UNIVERSIDAD DE EL SALVADOR
FACULTAD MULTIDISCIPLINARIA DE OCCIDENTE
DEPARTAMENTO DE INGENIERÌA Y ARQUITECTURA

TRABAJO DE GRADUACIÓN

TEMA:

“PROPUESTA DE DESARROLLO DE SOFTWARE PARA MANEJO DE LA PLANILLA DE LA FACULTAD MULTIDISCIPLINARIA DE OCCIDENTE”

PARA OPTAR AL GRADO DE:
INGENIERO EN SISTEMAS INFORMÀTICOS

PRESENTADO POR:

VILLA CONTRERAS, SARA MARGARITA
CERNA ESCOBAR, DOUGLAS ALJADYZ

DOCENTE DIRECTOR:
ING CARLOS STANLEY LINARES PAULA

SEPTIEMBRE, 2008

SANTA ANA EL SALVADOR CENTROAMÉRICA

FACULTAD MULTIDISCIPLINARIA DE OCCIDENTE

DECANO
LICDO. JORGE MAURICIO RIVERA

VICE-DECANO
LICDO. Y MASTER ELADIO EFRAIN ZACARÌAS ORTEZ

SECRETARIO DE FACULTAD
LICDO. VICTOR HUGO MERINO QUEZADA

JEFES DE DEPARTAMENTOS DOCENTES

DEPTO. DE CIENCIAS ECONÓMICAS	: LICDO. EDUARDO ZEPEDA GUEVARA
DEPTO. DE CIENCIAS JURÍDICAS	: LICDO. JOSÉ ROBERTO REYES GUADRÓN
DEPTO. DE CIENCIAS SOCIALES	: LICDO. Y MASTER FRANCISCO JAVIER ESPAÑA
DEPTO. DE BIOLOGÍA	: LICDO. Y MASTER RICARDO FIGUEROA CERNA
DEPTO. DE QUÍMICA	: LICDO. MARIO ANTONIO SANTAMARÍA CHILÍN
DEPTO. DE FÍSICA	: LICDO. CARLOS JOAQUÍN AGUILAR
DEPTO. DE MEDICINA	: DRA. SANDRA PATRICIA GÓMEZ DE SANDOVAL
DEPTO. DE INGENIERÍA Y ARQUITECTURA	: ING. RAÚL ERNESTO MARTÍNEZ BERMUDEZ
DEPTO. DE IDIOMAS	: LICDA. SONIA ELIZABETH DÍAZ DE MARROQUÍN
DEPTO. DE MATEMÁTICAS	: LICDO. WALTER WILLIAM ARANA

UNIVERSIDAD DE EL SALVADOR

RECTOR

ING. Y MSC. RUFINO ANTONIO QUEZADA SÀNCHEZ

VICE-RECTOR ACADÉMICO

ARQ. Y MASTER MIGUEL ÁNGEL PÈREZ RAMOS

VICE-RECTOR ADMINISTRATIVO

LICDO. Y MASTER OSCAR NOE NAVARRETE

SECRETARIO GENERAL

LICDO. DOUGLAS VLADIMIR ALFARO SANCHEZ

FISCAL GENERAL

DR. RENÈ MADECADEL PERLA JIMÈNEZ

Agradecimientos

a Dios... por la fuerza y la humildad

a mis padres .... por la paciencia y amor sin límites

a mi compañero de tesis... por la solidaridad y entrega

a mi abuelita... por el ejemplo

Sara Margarita Villa

A Aquel por dejarme encontrarlo.

Al Viejo y a la Vieja por aguantar las balas por mi.

A mi compañera de tesis por jamás jamás jamás jamás jamás perder la paciencia.

Y a toda la comunidad de Software Libre por lo enseñado.

Douglas Aljadyz Cerna

Índice de Contenido

Índice de tablas y figuras

Índice de ilustraciones

i. Resumen Ejecutivo

La Facultad Multidisciplinaria de Occidente cuenta con un departamento de Recursos Humanos encargado, entre otras funciones, de la elaboración de las planillas de pago de los empleados de dicha institución. Para ello se apoya en el software desarrollado por el Ministerio de Hacienda llamado SIRH-I en 1995 usando foxpro para DOS versión 2.6a, un lenguaje que actualmente ya no es soportado por Microsoft. Dicho software es utilizado también por la Subgerencia de Personal para centralizar las planillas de todas las facultades. El sistema está instalado en cada una de la Facultades y es completamente independiente.

Pero como toda organización, que obedece a las teorías de la evolución Recursos Humanos ha tenido cambios ocasionados por factores externos e internos: organismos gubernamentales como el Ministerio de Hacienda, mejora constantes de acuerdo a las administraciones y finalmente el dramático avance de la tecnología. Esta natural evolución que ha excluido al software SIRH ocasionó la obsolescencia del mismo.

Y es por esta problemática que surge el presente proyecto: desarrollar una alternativa en una nueva plataforma tecnológica que cumpla con las funciones de administración de la planilla salarial del personal de la FMO y a la vez sea tan flexible que pueda ajustarse a las nuevas necesidades y cambios que surjan.

La definición de los requerimientos para el sistema de administración de planillas fue establecida después del desarrollo de entrevistas con personas claves en la administración de la planilla, la investigación del marco regulatorio en el país y el análisis del sistema informático actual SIRH. El sistema de administración desarrollado en este proyecto fue elaborado bajo el paradigma de programación orientada a objetos y se aprovecharon las bondades de utilizar exclusivamente software libre en su desarrollo. Se aplicaron algunas prácticas de metodologías ágiles de desarrollo como el control de versiones y pruebas de funcionalidad descritos en el capítulo II Marco Metodológico siempre valiéndose del uso de software libre que facilita el ejercicio de éstas.

La principal conclusión derivada del desarrollo de este proyecto es que la migración de la plataforma tecnológica por sí sola, no solventará la problemática de evolución de la organización sino forma parte de un proyecto de mejora de todos los procesos involucrados en la generación de la planilla.

Por ello, la principal recomendación es aplicar reingeniería al proceso de generación de planillas considerando la integración de las facultades e interconexión de los sistemas de recursos humanos tanto entre facultades como con otros sistemas existentes dentro de la Universidad de El Salvador.

ii. Introducción

A mediados de los 90's el Gobierno de El Salvador dio inicio al Programa de Modernización del Sector Público [Banco Mundial, 2004], el cual perseguía, entre otros objetivos, hacer frente a la ineficiencia funcional causada por la débil administración del recurso financiero y humano del estado. Uno de los componentes fundamentales de dicho programa era el manejo de los recursos humanos mediante el desarrollo del Sistema de Administración de Recursos