, AXIS2[57] is a XML based Web service framework. It provides an API to create and develop web services. It has been developed by the Apache Foundation.

Deciding what framework choose use to be one of the first decisions that should be taken. This decision is much related with the application server that is going to be used in the system. Normally, Axis2 is chose when an Apache server is used and JAX-WS is chose for Glassfish servers. However, it is possible to use both in both servers.

In this project JAX-WS has been chose due to the Glassfish server and its interoperability with NetBeans. The IDE creates JAX-WS services by default. However, several sources [58, 59] indicate that JAX-WS is one step ahead Axis2 in issues related with generated code and readability.

4.3. Server

Glassfish has been the server used in this project. The motivation for using this server has been its total integration with the NetBeans environment. In addition, Glassfish implements the JavaEE5 platform. This platform supports technologies like JSP, JSF, Servlets, EJBs, Java API for Web services (JAX-WS), Java architecture for XML (JAXB), etc.

JBOSS [60] would have been the main alternative to Glassfish. JBOSS is a J2EE application server. It is open-source and it is implemented in Java. This server has been proved to be a very good option for enterprise solutions and it has the confidence of the business world.

What makes both solutions suitable for the deployment of web applications in a development scenario is their user community. These communities offer manuals, tutorials and help. These communities also develop side projects to improve the capabilities of the servers (Metro or Hudson in Glassfish).

4.4. Existing Systems

The project has been limited to develop a series of web applications and web services and put them in a server. This is enough for a concept test and to study how these applications would suit in the enterprise. The aim of this project is not to develop a professional solution; it is study how automated scheduling and web services could work together. Therefore, the first step in order to achieve our goal is create a simple environment where web services work and could be possible perform efficiency tests. If the tests and the system are successful, the next step would be use one of the following platforms to implement the system in a professional environment.

If the study meets the expectations and the enterprise desires add these services to their working systems more powerful tools are needed. The project development, without using any professional tool for web services, will give us a good overview and understanding of how web services and rostering could work together. If the results are successful, using one of the next tools will always be beneficial for the development.

There are several platforms which could serve to this aim. These platforms could be divided into corporative and open-source solutions:

Corporative Packages:

HydraSCA [11]
It's a framework focused on the development of C++ web services. This package gives special importance to XML, SOAP and WSDL, generating automatically skeletons for the web services' description. The main feature is the capacity to incorporate new standards or integration requirements without disrupting the rest of the application.

IBM Pack for SOA [12]
This package includes: WebSphere Portal Server, WebSphere Portlet Factory, Lotus ActiveInsight, and Lotus Forms Server. The combination of these products allows the user to work in a service orientated paradigm focused on the services by themselves not in the implementation of them. The package allows the user to separate the different layers of the SOA business logic and adapt the existing services.

Oracle SOA Suite [13]
It's a suite which integrates the necessary products to design, develop, assemble and manage web services. The suite works 100 percent standards-based and operates with existing web services. The suite includes Oracle Service Bus which gives the user a high scalability. The main features of this suite are: an integrated SCA-based designer (drag and drop features), unified events and services and end to end instance tracking which gives the user a total visibility of the implementation.

Open source Packages.

Apache Tuscany [14]
The main aim of Apache Tuscany is simplify the development of SOA solutions. The package provides a model to create the following: composite applications in an easy way, reusable services of the business layer, easy adaptable applications through pluggable bindings and a modular architecture to integrate different technologies. It allows components in Java, C++, BPEL, Spring and scripting.

Fabric3 [15]
This suite is the first one to comply with SCA Assembly 1.1 and it is architected by one of the SCA inventors. It provides the user with Enterprise-class features typically found only in commercial offerings. The main feature of Fabric3 is its capacity to work with reusable services generating other services which can be reused and so on. Fabric3 creates modular services which can run in diverse middleware environment without major changes

SCOrWare [16]
This project is very focused on the standards and their accomplishment. This package consists on a runtime platform and a design/development assistance tool. It supports a binding factory for different protocols, a transaction service, a semantic trading service of SCA components and it is integrated into Eclipse.

In order to choose a platform the most important features to look for are:

- Support the SCA and SDO standards and specifications.
- Ability to deal with different types of bindings, components and technologies.
- Lightweight runtimes.
- Compatibility with another open source tools which can be very useful in the implementation of a web service such as Apache Tomcat and Eclipse STP/SCA
- Composite Designer

Apache Tuscany and Fabric3 could be suitable for a SME because of the open source feature of the packages which fits into the structure and limitations of an SME. The commercial packages are very

expensive for an SME and they go beyond the SME's environment. SCOrWare has not been chosen because it is in a very preliminary phase and the documentation is not enough to make a project accessible for an SME.

ANEXO II – SYSTEM DIVISION

1. SYSTEM DIVISION

The system is divided in three parts. Each of these parts corresponds with the different physical scenarios where the system is deployed.

The figure 19 illustrates the parts of the system:

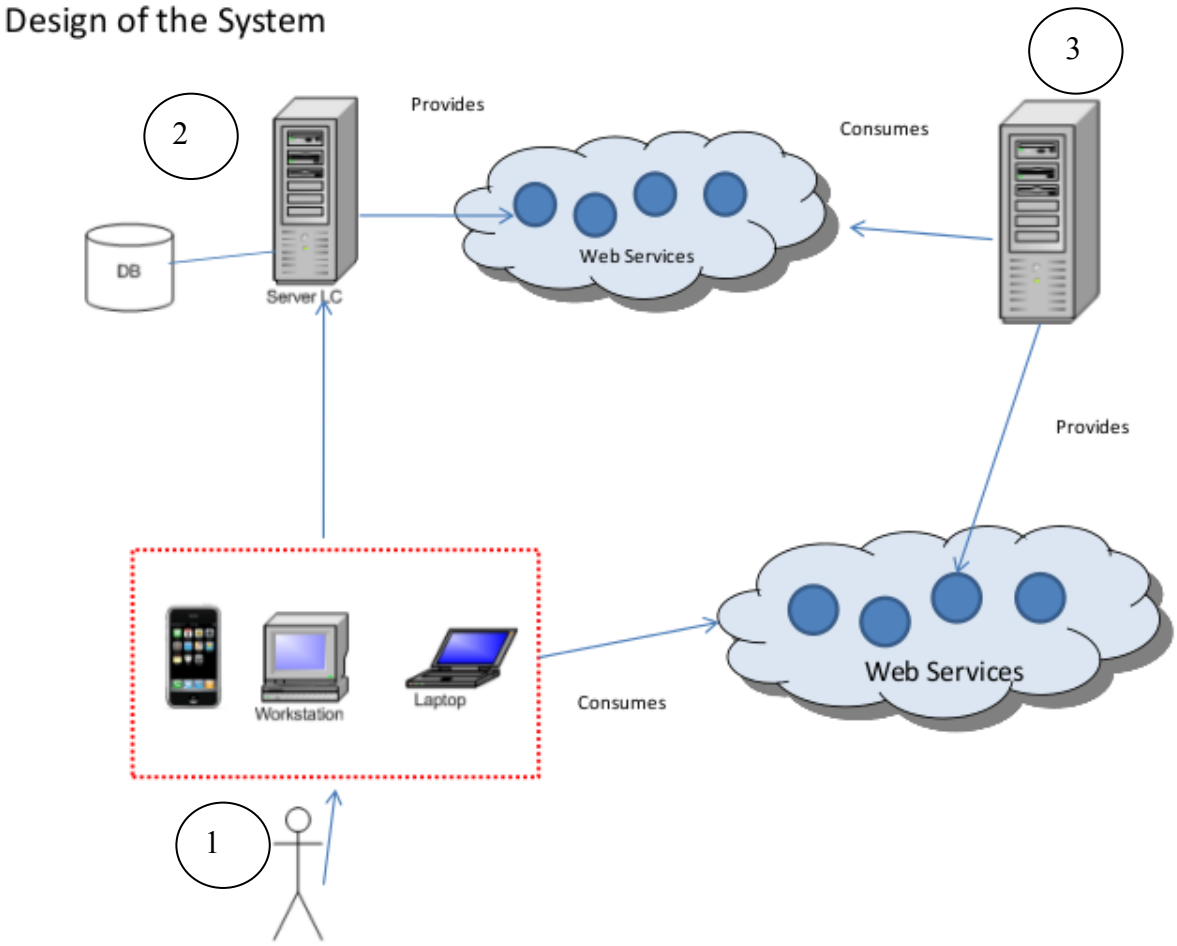


Figure 19 System's parts.

The first part has not a fixed location. The manager and the employees of the centre will use a desktop application to obtain rosters. The manager will also be able to use the application to add, modify and delete employees of the system and manage the work demand of the centre. The application could be located in any computer; it does not need to be in the leisure centre.

The application is called RostApp and there are two versions, one which works with SOAP web services (Annex III) and another which works with REST web services (Annex VII)
The SOAP version has three different versions which work with different levels of security (see annex IX)

The second part is located in the Leisure centre's server. This server could be in the same location that the computers where RostApp is executed or in a different place. The database of the leisure centre is located in this server.

The server contains two applications:

LeisureCenterServer: This web application provides a web page to the manager and the employees to obtain a roster (Annex VII).

LeisureCenter: This web application provides data from the leisure centre to third parties. In this project the data is provided to MidlandHR in order to create a roster. The data is provided through web services. There are two versions of this application one with SOAP web services (annex V) and one with REST web services (chapter VII)

The SOAP application also has three different versions, one for every security configuration.

The third part is located in the MidlandHR's place. MidlandHR is the enterprise which creates rosters. MidlandHR offers a web service to request a roster with the possibility of selecting the dates of the roster. In order to provide the roster one application has been created, the application is called MidlandHR. This application consumes the web services provided by the leisure centre's application to calculate the roster.

There is also two versions for this application one for SOAP web services with 3 different security configurations (see annex IV) and one for REST web services (see annex VII).

A normal interaction with the system would be:

1. The manager wants to calculate the roster for the next week, or the employee wants to know what are his/her working times for a period of time. The user runs the RostApp application and chooses the roster option, selects the dates and asks for a roster.
2. MidlandHR receives the roster request and starts to calculate the roster. In order to calculate the roster it needs data from the leisure centre, therefore it consumes the leisure centre's web services to obtain these data.
3. The leisure centre receives the request for the enterprise's data and answer with the data to the MidlandHR's requests
4. MidlandHR obtains all the data and creates the rostering solution. The solution is sent to the client.
5. The client receives the solution and visualizes it in their computer.

In the first step, instead using RostApp they could have used the web page that the leisure centre provides to their manager and employees.

In the next annexes each part will be explained in detail (requirements, analysis, design, implementation and tests).

ANEXO III – ROSTAPP

1. ROSTAPP

ROSTAPP is the application created for the leisure centre. This application allows the centre to manage the employee's data, the work demands and the availabilities. It also allows to obtain rosters through web services.

1.1. Requirement specification

Functional requirements

Requirements related with the usage of the application.

- **FR01 – Users.** Two types of user are distinguished: Manager and Employee
- **FR02 – Add Employee.** The application will allow to add new employees with their own data (personal, availability, marks).
- **FR03 – Modify Employee.** The application will allow to modify the data of the employees. These modifications can be done by the manager (modification of all the data) or by the employees (modification of their availability).
- **FR04 – Delete Employees.** The application will allow to delete employees entered in it.
- **FR05 – Obtainment of a roster solution.** The application will allow the obtainment of complete roster solution through the consumption of the web services provided by MidlandHR.
- **FR06 – Views.** The information from the roster could be visualized according to three different criteria: Task, Person and a mix of task and person organised by time.
- **FR07 – Add demand.** The application will allow to add new demands for a particular workstation.
- **FR08 – Delete Demand.** The application will allow to delete the demands previously added.
- **FR09 – Add Availabilities.** The application will allow to add availabilities for each employee.
- **FR10 –Modify Availabilities.** The manager can modify the availabilities. The employees can change their presence in the availability
- **FR11– Delete Availabilities.** The application will allow to delete the availabilities previously added.
- **FR12 – Modify skills assessment.** The application will allow to put values for the skills of each employee in the different workstations. By default, if no value is added, the skill is marked with 0.0

Non Functional requirements

- **NFR01 – Concurrency**. The application could be used by several users at the same time.
- **NFR02 – GUI Usage.** The application will be totally managed by a GUI. This GUI will allow the user to manage all the functionalities described in this document.
- **NFR03 – Help**. The application will provide the user with a "help" option with which the user may resolve all the problems that he found during the usage of the application.
- **NFR04– Oracle database.** The system provided by the client will count with an Oracle database in one of their machines or in a remote server.
- **NFR05– Java Virtual Machine 1.6.** The computer in which the application will be executed must have the version 1.6 of the JVM.
- **NFR06 – Access password.** To start the application a password will be needed. Each user will have a personal password.
- **NFR07 – Internet Connection**. The application must have access to the Internet.

1.2. Object Model

In this part, several diagrams will be shown in order to define the object structure of the application.

1.2.1. Class Diagram

This diagram represents the class organisation of the application. It shows that there are two types of classes: entity classes and managing classes.
The entity classes only have attributes and getters and setters for them (they have been omitted in the diagram). The managing classes contain with methods to manipulate the entity classes. The figure 20 corresponds to the class diagram.

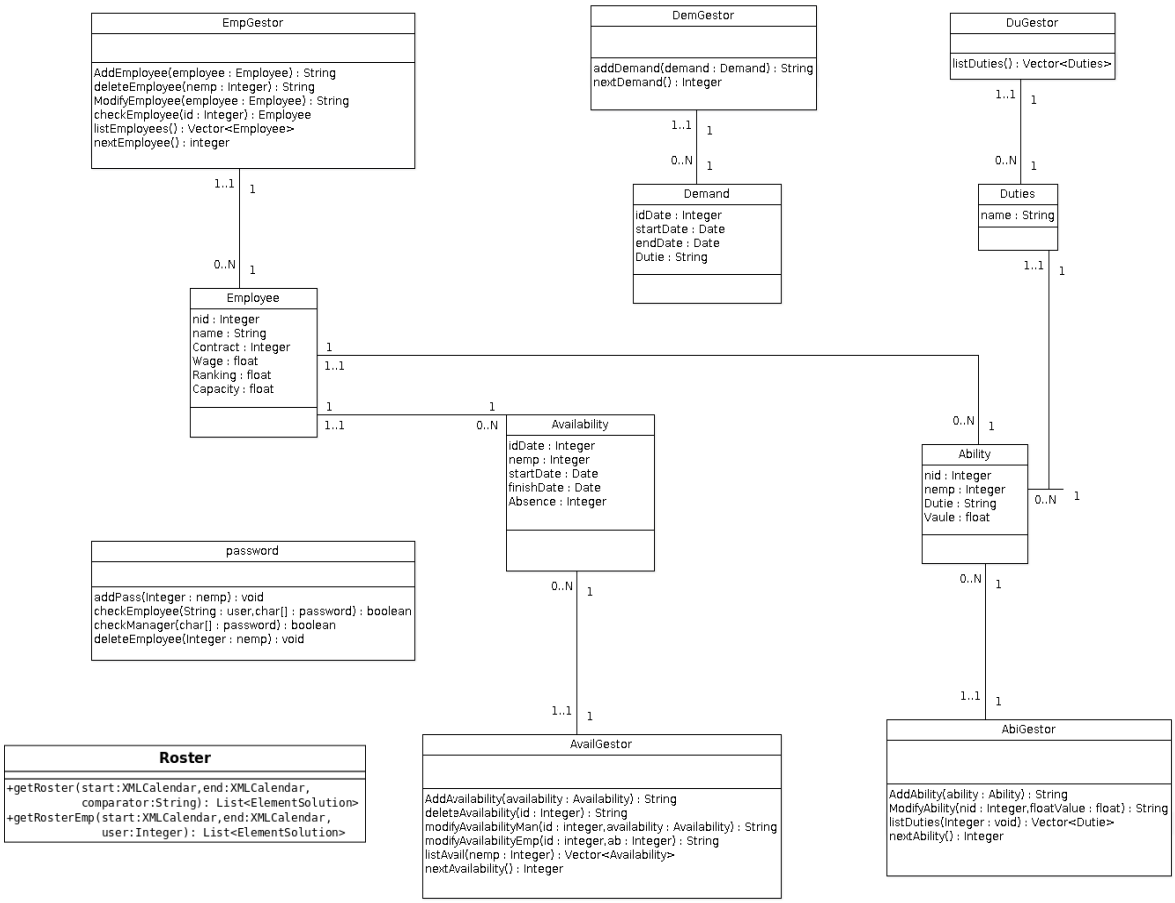


Figure 20 RostApp's class diagram

1.2.2. Class dictionary

In this section each class and its attributes are described.

EMPLOYEE

This class represents the enterprise's employees. Each employee has different characteristics and availabilities that will be taken into account when the roster is elaborated.

ABILITY

This class represents the values assigned to the employees' skills for each workstation.

AVAILABILITY

This class represents the availabilities of each employee. In the system, availability represents the temporal period of time in which the employee is not available to work.

There is a special attribute called Absence which can be modified by the manager or by the employees. It allows the employees to inform the enterprise that they will be able to perform that shift (cancel the availability)

DEMAND

This class represents which shifts need to be completed by the roster. It contains the basic data of shift.

DUTIES

This class represents the different duties performed in the enterprise.

ROSTER

This class obtains a roster through MidlandHR's web services.

PASSWORD

This is a static class which stores the user's passwords of the application. It checks the typed passwords and allows or blocks the access to the application.

MANAGERS

These classes will not have any attribute and their function will be to manage all the data and instances of the classes described before.

1.3. Dynamic model

1.3.1. Dynamic model's objectives

The system to be shaped in this software project is a transactional system of queries and operations related to databases. The dynamic evolution of the classes is not a main factor in the analysis model itself. The classes of the system will not have a very sophisticated dynamic model and they will have the basic operations of creation, modification, elimination and some operations to return attributes.

The use cases and scenarios of the system will be shown

1.3.2. Use cases

The use cases will catch the potential requirements of our system. Each use case provides one or more scenarios which show how the system should interact with the user or with other system to accomplish one specific objective. The scenarios will be show in the next section. Figures 21 and 22 depict the use cases.

- Manager

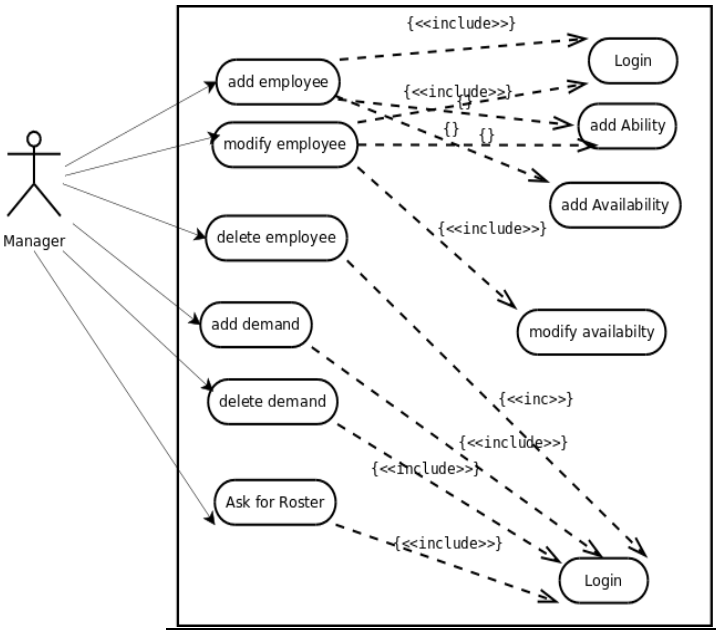


Figure 21. Manager's use case

- Employees

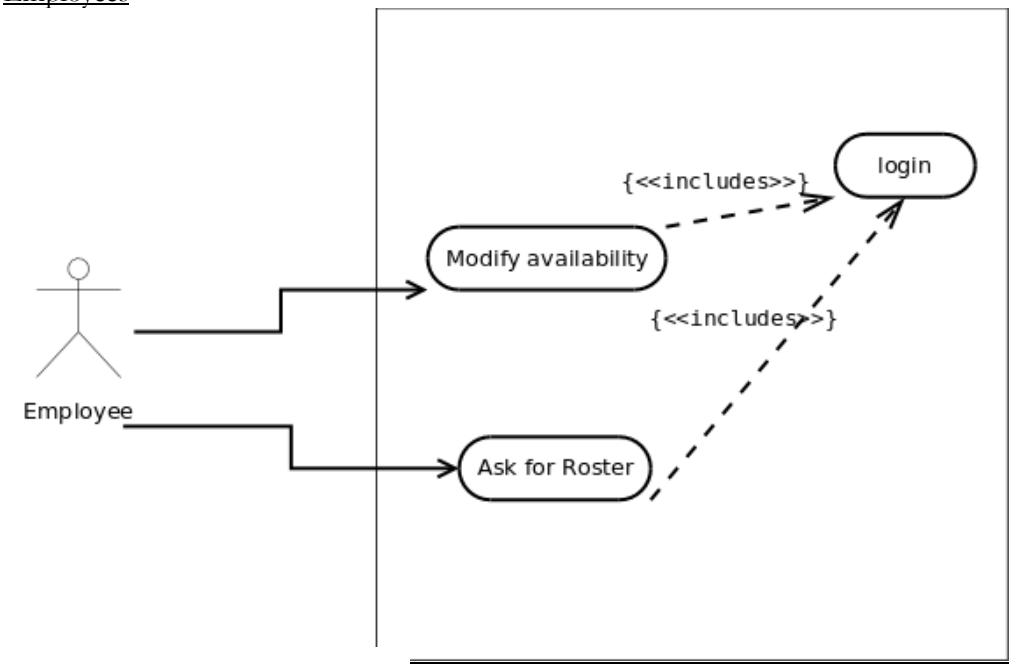


Figure 22. Employee's use case

1.3.3. Scenarios

Scene 1 – Start of the application

Normal Situation
The user starts the application.
Identification of the user.
The user types his user name and password.
The application starts to work,

Exceptional situation
The password is wrong. The application asks for the user name and password again.
The user name is wrong. The application asks for the user name and password again.

Scene 2 – Creation of an employee.

Normal situation.
The manager needs to add a new employee.
The manager creates a new employee and fills all the data of the employee's data.
The new employee has been created.

Exceptional situation
Some of the given data have a wrong value.
The manager makes a mistake typing the data.
The application warns about the wrong data and allows the manager to type the data again.

Scene 3 – Delete an employee.

Normal situation.
A employee has been dismissed or he has abandoned his job.
The manager selects the employee.
The manager deletes the employee.

Scene 4 – Modification of employee's data

Normal situation
The data of one employee needs to be modified.
The manager looks for the employee's information.
The manager modifies the data.

Exceptional situation
Some of the given data have a wrong value.
The manager makes a mistake typing the data.
The application warns about the wrong data and allows the manager to type the data again.

Scene 5 – Adding availabilities to an employee.

Normal situation
The manager needs to add new availabilities of an employee.
The manager starts the application.
The manager adds the availabilities.

Exceptional situation.
Some of the given data have a wrong value.
The manager makes a mistake typing the data.
The application warns about the wrong data and allows the manager to type the data again

Scene 6 – Adding abilities to an employee.

Normal situation.
A new activity has been added in the enterprise or it is needed to modify the skills of an employee in the performance of an activity.
The manager starts the application.
The manager adds new reviews or modify some of them.

Exceptional situation
Some of the given data have a wrong value.
The manager makes a mistake typing the data.
The application warns about the wrong data and allows the manager to type the data again.

Scenario 7. Modifying data of an employee (availability)

Normal situation
An employee needs to modify his availability
The employee starts the application.
The employee modifies his availability (changes his presence).

Scene 8 – Adding of a work demand

Normal situation.
The manager need to add a new work demand (for a defined duty)
The manager starts the application.
The manager adds the new demand.

Exceptional situation
Some of the given data have a wrong value.
The manager makes a mistake typing the data.
The application warns about the wrong data and allows the manager to type the data

Scene 9 – Removing a work demand

Normal situation.
The manager wants to remove a work demand (for a defined duty)
The manager starts the application.
The manager removes the demand.

Scene 10 – Obtaining of a roster.

Normal Situation.
The manager wants a roster for a period of time.
The manager selects the start date and the end date of the period.
The manager selects a comparator.
The manager obtains a roster (default view)
The manager can change between different views.

Exceptional situation
Some of the given data have a wrong value.
The manager makes a mistake selecting the dates.
The application warns about the wrong data and allows the manager to type the data again.

1.3.4. Deployment diagram

The application will be located in the client enterprise. Internet connection will be needed to connect with the database. The application is composed by several packages:

- AbilityManager
- AvailabilityManager
- Demands
- Duties
- EmployeeManager
- Password
- db: This package controls and manages the access to the database.
- GUI: All the graphical interface.
- RosterManager: This package transforms the roster obtained through web services into a graphical representation of it.

Figure 23 represents the deployment diagram:

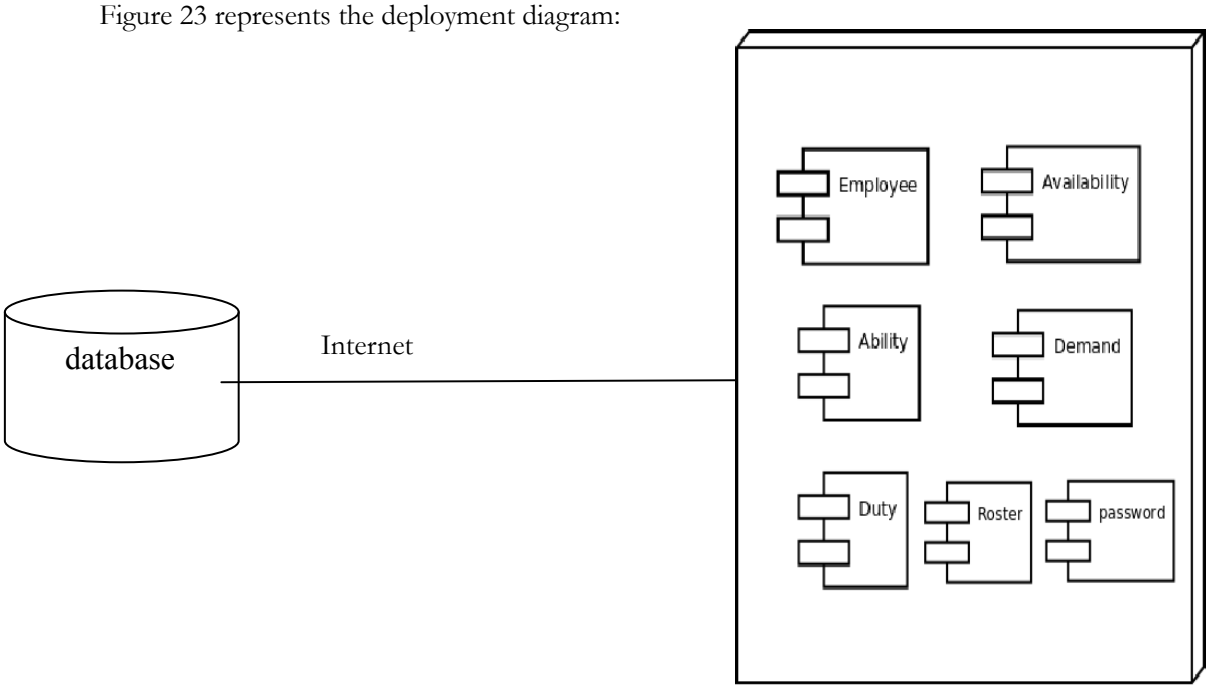


Figure 23. Deployment diagram

1.4. Functional model

The functional model describes the system's functions. In order to do this, data flow diagrams are made. DFDs specify the different functions at different levels and the input and output flow of data of all of them.

1.4.1. Data flow diagram

Level 0.

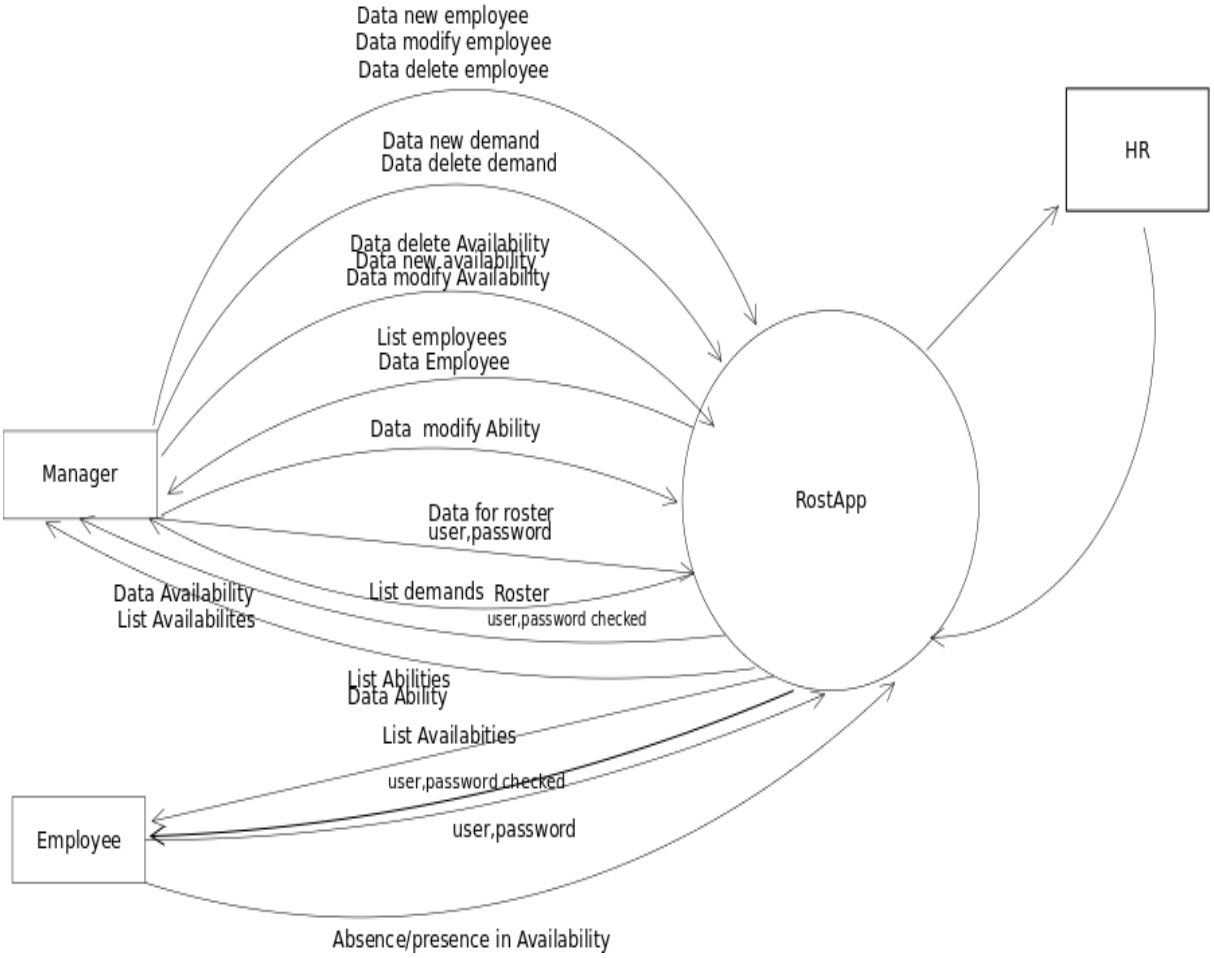


Figure 24. RostApp's DFD L0

In this level, the application interacts with the manager, the employees and with MidlandHR (through web services). The manager and the employee provide data to the application. The application operates with the data and returns the requested data. Figure 24 depicts the DFD of level 0.

Level 1

Figure 25 depicts the DFD of level 1.

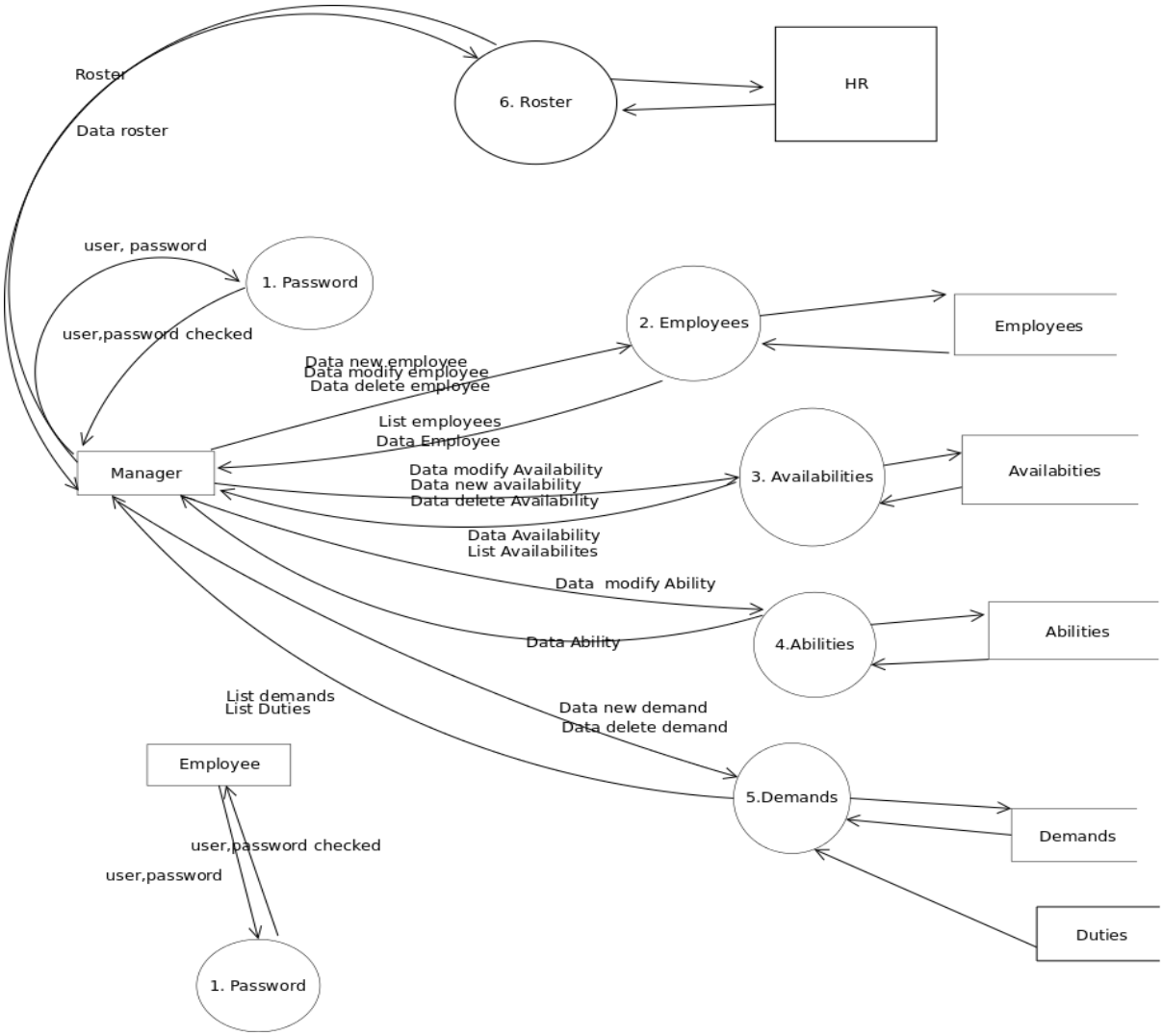


Figure 25. RostApp's DFD L1

The process 1 "Password" checks the user name and password from the user (manager or employee) and returns a Boolean value depending if the data is correct.

The process 2 "Employees" is responsible for the additions, modifications and deletions of the employees. The manager will provide the data to this process and it will perform all the operations.

The process 3 "Availabilities" is responsible for the additions, modifications and deletions of the availabilities. The manager will provide the data to this process and it will perform all the operations. This process is also used by the employees. They can change their absence/presence in their availabilities.

The process 4. "Abilities" is responsible for the additions, modifications and deletions of the abilities. The manager will provide the data to this process and it will perform all the operations.

The process 5 "Demands" is responsible for the additions and deletions of the demands. The

manager will provide the data to this process and it will perform all the operations.

The process 6 "Roster" is responsible for connection with MidlandHR's system to ask for a roster and get the solution of it. This process is managed by the GUI and for an external package, so the DFD for it will not be included here.

Level 2.

Figure 26 depicts the DFD of level 2.

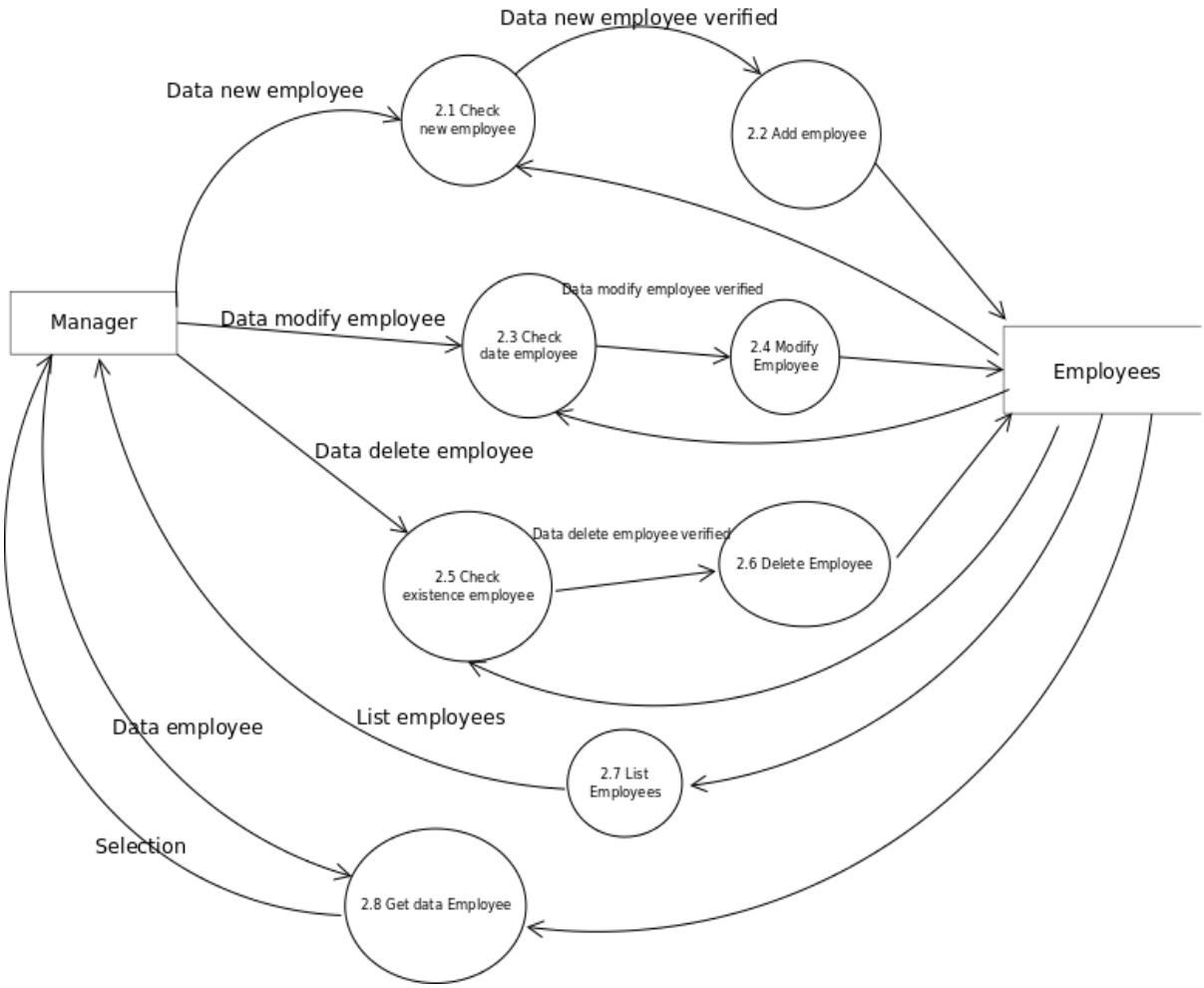


Figure 26. RostApp's DFD L2

The process 2.1 "Check new employee" verifies that the data of the new employee is correct and the employee is not stored in the database. It returns the verified data.

The process 2.2 "Add employee" adds a new employee to the database.

The process 2.3 "Check employee's data" verifies that the data of the employee is correct and that the employee is already in the database. It returns the verified data.

The process 2.4 " Modify employee" modifies the data of the employee in the database.

The process 2.5 "Check employee's existence" checks that the employee is stored in the database.
The process 2.6 "Delete employee" removes the employee from the database.

The process 2.7 " List employees" returns a list of all the employees sorted by name.

The process 2.8 "Get data employee" returns all the data of an employee. The employee is selected by the manager in the GUI.

Level 3

Figure 27 depicts the DFD of level 3.

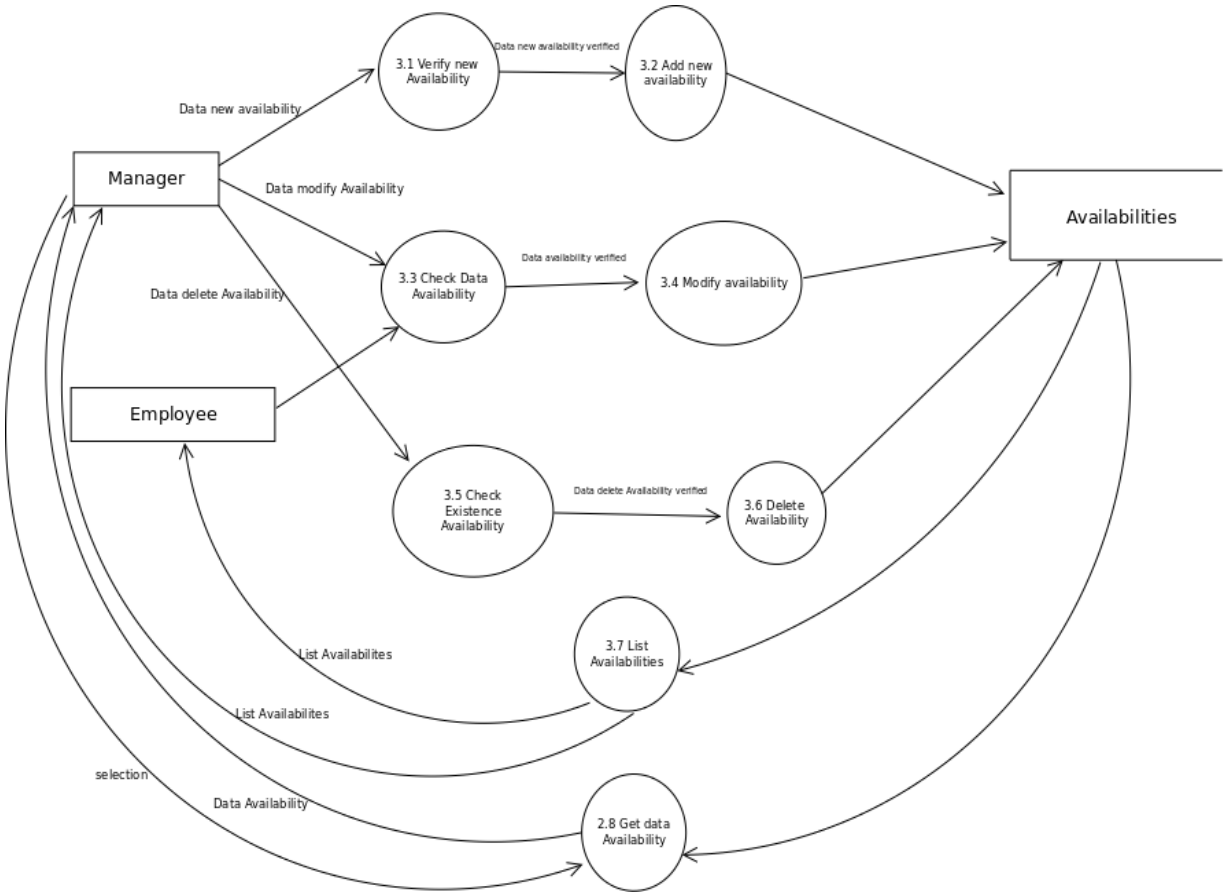


Figure 27. RostApp's DFD L3

The process 3.1 "Verify new Availability" verifies that the data of the new Availability is correct and the Availability is not stored in the database. It returns the verified data.

The process 3.2 "Add Availability" adds a new Availability in the database.

The process 3.3 "Check Availability data" verifies that the data of the Availability is correct and that the Availability is already in the database. It returns the verified data.

The process 3.4 "Modify Availability" modifies the data of the Availability in the database.

The process 3.5 "Check Availability existence" checks that the Availability is stored in the database.

The process 3.6 "Delete Availability" removes the Availability from the database.

The process 3.7 " List Availability" returns a list of all the Availability sorted by date.

The process 3.8 "Get Availability data" returns all the data of an Availability. The Availability is selected by the manager in the GUI.

Level 4

Figure 28 depicts the DFD of level 4.

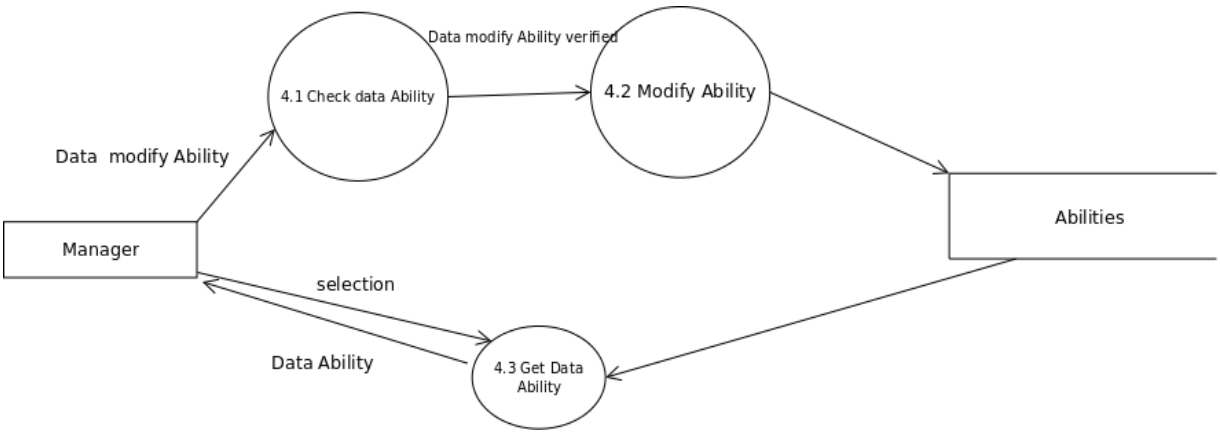


Figure 28. RostApp's DFD L4

The process 4.1 "Check new Ability" verifies that the data of the new Ability is correct and the Ability is not stored in the database. It returns the verified data.

The process 4.2 "Modify Ability" modifies the data of the Ability in the database.

The process 4.3 "Get Ability data" returns all the data of an Ability. The Ability is selected by the manager in the GUI.

Level 5

Figure 29 depicts the DFD of level 5.

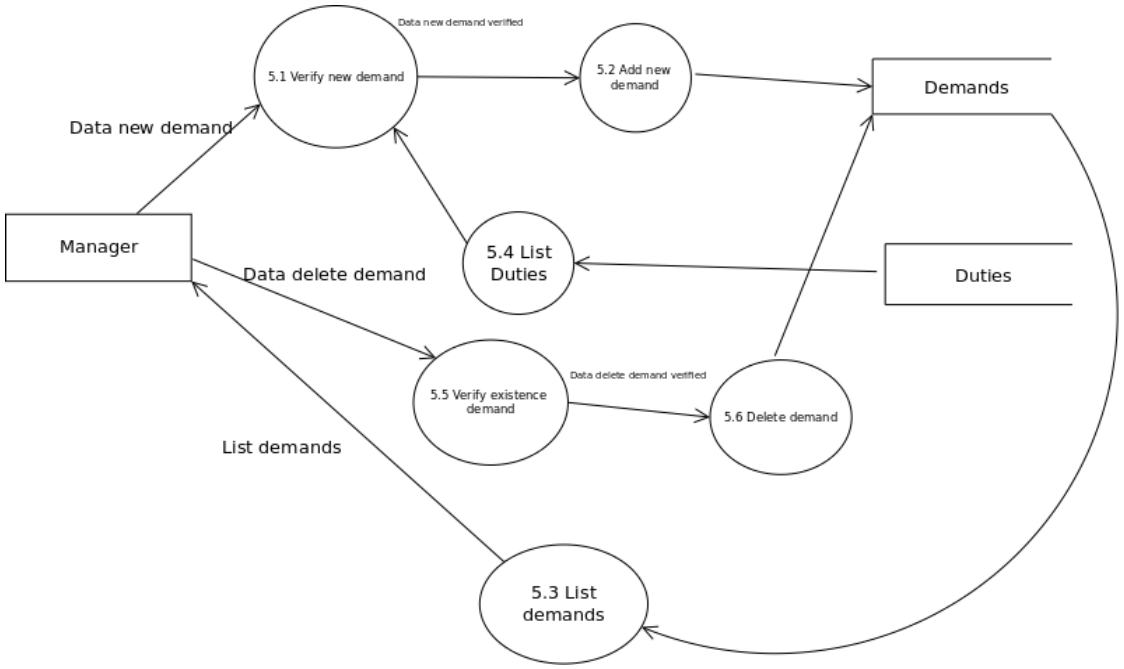


Figure 29. RostApp's DFD L5

The process 5.1 "Verify new demand" verifies that the data of the new demand is correct and the demand is not stored in the database. It returns the verified data.

The process 5.2 "Add demand" adds a new demand in the database.

The process 5.3 "List demands" returns a list of all the demand sorted by date.

The process 5.4 "List duties" returns a list of all the duties sorted by letter.

The process 5.5 "Verify demand existence" checks that the demand is stored in the database.

The process 5.6 "Delete demand" removes the demand from the database.

1.5. Subsystem design

1.5.1. General considerations

The first step in the process consists in establishing the modular division of the application. This division will be made identifying the different parts that will be in our application, the starting point is the analysis phase.

Therefore, it is necessary to analyse all the characterized models in the analysis phase in order to identify the functionalities that the application will offer. It is necessary to identify what classes are responsible for offering them and what dependencies exist between the different elements of the application. This study is focused on obtaining the best organization with a partitioning of the application into different modules which make easier the implementation of the application.

Some of the main criteria followed to identify the different subsystems are: identifying the elements which offer the same service or similar services, elements which share common properties, elements which are tightly coupled. All the elements which meet the above criteria will be grouped in the same subsystem.

Besides, it is necessary to think in aspects such as a well-defined and organized interface according to the design of the subsystems of the application. Thus, the interaction between the different modules and the GUI will be performed in a natural and easy way. This organization will not need the interaction of different modules to offer the same functionality, which would make the communication and the control of the application more difficult.

Likewise, it is also necessary to stress that the organization that is proposed here have to be implemented in a programming language (java). This could make that the division here proposed could be different in the implementation. Probably more and small subsystems will be created to ease the implementation of the application.

Aspects such as different users, concurrency, access to the database, usage of web services, client-server architecture for the application are very important to make the division of the system. These aspects will determine the need of adding new specific modules for the communications, concurrency control...

In order to communicate the different modules there are two main models to define the system's architecture: The layered model and the partition model.

The first one proposes a division into different levels. Each level represents a virtual machine for the upper level. A client-server relationship is established between the layers, the upper layer requests the services of the lower layer. This model can be an open model if each layer can establish relationships with every layer of a lower level or it can be a closed model if each layer only establishes relationships with the layers of the immediately below level.

On the other hand, the partition model proposes a vertical division into different independent subsystems which provide services at the same abstraction level. Therefore the different subsystems have to be loosely coupled.

Finally, the last important issue is the identification of the concurrency of our system. It is necessary to identify the tasks that need to be performed at the same time during the application's execution.

1.5.2. Application's subsystems design

This application can be used concurrently: some employees changing the availability at the same time, manager entering data and employee modifying at the same time...therefore the database system has to be able to manage concurrency (mechanisms to control data integrity).

The application will use the layer model. The partition model could be used in more granular divisions. The model will be a closed model. The closed model provide different advantages such us as high portability of the modules due to these have been built exclusively with the services of the below layer. It also reduces the dependencies between the layers and allows easier changes because the interface of one layer only affects to another. These points make this methodology suitable for this system.

Figure 30 shows the subsystem diagram:

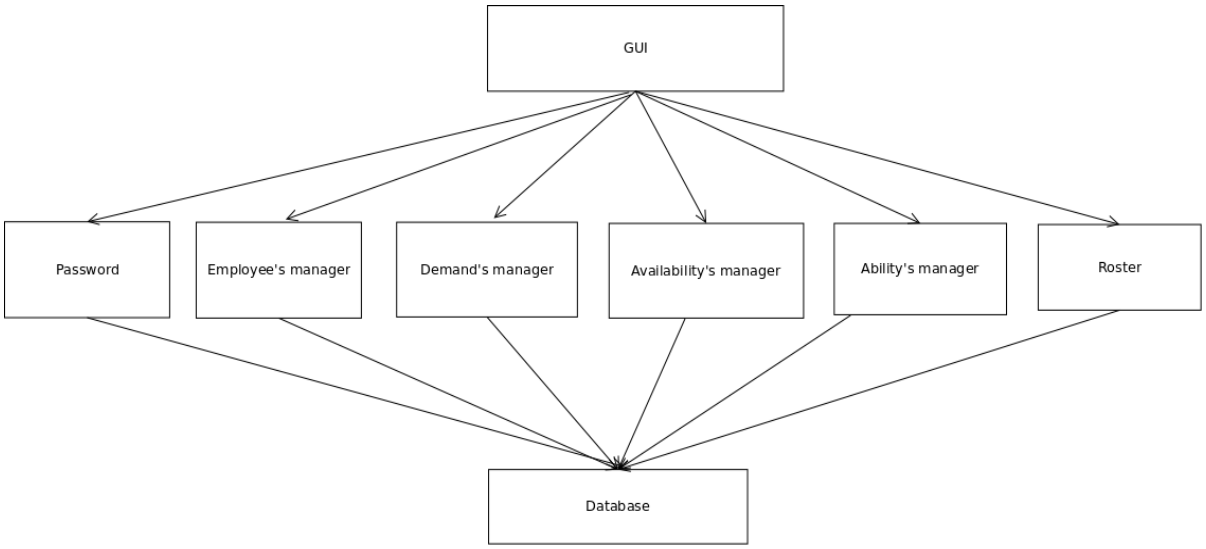


Figure 30. RostApp's subsystem diagram

This diagram shows three different levels: a lower level which manages the database, a middle level which manages all the data obtained from the database and the GUI and an upper level composed by the GUI which controls the user's interaction and the presentation of the data managed by the middle level.

The "intelligence" of the system is at the middle level. In this level all the functionalities are implemented. This level works like a bridge between the database and the GUI, it work with the data in order to offer the functionalities offered in the requirement analysis.

Finally, the upper level will manage the entire user's interaction. It will take all the data that the user enters and it will send these data to the middle level so that it processes and sends it to the database.

The middle level is composed of different modules:

The employee's manager module controls the addition, modification and deletion of the employees of the enterprise. It passes these actions to the database module. This module implements the functional requirements FR02, FR03 and FR04.

The roster module manages the transformation of the roster data obtained from the application into different graphical views. It implements the functional requirements FR05, FR06.

The manager's demands module manages the addition and deletion of demands in the database. It implements the functional requirements FR07 and FR08.

The availabilities' manager module manages the addition, modification and deletion of availabilities. It implements the functional requirements FR09, FR10 and RF11.

The abilities' manager module manages the modification of the employee's skills. It implements the functional requirement RF12.

Finally the security module manages the security of the application. It verifies the user name and the password of the users of the application. The verification is made with a file stored in the application files. It implements the non-functional requirement NFR06.

1.5.3. Object design

The object and operations design was made in order to get the simplest application with a logical separation which eases the implementation. It could seem that unnecessary objects are being created because they only stored intermediate data, however, this will allow us to organize the implementation in an efficient way.

Therefore, the subsystems' design and the classes' design make that the application be divided in different packages:

- GUI package (GUI)

  It contains all the necessary classes for the GUI. It will be detailed later.

- Database package (db)

  It contains all the elements (connectors, queries) to manage the database. It will be detailed later.

- Security package (password)

  This package implements the password class. This class will only have an instance in the system. The password object will access the password file of the application.

- Employee package (EmployeeManager)

  This package will implement the classes Employee and EmpGestor. The main function of this package will be managing all the aspects related to the addition, modification and deletion of employees.

  Therefore, the class Employee will work as a bridge between the data obtained from the GUI and the data introduced in the database. This will allow us to abstract from the data representation in the GUI and the data representation in the database.

  The next packages work in the same way that the Employee package.

- AvailabilityManager package.

  It contains the class Availability and the AvailGestor.

- AbilityManager package

  It contains the class Ability and the AbiGestor.

The Ability package and the availability package have been created in order to ease the implementation. The data that these classes contain could be added to the employee's data and the employee manager would have managed the operations into the different tables of the database. This separation gives us an easier management and a better treatment of the information.

- Demand package

  This package contains the Demand class and the DemGestor.

- Duties package

This package contains the Duties class and the DuGestor. The Duties class is a very simple class that is used by the demand and the ability packages. It would be possible add this class in those packages but with the separation of this class a more modular design is obtained.

- RosterManager package

This package manages all the views of the roster through the GUI. It has a Roster class which obtains the roster via the MidlandHR's web services. The other classes have been externally designed and manage GUI issues.

- Password package

This package implements the class password. This class checks the password and the user name entered for the user.

1.6. Database Design

In order to store all the information managed in the application and into the enterprise a database solution was chosen.

Then the design of this database is shown.

1.6.1. Entity model.

The entity diagram of the database is shown in figure 31.

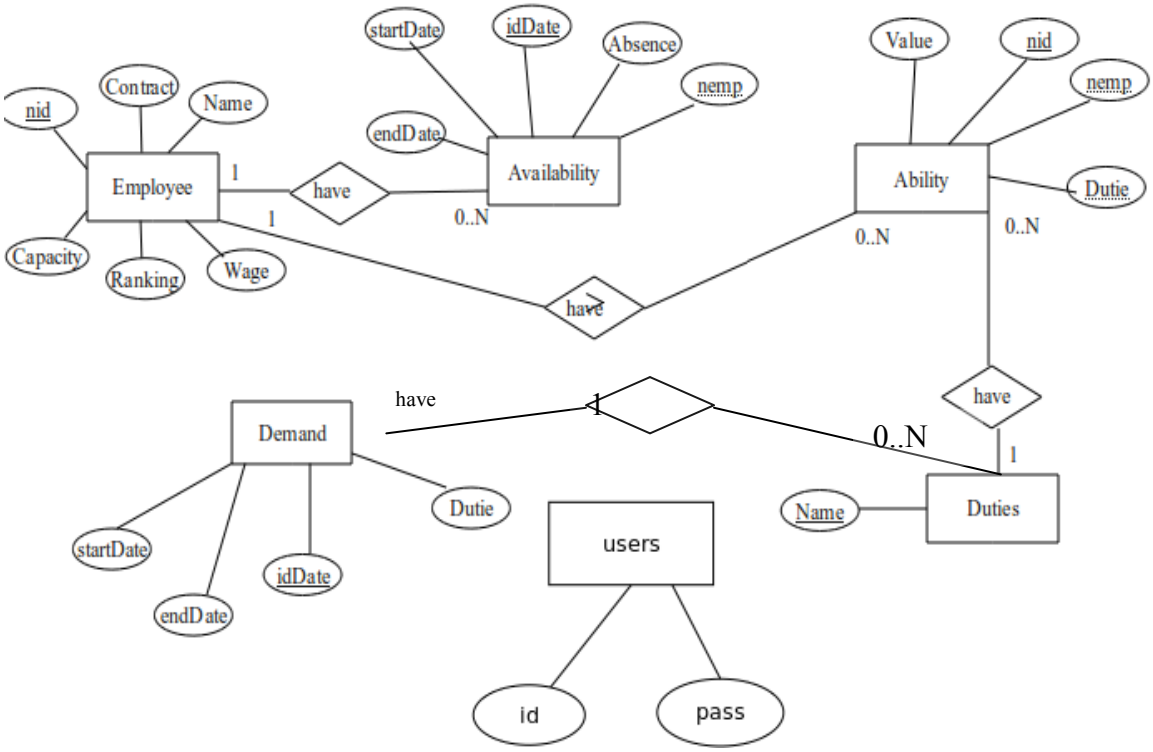


Figure 31. Entity Diagram

1.6.2. Relational model

The entity diagram will be the starting point of the relational model. The different entities in this model will be the next ones:

Employee:

nid: INTEGER (primary key)
Name: CHAR (30)
Wage: REAL
Capacity: REAL
Ranking: REAL
Contract: INTEGER

Availability:

idDate: INTEGER (primary key)
nemp: INTEGER (Foreign key - employee)
Absence: BOOLEAN
startDate: DATE
endDate: DATE

Ability

nid: INTEGER (primary key)
nemp. INTEGER (Foreign key - employee)
dutie: CHAR (30) (Foreign key - Duties)
Value: REAL

Demand

idDate: INTEGER (primary key)
startDate: DATE
endDate: DATE
dutie: CHAR (30) (Foreign key – duties)

Duties

id: INTEGER
name : CHAR (30)

There is not any N:N relationship in the model, so there is no need for additional tables. Instead of that, key propagation has been used. In the next section, the queries made to the database will be shown.
The queries are prepared to be used with any value; this will make easier the usage of them. All the queries follow the same path of execution. An example of usage of a query:

Definition of the query:

```
private static final String ModifyEmployee = "UPDATE Employee " +
            "SET name=?, Contract=?, Wage=?, " +
            "Capacity=?, Ranking=?"+
            "WHERE nid=?";
```

Definition of the Prepared Statement:

```
static PreparedStatement PSmodifyemployee;
```

Association between elements:

```
PSmodifyemployee = conn.prepareStatement(ModifyEmployee);
```

Execution of the query:

```
PSmodifyemployee.setInt(6,nid);
PSmodifyemployee.setString(1,name);
PSmodifyemployee.setInt(2,Contract);
PSmodifyemployee.setFloat(3,Wage);
PSmodifyemployee.setFloat(4,Capacity);
PSmodifyemployee.setFloat(5,Ranking);
PSmodifyemployee.executeUpdate();
```

1.7. Graphical User Interface

The interface corresponds to a Java Swing development. It is clear, simple and very intuitive for the user. A document showing all the interface design and the navigation between the different parts is located in the appendix section (Appendix V).

The most important feature of the GUI is its ability to draw the roster solution. For this purpose, the "Algo" package is used. This package was provided by Arturo Castillo and has been separated into different parts in order to create a web services oriented solution for the rostering process. In this application the GUI part of the package has been taken. In order to manage all the classes and data

structures necessary for the drawing, the package "rostering" has been added as a library to the project. This package was also provided by Arturo Castillo and contains all the data structures to work with rosters.

Some modifications have been made in order to adapt the solution obtained via web services to the data format with which the package works.

Views of the solution are shown in figures 32, 33 and 34.

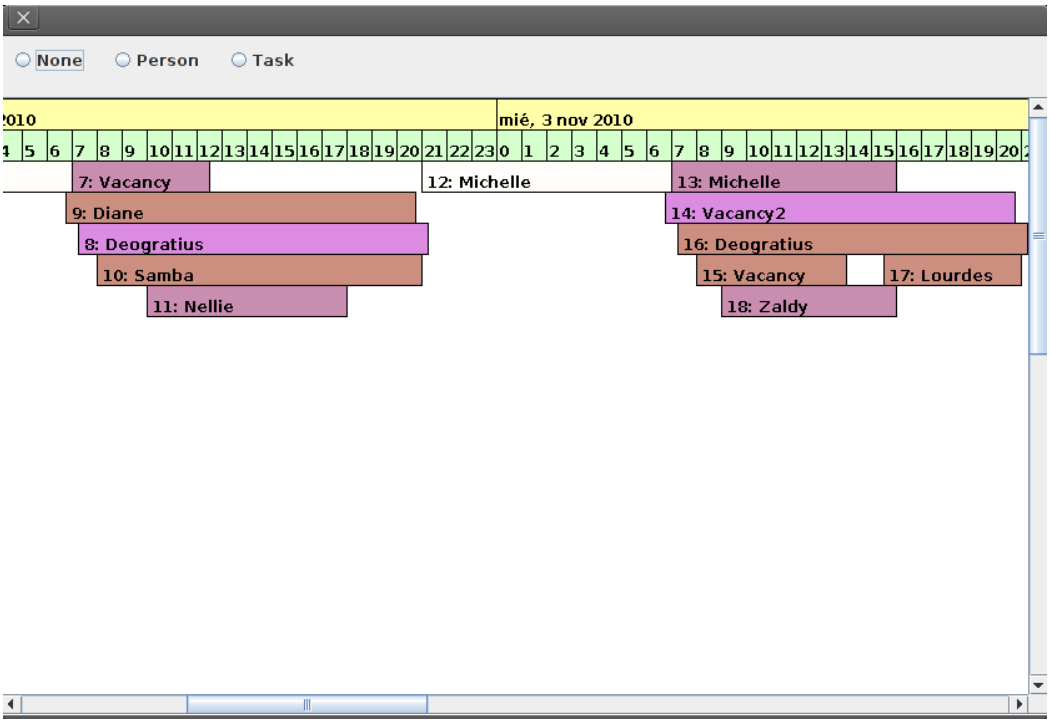


Figure 32 – Roster – Complete view.

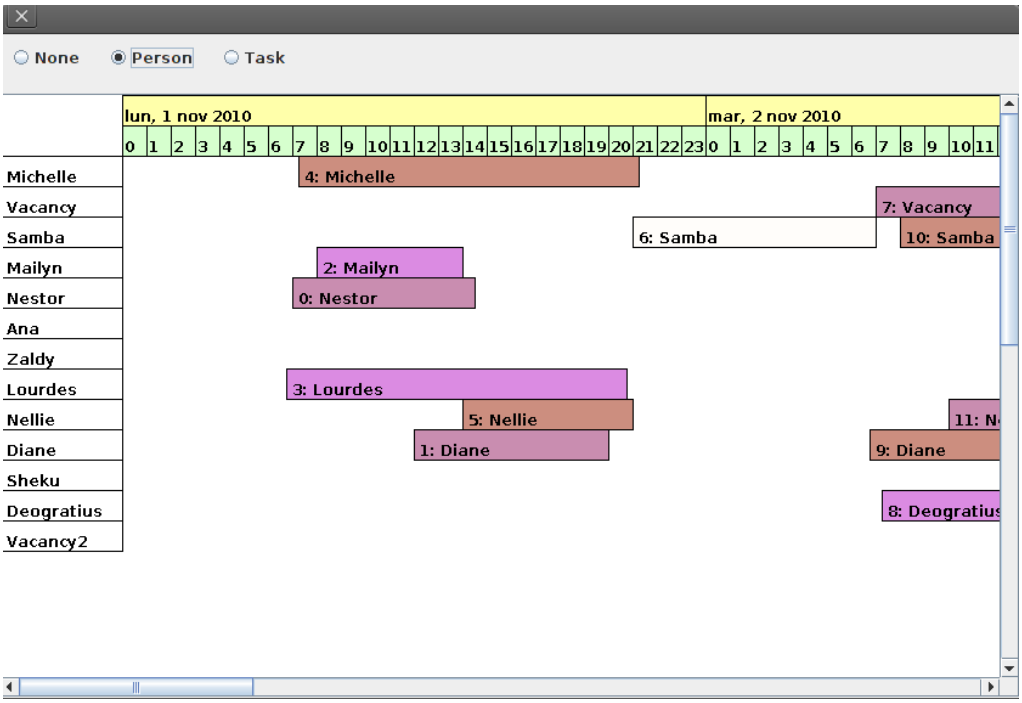


Figure 33– Roster – Person's view.

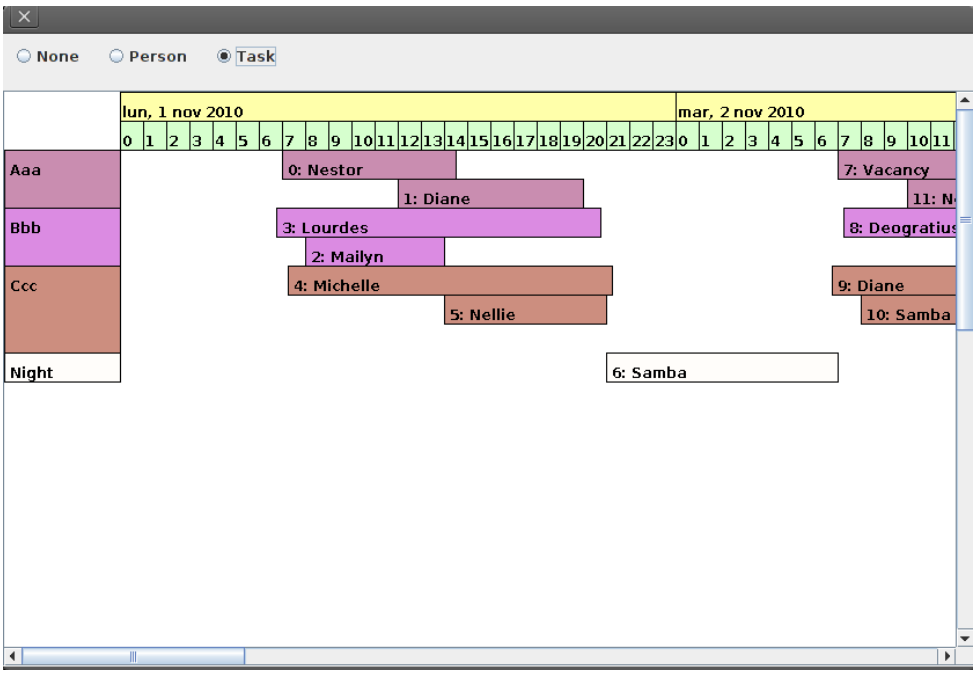


Figure 34 – Roster – Task's view.

1.8. Web Services

As mentioned above, the RosterManager package manages the obtaining of a roster solution. Obtaining the solution is very simpler due to the IDE. It is only necessary to add a call to the web service as it was a function developed by ourselves.

Example:

```
sol = askRoster(date1, date2,comparator,"LC","osuosu21");
```

The parameters of the web service will be explained in MidlandHR's annex. In this case, the dates for the roster, a comparator (development variable), the name of our enterprise and the password of the enterprise in the rostering company are sent. The web service returns a roster solution which is stored in the "sol" variable. It is not necessary to import the data definition of the solution; the IDE does it for us.

The sent dates have to be transformed from java dates to an XMLGregorianCalendar in order to be "shippable" through the web service:

```
GregorianCalendar calendar = new GregorianCalendar();

XMLGregorianCalendar date =
DatatypeFactory.newInstance().newXMLGregorianCalendar(calendar);
```

1.9. Tests

All the functionalities of the application have been tested. A complete report of the tests results is included in the appendices. (Appendix XIII).

ANEXO IV – MidlandHR

1. MidlandHR

MidlandHR is the core of the system. This application is responsible for calculating the roster. It works in the rostering company's server.

1.1. Requirements specification

Functional requirements

Requirements related to the usage of the application.

- **FR01 – Roster.** It elaborates two different solutions: a complete one and a reduced one for the employees.
- **FR02 – Web Services.** It provides web services to roster's request and sends back the solution
- **FR03 – Data**. It obtains all the necessary data for the roster via web services.

Non Functional requirements

- **NFR01 – Concurrency**. The application must be highly concurrent.
- **NFR02 – Password.** To start the application a password will be needed. Each enterprise will have a password.
- **NFR03 - Availability**. The application should be online 24/7

1.2. Object Model

In this part, several diagrams will be shown in order to define the object structure of the application.

1.2.1. Class diagram

The class diagram represents very properly the working of the application. The application has three different parts that will be explained later, but they can be intuited in the diagram. The figure 35 corresponds to the class diagram.

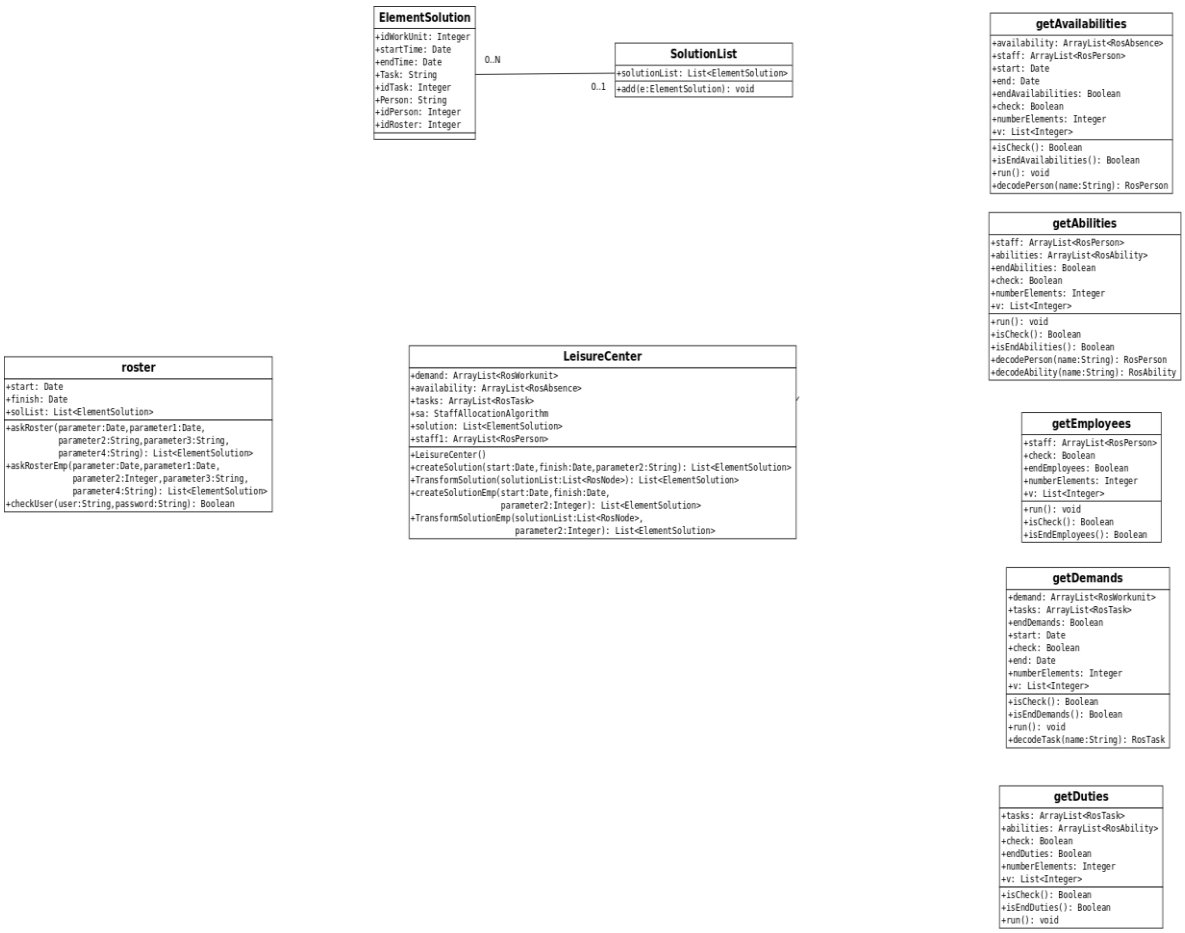


Figure35. MidlandHR's class diagram

1.2.2. Class Dictionary

In this section each class and its attributes are described.

Roster

This class implements the web services to obtain a roster. It also checks the user's password in a file.

LeisureCenter

This class is responsible for creating the solution. It retrieves all the necessary information and then it calls the rostering algorithm.

ElementSolution

The roster provided by the rostering algorithm contains not useful information for the client and it has a complex structure. In order to return it via web services, a new class with simpler structure has been created. An ElementSolution instance contains one roster's workunit.

SolutionList

SolutionList is a list of ElementSolution and it is what will be returned to the client.

getAvailabilities

This class obtains the availabilities of the Leisure centre via web services

getAbilities

This class obtains the abilities of the Leisure centre via web services

getEmployees
This class obtains the employees of the Leisure centre via web services

getDemands
This class obtains the demands of the Leisure centre via web services

getDuties
This class obtains the duties of the Leisure centre via web services

1.3. Dynamic model

The dynamic model in this application is not relevant. The application starts to run when a request arrives. Once the request has been validated all the data is collected and then the solution is created and returned.

This behaviour is easily seen in an activity diagram (Figure 36)

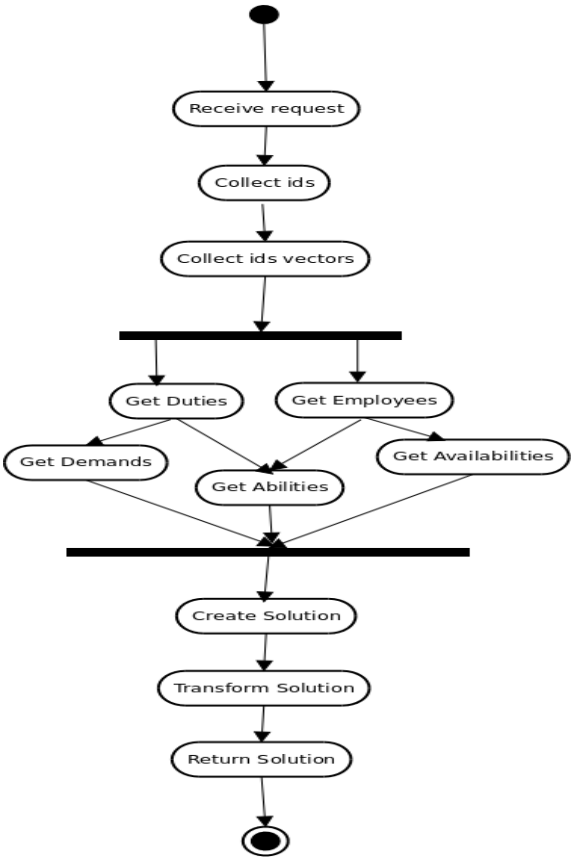


Figure 36. MidlandHR's activity diagram

1.4. Functional model

The functional model presented for this application is very simple and it only aims to show how the system works and what the main data flows are.

1.4.1. Data flow diagram

Level 0.
Figure 37 depicts the DFD of level 0.

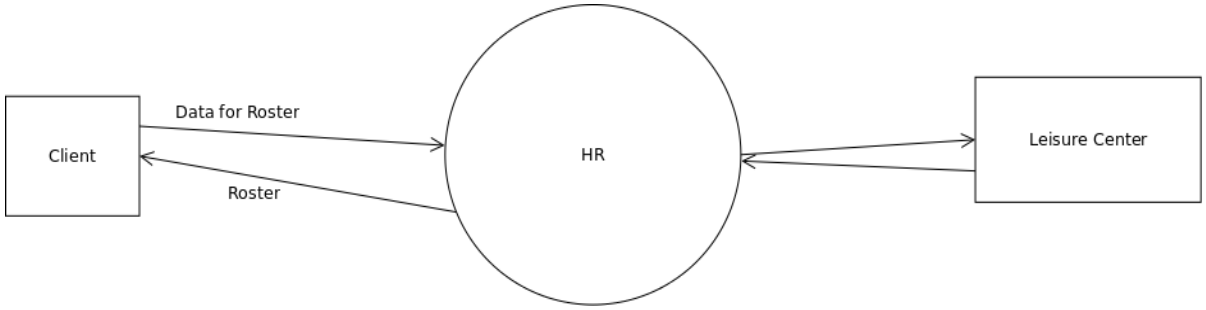


Figure 37. MidlandHR's DFD L0

In this level, the application interacts with the client and with the leisure centre. Both interactions are done via web services. The client provides data to the application; the application works these data and asks for more data to the leisure centre.

Level 1
Figure 38 depicts the DFD of level 1.

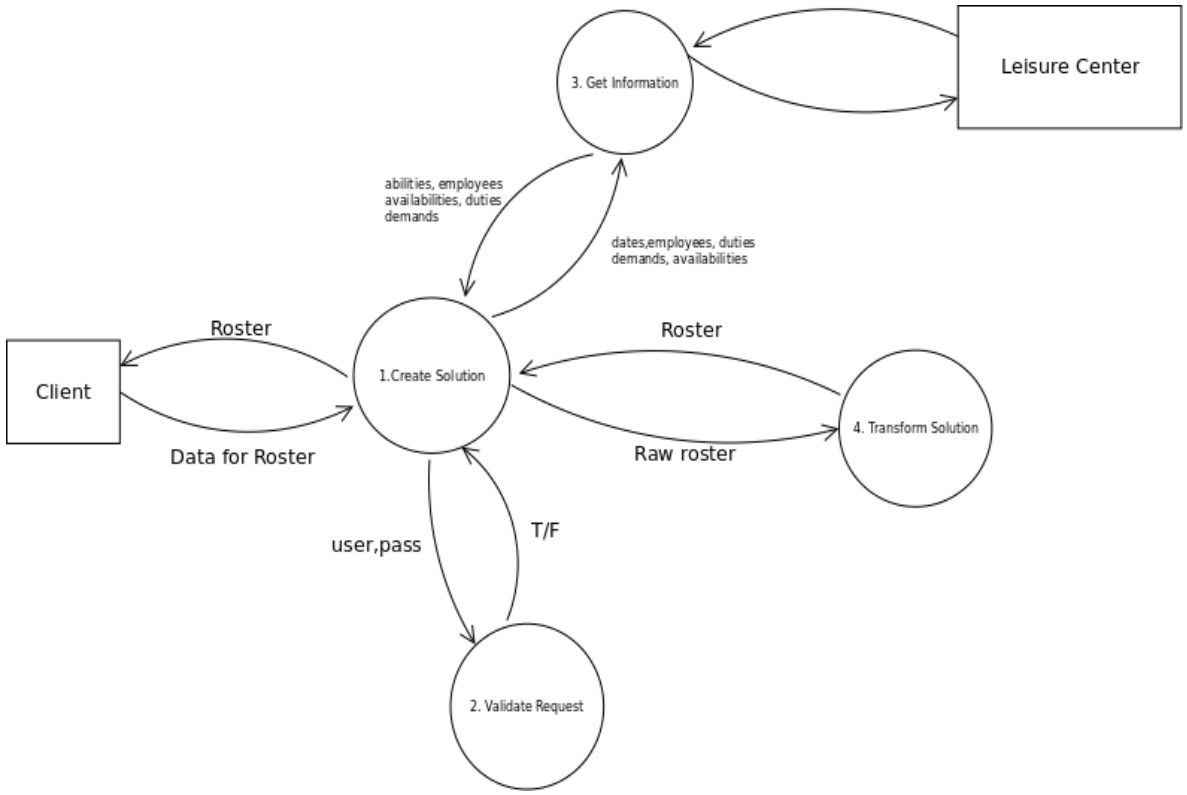


Figure 38. MidlandHR's DFD L1

The process 1 "Create solution" receives the data and begins the creation of the solution. This process is the central part of the application and it controls all the data flows in the rostering process.

The process 2 "Validate Request" checks the user name and password from the client and returns a boolean value depending if the data is correct.

The process 3 "Get Information" obtains all the needed information from the Leisure centre via web services.

The process 4 "Transform Solution" receives a raw roster solution and transforms it in a shippable solution.

1.5. Design

Rostering is a very complex process. The mix of a web services approach and rostering could result in a complex application. For this reason, the design's goal of this application has been keeping both parts independent. The rostering algorithm and all its dependencies are kept in separated packages and the application only invokes the algorithm once.

Rostering works as a black box; the application obtains all the necessary information and passes it to the rostering algorithm. If instead of a roster we wanted calculate another type of solution with the same data, we would only have to change the rostering packages for the new solution's ones.

As seen in the analysis part, the application works like a transactional system. The class diagram shows three different parts: One to receive the request, another to process the request and create the solution and another to obtain data from the leisure centre.

The design of the objects corresponds with the classes created in the analysis phase.

The parts have been grouped in packages. The packages are:

Servicios

This package contains the roster class which implements the web services.

Solutions

This package contains the LeisureCentre class. If besides the leisure centre MidlandHR would be working with more enterprises, the classes for each of those enterprises would be located in this package.

Resources

This package contains the classes which retrieve data from the leisure centre.

Algo.allocation

These series of packages were provided by Arturo Castillo and they contain the algorithm to create a roster. The methods of these classes are invoked from the LeisureCentre class.

Two classes have been added to these packages: ElementSolution and SolutionList. These classes represent a shippable version of the roster solution.

1.6. Implementation

1.6.1. Web service implementation

The MidlandHR's web services headers are:

askRoster: Creates a roster.

```
@WebMethod(operationName = "askRoster")
 public List<ElementSolution> askRoster(
 @WebParam(name = "parameter")
 Date parameter,
 @WebParam(name = "parameter1")
 Date parameter1,
 @WebParam(name = "parameter2")
 String parameter2 ,
 @WebParam(name = "parameter3")
```

```
        String parameter3 ,
    @WebParam(name = "parameter4")
    String parameter4) {
```

Parameters "parameter" and "parameter1" are the dates in which the roster lies. "Parameter2" is the comparator for the roster (development variable). "Parameter3" and "Parameter4" are the user name and password of the client in the MidlandHR's system.

AskRosterEmp: Creates a roster for one employee.

```
    WebMethod(operationName = "askRosterEmp")
    public List<ElementSolution> askRosterEmp(
    @WebParam(name = "parameter")
    Date parameter,
    @WebParam(name = "parameter1")
    Date parameter1,
    @WebParam(name = "parameter2")
    Integer parameter2 ,
    @WebParam(name = "parameter3")
    String parameter3 ,
    @WebParam(name = "parameter4")
     String parameter4) {
```

In this service there is not comparator (it is set by default) and instead of it "parameter2" corresponds to the id of the employee who has asked for the roster.

The algorithm of these web services is simple and it just invoked to another class to create the roster:

```
if (checkUser(parameter3,parameter4))

        solList=LC.createSolutionEmp(start,finish,parameter2);

}
return solList;
```

First, the enterprise which wants to consume our web services is checked (checkUser) and then the LeisureCentre class is called to create a solution which is returned. All the calculating process could have been located in this class but it would not be good for the modularity and design of the application.

Keeping the service simple allows future changes to be achieved easily. The SOAP messages have the following structure:

SOAP message:

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ser="http://servicios/">
   <soapenv:Header/>
   <soapenv:Body>
      <ser:askRoster>
         <parameter>2010-01-04T00:00:00.000+01:00</parameter>
         <parameter1>2011-01-04T00:00:00.000+01:00</parameter1>
         <parameter2>Task</parameter2>
         <parameter3>LC</parameter3>
        <parameter4>osuosu21</parameter4>
      </ser:askRoster>
   </soapenv:Body>
</soapenv:Envelope>
```

SOAP message:

```
<soapenv:Envelope
```

```
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ser="http://servicios/">
   <soapenv:Header/>
   <soapenv:Body>
      <ser:askRosterEmp>
         <parameter>?</parameter>
         <parameter1>?</parameter1>
         <parameter2>?</parameter2>
         <parameter3>?</parameter3>
         <parameter4>?</parameter4>
      </ser:askRosterEmp>
   </soapenv:Body>
</soapenv:Envelope>
```

1.6.2. Ids and vectors

The resources package obtains data from the leisure centre. The data is expected to be right; however, while the application is obtaining the users of the leisure centre could add or erase data. This will invalidate our roster because the data retrieved would not be coherent with the data stored in the leisure centre or even the different batches of data retrieved could be incoherent.

In order to avoid this, the leisure centre provides two web services. The first of them returns the number of elements in each of the categories of data needed. The second one returns the identification numbers of all the elements.

These services are invoked at the beginning of the leisure centre class:

```
//WS to get the number of elements of each category
v = askNumberElements(dateX1,dateX2);
//WS to get the id's vectors of each category.
v2 = askVectors(dateX1, dateX2);
```

These numbers and the ids are stored in two vectors (v and v2). These vectors are passed to the threads which obtain the data from the leisure centre. The first verification is check if the size of v2 is the same as the number of elements in each category:

```
if (numberElements == v.size()){
```

If the verification is successful, the thread starts to ask for the elements of the category (employees, availabilities, abilities...). The thread asks for each of the identification numbers stored in the vector

```
for(int i = 0; i<v.size(); i++)
{
      e = askEmployee(v.get(i));
```

This mechanism allows MidlandHR to work with the data stored at the moment which it starts to create the roster and it does not depend on the possible modifications of the data while the roster is being created.

In the final version only these two web services are used and they return all the data needed. Nevertheless, an early implementation used one service for each category. This was very inefficient due to the creation of ten connections and the few data that they receive.

The amount of data received by a normal roster makes these services efficient, but if the size of the roster is increased and more data has to be returned, the first option should be considered again.

1.6.3. Threads

As it can be seen in the activity diagram some tasks run parallel. Some of the classes which obtain data from the leisure centre can be executed at the same time. For this purpose, these classes have been implemented as threads. The threads are created and run in the LeisureCentre class.
At the beginning, the implementation was done in a linear way, executing one class after another. This was the first upgrade done; it is easy to see and it increases the performance of the system with only a few modifications.

However, the threads must be run in a particular order due to the dependencies among the information they collect.

The dependencies are:

getDemands depends on getDuties
getAvailabilities depends on getEmployees
getAbilities depends on getDuties and getEmployees.

Therefore, the threads must be synchronised and they have to wait each other to respect the dependencies. This is achieved with the endNameOfTheClass and checkNameOfTheClass variables.
The LeisureCenter class controls the threads with an active waiting:

```
While (name_of_thread.isAlive());
```

1.6.4. Getting the information

The data obtained has to be transformed into the structures needed by the rostering algorithm. In each category, a consumption of the web service is performed to every element. The identification vector is used and the web service is asked for every id in the vector.
The IDE automates the process, it is only necessary to call the service:

```
av = askAvailabilites(v.get(i));
```

and the IDE creates the port of the web service call:

```
private Availabilities askAvailabilites(int parameter) {
askAvailabilities.AskAvailabilitiesWSService service = new
askAvailabilities.AskAvailabilitiesWSService();
askAvailabilities.AskAvailabilitiesWS port =
service.getAskAvailabilitiesWSPort();
return port.askAvailabilites(parameter);}
```

Once the element has been received it is transformed into the rostering's structures. In the prototype provided by Arturo Castillo all the data was read from a file and transformed into the rostering's structures. In this case, instead of a file there are web services but the code has been adapted.
This shows the versatility of the web services, it is easy to adapt programming code to a services approach.

1.6.5. Obtaining the roster

Once, all the information has been retrieved the roster is obtained in three steps:

```
sa = new StaffAllocationAlgorithm(demand, staff1, availability);
sa.setSortingComparator();
sa.solve();
```

After obtaining the roster, a new line in the log file is generated. This line indicates if the solution has been achieved or not and the solution's characteristics.

1.6.6. Transforming the solution.

The solution provided by the rostering algorithm is complex and it needs to be modified in order to be send through web services. The transformation process is very simple. Every element of the solution is picked and transformed into an ElementSolution object which is stored in a SolutionList and returned.

Every element of the original solution has this structure:

```
private RosWorkunit workUnit;
private List<RosPair> combined;
private int availableSize;
private List<RosNode> clashes;
private RosPerson person;
private List<AssignationListener> assignationListeners;
```

However, the client does not need all this information to get a roster in his screen. Therefore, a simpler structure is provided:

```
Integer idWorkUnit
Date startTime: start time of the shift
Date endTime: end time of the shift
String Task: Task's name
Integer idTask
String Person: Employee's name
Integer IdPerson
Integer idRoster
```

The process:

```
for (int i=0; i<solutionList.size();i++){
          ElementSolution e = new ElementSolution();
          wu = solutionList.get(i).getWorkUnit();
          person = solutionList.get(i).getPerson();
          //IdworkUnit
          e.setIdWorkUnit(wu.getIdWorkUnit());
          //Start Time
          e.setStartTime(wu.getStartTime());
          //End Time
          e.setEndTime(wu.getEndTime());
          //Get Task
          e.setTask(wu.getNameTask());
          e.setIdTask(wu.getIdTask().getIdTask());
          //Person
          e.setIdPerson(person.getIdPerson());
          e.setPerson(person.getNamePerson());
          //idRoster
          e.setIdRoster(wu.getNumberRoster());
          l.add(i,e);
       }
```

Sending the original solution had also been possible but methods to codify the data into a valid XML schema would have been necessary. The adopted solution uses the advantages of the IDE and no methods to codify the solution are needed, the JAX-WS tools of the IDE does it automatically.

This method has another usage: when an employee asks for his roster, the whole roster has to be created but only his part has to be sent. In the case of an employee's roster, this method selects only the employee's work units. This is made with a simple "if" structure:

```
if (person.getIdPerson() == parameter2)
```

Where parameter2 is the id of the employee who has asked for the roster.

1.7. Test

The tests for this application will be performed with soapUI. Testing this application could be reduced to test the web services. SoapUI allows to check the conformance of the web services with the standards and to check the correct working of them. The results of these tests are shown in the annex X.

ANEXO V - LEISURE CENTER

1. LEISURE CENTRE

This web application provides data from the leisure centre to third parties. In this project the data is provided to MidlandHR in order to create a roster. The data is provided through web services. There are two versions of this application one with SOAP web services and one with REST web services (annex VII)

1.1. Requirement specification

Functional requirements

Requirements related to the usage of the application.

- **FR01 – Web Services**. The data is provided to third parties via web services.

Non Functional requirements

- **NFR01 – Concurrency**. The application must be highly concurrent.
- **NFR02 - Availability**. The application should be online 24/7

1.2. Object Model

In this part, several diagrams will be shown in order to define the object structure of the application.

1.2.1. Class diagram

The class diagram gives an idea of how the application works. There are three different parts: one for the web services, one for the logic and one for the database.
All the getters, setters and constructors have been omitted. The methods for executing queries from the database have been omitted as well. The figure 39 corresponds to the class diagram.

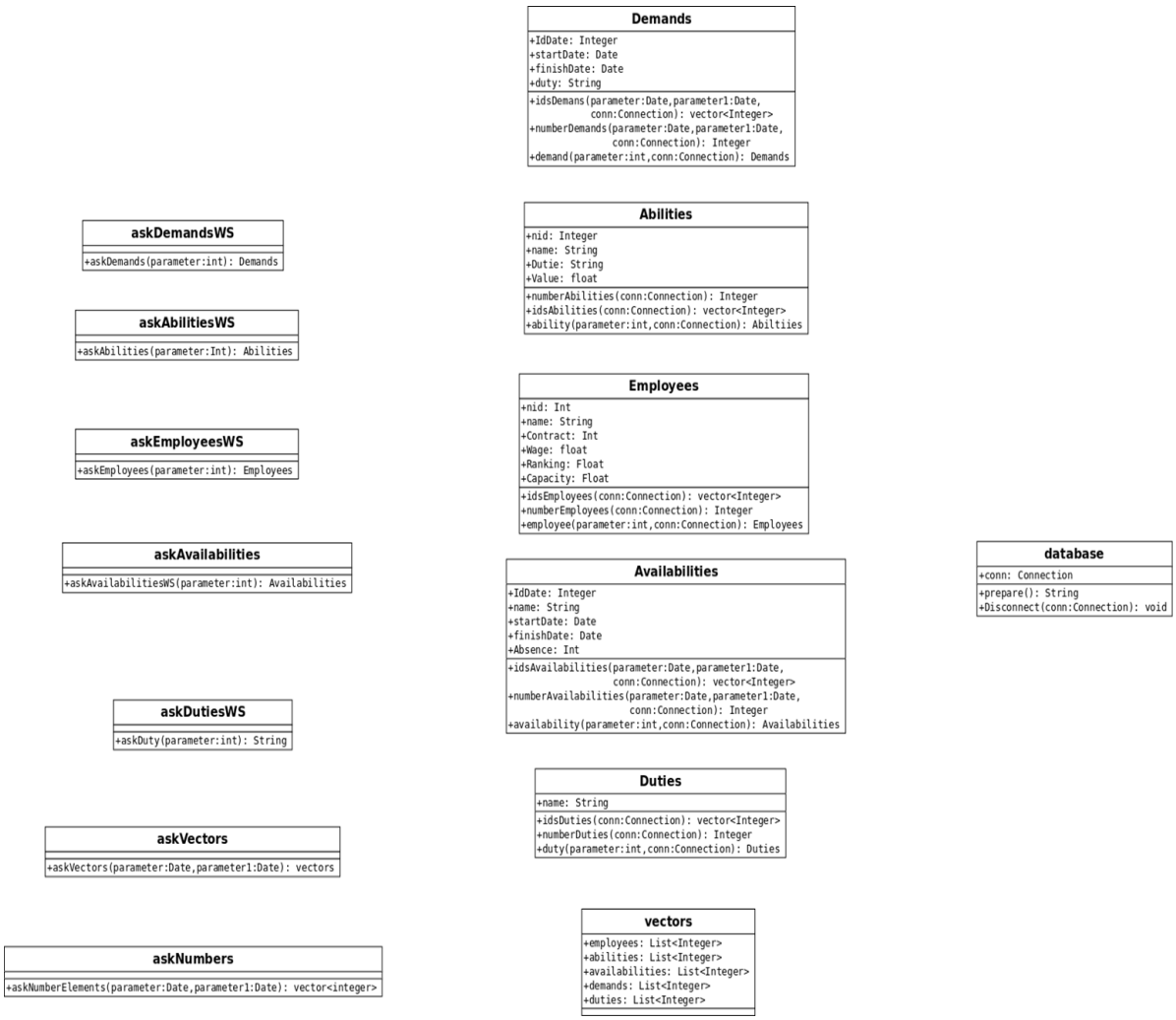


Figure 39. LeisureCenter's class diagram

1.2.2. Class dictionary

In this section each class and its attributes are described:

Web service classes:

The following classes:AskAbilitiesWS, AskAvailabilitiesWS, AskDemandsWS, AskDutiesWS, AskEmployeesWS, AskNumbers, AskVectors, do not have attributes. These classes implement the web services of the application.

Availabilities

This class represents the availabilities of the system. An availability is a period of time in which the employee cannot work.

Abilities

This class represents the performance of the employees in the different duties. In every duty the employee has a grade. These grades are used to calculate the roster.

Demands

This class represents the work demands stored in the system.

Duties

This class represents the different tasks performed in the leisure centre.

Employees

This class represents the employees of the leisure centre.

Vectors

This class has been created to ease the implementation. The usage of the identifiers vectors was explained in the previous annex. This class joins the different vectors into a vector of vectors. This allows the creation of only one web service instead one web service per vector.

Database

This class is used to connect with the system database and execute the queries to retrieve information.

1.3. Dynamic model.

In this application the dynamic model is also irrelevant. When one web service is consumed it calls the correspondent class which performs the query in the database and returns the data to the client. There are different web services but all of them work in the same way.
Instead of developing use cases, scenarios and state machine diagrams, an activity diagram is showed. It represents the performance of the system better: Figure 40 shows the activity diagram.

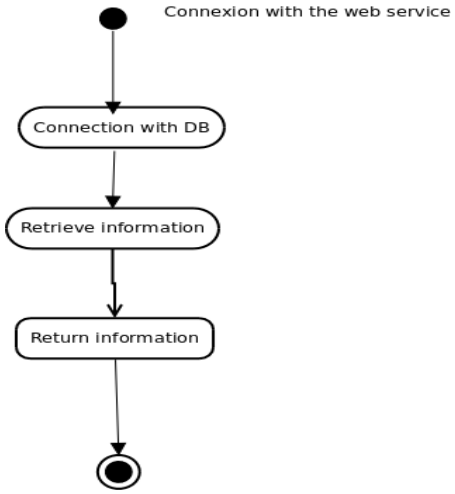


Figure 40. Leisure centre's activity diagram.

This is what happens every time a web service is consumed. In a real scenario, the web services are consumed concurrently by clients.

1.4. Functional model

1.4.1. Data Flow Diagrams

The functional model presented for this application is very simple and it only aims to show how the system works and what the main data flows are. Instead of create a diagram for every web service; a generic diagram has been created.

Level 0.
Figure 41 depicts the DFD of level 0.

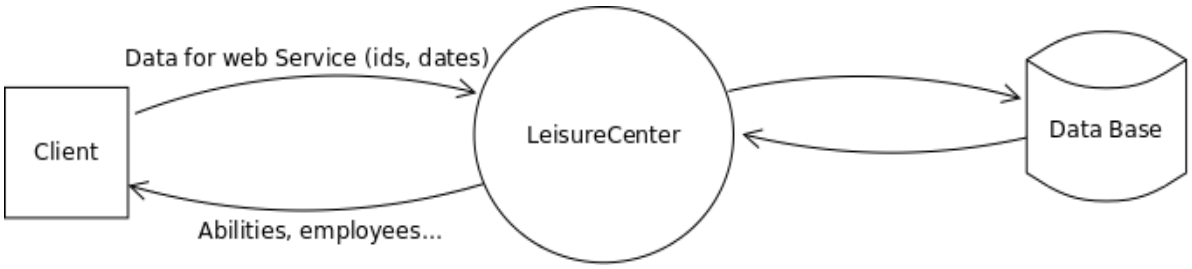


Figure 41. LeisureCenter's DFD L0

In this level, the client consumes one of the web services and receives the data requested. In order to return the data, the application obtains the data from the database.

Level 1
Figure 42 depicts the DFD of level 1.

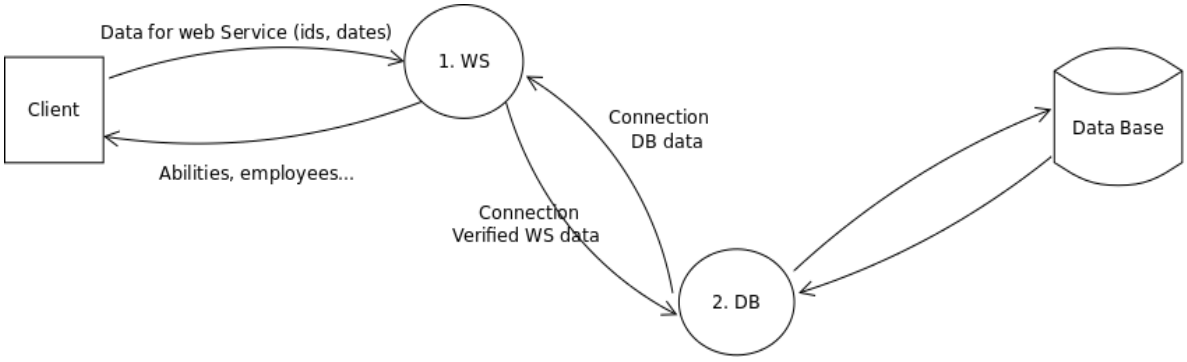


Figure 42. LeisureCenter's DFD L1

The process 1 "WS" receives the request and creates the structure of requested data. To fill the structure contacts with the process 2.
The process 2 "DB" interacts with the database to retrieve the data asked by the process 1.

1.5. Design

This application deals with the data base very similar to how RostApp does it. The analysis phase has determined that three layers are needed for a good design. The design principles of modularity and simplicity have also been applied here.

As in MidlandHR, the web service gives all the work to another classes and it just returns the solution.

The application was developed after RostApp, and it has been tried to reuse the maximum number of components. If the components were reused successfully it meant that the level of modularity achieved in RostApp was optimal. The changes were minimal and they were due to the concurrency and service approach of this application.

The design of the objects corresponds with the classes created in the analysis phase.
The parts have been grouped in packages. The packages are:

Services

This package contains all the classes that implements web services.

Database

This package contains the class which interacts with the database

Logic

This package contains the classes which represent the entities of the system: Employees, availabilities, abilities, duties and demands.

Vectors

This package contains the vector class. This class could have been in the logic package, however this a class created only to ease one of the web services. Therefore, its functionality is different of the classes located in the logic package, so it has been separated.

1.6. Implementation

1.6.1. Web services

The following web services have the same structure. In of all them, the parameter "parameter" corresponds with the identifier of the resource in the database.

AskAbilitiesWS

```
@WebMethod(operationName = "askAbilites")
    public Abilities askAbilites(
  @WebParam(name = "parameter") int parameter) {
```

SOAP message:

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ser="http://services/">
   <soapenv:Header/>
   <soapenv:Body>
      <ser:askAbilites>
         <parameter>?</parameter>
      </ser:askAbilites>
   </soapenv:Body>
</soapenv:Envelope>
```

AskAvailabilitiesWS

```
@WebMethod(operationName = "askAvailabilites")
 public Availabilities askAvailabilites(
```

```
@WebParam(name = "parameter") int parameter) {
```

SOAP message:

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ser="http://services/">
   <soapenv:Header/>
   <soapenv:Body>
      <ser:askAvailabilites>
         <parameter>?</parameter>
      </ser:askAvailabilites>
   </soapenv:Body>
</soapenv:Envelope>
```

AskDemandsWS

```
@WebMethod(operationName = "askDemand")
public Demands askDemand(
@WebParam(name = "parameter") int parameter) {
```

SOAP message:

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ser="http://services/">
   <soapenv:Header/>
   <soapenv:Body>
      <ser:askDemand>
         <parameter>?</parameter>
      </ser:askDemand>
   </soapenv:Body>
</soapenv:Envelope>
```

AskDutiesWS

```
@WebMethod(operationName = "askDuty")
    public String askDuty(
@WebParam(name = "parameter") int parameter)
```

SOAP message:

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ser="http://services/">
   <soapenv:Header/>
   <soapenv:Body>
      <ser:askDuty>
         <parameter>?</parameter>
      </ser:askDuty>
   </soapenv:Body>
</soapenv:Envelope>
```

AskEmployeesWS

```
@WebMethod(operationName = "askEmployee")
    public Employees askEmployee(
@WebParam(name = "parameter") int parameter) {
```

SOAP message:

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ser="http://services/">
   <soapenv:Header/>
   <soapenv:Body>
      <ser:askEmployee>
         <parameter>?</parameter>
      </ser:askEmployee>
   </soapenv:Body>
</soapenv:Envelope>
```

The next two web services are different from the previous ones. Both have two parameters. These parameters are the dates in which the roster lies.

AskNumbers

```
@WebMethod(operationName = "askNumberElements")
    public Vector<Integer> askNumberElements(
@WebParam(name = "parameter") Date parameter,
@WebParam(name = "parameter1") Date parameter1)
```

SOAP message:

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ser="http://services/">
   <soapenv:Header/>
   <soapenv:Body>
      <ser:askNumberElements>
         <!--Optional:-->
         <parameter>?</parameter>
         <!--Optional:-->
         <parameter1>?</parameter1>
      </ser:askNumberElements>
   </soapenv:Body>
</soapenv:Envelope>
```

AskVectors

```
@WebMethod(operationName = "askVectors")
public vectors askVectors(
@WebParam(name = "parameter") Date parameter,
@WebParam(name = "parameter1") Date parameter1) {
```

SOAP message:

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ser="http://services/">
   <soapenv:Header/>
   <soapenv:Body>
      <ser:askVectors>
         <!--Optional:-->
         <parameter>?</parameter>
         <!--Optional:-->
         <parameter1>?</parameter1>
      </ser:askVectors>
   </soapenv:Body>
```

```
</soapenv:Envelope
```

1.6.2. Obtaining data

The way of obtaining data is the same in 5 of the web services. These services are the ones which correspond to the database entities. To illustrate it, an example obtaining an ability is shown. The process is divided in steps to ease the comprehension:

First step.

AskAbilitiesWS

```
database db = new database(); - The instance of the database is
created
message = db.prepare(); - Connection to the database

conn =  (Connection) db.getConn(); - Get the connection object.
ab1 =  ab.ability(parameter,conn); - Call to obtain the ability.
```

Second step.

Abilities class.(ability method)

```
rs = db.ability(parameter,conn); - Execution of the query.

ab = new Abilities( rs.getInt(1), - Creation of the new ability.
n,
rs.getString(3),
rs.getFloat(4));

return ab;
```

Third step.

AskAbilitiesWS

```
db.dissconect(conn); - Disconnection of the database

return ab1; - Return of the requested ability.
```

The services askNumbers and askVectors work in a different way

askNumbers.

The working of this web service could also be explained in three steps. Every logic class has a method to get the number of elements of that class in the database.

First step.

AskNumbers

```
Vector<Integer> v = new Vector<Integer>(5); - Vector which will be
filled with the numbers.
message = db.prepare();

 conn =  (Connection) db.getConn(); - Connection to the database

 --This adds the different numbers to the vector.
```

```
v.add(dt.numberDuties(conn));
v.add(emp.numberEmployees(conn));
v.add(dem.numberDemands(parameter,parameter1,conn));

v.add(av.numberAvailabilities(parameter,parameter1,conn));
v.add(ab.numberAbilities(conn));
db.dissconect(conn);}
```

Second step

(Abilities has been chose for this example, but it could have been any other one) Method numberAbilties()

```
rs = db.numberAbilities(conn); – Execute the query

n =  rs.getInt(1); – Get the number

return n; – Return the number.
```

Third Step.

AskNumbers

```
--Return the vector
return v;
```

The askVectors service works in the same way but instead of using a generic vector of integer a "vectors" object is used:

```
vectors v = new vectors();

v.setAbilities(ab.idsAbilities(conn));
v.setAvailabilities(av.idsAvailabilities(parameter,
parameter1,conn));
v.setDemands(dem.idsDemands(parameter, parameter1,conn));
v.setDuties(dt.idsDuties(conn));
v.setEmployees(emp.idsEmployees(conn));
```

In this case all the classes in the logic package have a method to get the a vector of identifiers.

1.7. Database

The database used for this application as is the same that is used by the RostApp application. The database class interacts with it in the same way that the database class does in RostApp.

The only change is due to the high concurrency of this application. Every web service's consumption needs a unique instance of the database (a unique connection resource) this connection is created at the beginning of the web service and is passed through the different methods until the query is executed.

In RostApp, there was only one user using the application in one computer at a time so a global connection could be created and used to execute all the queries.

Too see the scheme and diagram of the data base, see the third appendix.

1.8. Test

See Appendix IV

ANEXO VI – LEISURECENTERSERVER

1. LEISURECENTERSERVER

LeisureCenterServer is a web application that the leisure centre provides to their employees to obtain a roster whenever they want and wherever they are. This application works serving a web page.

1.1. Requirement specification

Functional requirements

Requirements related to the usage of the application.

- **FR01 – Users.** Two types of user are distinguished: Manager and Employee
- **FR02 – Obtaining of a rostering solution.** The application will allow the obtaining of complete rostering solution through the consumption of the web services provided by MidlandHR.
- **FR03 – Views.** The information from the roster could be visualized according to three different criteria: Task, Person and a mix of task and person organised by time.

Non Functional requirements

- **NFR01 – Concurrency.** The application must be highly concurrent.
- **NFR02 – Password**. To start the application a password will be needed. Each employee will have a password.
- **NFR03 - Availability**. The application should be online 24/7

1.2. Design

This application has two types of elements: web pages and servlets. The web pages will interact with the user and the servlets will manage the data entered by the user and they will process the roster.

1.2.1. Web pages.

The web pages are written in JSP and they also have some Java fragments. The visual design is very simple but effective. A schema of the navigation among the pages:

index.jsp

This is the main web page of the application. It is the login page. The user types his/her user name and password and clicks a button to access the rostering service. The page takes the data entered and sends it to servlet (validateUser). If the data is correct the servlet redirects the user to their correspond web page (selectDate or selectDateEmp). If the data is wrong it redirects the user to an error page. Figure 43 shows the index.jsp page.

The data is obtained with two text boxes and sent to the servlet with a JSP form (post method).

Figure 43– index.jsp

SelectDate.jsp

This web page is accessed by the manager. This web page allows the manager to enter the roster's dates and the roster's views that he/she wants. A JSP form sends the information to the validateDate servlet. The servlet will redirect the user to the roster solution. Figure 44 shows the SelectDate.jsp page.

Figure 44 – SelectDate.jsp

SelectDateEmp.jsp

This web page is accessed by the employees. This web page allows the employees to enter the roster's dates and what roster's view that they want. A JSP form sends the information to the validateDateEmp servlet. The servlet will redirect the user to the roster solution.

Error.jsp

This page informs about an error entering the user and password information.

ErrorRoster.jsp

This page informs about an error in the rostering process.

RosterImage.jpg

Finally, if the data is correct the user obtains an image of the roster solution. This image is generated by Roster package and served by the validateDate/validateDateEmp servlets. Figures 45 and 46 show the RosterImage.jpg.

Figure 45 – RosterImage.jpg

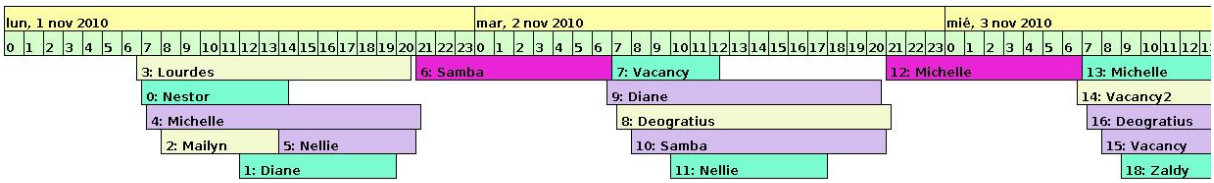


Figure 46 – RosterImage.jpg zoom.

1.2.2. Servlets

There are three servlets in the server. These servlets do not work alone and they need extra classes to support their work. A schema showing the servlets and the classes:

validateUser

This servlet checks the user's name and password. If the data is validated it redirects the user to the selectDate/selectDateEmp pages. If the user is the manager, the servlet validates the password but if the user is an employee the username and password have to be checked in the system's database.

Database

This class supports the validateUser servlet. It has the same stub than the other database classes used in the project. It creates the connection with the database, executes the query and sends the result to

the servlet.

The query is:

SELECT id FROM Users WHERE id = ? AND pass = ?

The servlet fills the "?" with the data entered by the user. If the user exists the redirection is performed, if not the user is redirected to an error page.

ValidateDate and validateDateEmp

These servlets work in the same way; the only difference is the web service that they invoke. ValidateDate invokes the askRoster web service in order to obtain a general roster and validateDateEmp invokes the askRosterEmp web service in order to obtain the roster of an employee.

If the dates are correct these servlets call the web service and obtain the roster solution. Once they have the solution they create the image and create a new web page with the image.

The creation of the image is a complex process. Drawing the roster requires the package provided by Arturo Castillo, however, that package draws the roster in a Java Swing element, not in a jpg file. Therefore the package has been adapted to draw the image in a jpg file.

In RostApp, once the roster was shown, the user could change between the different views. In this case it works differently; there were two possibilities:

1. Creating and arraying with the three views and redirecting the user to the web page where he could choose the image. This solution is more interactive but sending three images in an array is a heavy load in an internet connection. The size and the complexity of the roster are not known, so this choice with several clients at the same time could not work properly.

2. The second choice was created only with one of the views and put it in a web page where the user is redirected. This solution was light but it also has its limitations. If the user wants to check the different views he/she has to call the web service three times.

Both solutions were valid but the second one was chosen because it is better for the server dealing with only one image.

One auxiliary class is needed. This class is rosterImage, it creates a new allocationPanel (adapted version) and returns the created image:

```
allocationPanel = new AllocationPanel();
allocationPanel.setView(view);
byteArray =  allocationPanel.setSolutionList(sol);
```

The new version of the allocationPanel writes the image in a BufferedImage instead of a swing panel. The BufferedImage is written in a ByteArrayOutputStream which is returned. For this purpose the methods setSolutionList and savePanel have been created.

The process to create the image is the next:

1. The view is set.

```
if (request.getParameter("view").equals("general")){
      select = 1;
}
else if (request.getParameter("view").equals("person"))
{
      select = 2;
}
else{
      select = 3;
```

```
}
```

2. A new instance to the rosterImage class is created. The roster solution and the view are passed.

```
roster = new rosterImage(sol,select);
```

3. Creation of the image.

```
byteArray = roster.createImages();
```

4. Writing the image in a buffer where the servlet redirects the user

```
ServletOutputStream bufferSalida = response.getOutputStream();
response.setContentType("image/jpeg");
response.setContentLength(byteArray.length);
bufferSalida.write(byteArray);

bufferSalida.flush();
 bufferSalida.close();
```

ANEXO VII – REST

1. REST

The applications presented above are based on the SOAP model. In this annex the REST applications are presented. The applications MidlandHR, LeisureCenter and ROSTAPP have been transformed into this REST approach.

1.1. HRRest

This application is the same as MidlandHR but instead of SOAP web services it uses REST web services. The structure of the application is the same and it works in the same way. Only one a few changes have been made to adapt the application to the REST approach.

First of all, a new package called webServices has been created. This package contains the classes that get the REST resources. One class has been created for every web service.

These classes are generated by the IDE but they need some changes to work properly. The classes are Jersey REST clients.

The second change is how the web services are consumed. The structure and parameters of these web services are explained in the next annex. Here, the only important thing to know is that the resources are strings with the fields separated by spaces. The parameters passed to the web service need to be transformed into strings. There is no problem with the integers but the dates need an agreed format between provider and consumer if an easy read is wanted.
Java provides a mechanism called SimpleDateFormat which allows to transform dates to strings and viceversa. It is a very simple process in both ways.

```
SimpleDateFormat formateador = new SimpleDateFormat("HH:mm
dd/MM/yyyy");
date1 = formateador.format(start);
```

In the SOAP version the web services returned objects, in this case we have to create the object and fill its fields with the different parts of the string. To consume a web service the following steps are needed:

1º Creating an instance to the Jersey client to consume the web service

```
webServices.Abilities client = new webServices.Abilities();
```

2º Using the method of the client to get the resource. A string with the id of the ability is passed and a string is obtained.

```
String response =
client.ability(String.class,v.get(i).toString());
```

3º. Now, it is time to parse the string, a scanner is created in this case, but there are several options to parse the string.

```
Scanner s = new Scanner(response);
s.useDelimiter(" ");
```

4º Getting the fields of the ability

```
ab.setNid(Integer.parseInt(s.next()));
ab.setName(s.next());
ab.setDutie(s.next());
ab.setValue(Float.parseFloat(s.next()));
```

5º .Once the ability is completed, closing the connection.

```
client.close();
```

The performing is the same with the different entities (availabilities, numbers,employees...). The layered division of the system has allowed us to modify the working of the system only changing a few lines in some classes. This is one of the advantages of modularity and a good analysis phase.

1.2. LeisureCenterRest

This application is simpler than the SOAP version and it shows the advantages of REST web services (see annex X).

Instead of having a three layers structure only two layers are needed:One layer for the web services and another for the database interaction.

Web services could not be the perfect term to write about the REST services. In REST, there are no special methods designed to be called by the client and return an agreed result. There is not WSDL. REST services are a direct way to offer resources; they create a URL which the client accesses to get them. For this reason, they will be referred as REST resources.

Normally the data returned by the REST resources is in XML, in this case it is just a string. This change has been adopted in order to ease the implementation of HRRest, otherwise the parsing of the XML to the system entities (employees, abilities...) would have been difficult.

In the SOAP LeisureCenter there were three methods in each entity: one for itself, one for the ids and one for the number of it in the database. This version keeps those methods but it is structured in a different way.

Classes:

Each class will represent a REST resource

Database

This class is used to connect with the system database and execute the queries to retrieve information.

Connection conn: Object to connect with the database

Abilities, availabilities, employees, demands and duties

These classes work similarly. They return a string with the fields of the requested entity.

IdsAvailabilities, idsAbilties, idsEmployees, idsDemands and idsDuties.

These classes work very similarly. They return a vector of numbers converted into a string.

Numbers

This class has methods to retrieve the quantity of each entity in the database and return the vector converted into a string.

The methods used in these classes are the same methods used in the SOAP version. In order to adapt them to the REST approach three changes have been made:

1. Adding the path at the beginning of the class:

```
@Stateless
@Path("/Employees")
public class Employees {
```

The IDE creates automatically all the necessary infrastructure to support this. To get an employee the path will be: http://host:8080/LeisureCenterWeb/Employees/id

2. Creating a REST method and indicating if it is a GET or POST method. It the method has parameters it is necessary to indicate what will be the name of the parameters so that the client can retrieve the information.

```
@GET
 public  String employee(@QueryParam("id")String parameter)
```

3. Returning a string. This is very easy to do, Java has methods to convert most of predefined types to strings (even vectors)

```
return id.toString() + " " + name + " " + contract.toString() +
"  "  +  Float.toString(wage)  +  "  "  +  Float.toString(ranking)  +
" " + Float.toString(capacity);
```

Applying these changes to the existent methods all the REST methods are created. As can be seen one class has been created for each id type but only one has been created for the numbers resources. This is because if only one method is used for the ids, a vector of vectors is obtained and converted into a string. This string is more difficult to be converted again into a vector than a simple vector string. Therefore, one REST resource has been created for each vector.

1.3. Tests.

In order to test the REST resources, the IDE creates a web environment to test the web services, allowing to change the parameters and to see the results.

1.4. RostAppRest

This application is the same application that RostApp, the only difference between them is RostApp uses the SOAP web services and RostAppRest uses the REST web services.

ANEXO VIII – TRABAJO FUTURO

Cómo ya se ha señalado este proyecto tiene varias áreas susceptibles a mejoras. Una de ellas es la forma en la que se tratan los mensajes procedentes de los servicios Web. En el diseño propuesto hay dos zonas en las que se consumen servicios Web; por un lado la empresa que solicita el roster y por el otro la empresa de *rostering* que consulta los datos necesarios para la elaboración del roster.

El proceso de consulta de los datos de la empresa solicitante puede ser mejorado a través de la utilización de *pools* de conexiones o técnicas más eficientes al obtener los datos de una base de datos.

En la otra zona, a priori, no hay ningún problema y el funcionamiento del proyecto ha sido satisfactorio. Sin embargo, solo una empresa está solicitando rosters, lo cual se encuentra lejos de un escenario real en el que más empresas solicitarían rosters al mismo tiempo. El procesado de estos mensajes y posterior creación y devolución de los rosters puede resultar muy pesado para un servidor que no gestione adecuadamente la llegada y tratamiento de estos mensajes. De la misma manera, futuros desarrollos dentro de la empresa prevén la integración entre diferentes plataformas lo cual requiere que varios servicios web funcionen de forma coordinada llamándose unos a otros.

Por ello, una de las posibles mejoras de este proyecto sería la utilización de un gestor de los mensajes de los servicios Web. En este tipo de situaciones se utilizan los conocidos como *brokers* de mensajes, que se encargan de recibir, procesar y distribuir los mensajes que llegan al servidor para garantizar un servicio lo más eficiente posible además de coordinar las posibles interacciones entre servicios Web (internos y externos).

Los *brokers* funcionan de acuerdo a algoritmos que se encargan de coordinar las coreografías entre los diferentes procesos Web. Entre las posibles alternativas, encontramos RLinda [28] este sistema de coordinación se basa en el paradigma Linda [61], que se usa como un lenguaje intermedio para la definición de interacciones. Existen varios desarrollos basados en este modelo tales como *JavaSpaces* [62]. *GigaSpaces*[63] o *XMLSpaces*[64].

Linda es un lenguaje de coordinación que provee comunicaciones a través de la producción y consumo de estructuras de datos pasivas en un espacio compartido. Los datos se representan como tuplas por lo que el espacio compartido se conoce como "espacio de tuplas".

Este espacio es utilizado por una serie de procesos para comunicarse e interaccionar a través de la inserción de tuplas (colecciones o conjuntos de elementos) en él. Este espacio posee una serie de operaciones: introducir una tupla, eliminar una tupla y consultar una tupla. Una vez introducidas, las tuplas se mantienen en el espacio esperando a ser consumidas o consultadas.

Para buscar las tuplas que han de ser consultadas o consumidas, RLinda cuenta con un conjunto de funciones de emparejamiento, que pueden ser utilizadas dependiendo del problema, contexto y escenario fijándose en la estructura de las tuplas o en sus atributos.

En nuestro proyecto las tuplas pasan a ser las peticiones de roster y los rosters. Las diferentes empresas introducirían sus peticiones a través de la invocación de la operación de entrada con un mensaje SOAP; estas peticiones se almacenarían en el espacio de tuplas en formato *XML*. El servidor de la empresa se encargaría de ir consumiendo esas peticiones y una vez consumidas y procesadas, los rosters creados son introducidos en el espacio de tuplas para ser recogidos por las empresas solicitantes.

La operación de consulta permite al servidor seleccionar las peticiones a tratar de forma que se maximice la eficiencia del servidor y se reduzca el tiempo de espera de los clientes. La figura 47 recoge la integración y funcionamiento de RLinda.

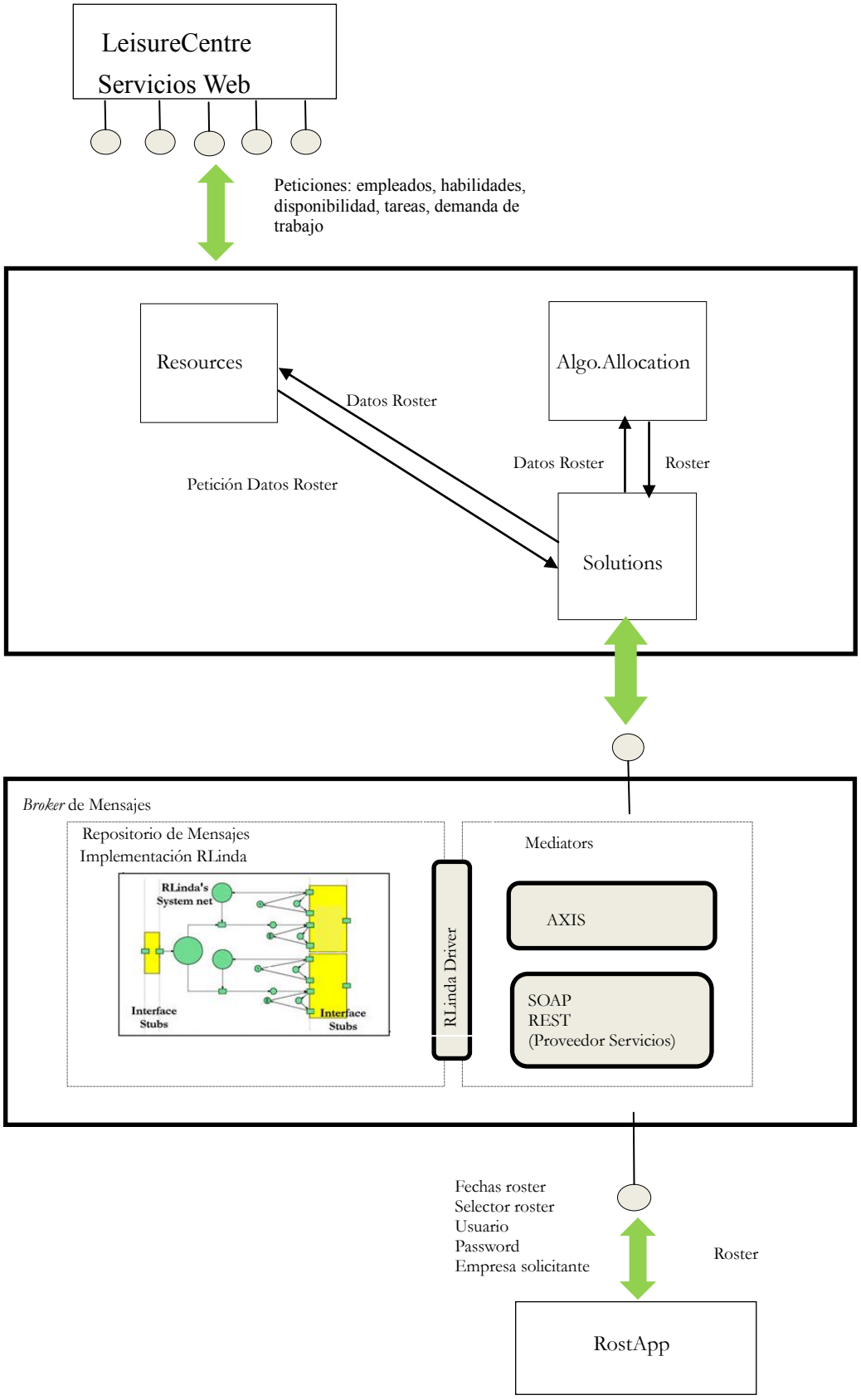


Figura 47– Integración RLinda

Dentro del bróker de mensajes podemos observar dos zonas. En una de ellas se encuentra el repositorio de mensajes RLinda que funciona de la forma explicada anteriormente. En la otra zona nos encontramos con dos componentes: uno Axis y otro SOAP/REST. El objetivo del componente *AXIS* es publicar los servicios Web que ofrece la empresa y el del componente SOAP/REST es devolver los resultados que ofrecen los servicios Web de la empresa al ser consumidos. Ambas partes se comunican a través de una serie de *drivers*.

Dentro de MidlandHR desparece el módulo *Servicios* que se encargaba del consumo y publicación de éstos. Esta labor recae ahora en el *bróker*. El *bróker* y MidlandHR se comunican a través de un servicio que el *bróker* publica y MidlandHR consume. De esta manera el bróker procesa las tuplas y envía las peticiones al módulo *Solutions* que se encarga de obtener los *rosters* y devolverlos al *bróker*.

A partir del trabajo desarrollado para RLinda los mismos autores han desarrollado DRLinda[29]. DRLinda es una implementación dinámica y distribuida del *broker* de mensajes introducido en RLinda. Este modelo distribuido intenta solucionar de manera eficiente algunos de los problemas que el modelo centralizado presentaba, mejorando y ampliando las características de RLinda. Puede ser configurado en tiempo de ejecución lo cual lo hace más adecuado para escenarios donde existan procesos más complejos y dinámicos. La figura 48 muestra la estructura de este sistema.

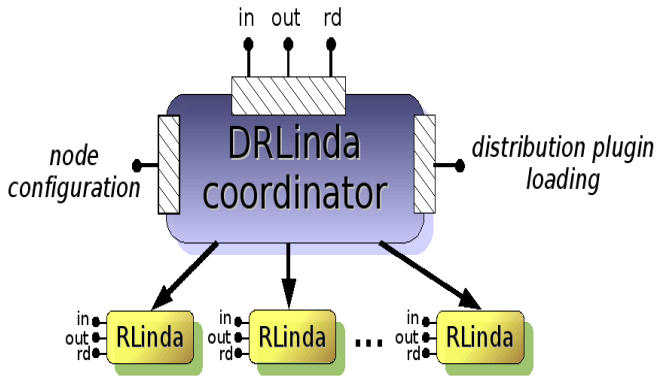


Figura 48 Resumen arquitectural de DRLinda

La estructura muestra como ahora existe un elemento central que se encarga de controlar diferentes espacios de tuplas. Para el cliente, la interfaz a usar es la misma (mismas operaciones), sin embargo, la tupla no irá directamente al espacio de tuplas, sino que pasará antes por un elemento central que la enviará a un espacio determinado dependiendo de situación del sistema en ese momento.

A su vez, existe una nueva interfaz para la configuración dinámica del sistema. Esta interfaz permite cambiar en tiempo de ejecución la configuración de los nodos (Número de nodos RLinda disponibles, localización, restricciones, limites, características…) y la función de distribución. De esta manera DRLinda puede adaptarse a cualquier escenario dinámico y a cualquier topología o distribución de nodos.

Más allá de la escalabilidad de este nuevo sistema, las posibilidades que ofrece al problema propuesto en el proyecto son amplias. Por un lado, la empresa pasa a tener un control total sobre la distribución de las tuplas, esto le permite la creación de sistemas de prioridades a los diferentes clientes que pueden ver mejorado su servicio a través de por ejemplo, la contratación de una mejora del servicio.

De la misma manera permite tratar la seguridad, distribución de la carga del sistema y la tolerancia a fallos de forma más efectiva. Las peticiones pueden ser almacenadas en varios espacios por si es necesario recuperarlas por el mal funcionamiento del sistema.

A lo largo de la memoria se ha señalado la complejidad del proceso de *rostering*, este tipo de algoritmos se encuentran en una fase de mejora continua. A través de la configuración de nodos y la utilización de *plugins* las peticiones pueden ser tratadas de la forma más eficiente para las diferentes fases de desarrollo en las que se encuentre el algoritmo de *rostering*. De la misma manera, el hecho de poder organizar las peticiones en diferentes espacios facilita un posible cacheo de datos a la hora de elabora los rosters.

Finalmente, la figura 49 muestra la integración del sistema desarrollado en este proyecto con DRLinda.

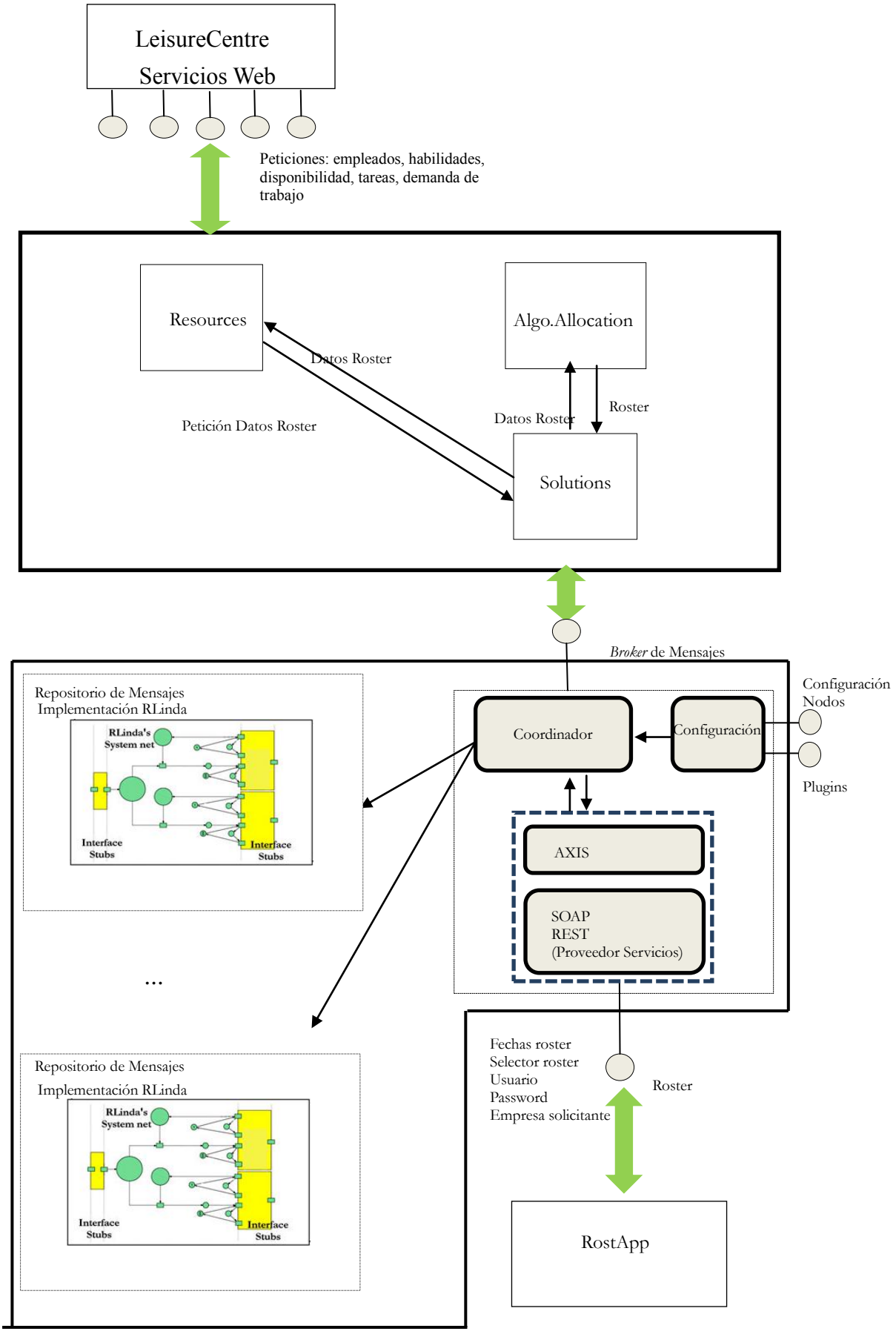


Figura 49 – Integración DRLinda

ANEXO IX – SECURITY

1. SECURITY

Security is a fundamental part of web services. In this project, private and confidential information is sent through them. Therefore, it is important to use the appropriate tools to provide confidentiality, integrity and privacy to the system.

In web services, security is related with efficiency. Adding security layers affects to the efficiency of the service because there is more data to be processed. In order to measure the relationship between security and efficiency, three different security configurations have been created and tested. The configurations have been created using WSIT.

WSIT is developed by Sun and Microsoft and it is a part of the Metro Web Services Stack. WSIT is an implementation of a number of open web services specifications to support enterprise features. In addition to message optimization, reliable messaging, and security,

By default, web services are able to use transport security working over SSL to provide point-to-point security. WSIT implements WS-Security (WS-S) in order to provide message interoperability with integrity and confidentiality. WS-S implements WS-Secure Conversation, which creates a shared security context. This context is used by consumer and provider to exchange a sequence of multiple messages. Therefore, the next messages use derived session keys that increase the overall security while reducing the security processing overhead for each message.
WS-S also implements Web Services Security Policy. This technology allows including assertions that represent security preferences and requirements for web service endpoints.

To understand these assertions and policies, a tool called wsimport is used. This tool obtains the WSDL from the web service's URL and creates the client according to the policies and assertions specified in the WSDL document.

1.1. Security Configurations.

The different security configurations will be applied to all the web services (MidlandHR's + Leisure Centre's). The configurations have been created following the WSIT tutorial [24]

1. **No security**

No security measures have been taken in this configuration. It will be the base case with which the other configurations will be compared to.

2. **SSL**

Secure Sockets Layer (SSL) and Transport Layer Security (TLS) are cryptographic protocols which provide secure communications through a network.

SSL provides authentication and privacy on the information sent. Normally, only the server is authenticated while the client is not authenticated. There are three basic steps in a SSL connection:

- Negotiation between the parts about the algorithm that will be used in the communication.
- Exchange of public keys and authentication based on digital certificates.
- Encrypting all the traffic (symmetric encryption).

In order to configure SSL in our system it is necessary to configure Glassfish and the application. In Glassfish the SSL certificate has to be selected, in this case (development) the predefined keystores and truststrores of Glassfish will be used in both sides (consumer and provider). The certificate used for this

configuration is "s1as" in the localhost scenario, for a distributed scenario X.509 certificates have been created (Appendix III).
To configure the application, the file "web.xml" has to be modified with this code:

```
<security-constraint>
        <display-name>Constraint1</display-name>
        <web-resource-collection>
            <web-resource-name>c1</web-resource-name>
            <description/>
            <url-pattern>/*</url-pattern>
        </web-resource-collection>
        <user-data-constraint>
            <description/>
            <transport-guarantee>CONFIDENTIAL</transport-guarantee>
        </user-data-constraint>
    </security-constraint>
```

"Constraint1" and "c1" are names chose by the programmer. The important part of the code is ask for confidentiality in all the URLs of the application (/*)

In order to perform the test, the original working scenario (localhost) needs to be distributed different computers. In the new scenario, the Glassfish's development certificates are not valid. A easy option is create our own certificates and sent them to enterprises like Verisign[51] or Comodo[65]. These companies sign trial certificates for free. In this case, these companies are useless, because they do not sign certificates for IP hosts (In the case of our private network: 192.168.1.100...). So it has been necessary to create our own certification authority (CA) to sign our own certificates.
The steps to create and sign the certificates are shown in the annex XIII.

3. Mutual Certificates + SSL

The Mutual Certificates Security mechanism adds security via authentication and message protection that ensures integrity and confidentiality. When using mutual certificates, a keystore and truststore file must be configured for the client and server sides of the application.
This level of security with SSL provides integrity, confidentiality and privacy and authentication in both sides.

In order to implement this policy it is necessary to configure the service and the client of the service. Netbeans offers a great support and automates all the creation process of the XML file of the service. In the service side, a certificate must be selected from the keystore (xws-security-server). Once selected, a new XML file of the server is created with all the security constraints. In the file there are two fields that represent the selected keys.

```
<sc:KeyStore wspp:visibility="private" location="/home/miguel/glassfish-
3.0.1/glassfish/domains/domain1/config/keystore.jks" type="JKS"
storepass="changeit" alias="xws-security-server"/>
```

The WSDL of the service is also automatically modified with the security constraints by the IDE

The first step in the client side is to get the WSDL of the service. Once received, the security has to be configured. Two steps are needed:

- Provide the client's private key by pointing to an alias in the keystore (xws-security-client)
- Provide the server's certificate by pointing to an alias in the client truststore.

Now, the service's XML file has a new field:

```
<wsp:Policy wsu:Id="rosterPortBindingPolicy">
        <wsp:ExactlyOne>
```

```
            <wsp:All>
                <sc:KeyStore wspp:visibility="private" alias="xws-security-
client" storepass="changeit" type="JKS" location="/home/miguel/glassfish-
3.0.1/glassfish/domains/domain1/config/keystore.jks"/>
                <sc:TrustStore wspp:visibility="private" peeralias="xws-
security-server" storepass="changeit" type="JKS"
location="/home/miguel/glassfish-
3.0.1/glassfish/domains/domain1/config/cacerts.jks"/>
            </wsp:All>
        </wsp:ExactlyOne>
    </wsp:Policy>
```

ANEXO X – TESTS

1. TESTS

Two types of tests have been performed: SOAP tests and Load tests. The SOAP tests have checked the correctness of the web services' messages and the Load tests have stressed the system in order to obtain conclusions of the suitability of the combination of web services and rostering.

Quality of service requirements for web services are: availability, accessibility, integrity, reliability, throughput, latency, conformance to the standards and security.[66]

SOAP tests.

These tests are automatically performed by SoapUI, all the SOAP messages have been validated by this program. All of them accomplish the standard of SOAP messages.

Load tests.

These tests simulate a real work environment. Different cases are simulated and performed in order to measure time, CPU, memory...

The scenario in these tests has 3 computers which share a private WiFi network. Each part of the system has been installed in one computer. The characteristics of the computers are:

- **Computer 1 – RostApp**
  Ubuntu 10.10 maverick
  Linux Core 2.6-35.25.
  Gnome 2.32.0
  Intel(R) Core(TM)2 Duo CPU T7300 @ 2.00GHz
  2GB DDR2
  Server Glassfish 3.0.1

- **Computer 2 – MIDLANDHR**
  Ubuntu 10.10 maverick
  Linux Core 2.6-35.25.
  Gnome 2.32.0
  Intel Pentium Dual Core Processor T2370 @ 1,73GHz
  Enhanced Intel SpeedStep technology
  2GB DDR2 Go SDRAM
  Server Glassfish 3.0.1

- **Computer 3 – LeisureCenter**
  Ubuntu 10.10 maverick
  Linux Core 2.6-35.25.
  Gnome 2.32.0
  Intel(R) Core(TM)2 Duo CPU T7300 @ 2.00GHz
  2GB DDR2
  Server Glassfish 3.0.1

The aim of these tests is to stress the system in order to obtain different measures to analyze the combination of web services and rostering. There are different load tests. It is important to explore all the possibilities in order to achieve a global knowledge of how the system responds to different situations. The system expects peak accesses of 15 concurrent users asking for different types of rosters: individual, global, 1 week, two weeks...

The tests have been performed using SoapUI to simulate the different users (threads) asking for rosters. VisualVM [27] has also been used to control the CPU and memory of the computers. The graphical results of VisualVM are included in CD-ROM in the folder "Results", the soapUI's reports are also included.

The following scenarios have been tested:

- Scenario 1 – No Security

In this scenario there is not any security configuration in the system.

- Scenario 2 – SSL

In this scenario the web services use the secure sockets layer.

- Scenario 3 – Mutual Certificates + SSL

In this scenario the web services use the secure sockets layer plus mutual certificates to authenticate among them.

- Scenario 4– REST Web Services + SSL

The SOAP service (roster) works over SSL and the services between MidlandHR and the Leisure Centre are REST web services.

For all these scenarios the following test has been performed:

	1 week	2 weeks	3 weeks
1 user			
5 users			
10 users			
15 users			

1.1. Results

Sizes and BPS

The sizes of the rosters are:

1 week roster: 10511 bytes
2 weeks roster: 21543 bytes
3 weeks roster: 32363 bytes

The bps in the different scenarios have been:

Scenario 1: 2659 bps
Scenario 2: 3598 bps
Scenario 3: 4256 bps
Scenario 4: 4309 bps

Scenario 1

The figure 50 shows the response times for the first scenario.

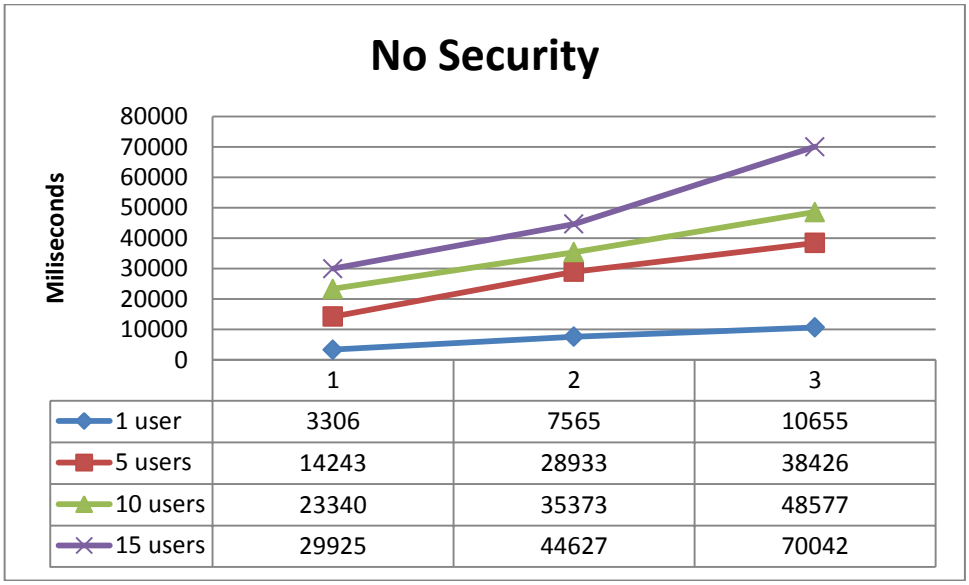


Figure 50. Scenario 1 Results

MidlandHR (CPU % / Memory MB)

	1 week	2 weeks	3 weeks
1 user	88/220	90/230	91/230
5 users	90/225	93/225	95/210
10 users	92/225	93/220	95/220
15 users	93/225	94/225	92/230

Leisure Centre (CPU/Memory)

	1 week	2 weeks	3 weeks
1 user	11/225	12/250	12/260
5 users	15/270	15/270	17/270
10 users	15/280	12/280	17/280
15 users	14/290	15/290	14/280

Scenario 2

The figure 51 shows the response times for the second scenario.

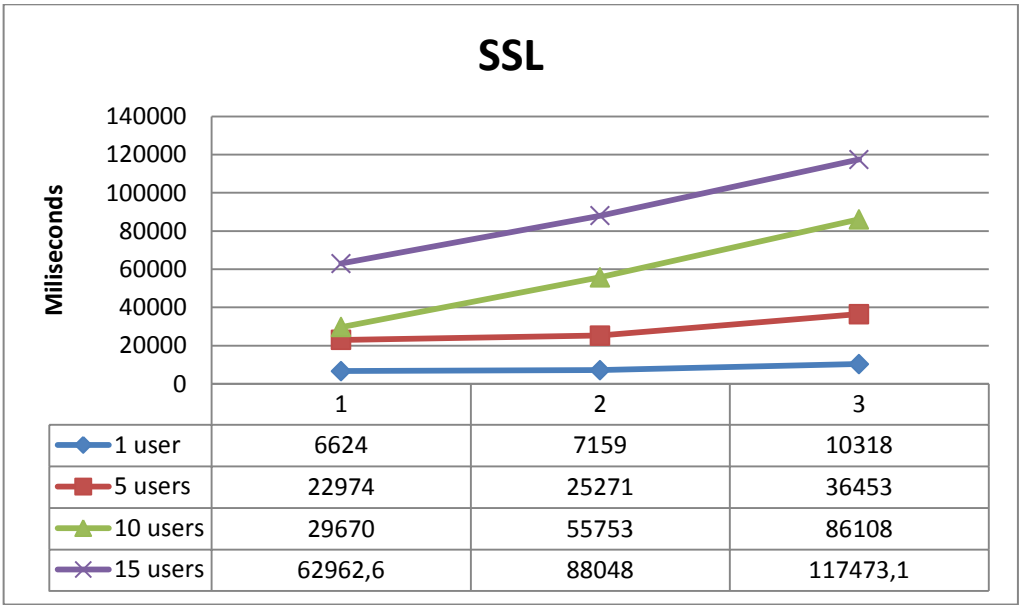


Figure 51. Scenario 2 Results

MidlandHR (CPU % / Memory MB)

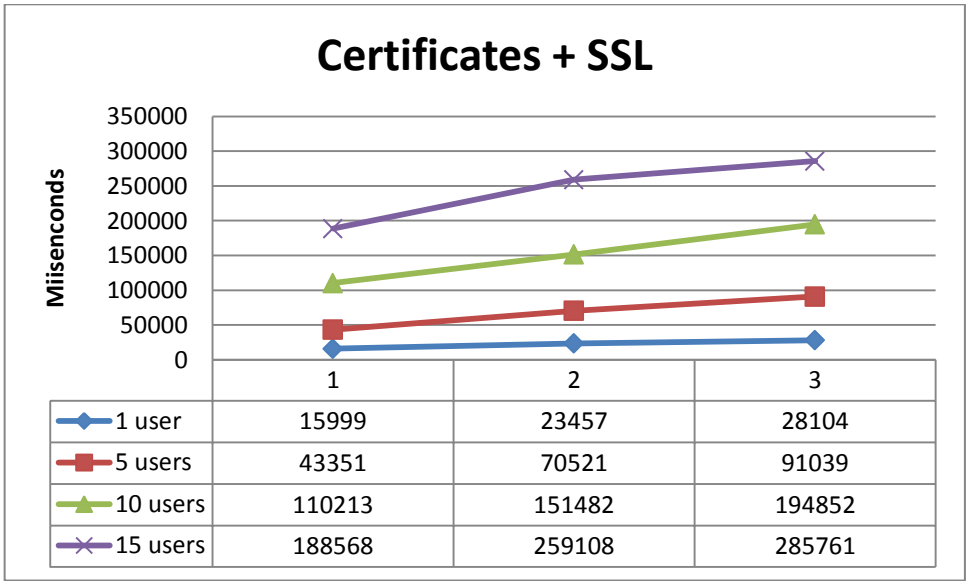
	1 week	2 weeks	3 weeks
1 user	90/175	90/250	90/200
5 users	95/170	95/280	95/260
10 users	93/150	93/260	95/280
15 users	97/280	96/280	97/300

Leisure Centre (CPU/Memory)

	1 week	2 weeks	3 weeks
1 user	27/325	32/350	30/400
5 users	36/325	40/360	36/350
10 users	35/360	45/400	40/400
15 users	45/350	48/400	45/400

Scenario 3

The figure 52 shows the response times for the third scenario.



	1	2	3
1 user	15999	23457	28104
5 users	43351	70521	91039
10 users	110213	151482	194852
15 users	188568	259108	285761

Figure 52. Scenario 3 Results

MidlandHR (CPU % / Memory MB)

	1 week	2 weeks	3 weeks
1 user	85/260	90/270	88/280
5 users	93/300	93/300	93/300
10 users	95/300	95/300	95/350
15 users	96/325	96/350	96/360

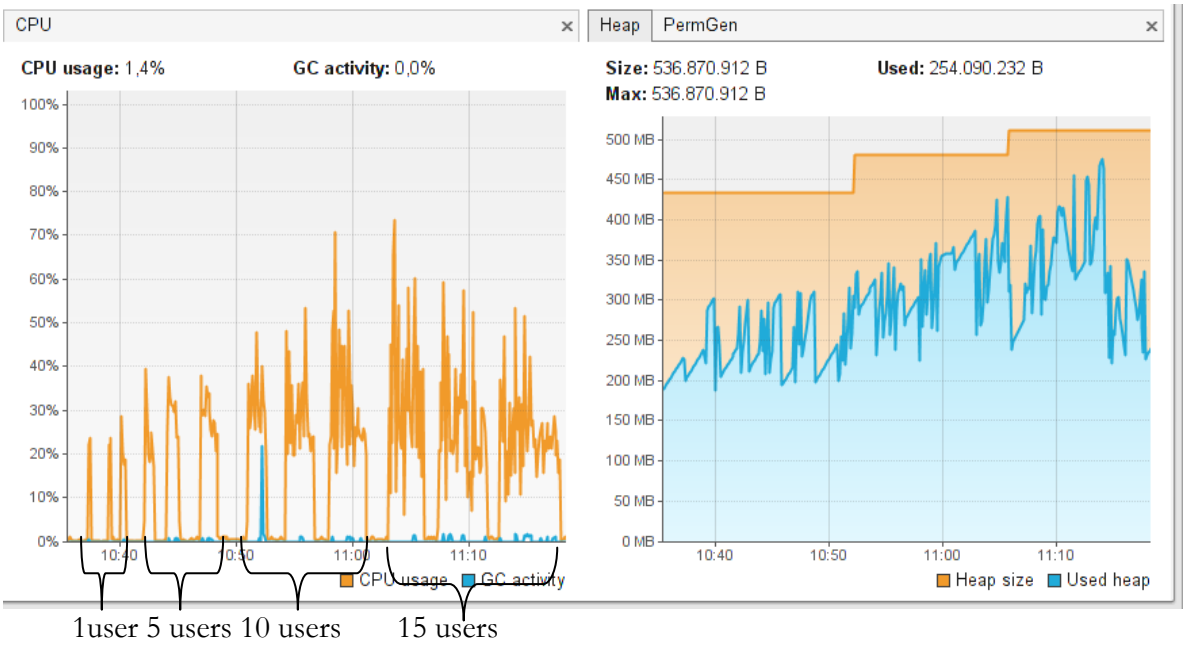


Figure 53. Scenario 3 performance – MIDLANDHR server

Leisure Cente (CPU/Memory)

	1 week	2 weeks	3 weeks
1 user	25/280	25/280	26/300
5 users	30/280	35/300	35/300
10 users	37/300	37/325	40/350
15 users	45/350	45/400	45/350

The figure 54 shows the performance of the server.

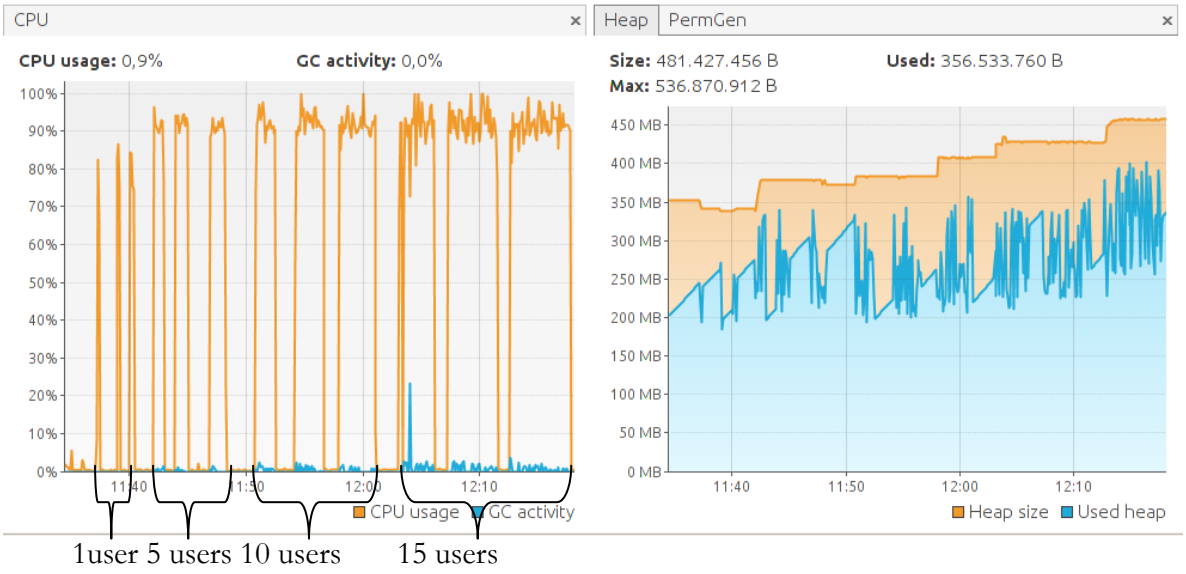


Figure 54. Scenario 3 performance – LeisureCenter server

The results of all these tests follow the expected pattern. The response times increase gradually with the number of users and the size of the roster. Only the scenario without security is under the acceptable response times (60 seconds). SSL gives acceptable times for 10 users or less and the third scenario is only useful for one user.

The usage of certificates doubles the SSL's response times, which makes this option not recommended. The benefits do not worth the cost. However SSL is a recommendable option, because the response times are very similar to the No security scenario and only with more than ten users the times are too high.

The CPU and memory usage are very similar in all the scenarios. As it can be seen, the scheduling process requires almost all the capacity of the processor. The computers used are laptops and they are not valid to work as a real server, however, these tests show the importance of acquiring a high performance server for this kind of purpose. (See 14.2 for more information about the server).

In the Leisure Centre side, the CPU usage increases with SSL and the certificates. The usage in these scenarios triplicates the usage in the no security one. In both cases, the usage is constant and there are not high differences in the different cases.
The memory consumption increases in the expected pattern and it is similar in MidlandHRand in the Leisure Centre. The memory consumption is not as critical as the CPU usage.

Scenario 4

The figure 55 shows the response times for the fourth scenario.

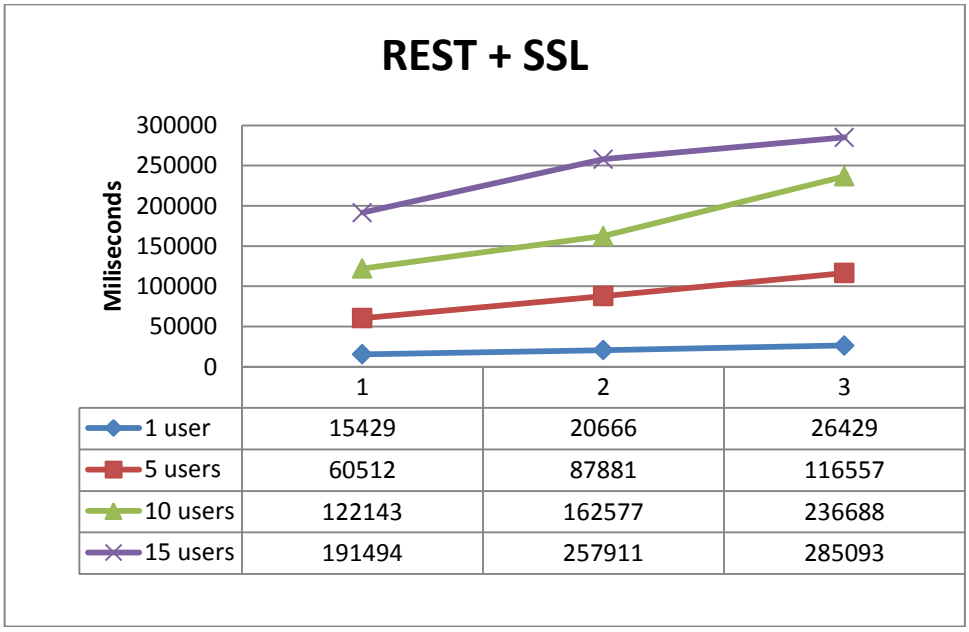


Figure 55. Scenario 4 Results

MidlandHR
The figure 56 shows the performance of the server.

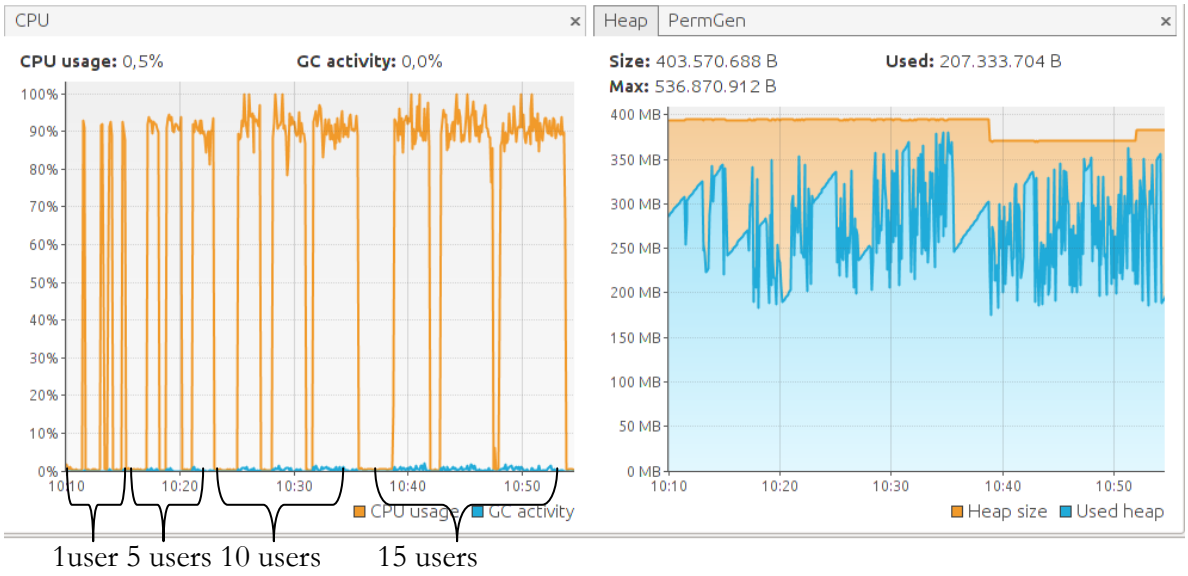


Figure 56. Scenario 4 performance – MidlandHR server

Leisure Centre
The figure 57 shows the performance of the server.

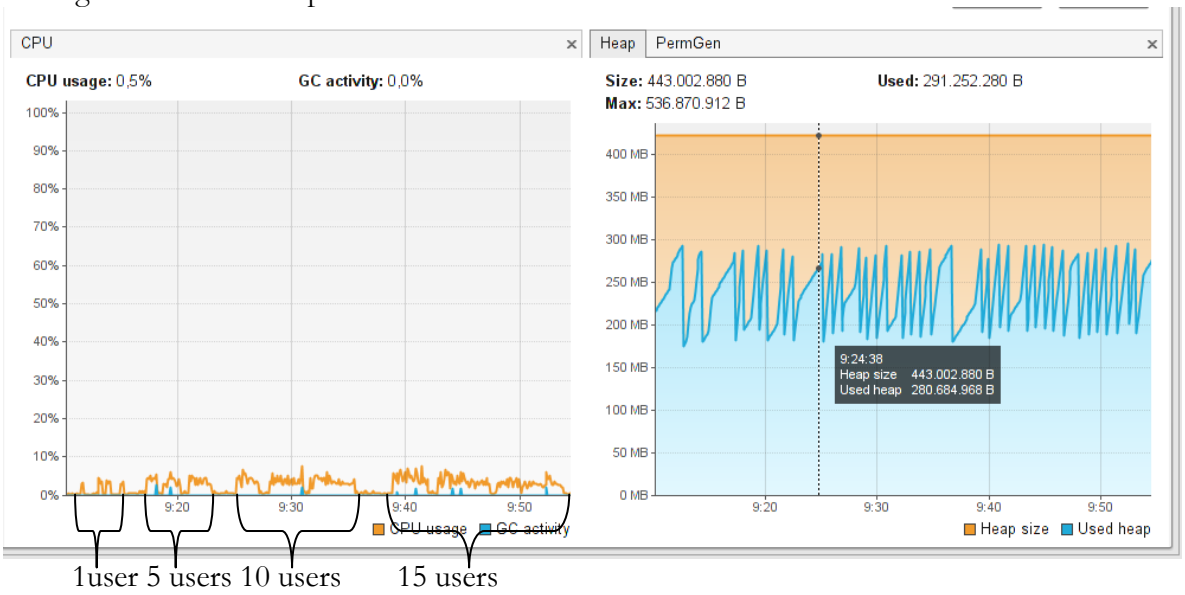


Figure 57. Scenario 4 performance – MidlandHR server

This scenario includes REST and SSL, a scenario without SSL was tested but the data obtained is very similar to the obtained in this one. This is because SSL is only applied to one web service consumption (roster service).
The results in this scenario are surprising. Theoretically REST web services are faster and lighter than SOAP web services but in the results in this scenario are similar to the Certs+SSL scenario.

At this point it can be seen that there is no point in a discussion between REST and SOAP. Both approaches are valid only if they are used in the right scenarios and in the right way. The key point is the implementation, not the kind of service.

The way of operating of the Leisure centre's web services corresponds to a REST approach more than to a SOAP approach. The problem is that the implementation of the REST web services in the system has been done following the SOAP approach so as not to modify the guidelines of the SOAP implementation done before.

In the implementation of the REST services, the data is obtained from the database and it is parsed, prepared and sent. However, there are tools to obtain database information with REST services very easily, quickly and in a XML format. The usage of these tools (EJB Beans, Netbeans provides them and automates the process.) had complicated the implementation of the MidlandHR´s system but it would have reduced the response times.

The previous results show that the CPU usage on the MidlandHR server is around 90%, if more work(parsing and preparing all the data) is added to this system, the system could become unbearable.

In the tested system there are 214 web service calls (218 with the REST services) between MidlandHR's server and Leisure centre's server. They are too many calls and they can even be more if there are more absences or work demands. Repeated SOAP-client/REST calls to access server state can choke a network and degrade the server performance. Data should be cached in the client whenever possible to avoid requests to the server. In this case, caching data cannot be done in a traditional way due to the nature of the system.

Another compromise has to be arranged: many small packets or few big packets. Small packets are easy to manipulate but they require many web service calls, big packets require less calls (in our system only 10) but parsing all the information and entering it into the system can be difficult and it would add more work to the overloaded MidlandHR's server.

Other factors affecting web service performance are: web server response time and availability;

web application execution time (like EJB/Servlets in Web application server); back-end database or legacy system performance.

In summary, the "strange" results in the REST scenario show the importance of choosing the proper technology. They also show the critical part in the system: the data interchange between MidlandHR and the different enterprises. Retrieving data is easy but retrieve it in the right way is not easy, besides the requirements of the enterprises are different. The requisites of the enterprises have to be deeply studied in order to achieve the best global system. Rostering is a heavy process and adding more load to the server could suppose serious problems, however significant response times could suppose problems as well.

The main problem of this aspect for MidlandHRis that every company with which they work has its own system, server, database…therefore configuring them in the proper mode to MidlandHRmight not be done; it depends on the external company.

To clarify this point, a new test was created. This new test aims to measure the proportion of time that the HR server spends retrieving data compared with the time that it spends creating the roster. The results are:

(Total time/ Time retrieving data/ Time calculating the solution (miliseconds))

	1 week	2 weeks	3 weeks
1 user- No security	3396/3316/28	5261/5120/61	6531/6348/92
1 user- REST	11823/11761/35	18194/18056/79	22676/22516/104

The bottleneck in the system is retrieving of the Leisure Centre's data. In the beginning, the most common thought was that the most part of the time was spent calculating the roster but as it can be seen it is not like that.

If the different scenarios are mixed and grouped by the number of users the following results are obtained:

1 user

The figure 58 shows the response times for one user.

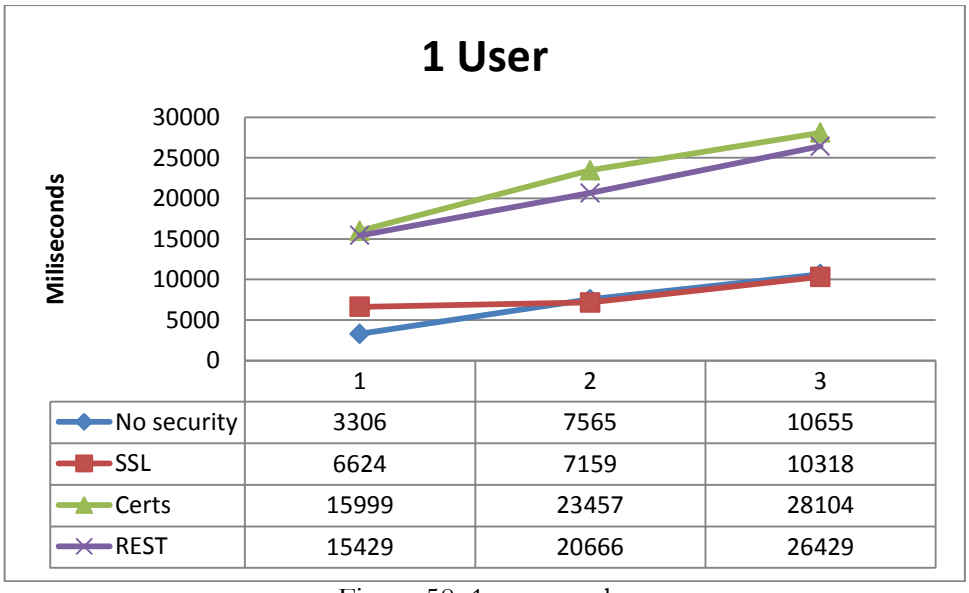
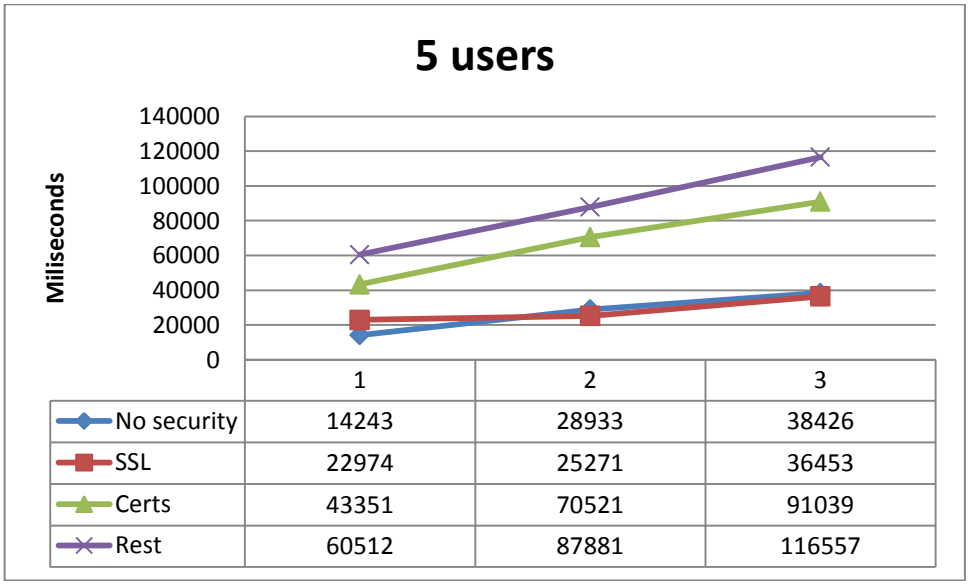


Figure 58. 1 user results

5 users

The figure 59 shows the response times for five user.

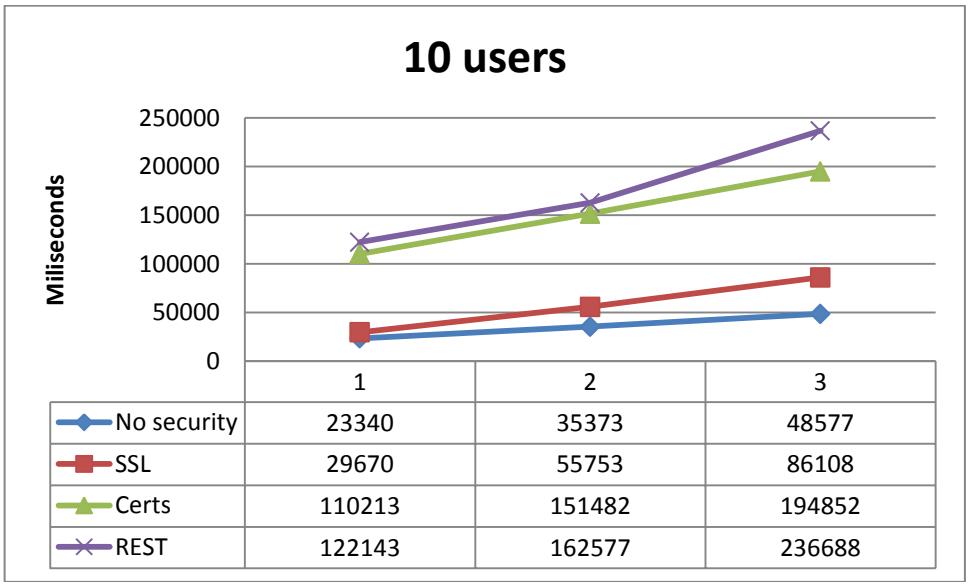


	1	2	3
No security	14243	28933	38426
SSL	22974	25271	36453
Certs	43351	70521	91039
Rest	60512	87881	116557

Figure 59. 5 user results

10 users

The figure 60 shows the response times for ten user.

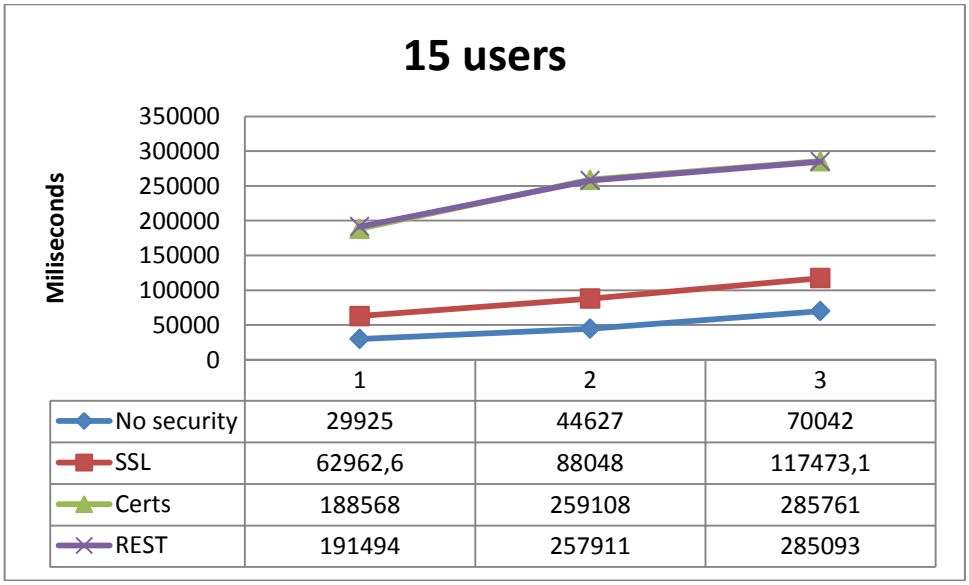


	1	2	3
No security	23340	35373	48577
SSL	29670	55753	86108
Certs	110213	151482	194852
REST	122143	162577	236688

Figure 60. 10 user results

15 users

The figure 61 shows the response times for fifteen user.



	1	2	3
No security	29925	44627	70042
SSL	62962,6	88048	117473,1
Certs	188568	259108	285761
REST	191494	257911	285093

Figure 61. 15 user results

These results show more clearly what has been written before: the Certificates and the REST scenarios are similar and the SSL and No security scenarios are similar.

1.2. Server.

The computers used in these tests are laptops and they are not prepared to work as a server machine. The Glassfish server has been configured to be stressed in these tests.

The high level of CPU usage on the MidlandHR server suggests that the performance of this machine needs to be optimized.

In the Glassfish server there are 5 points that need to be studied and tested in order to provide the best possible results in a production environment:

1. **JVM**
   The last JVM version should be installed. The JVM should be configured in server mode and the memory limits should be modified according to our server characteristics.

2. **Acceptor threads**
   The acceptor threads are the execution threads assigned to a socket in order to process HTTP requests. The default number is 1, but for a better configuration it is better to configure the threads according to the number of the machine's cores. The proportion is 1 thread for each 1-4 core.

3. **Thread pool**
   Processing threads are located in the thread pool. These threads execute the HTTP requests. The default number is 5 but this numbers should be adjusted with the number of cores of the machine.1 Thread for 1 core.

4. **Keep Alive**
   This option is used to optimize the network usage. Normally, a HTTP request requires opening a connection, sending the data and closing the connection. However, if the same connection sends data in different times it is better to keep this connection open. The "keep

alive" parameter allows the administrator to select the number of threads that should be kept opened.

5. **Cache**
   If the system uses static files, caching them in the server is the best option. It is necessary to calculate and allocate the space to store all these data. Normally, caching data supposes a significant performance improvement.

Glassfish has mechanisms to configure all these parameters easily. It counts with an administrator console run by commands and with a visual environment to administrate the server.

ANEXO XI – CONCLUSIONS

1. CONCLUSIONS

The project has achieved its objectives but achieving these objectives, new objectives have appeared in the process. The proposed system has been implemented and it has shown the advantages, disadvantages, key points and weak points of the combination of automated scheduling and web services.

The project has shown how this combination is possible. Web services interoperability eases the implementation of the rostering process in a distributed scenario. Security is an important part of web services and as it can be seen, implementing it is not a hard task. The problem is that security could be expensive and in some cases the rostering company will have to adapt its security to the other company's security policies, which could complicate the whole system.

The tests have shown unfavorable results but they have highlighted the bottleneck of the system: the data interchange of the roster company and the rest of companies. However, the most important issue is not the bottleneck but the external companies and what they expect of the system which they hire.

Mentioning the words "web services" means speed to the most part of companies. Therefore, all the efforts should be focused on obtaining this feature. This is not an easy task, if the roster company works with different companies and each company have its own server and technology, obtaining the best results for each of them requires a significant amount of effort and dedication.

Every client has to be studied with detail and the collaboration between both parts is essential. However, for the client, collaborating with the roster company cannot suppose an extra effort because in that case they will not want the system. It is important to remember that they just want to get services and pay for them. The roster company has to be careful and keep the appropriate distance with the clients.

The purpose of the system requires obtaining a new roster with every request, but several requests could ask for the same roster. Creating the same roster again is a waste of time and resources. However, the roster depends on external companies that can change their data instantaneously which forces the roster company to create a new roster with every new request.

For the reasons mentioned above, caching in the traditional way is not possible. Therefore new methods, techniques and strategies should be developed to reuse data and avoiding the waste of resources. In order to develop these techniques, two things are needed:

- Studios of the real usage of the system, which could lead to generate automatically the most asked rosters to avoid calculating them with every new request.

- Methods to discover if the client's data have been modified. This leads again to establish collaboration between the both parts.

This project and the developed system have shown how web services and automated scheduling are suitable to work together and at the same time they have shown what is the most important characteristic in this combination: customization. Normally, web services gives a global service to everybody, but for this topic that approach is not valid. Each web service has to be customized for the client. This characteristic should focus the orientation of the next steps in this area.

This project was also aimed to show SMEs how web services are a good and easy solution. The dissertation could be easily followed and understood. It remarks the simplicity of web services and it gives a good example of how useful they could be.

Personally, I am happy with the work done and I am also curious about how this solution will evolve and will be implemented in the industry. At this point I think that this is an unexplored area full of possibilities. However I think that the deployment of this idea could be complicated. In addition to the retrieving data problem the fact of transform the centralized approach to the distributed one changes the focus of the company. Up to now, one of these companies was mostly concerned about its software but

with the change this focus disappears. It is necessary to think in a 2.0 way with which it implies and it could lead to bad quality software if the company pays more attention to web issues than to the product itself.

The Internet is democratic; everybody and every enterprise have the same opportunities on it. It also promotes a high competitiveness where the best product not always triumphs. This makes the rostering company needs a new business strategy. The fact of paying to obtain a service instead paying to get a software solution could create conflicts between companies; maybe a company will not be willing to pay the same for the service that for the software when the cost of the service could be higher. This combination transforms the rostering company into a web company. Personally, I think that this change is complicated and dangerous and not all the companies are qualified to take it.

I think that this project is a good beginning to explore the combination of both technologies in future projects. This project has given me a global and complete understanding of both areas. Several improvements can be made. I think that the improvements related with the rostering theory and techniques are important, but the ones related with the web issues are more important. They are going to provide the company an extra value against its competitors. Some of the improvements could be:

- Development of an accessible and easy to use web interface (eg. Interactive calendar)
- Potential open source release for community development
- Enhanced availability specification and automated task allocation
- Adaptation into a plugin/component for content
- Management systems (eg. Joomla, Drupal)

This project is finished. Everything in the project can be improved, specially the retrieving data part where a connection pool or another technique for the same purpose could significantly change the results.I think that this project is a good beginning to explore the combination of both technologies in future projects. This project has given me a global and complete understanding of both areas.

This project is finished. It has been a challenging project, it is a first valid approach to the area and it opens the door and shows the path to follow to future projects on this topic.

ANEXO XII – KNOWLEDGE, SKILLS, RESOURCES AND SCHEDULING AT NOTTINGHAM UNIVERSITY

1. KNOWLEDGE, SKILLS AND RESOURCES

This project has been written in Java, using the 1.6 development kit version. All the code has been written using the NetBeans v6.9.1 IDE [67]. This version of the IDE has multiple plug-ins to help the development of web applications.

The whole project has been assisted by the JAVA EE [68] plug-in. This plug-in provides support for Java EE applications, SOAP web services, RESTful web services with the Jersey Library plug-in [69] and Java web applications.

The system uses a MySQL[70] database. The database was created and administrated using the MySQL packages v5.1 for Ubuntu System. In order to administrate the database, PhpMyAdmin [71] and the support for the NetBeans IDE have been used. NetBeans Database Explorer supports any relational database for which there is a JBDC driver.

The system uses a Glassfish Server v.3.0.1 [72]. It is an open-source application server developed by Sun Microsystems (Oracle). The server implements the technologies defined in the Java EE platform and it is able to run the applications under this specification. It is free and it is distributed under a CDDL and GNU GPL license.

In order to test the web services, soapUI Pro v3.6.1 [26] was used. This program has two versions: one open-source version and one professional and commercial version. The professional version was used because it allows a 15 days free trial. The functionality of this program mainly covers Web Service inspection, invoking, development, simulation, load testing and compliance testing,

The different versions of the project were stored and backed up using DropBox v.2.0 [74]. This software has a perfect integration with the O.S and allows an easy way to save the different parts of your work in the cloud

The dissertation has been written using OpenOffice v.3.2.1 [74].To draw the diagrams Dia[75] was used. The entire project has been developed under Ubuntu 10.10 Desktop version [76]. All the tools and resources used are open-source. This is has been done consciously following the guideline of the project. SMEs could see how the economic disbursement can be decreased with the usage of these tools.

The knowledge and skills in order to develop the project are a good knowledge in Java programming language and XML language. It is necessary a global knowledge about how the World Wide Web works and how the web services work. The understanding about the working platforms will be developed along the project. An important research in the web services and automated scheduling will be made in order to comprehend the system, the limitations of it, how it works and the aim of this program.

In order to achieve the Project it will be necessary to follow the established schedule. Good time-management and motivation will be required to complete the project to a high standard and within the targeted time frames

2. SCHEDULE

The next chart shows the approximate dates in the development of the project and its different parts. This chart is and will serve as a way to control the development of the project. On the same way, during the development of the project checking and control tasks will be made. They will be useful to evaluate the work done and to evaluate the chosen schedule and organization.

The project idea has suffered important changes and therefore the schedule has suffered a strong reorganization. The schedule has been maintained but some of the parts of the project have changed during this time, so new stages have been added and others have been eliminated.

During all this period, regular meetings have been arranged with my tutors. Despite the wide margins the work has been continuous and the project was finished three weeks before the deadline. This gives us three weeks to change small details and to improve the quality of the dissertation.

Task Description	Start Date	Completion date
Initial research: SOA, web services, SMEs and Automated scheduling.	04/10/10	17/10/10
Elaboration of the project proposal	18/10/10	24/10/10
Set up the work equipment: IDE, server, testing tools.	25/10/10	31/10/10
Design of the system and web services: Parts of the system and the interaction between them.	01/11/10	07/11/10
System Implementation (SOAP)	08/11/10	20/12/10
Preparation Demo	22/11/10	29/11/10
Addition of the security layer	24/12/10	29/12/10
Implementation's review and improvements	30/12/10	10/01/11
Break for exams and revision		
Testing RostApp	24/01/11	30/01/11
Implementation of new functionalities: Employee's roster.	01/01/11	06/01/11
System Implementation (REST)	07/02/11	13/02/11
Web service tests: Acceptance and correctness tests.	14/02/11	28/02/11
Final Dissertation Report	28/02/11	24/04/11
Web service tests II: Load tests	01/04/11	16/04/11
Demo preparation	01/05/11	08/05/11

The main change from the original schedule has been the delay in the dissertation. This has been done consciously. During the project it was seen that new features could be added.

The first version of the schedule was quite conservative. This could have been restrictive or have deteriorated the project's quality due to the limited dates but it created a good working pace.

ANEXO XIII – APPENDICES

Appendix I -- Installation

El código fuente del proyecto se encuentra en la carpeta "fuentes". Dentro de cada carpeta existe una carpeta para cada una de las aplicaciones del proyecto. Las carpetas de estas aplicaciones corresponden al proyecto del IDE NetBeans. De esta forma las aplicaciones pueden ser añadidas a NetBeans de forma fácil y cómoda.

Las librerías necesarias para el funcionamiento del proyecto se encuentran en la carpeta "rostering" dentro de la carpeta "fuentes". El usuario puede introducir sus propios datos o usar los datos que se han utilizado en los test. Los datos del test están localizados en el archivo "Roster.sql" dentro de la carpeta "BD". Los resultados de los test se encuentran en la carpeta "TEST"; dentro de esta carpeta

Para hacer funcionar las aplicaciones Web, una vez abierta la aplicación en NetBeans hacer *click* en la opción de "desplegar". De esta manera la aplicación comenzará a funcionar en el servidor. Para las aplicaciones de escritorio (RostApp), acceder a la opción "build" y tras ello seleccionar ls opción "run".

Para usar las aplicaciones el formato de contraseñas es:

Usuario : contraseña
manager:osuosu21

Cada empleado tiene un id y su correspondiente *password* es "passNumeroID"
Ejemplo
1:pass1
2:pass2
3:pass3
...

Finalmente para hacer funcionar las aplicaciones es necesario realizar cambios en las rutas de los ficheros de log y *password*. Para ello hay que modificar las siguientes líneas de forma que señalen la localización de los ficheros.

Fichero: HRRest/src/java/Solutions/log.java Línea 29
Fichero: MidlandHR/src/java/Solutions/log.java Línea 28
Fichero: HRRest/src/servicios/roster.java Línea 142
Fichero: MidlandHR/src/servicios/roster.java Línea 139

Appendix II

Data used to test the system:

//Duties

```
Aaa,Bbb,Ccc,Night
```

//Staff

```
Michelle,Vacancy,Samba,Mailyn,Nestor,Ana,Zaldy,Lourdes,Nellie,Diane,Sheku,D
eogratius,Vacancy2
```

//Details (contract , wage, ranking, capacity)

```
1,8.5,40,0.75
1,8.5,40,0.78
1,8.0,40,0.66
1,8.0,40,0.52
1,7.75,40,0.8
1,7.75,40,0.58
1,7.7,40,0.71
1,7.7,40,0.49
1,7.5,40,0.77
1,7.5,30,0.8
1,7.0,30,0.7
1,7.0,30,0.4
1,7.0,30,0.4
```

//Demand

Aaa,01/11/2010 07:00:00,01/11/2010 14:30:00
Aaa,01/11/2010 12:00:00,01/11/2010 20:00:00
Bbb,01/11/2010 08:00:00,01/11/2010 14:00:00
Bbb,01/11/2010 06:45:00,01/11/2010 20:45:00
Ccc,01/11/2010 07:15:00,01/11/2010 21:15:00
Ccc,01/11/2010 14:00:00,01/11/2010 21:00:00
Night,01/11/2010 21:00:00,02/11/2010 07:00:00

Aaa,02/11/2010 07:00:00,02/11/2010 12:30:00
Bbb,02/11/2010 07:15:00,02/11/2010 21:15:00
Ccc,02/11/2010 06:45:00,02/11/2010 20:45:00
Ccc,02/11/2010 08:00:00,02/11/2010 21:00:00
Aaa,02/11/2010 10:00:00,02/11/2010 18:00:00
Night,02/11/2010 21:00:00,03/11/2010 07:00:00

Aaa,03/11/2010 07:00:00,03/11/2010 16:00:00
Bbb,03/11/2010 06:45:00,03/11/2010 20:45:00
Ccc,03/11/2010 08:00:00,03/11/2010 14:00:00
Ccc,03/11/2010 07:15:00,03/11/2010 21:15:00
Ccc,03/11/2010 15:30:00,03/11/2010 21:00:00
Aaa,03/11/2010 09:00:00,03/11/2010 16:00:00
Night,03/11/2010 21:00:00,04/11/2010 07:00:00

Aaa,04/11/2010 10:00:00,04/11/2010 15:00:00
Bbb,04/11/2010 08:00:00,04/11/2010 14:00:00
Bbb,04/11/2010 06:45:00,04/11/2010 20:45:00
Ccc,04/11/2010 07:00:00,04/11/2010 13:00:00
Ccc,04/11/2010 07:15:00,04/11/2010 21:15:00
Ccc,04/11/2010 14:30:00,04/11/2010 21:00:00
Night,04/11/2010 21:00:00,05/11/2010 07:00:00

Aaa,05/11/2010 08:00:00,05/11/2010 17:00:00

Bbb,05/11/2010 06:45:00,05/11/2010 20:45:00
Ccc,05/11/2010 07:15:00,05/11/2010 21:15:00
Ccc,05/11/2010 07:00:00,05/11/2010 13:30:00
Ccc,05/11/2010 14:00:00,05/11/2010 21:00:00
Aaa,05/11/2010 10:00:00,05/11/2010 17:00:00
Night,05/11/2010 21:00:00,06/11/2010 07:00:00

Bbb,06/11/2010 06:45:00,06/11/2010 20:45:00
Ccc,06/11/2010 07:15:00,06/11/2010 21:15:00
Ccc,06/11/2010 12:30:00,06/11/2010 17:30:00
Ccc,06/11/2010 08:00:00,06/11/2010 21:00:00
Night,06/11/2010 21:00:00,07/11/2010 07:00:00

Bbb,07/11/2010 08:00:00,07/11/2010 18:00:00
Bbb,07/11/2010 06:45:00,07/11/2010 20:45:00
Ccc,07/11/2010 15:30:00,07/11/2010 21:00:00
Ccc,07/11/2010 07:15:00,07/11/2010 21:15:00
Aaa,07/11/2010 11:00:00,07/11/2010 17:00:00
Night,07/11/2010 21:00:00,08/11/2010 07:00:00

Aaa,08/11/2010 07:00:00,08/11/2010 14:30:00
Aaa,08/11/2010 12:00:00,08/11/2010 20:00:00
Bbb,08/11/2010 08:00:00,08/11/2010 14:00:00
Bbb,08/11/2010 06:45:00,08/11/2010 20:45:00
Ccc,08/11/2010 07:15:00,08/11/2010 21:15:00
Ccc,08/11/2010 14:00:00,08/11/2010 21:00:00
Night,08/11/2010 21:00:00,09/11/2010 07:00:00

Aaa,09/11/2010 07:00:00,09/11/2010 14:00:00
Bbb,09/11/2010 07:15:00,09/11/2010 21:15:00
Ccc,09/11/2010 06:45:00,09/11/2010 20:45:00
Ccc,09/11/2010 08:00:00,09/11/2010 21:00:00
Aaa,09/11/2010 12:00:00,09/11/2010 18:00:00
Night,09/11/2010 21:00:00,10/11/2010 07:00:00

Aaa,10/11/2010 07:00:00,10/11/2010 16:00:00
Bbb,10/11/2010 06:45:00,10/11/2010 20:45:00
Ccc,10/11/2010 08:00:00,10/11/2010 14:00:00
Ccc,10/11/2010 07:15:00,10/11/2010 21:15:00
Ccc,10/11/2010 15:30:00,10/11/2010 21:00:00
Aaa,10/11/2010 09:30:00,10/11/2010 16:00:00
Night,10/11/2010 21:00:00,11/11/2010 07:00:00

Aaa,11/11/2010 09:00:00,11/11/2010 15:30:00
Bbb,11/11/2010 08:00:00,11/11/2010 14:00:00
Bbb,11/11/2010 06:45:00,11/11/2010 20:45:00
Ccc,11/11/2010 07:00:00,11/11/2010 13:00:00
Ccc,11/11/2010 07:15:00,11/11/2010 21:15:00
Ccc,11/11/2010 15:30:00,11/11/2010 21:00:00
Aaa,11/11/2010 12:00:00,11/11/2010 18:30:00
Night,11/11/2010 21:00:00,12/11/2010 07:00:00

Aaa,12/11/2010 08:00:00,05/11/2010 17:00:00
Bbb,12/11/2010 06:45:00,12/11/2010 20:45:00
Ccc,12/11/2010 07:15:00,12/11/2010 21:15:00
Ccc,12/11/2010 07:00:00,12/11/2010 13:30:00
Ccc,12/11/2010 14:00:00,12/11/2010 21:00:00
Aaa,12/11/2010 08:00:00,12/11/2010 17:00:00

Night,12/11/2010 21:00:00,13/11/2010 07:00:00

Aaa,13/11/2010 11:00:00,13/11/2010 17:00:00
Bbb,13/11/2010 06:45:00,13/11/2010 20:45:00
Ccc,13/11/2010 07:15:00,13/11/2010 21:15:00
Ccc,13/11/2010 08:00:00,13/11/2010 21:00:00
Night,13/11/2010 21:00:00,14/11/2010 07:00:00

Bbb,14/11/2010 08:00:00,14/11/2010 18:00:00
Bbb,14/11/2010 06:45:00,14/11/2010 20:45:00
Ccc,14/11/2010 15:30:00,14/11/2010 21:00:00
Ccc,14/11/2010 07:15:00,14/11/2010 21:15:00
Ccc,14/11/2010 11:00:00,14/11/2010 17:00:00
Night,14/11/2010 21:00:00,15/11/2010 07:00:00

Aaa,15/11/2010 07:00:00,15/11/2010 14:30:00
Aaa,15/11/2010 12:00:00,15/11/2010 20:00:00
Bbb,15/11/2010 08:00:00,15/11/2010 14:00:00
Bbb,15/11/2010 06:45:00,15/11/2010 20:45:00
Ccc,15/11/2010 07:15:00,15/11/2010 21:15:00
Ccc,15/11/2010 14:00:00,15/11/2010 21:00:00
Night,15/11/2010 21:00:00,16/11/2010 07:00:00

Aaa,16/11/2010 07:00:00,16/11/2010 14:00:00
Bbb,16/11/2010 07:15:00,16/11/2010 21:15:00
Ccc,16/11/2010 06:45:00,16/11/2010 20:45:00
Ccc,16/11/2010 08:00:00,16/11/2010 21:00:00
Aaa,16/11/2010 12:00:00,16/11/2010 18:00:00
Night,16/11/2010 21:00:00,17/11/2010 07:00:00

Aaa,17/11/2010 07:00:00,17/11/2010 15:00:00
Bbb,17/11/2010 06:45:00,17/11/2010 20:45:00
Ccc,17/11/2010 08:00:00,17/11/2010 14:00:00
Ccc,17/11/2010 07:15:00,17/11/2010 21:15:00
Ccc,17/11/2010 15:30:00,17/11/2010 21:00:00
Aaa,17/11/2010 11:00:00,17/11/2010 18:00:00
Night,17/11/2010 21:00:00,18/11/2010 07:00:00

Aaa,18/11/2010 09:00:00,18/11/2010 15:30:00
Bbb,18/11/2010 08:00:00,18/11/2010 14:00:00
Bbb,18/11/2010 06:45:00,18/11/2010 20:45:00
Ccc,18/11/2010 07:00:00,18/11/2010 13:00:00
Ccc,18/11/2010 07:15:00,18/11/2010 21:15:00
Ccc,18/11/2010 15:00:00,18/11/2010 21:00:00
Night,18/11/2010 21:00:00,19/11/2010 07:00:00

Bbb,19/11/2010 06:45:00,19/11/2010 20:45:00
Ccc,19/11/2010 07:15:00,19/11/2010 21:15:00
Ccc,19/11/2010 07:00:00,19/11/2010 13:30:00
Ccc,19/11/2010 14:00:00,19/11/2010 21:00:00
Aaa,19/11/2010 08:00:00,19/11/2010 17:00:00
Night,19/11/2010 21:00:00,20/11/2010 07:00:00

Aaa,20/11/2010 11:00:00,20/11/2010 17:00:00
Bbb,20/11/2010 06:45:00,20/11/2010 20:45:00
Ccc,20/11/2010 07:15:00,20/11/2010 21:15:00
Ccc,20/11/2010 08:00:00,20/11/2010 21:00:00
Night,20/11/2010 21:00:00,21/11/2010 07:00:00

Bbb,21/11/2010 08:00:00,21/11/2010 18:00:00
Bbb,21/11/2010 06:45:00,21/11/2010 20:45:00
Ccc,21/11/2010 15:30:00,21/11/2010 21:00:00
Ccc,21/11/2010 07:15:00,21/11/2010 21:15:00
Aaa,21/11/2010 11:00:00,21/11/2010 17:00:00
Night,21/11/2010 21:00:00,22/11/2010 07:00:00

//Availability

```
1, 1, 2010-11-04 15:00:00.000, 2010-11-05 07:00:00.000
2, 1, 2010-11-06 15:00:00.000, 2010-11-07 07:00:00.000
3, 1, 2010-11-07 15:00:00.000, 2010-11-08 07:00:00.000
4, 1, 2010-11-11 15:00:00.000, 2010-11-12 07:00:00.000
5, 1, 2010-11-13 15:00:00.000, 2010-11-14 07:00:00.000
6, 1, 2010-11-14 15:00:00.000, 2010-11-15 07:00:00.000
7, 1, 2010-11-18 15:00:00.000, 2010-11-19 07:00:00.000
8, 1, 2010-11-20 15:00:00.000, 2010-11-21 07:00:00.000
9, 1, 2010-11-21 15:00:00.000, 2010-11-22 07:00:00.000
```

//Ability (Aaa, Bbb, Ccc, Night)

```
1.0,0.59,0.69,0.59
0.59,0.79,0.6,0.59
0.69,0.89,0.5,0.5
0.59,0.89,0.5,0.54
0.6,0.69,0.89,0.65
0.5,0.59,0.99,0.5
0.55,0.69,0.92,0.72
0.59,0.79,1.0,0.52
0.69,0.69,0.87,0.79
0.99,0.79,0.5,0.66
0.69,0.75,1.0,0.61
0.69,0.51,0.89,0.5
0.58,0.69,0.79,0.55
```

Appendix III – Security

Creating the CA

```
openssl req -x509 -newkey rsa:2048 -keyout cakey.pem -days 3650 -out
cacert.pem
```

Creating certificates:

Steps:

Generate the keys for the certificates:

```
keytool -genkey -alias certname -keypass changeit -validity 365 -keystore
keystore.jks -storepass changeit
```

Generating CSR petition

```
keytool -certreq -alias certname -file certname.csr -keystore keystore.jks
-keypass changeit -storepass changeit

mv certname.csr firmar.pem
```

Create config1.txt

```
-config1.txt-
basicConstraints = critical,CA:FALSE
extendedKeyUsage = serverAuth
subjectAltName   = IP:192.168.1.100 -This line is to generate certificates
for IP hosts.

openssl x509 -CA cacert.pem -CAkey cakey.pem -req -in firmar.pem -days 3650
-extfile config1.txt -sha1 -CAcreateserial -out firmado.pem
```

Import the CA into the system's keystores.

```
keytool -import -alias PFC -keypass changeit -file cacert.pem -storepass
changeit
```

Import the certificates into the system's keystore and truststore.

```
sudo keytool -import -alias PFC -keypass changeit -file cacert.pem -
keystore /usr/lib/jvm/java-6-openjdk/jre/lib/security/cacerts -storepass
changeit

keytool -import -alias certname -keypass changeit -keystore keystore.jks -
file /…/firmado.pem -storepass changeit -trustcacerts

keytool -import -alias certname -keypass changeit -keystore cacerts.jks -
file /…./firmado.pem -storepass changeit -trustcacert
```

APPENDIX IV – RostApp's Tests

1. BLACK BOX TESTING

a. General description

Black box testing will test that the different inputs in the system generate the expected outputs.

The tests will be sorted by functionality in several categories numbered by Roman numerals. Likewise, inside each category, every test will be numbered with a cardinal numeral. One test could need more that one sub tests and these will be numbered alphabetically.

To refer to a particular test, the following notation will be used:

Roman numeral – cardinal numeral – letter of the test (if it is necessary)

Example:

Test i-1.
Test i-2-a.

Every category has a description attached of the functionality of the system which is desired to test. Every test has a name and the expected result.

b. Implementation methodology

Every test in a category will be tested twice. The sequence of the test will be decided by the tester. The sequence will be written down with the results of the test (success, failure). The results will be written down in special sheets.

c. Detailed list of tests

i. Application's start tests

These tests will check if the application starts correctly.

1. Start the application without Internet connection.

The application shows a window with the error when it try to access to the database. If the error window is closed, the application is closed.

2. Execution of the application with internet connection:

a. **Try to access to the application without user name or password**: The application shows a message reporting some data is wrong.

b. **Type the password "euwht3qurh" and try to access:** The application shows a message reporting some data is wrong.

c. **Type the right password with the right user (Manager and employee)**: The application is accessed.

d. **Close the window of the application:** The application ends.

ii. Navigation tests:

Navigation test check that every menu option can be correctly accessed.

Manager option

1. File Menu:

a. **Click the exit option:** the application ends.

2. Employees Menu:

a. **Click "New employee" option:** The add employee window is shown.

b. **Click "Modify employee" option:** The modify employee window is shown.

c. **Click "Delete employee" button:** The delete employee window is shown.

3. Roster Menu:

a. **Click "Roster" option:** The roster window is shown**.**

4. Demand Menu:

a. **Click "New demand" option:** The new demand window is shown.

b. **Click "Delete demand" option:** The delete demand window is shown.

5. Help menu:

a. Click "Help" option: The help window is shown.

Employee option

6. File Menu :

a. **Click the exit option:** the application ends.

7. Employee Menu:

a. Click "Modify availability" option: The modify availability window is shown.

8. Roster Menu

a. Click "Roster" option: The roster window is shown.

9. Help menu:

a. **Click "Help" option:** The help window is shown**.**

iii. **Employees' insertion test:**

In every test, the not tested fields will have a right value. Then, combinations of the fields with wrong values will be made.

testX + … + testN

E.g test **iv-1 + iv-4** for one right addition with wrong values for the name and the capacity.

1. Addition of an employee without name: A window reporting "Some data is wrong" is shown.

2. Addition of an employee without contract: A window reporting "Some data is wrong" is shown.

3. Addition of an employee without wage: A window reporting "Some data is wrong" is shown.

4. Addition of an employee without ranking: A window reporting "Some data is wrong" is shown.

5. Addition of an employee without capacity: A window reporting "Some data is wrong" is shown.

6. Addition of an employee with a non numerical wage: A window reporting "Some data is wrong" is shown.

7. Addition of an employee with a non numerical capacity: A window reporting "Some data is wrong" is shown.

8. Addition of an employee with a non numerical ranking: A window reporting "Some data is wrong" is shown.

9. Addition of an employee with a non numerical contract: A window reporting "Some data is wrong" is shown.

10. Addition of an availability with wrong dates(start date > end date): A window reporting the wrong dates is shown.

11. Addition of an ability without value: A window reporting "Some data is wrong" is shown.

12. Addition of an ability with a non numerical value: A window reporting "Some data is wrong" is shown.

13. Addition of an employee with the right data: The employee is successfully added and the "add availability" and "add ability" buttons are unlocked.

14. Addition of a right availability: The availability is successfully added.

15. **Addition of a right ability**: The ability Is successfully added.

iv. Employees' modification tests:

In every test, the not tested fields will have a right value. Then, combinations of the fields with wrong values will be made.

testX + … + testN

E.g test **iv-1 + iv-4** for one right addition with wrong values for the name and the capacity.

1. **Modification of an employee removing his name:** A window reporting "Some data is wrong" is shown.

2. **Modification of an employee removing his contract:** A window reporting "Some data is wrong" is shown.

3. **Modification of an employee removing his wage:** A window reporting "Some data is wrong" is shown.

4. **Modification of an employee removing his ranking:** A window reporting "Some data is wrong" is shown.

5. **Modification of an employee removing his capacity:** A window reporting "Some data is wrong" is shown.

6. **Modification of an employee with a non numerical wage:** A window reporting "Some data is wrong" is shown.

7. **Modification of an employee with a non numerical capacity:** A window reporting "Some data is wrong" is shown.

8. **Modification of an employee with a non numerical ranking:** A window reporting "Some data is wrong" is shown.

9. **Modification of an employee with a non numerical contract:** A window reporting "Some data is wrong" is shown.

10. **Modification of an availability with wrong dates(start date > end date):** A window reporting the wrong dates is shown.

11. **Modification of an ability without value:** A window reporting "Some data is wrong" is shown.

12. **Modification of an ability with a non numerical value:** A window reporting "Some data is wrong" is shown.

13. **Addition of an availability with wrong dates(start date > end date):** A window reporting the wrong dates is shown.

14. **Addition of an ability without value:** A window reporting "Some data is wrong" is shown.

15. **Addition of an ability with a non numerical value:** A window reporting "Some data is wrong" is shown.

16. **Modification of a employee with the right data:** The employee is successfully added and the "add availability" and "add ability" buttons are unlocked.

17. **Modification of an availability with right dates**: The availability is successfully modified.

18. **Modification of an ability with right value**: The ability is successfully modified.

19. **Addition of a right availability**: The availability is successfully added.

20. **Addition of a right ability**: The ability Is successfully added.

21. **Modification of the presence/absence in an availability (employee menu):** The value is correctly changed.

v. **Employees' deletion tests:**

1. **All the existent employees are showed in a list in the window.**

2. **A employee is deleted**: All his availabilities and abilities are deleted as well.

vi. **Roster tests:**

1. **A roster is requested with wrong dates (start date > end date).:** A window reporting the wrong data is shown.

2. **A roster is requested with right dates:** A roster is obtained an shown in a window.

3. **Change between views of the roster:** The views change between them and they are shown in the same window.

vii. **Demands' insertion test:**

1. **Addition of a demand with the wrong dates (Start date > end date):** A window reporting the wrong dates is showed.

2. **Addition of a demand with the right dates :** The demand is successfully added.

viii. **Demands' deletion tests:**

1. **Demand selection:** All the existent demands are listed in a comboBox.

2. **Deletion of a demand:** The selected demand is deleted by clicking the delete button.

ix. **Availabilities' modification tests (employee):**

1. **Presence change:** If the attribute "presence" is changed and the availability is select again, the attribute is changed.

2. CONCURRENCY TESTS

a. Concurrency tests' motivation

The application is designed to be used in several machines. The focus of the concurrency test is the database (Web services concurrency is tested in other document). The database system (Mysql database with InnoDB engine) is designed to be concurrent. No problems should appear in this tests, however they have been made to check if any external problem or code problem appears with them.

b. Development of concurrency test

The tests in the A and B machines will be developed in a parallel way.

The concurrency test will consists of:

- **Start:**

  Start of the application in the A machine. (The machine must meet the specifications)

  Start of the application in the B machine. (The machine must meet the specifications)

  - **Employee addition tests:**

  Some employees will be added in the A machine.

  Some employees will be added in the B machine..

  Expected result: The new employees will appear in both machines without any difference.

  - **Employee deletion tests:**

  Some employees will be deleted in the A machine.

  The user in the B machine will try to delete the same employees at the same time.

  Expected result: The application will inform that there is some problem with the operation in the B machine. Once the employee in the machine A has been deleted the user on the B machine won't see the employee in his application (it won't appear in the list).

These tests will be performed with: the availabilities of the employees, the abilities of the employees and the demands. The objective is to see that once that if two values are entered at the same time only one of them remains.

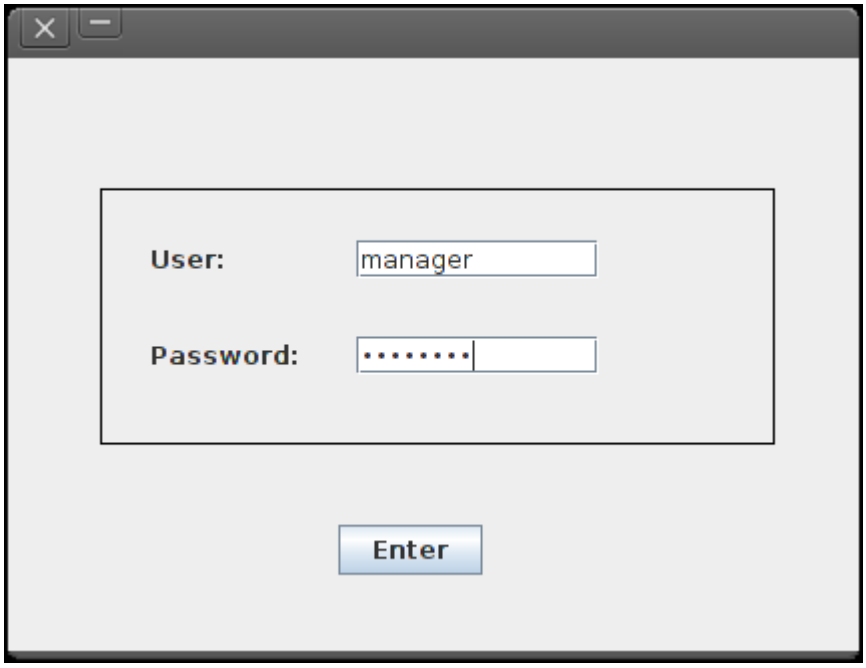
3. PERSISTENCY TESTS

The application will be running for 5 hours and then it will be check that all the options and the database works correctly.

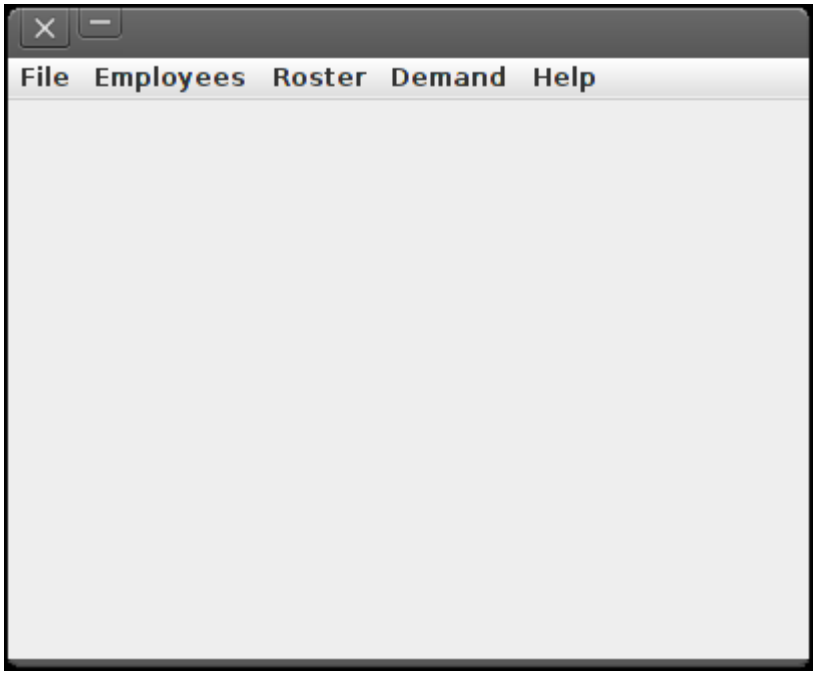
Appendix V – RostApp's GUI

In the section, all the GUI's windows will be show along the navigation between them.

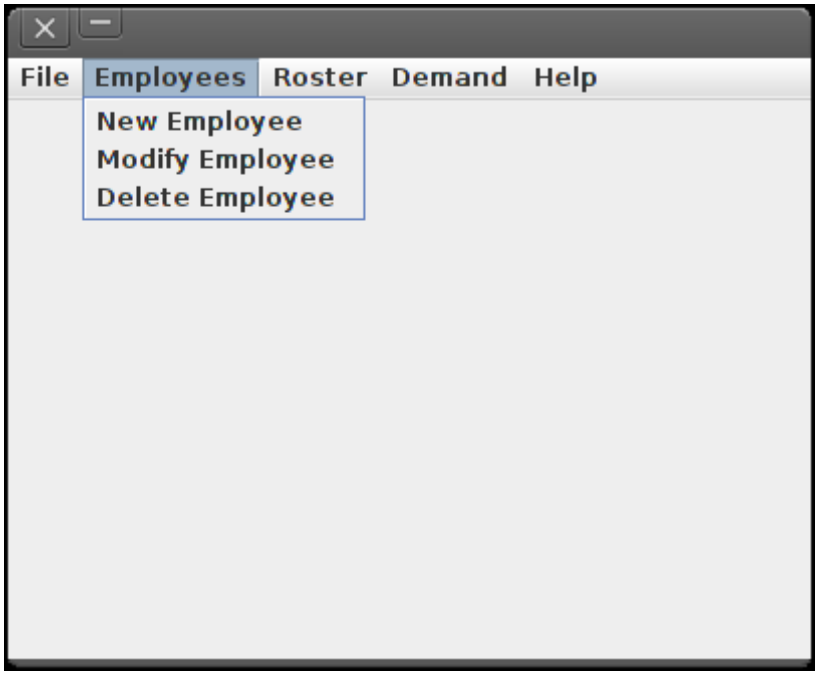
- Main



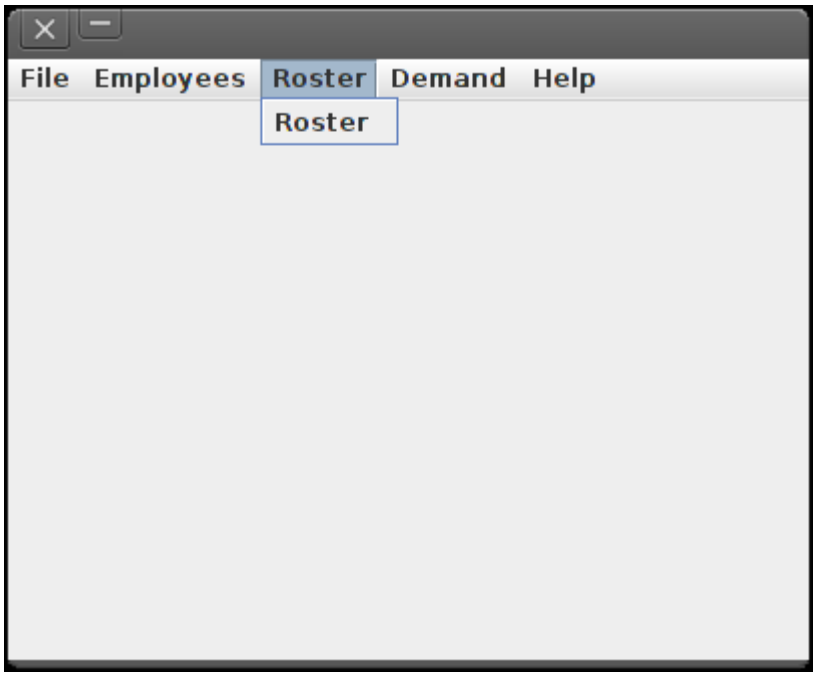
- Main Manager



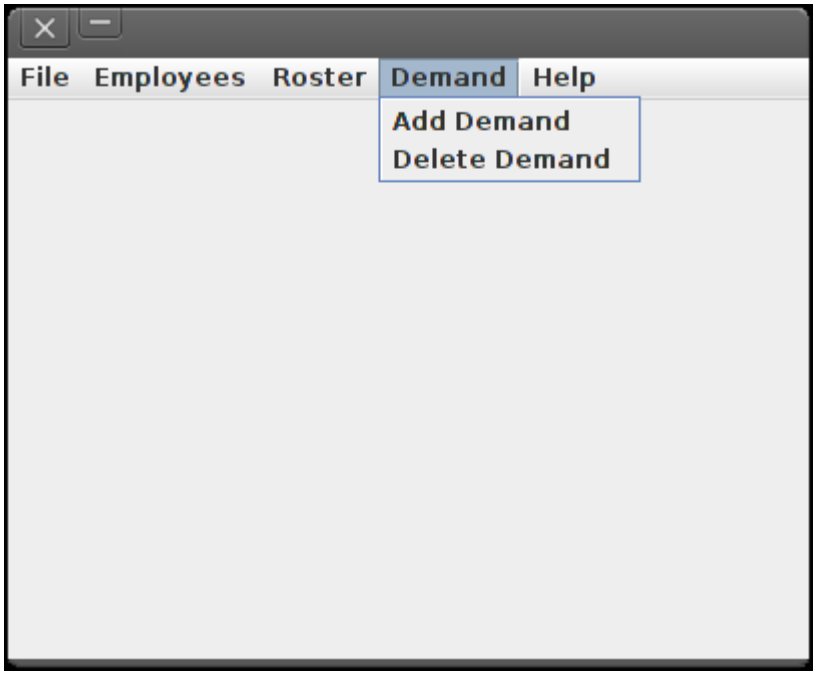
- Employee Menu



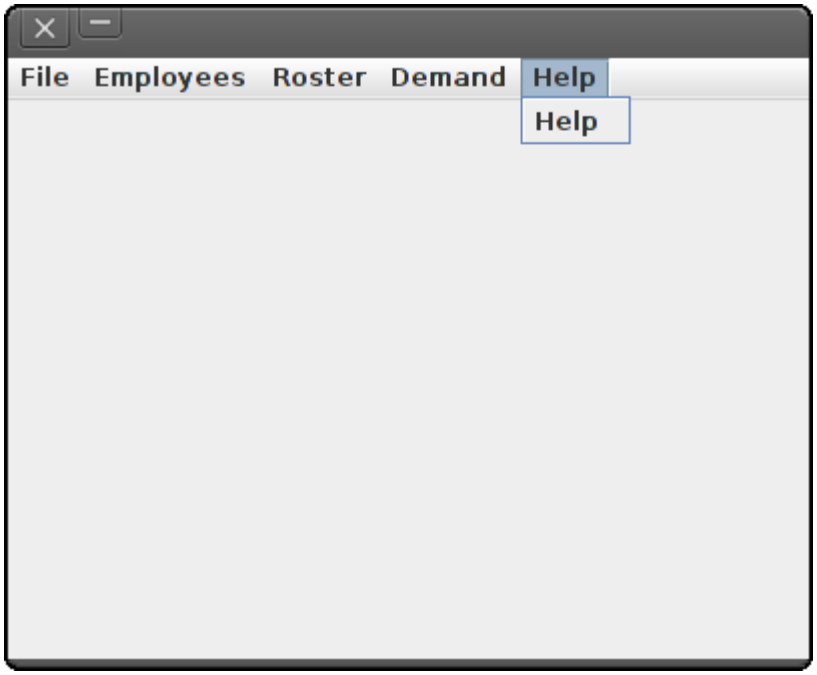
- Roster Menu

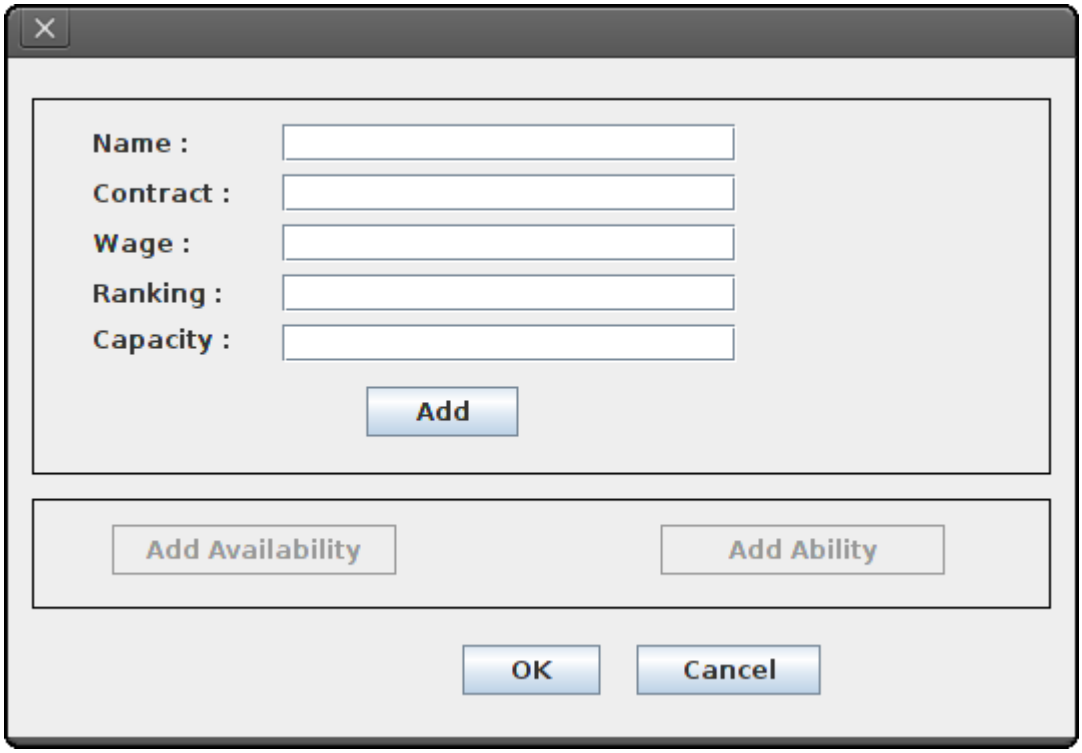


- Demand menu



- Help Menu

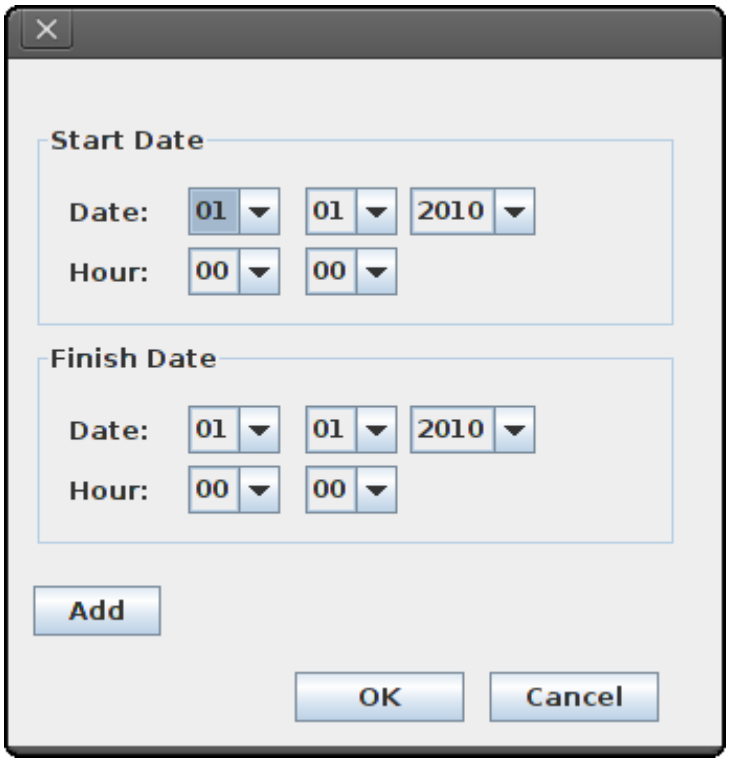




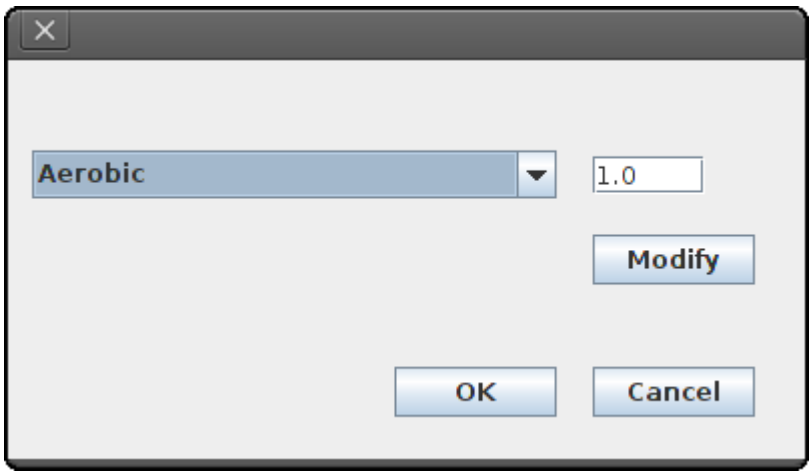
- Employee menu → Add Employee

This menu allows the manager to add a new employee. Once the employee had been added the manager will be able to add availabilities and abilities to the employee.

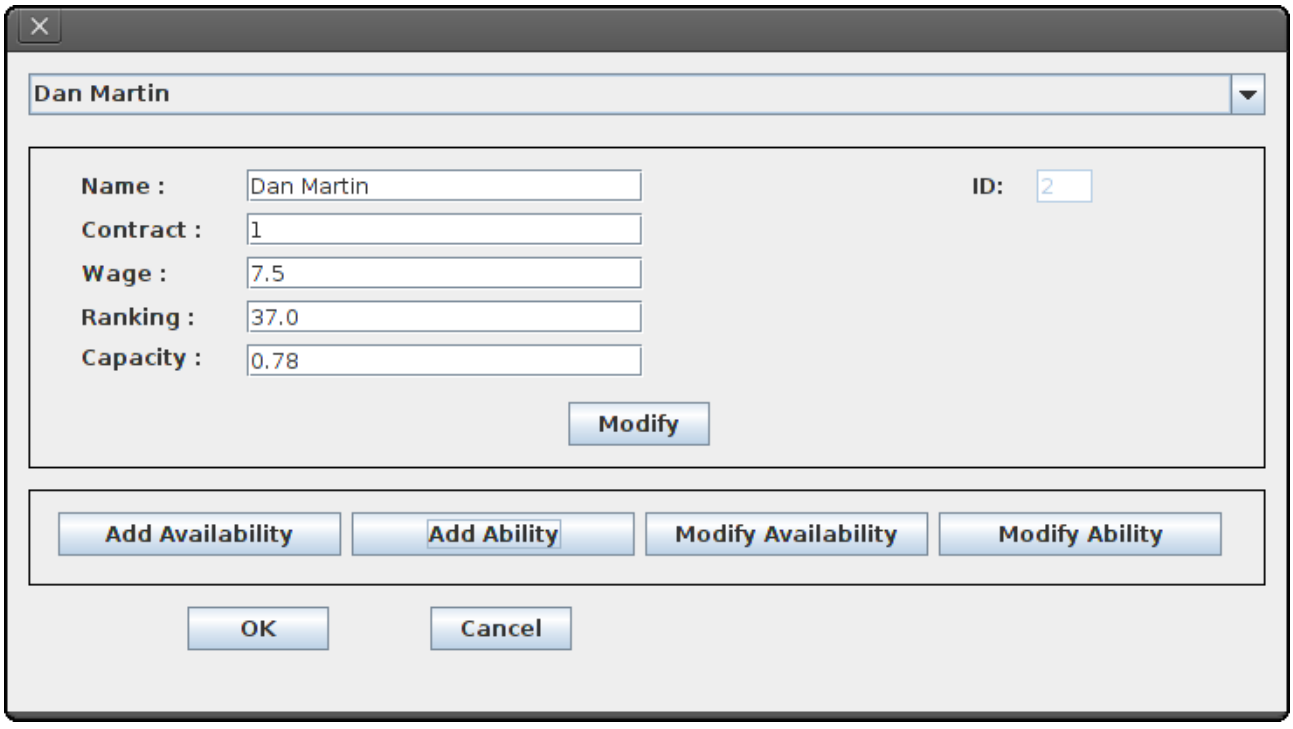
- Add Employee → Add Availability



- Add Employee → Add Ability

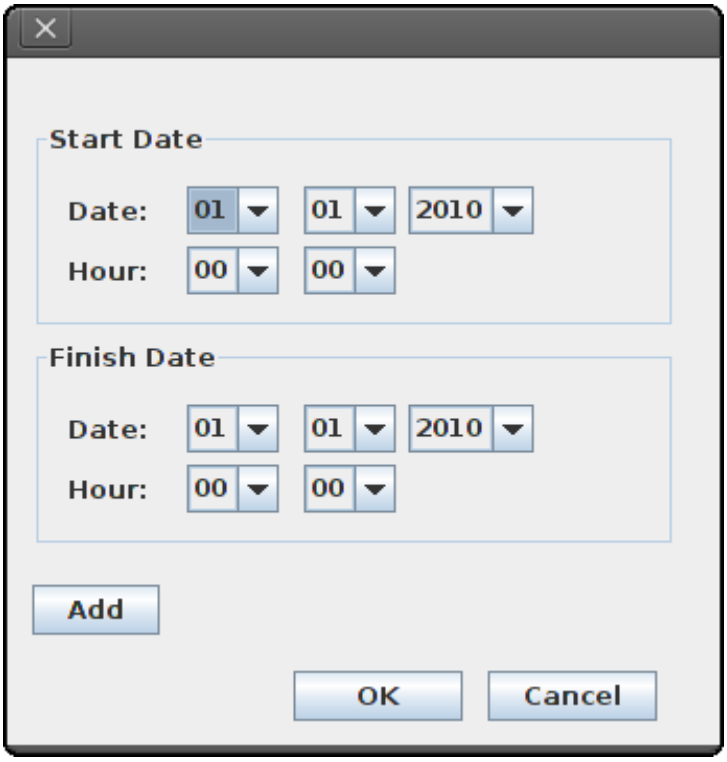


- Employee menu → Modify Employee

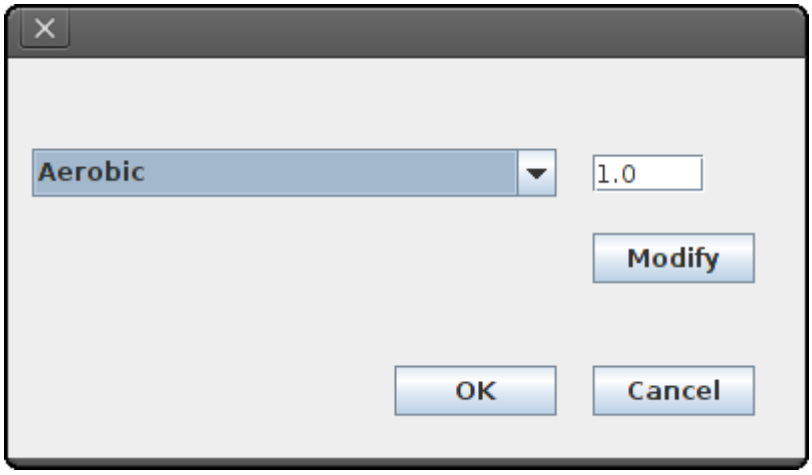


In this menu the user can modify the data of the employee. New availabilities and abilities could be added. The existent availabilities and abilities can be modified.

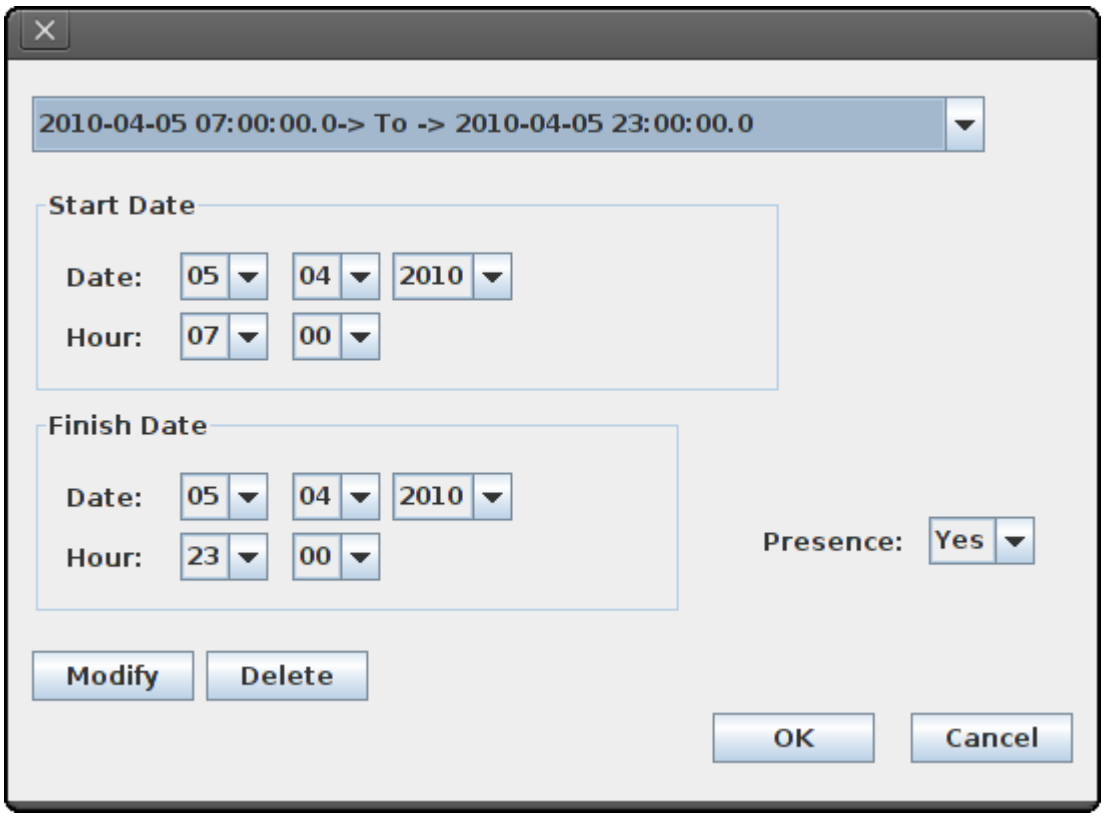
- Modify Employee→ Add Availability



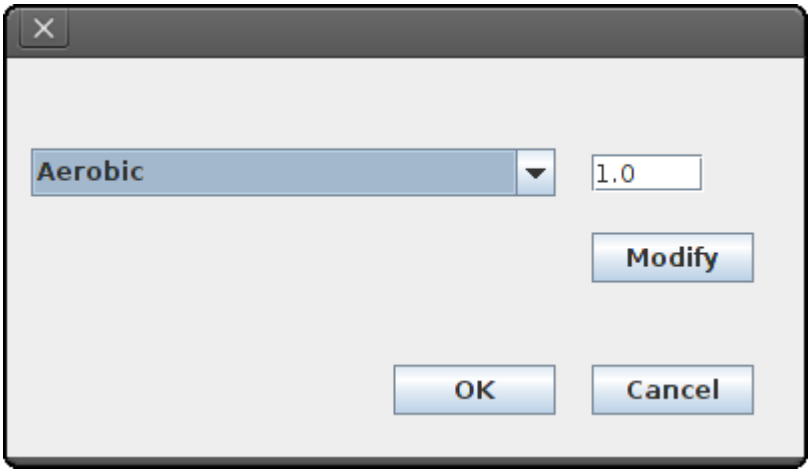
- Modify Employee → Add Ability



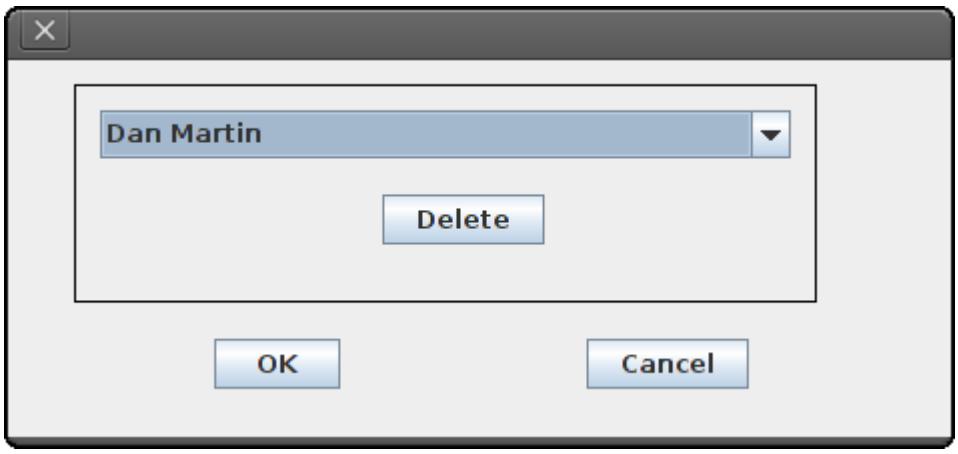
- Modify Employee→ Modify Availability



- Modify Employee → Modify Ability

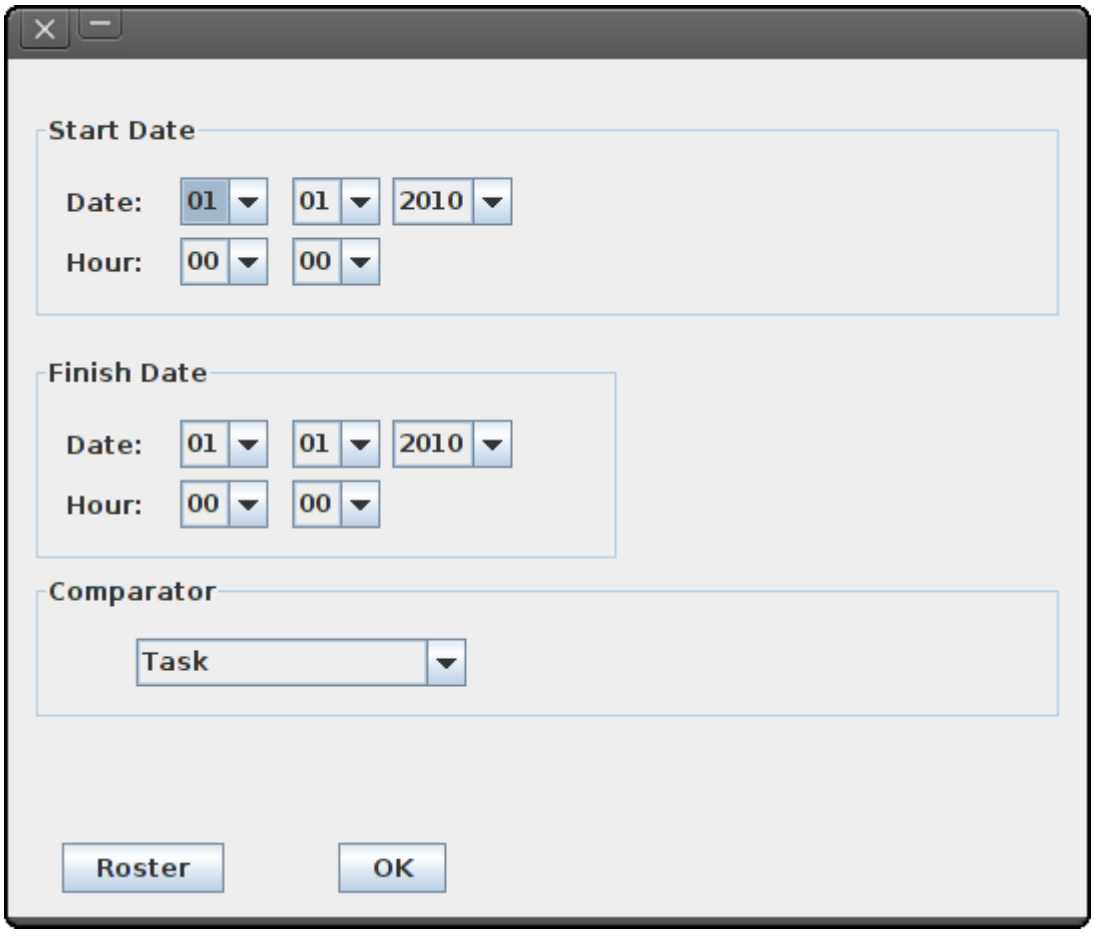


- Employee Menu → Delete Employee



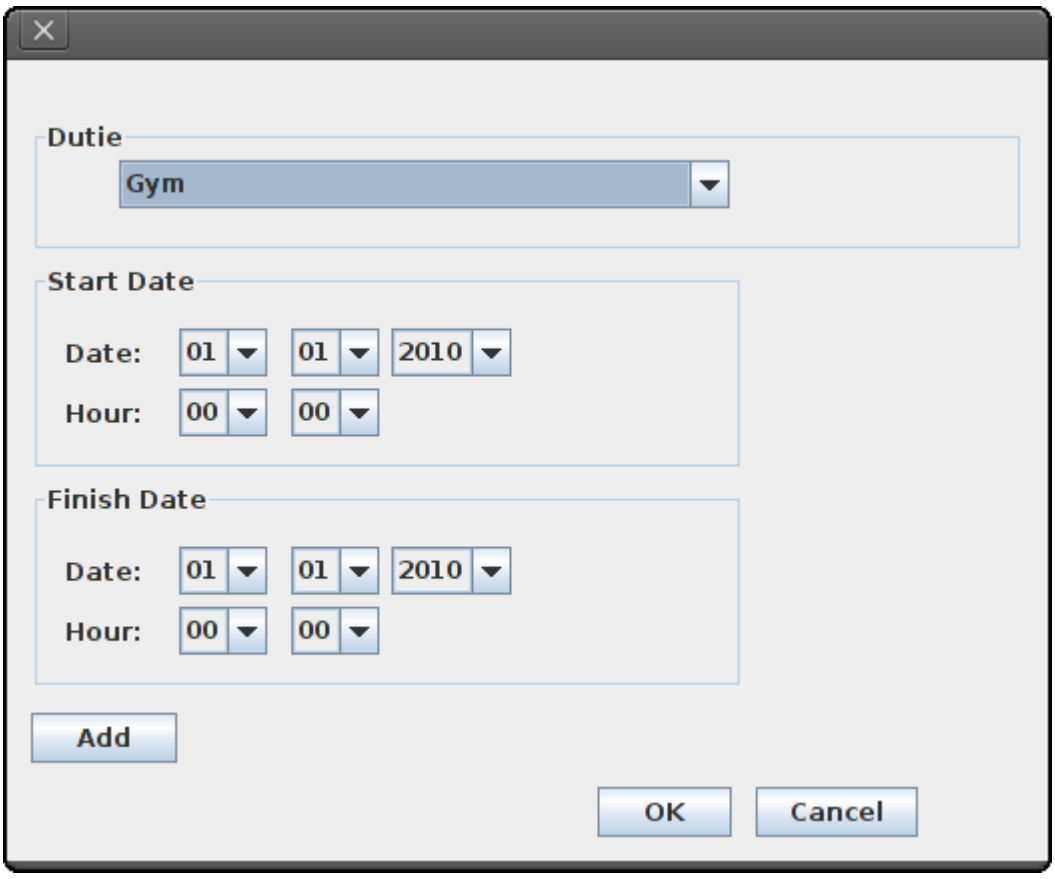
In this menu the user can select the employee from the list in order to delete it.

- Roster Menu → Roster



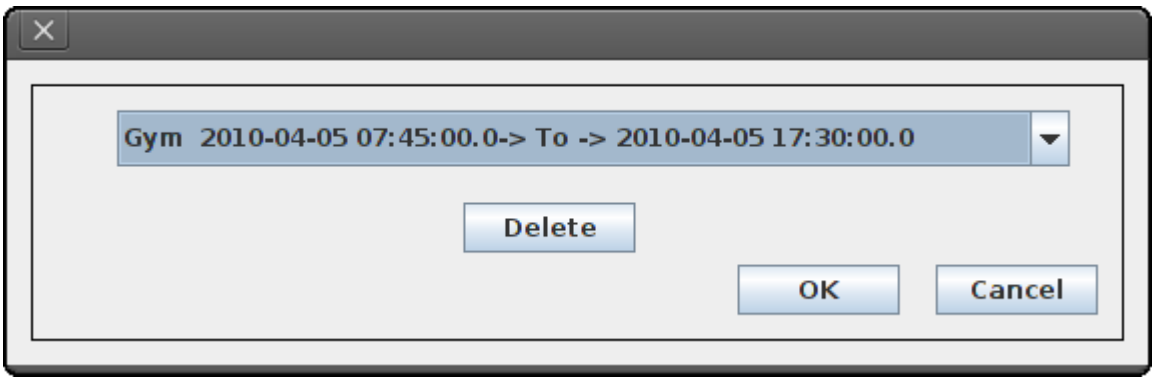
In this menu the user can ask for a roster solution. He has to chose the limit dates for the roster and the comparator.

- Demand Menu → Add demand



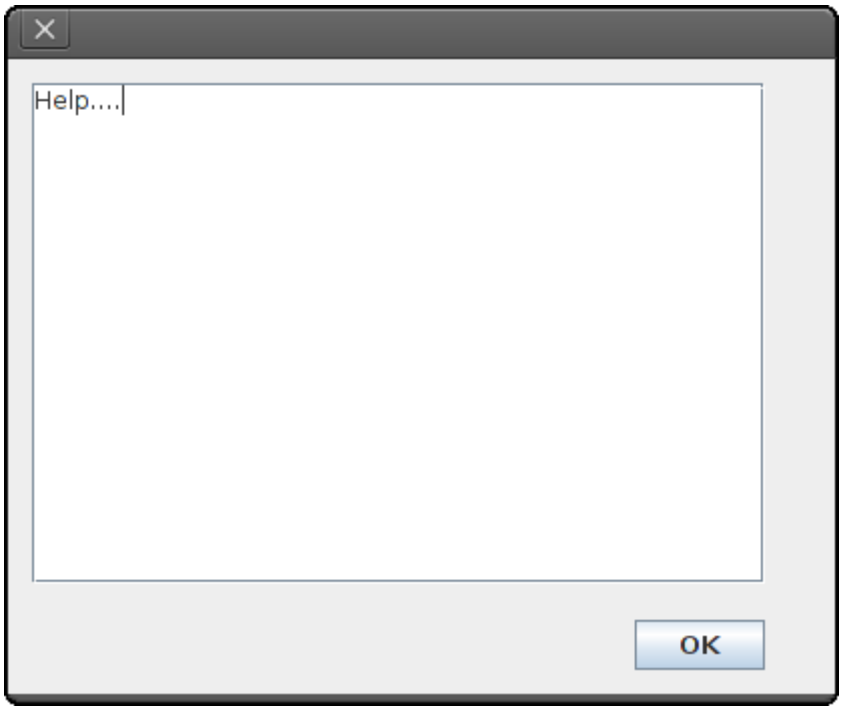
Menu to add a new demand

- Demand Menu → Delete Demand

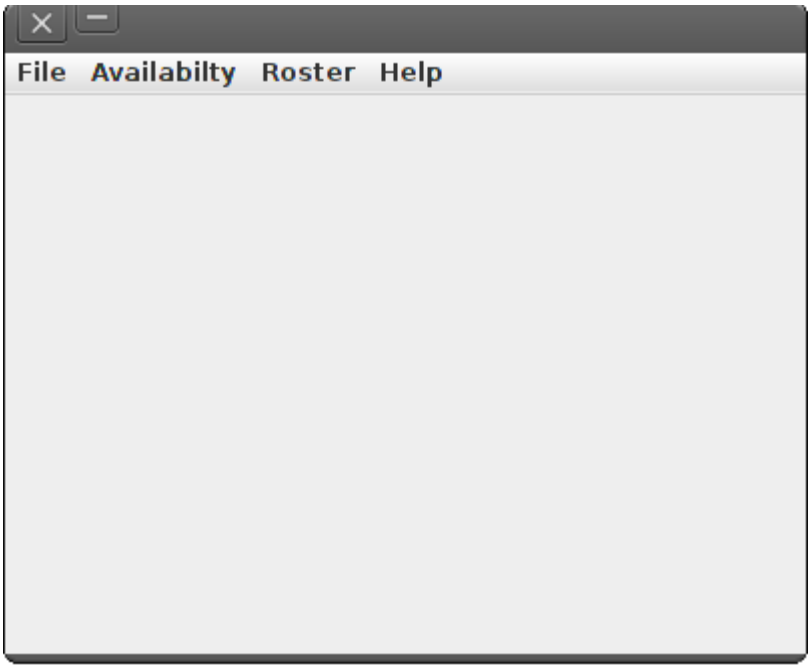


The user can delete a demand by selecting from the list.

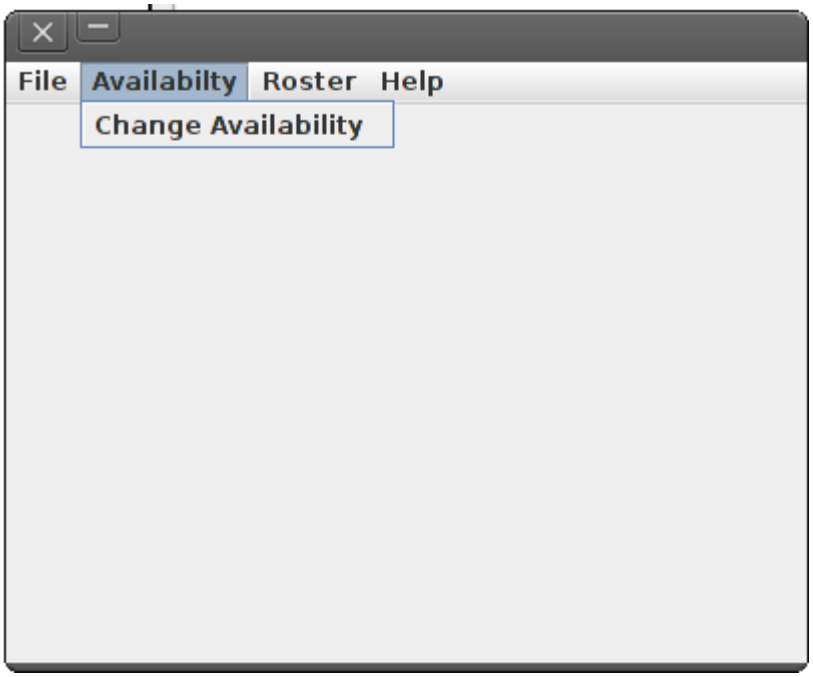
- Help Menu → Help



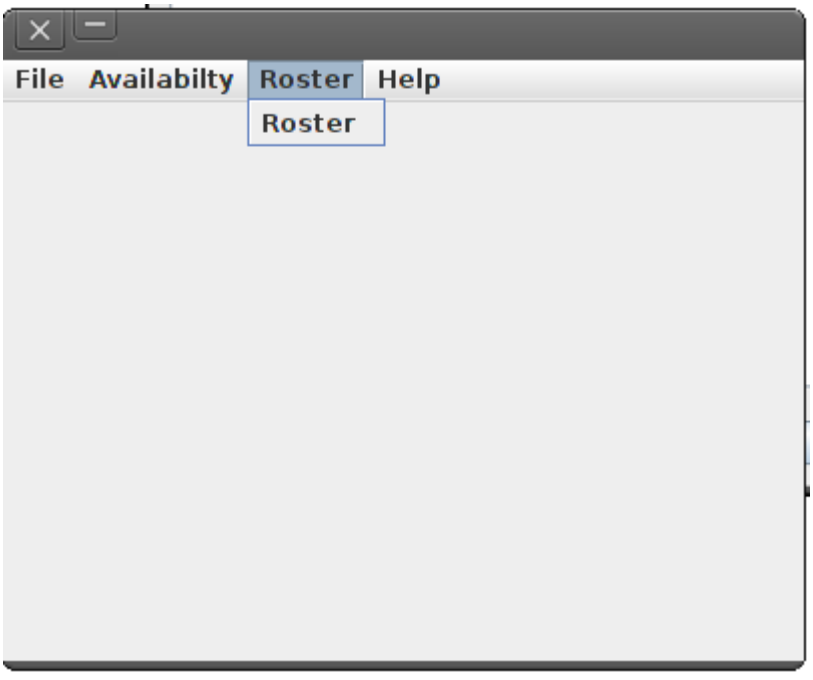
- Main Employee



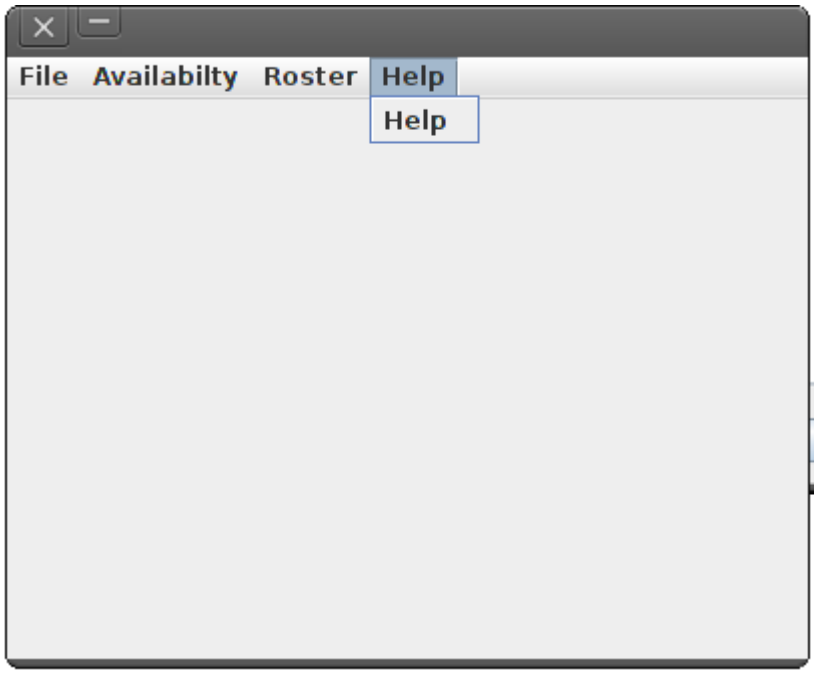
- Availability Menu



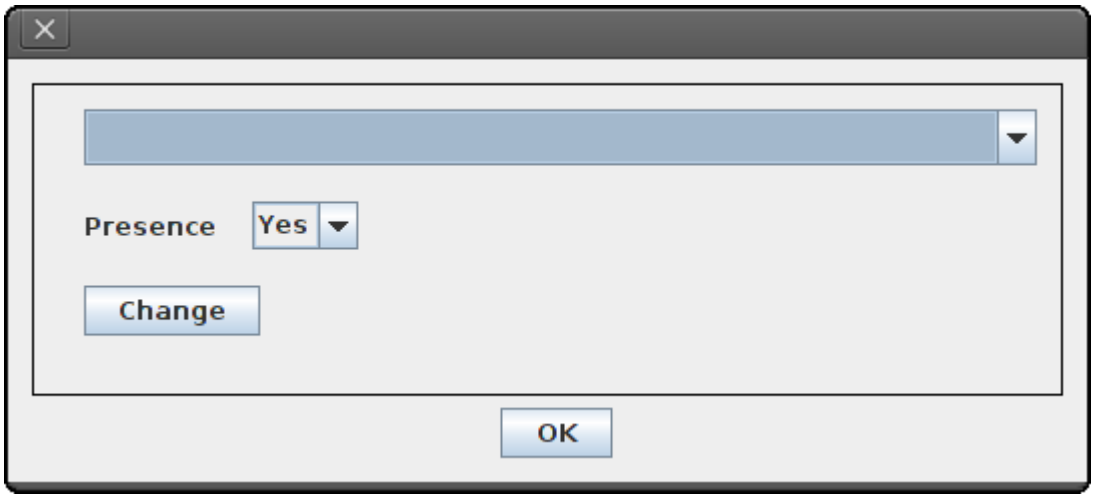
- Roster Menu



- Help Menu

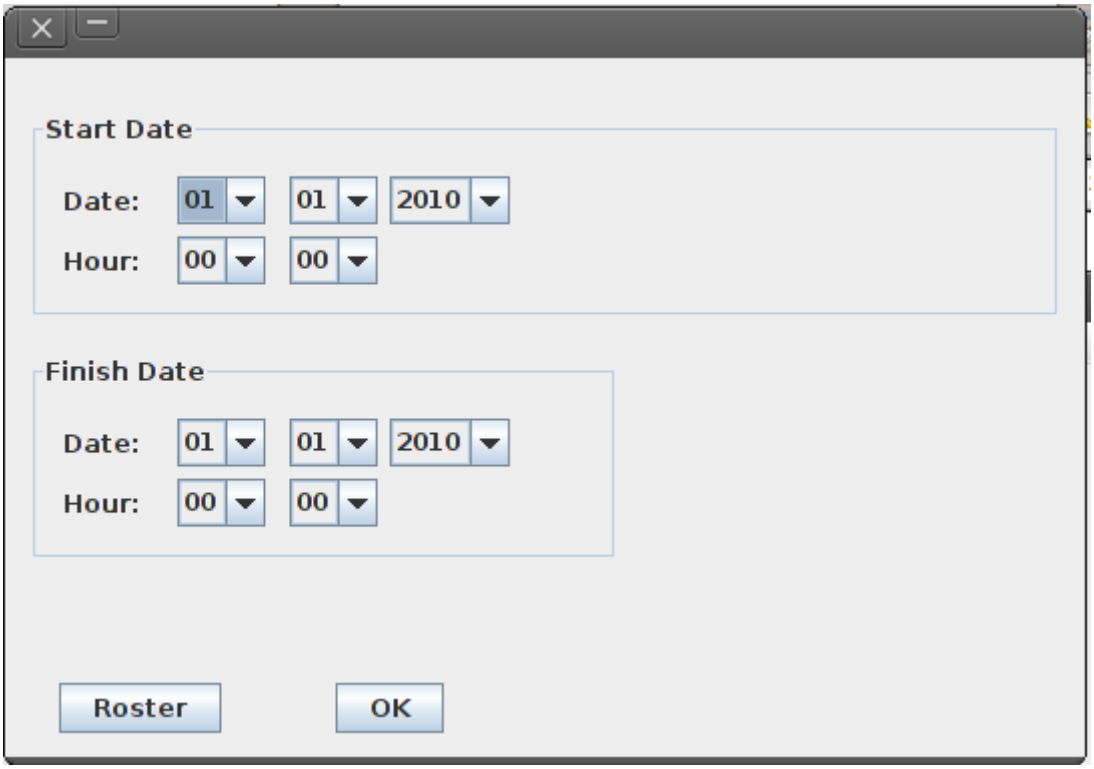


- Availability Menu → Change Availability.
-



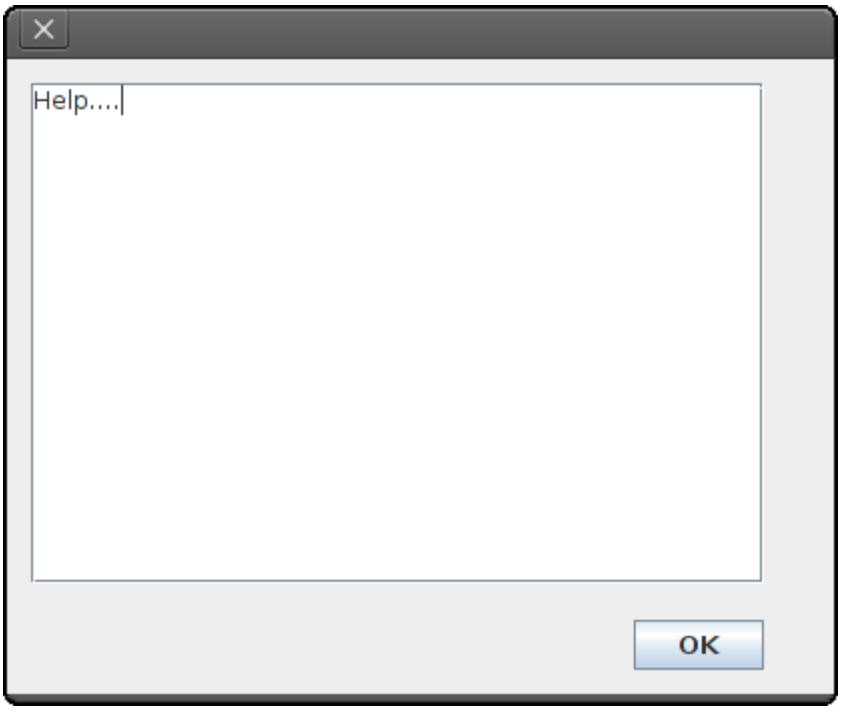
In this menu the user can change his presence/absence in the selected availability.

- Roster Menu → Roster



In this menu, the employee obtains his/her personal roster.

- Help Menu → Help.



BIBLIOGRAFÍA

[1] Software & Information Industry Association. ***Software As A Service: Strategic Backgrounder***. Washington, D.C.: 31 February 2001.

[2] OASIS - **SOA Reference Model**. 2011
http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=soa-rm.

[3] Michael Pinedo. **Scheduling, Theory, Algorithms, and Systems.** Prentice Hall, 1995, or new: Second Addition, 2002

[4] Michael Pinedo**. Scheduling, Theory, Algorithms, and Systems**, Prentice Hall, 1995, or new: Second Addition, 2002

[5] Timothy Curtois **Staff Rostering Benchmark data Sets**. 2011
http://www.cs.nott.ac.uk/~tec/NRP/

[6] Edmund Burke and Peter Ross - **Practice and Theory of Automated Timetabling: First International Conference, Edinburgh**, UK, August 29 - September 1, 1995. Selected Papers (Lecture Notes in Computer Science) (Paperback - 2 Oct 1996)

[7] Jason Levitt . **From EDI To XML And UDDI: A Brief History Of Web Services.** 2001
http://www.informationweek.com/news/development/tools/showArticle.jhtml?articleID=6506480

[8] W3C. Web **Services Description Language (WSDL) Version 2.0 Part 1: Core Language** 2003 http://www.w3.org/TR/2003/WD-wsdl20-20031110/#intro

[9] **UDDI XML.org**. 2011
http://uddi.xml.org

[10] Employee/Personal Scheduling Software. 2011
http://www.hr-software.net/pages/217.htm

[11] **HydraSCA**. 2011
http://www.roguewave.com/products/hydraexpress/index.php

[12] **IBM pack for SOA**. 2011
http://www-01.ibm.com/common/ssi/cgi-bin/ssialias?subtype=ca&infotype=an&appname=iSource&supplier=897&letternum=ENUS208-151

[13] **Oracle SOA Suite**. 2011
http://www.oracle.com/us/technologies/soa/soa-suite/index.html

[14] **Apache Tuscany**. 2011
http://tuscany.apache.org/home.html

[15] **Fabric3**. 2011
http://www.fabric3.org/overview.html

[16] **SCOrWare**. 2011
http://www.scorware.org/projects/en

[17] OASIS **Open CSA**. 2011
http://www.oasis-opencsa.org/sca

[18] Open SOA collaboration ,**Service Data Objects Home.** 2011
http://www.osoa.org/display/Main/Service+Data+Objects+Home

[19] OASIS, **Web Services Security (WSS)** . 2011
http://www.oasis-open.org/committees/wss

[20] W3C, **Web Services Policy 1.2** . 2011
http://www.w3.org/Submission/WS-Policy

[21] OASIS, **Web Services Compossite Application Framework**. 2011
http://www.oasis-open.org/committees/ws-caf/

[22] OASIS, **WS-Context especification**. 2011
http://www.docs.oasis-open.org/ws-caf/ws-context/v1.0/wsctx.html

[23] Mark Little -**The OASIS WS-CAF Approach to Web Services Business Transactions**, 25 January 2005
http://www.webservices.org/weblog/mark_little/the_oasis_ws_caf_approach_to_web_services_bussines_transactions

[24] **WSIT project**. 2011
http://wsit.java.net/

[26] **WSIT tutorial**. 2011
http://download.oracle.com/docs/cd/E17802_01/webservices/webservices/reference/tutorials/wsit/doc/index.html

[26] **soapUI**. 2011
http://www.soapui.org/

[27] **VisualVm**. 2011
http://visualvm.java.net/

[28] J. Fabra, P. Álvarez, J.A. Bañares, and J. Ezpeleta -**RLinda: A Petri Net Based Implementation of the Linda Coordination Paradigm for Web Services Interactions –** In: Bauknecht, K., Pr¨oll, B., Werthner, H. (eds.) EC-Web 2006. LNCS, vol. 4082, pp. 184–193. Springer, Heidelberg (2006)

[29] J. Fabra J, Álvarez P, Ezpeleta J. **DRLinda: a distributed message broker for collaborative interactions among business processes.** In *8th International Conference on Electronic Commerce and Web Technologies*, Vol. 4655, Lecture Notes in Computer Science. Springer Verlag: Berlin, Germany, 2007; 212–221.

[30] **SOA Manifesto**. 2011
http://www.soa-manifesto.org/

[31] **SOA principles web**. 2011
http://www.soaprinciples.com/

[32] No author - **Lean Development Applied To SOA** (22-09-2010)
http://www.manageability.org/blog/stuff/lean-soa

[33] Julio Alba. **Qué es...SOA, Arquitectura orientada al servicio.** Bit Magazine 167. 2008

[34] Wikipedia **. SOA, Web service approach**. 2011.
http://en.wikipedia.org/wiki/Service-oriented_architecture#Web_services_approach

[35] W3C **Web services Architecture.** 2004
http://www.w3.org/TR/ws-arch/

[36] W3C. **Extensible Markup Languange**. 2008
http://www.w3.org/TR/xml/#sec-origin-goals

[37] W3C. **SOAP Version 1.2 Part 1: Messaging Framework (Second Edition).** 2008
http://www.w3.org/TR/soap12-part1/#intro

[38] Michael Przybilsk. REST - **REpresentational State Transfer**. 2011
http://www.cs.helsinki.fi/u/chande/courses/cs/MWS/reports/MichaelPrzybilski_REST.pdf

[39] **Tema 3.2: Servicios Web REST: Conceptos Básicos. Facultade da Informatica de A Coruna**. 2011
http://www.fic.udc.es/files/asignaturas/39ADOO/Tema3Apartado3.2.pdf

[40] **Advantages & Disadvantages of Webservices**. 2011
http://social.msdn.microsoft.com/forums/en-US/asmxandxml/thread/435f43a9-ee17-4700-8c9d-d9c3ba57b5ef/

[41] Carlos Arturo Vega Lebrún **Beneficios de los servicios web**. 2008
http://www.eumed.net/tesis/2007/cavl/Beneficios%20de%20los%20servicios%20Web.htm

[42] IBM. **Good timing: Realizing value from investments in labor-scheduling**. 2008

[43] University of Strathcydle Glasgow. **Small and Medium Sized Enterprises Definitions.** 2011
http://www.lib.strath.ac.uk/busweb/guides/smedefine.htm

[44] Department for Business Innovation and Skills. **Small and Medium-sized Enterprise (SME) Statistics for the UK and Regions 2009',**. 2009
http://stats.bis.gov.uk/ed/sme/Stats_Press_release_2009.pdf

[45] Gary Howes. **UK SMEs need to play catch up on IT knowledge**. 2010.
http://www.smeweb.com/content/view/2345/117/

[46] Brian Eagles .**Data protection change for SMEs explained**. 2010.
http://www.smeweb.com/content/view/2053/106/

[47] Brian Eagles .**Data protection change for SMEs explained**. 2010.
http://www.smeweb.com/content/view/2053/106/

[48] Rajeev Trikha. **SOA recommendations for SMEs**. 2010
http://itis-consultants.blogspot.com/2010/01/soa-recommendations-for-sme.html

[49] Dan Rosanova. **The SOA knowledge gap**. 2009
http://www.cio.com.au/article/271922/soa_knowledge_gap/

[50] Francisco Carrillo. **SOA para empresas grandes o dinámicas.** 2010
http://franciscocarrillo.wordpress.com/2010/04/20/soa-para-pymes/

[51] **Verisign – SSL Certificates**. 2011
http://www.verisign.co.uk

[52] Wikipedia .**Web services specifications** . 2011.
http://en.wikipedia.org/wiki/List_of_Web_service_specifications

[53] Pete Freitag. **SOAP vs REST.** 2011
http://www.petefreitag.com/item/431.cfm

[54] **REST vs SOAP. The right web service.** 2009
http://www.taranfx.com/rest-vs-soap-using-http-choosing-the-right-webservice-protocol

[55] **JAX-WS**. 2011
http://en.wikipedia.org/wiki/Java_API_for_XML_Web_Services

[56] **Metro Stack**. 2011
https://metro.dev.java.net/

[57] **Apache Axis2**. 2011
http://axis.apache.org/axis2/java/core/docs/userguide.html#intro

[58] Kohsuke Kawaguchi**. JAX-WS RI 2.1 benchmark details.** 2007
http://weblogs.java.net/blog/kohsuke/archive/2007/02/jaxws_ri_21_ben.html

[59] **Tutorial web services Que elegir, Axis2 o JAX-WS?.** 2008
http://tundidor.com/blog/?p=22

[60] **JBOSS**. 2011
http://www.jboss.org/

[61] D. Gelernter: **Generative communication in Linda**. ACM Transactions on Programming Languages and Systems **7** (1985) 80–121

[62] Sun Microsystems, Inc.: **JavaSpaces Service Specification. Technical report,** 2000

[63] Sun Microsystems - **GigaSpaces Technologies: GigaSpaces. Technical report**, 2000

[64] R. Tolksdorf and D. Glaubitz: **Coordinating Web-Based Systems with Documents in XMLSpaces.** In: 9th International Conference on Cooperative Information Systems. Trento, Italy, September 5-7, 2001. Volume 2172 of Lecture Notes in Computer Science. Springer Verlag (2001) 356–370

[65] **Comodo**. 2011
http://www.comodo.com/

[66] Anbazhagan Mani and Arun Nagarajan . **Understanding quality of service for Web services** (1-06-2002)
http://www.ibm.com/developerworks/webservices/library/ws-quality.html

[67] **NetBeans IDE**. 2011
http://netbeans.org/

[68] **Java EE application**. 2011
http://netbeans.org/features/web/web-app.html

[69] **Jersey**. 2011
http://jersey.java.net/use/getting-started.html

[70] **MySQL**. 2011
http://www.mysql.com/

[71] **PhpMyAdmin**. 2011
http://www.phpmyadmin.net

[72] **Glassfish**. 2011
http://glassfish.java.net/

[73] **Dropbox**. 2011
http://www.dropbox.com/

[74] **OpenOffice**. 2011
http://www.openoffice.org/

[75] **Dia Diagram Editor**. 2011
https://help.ubuntu.com/community/Dia

[76] **Ubuntu**. 2011
http://www.ubuntu.com/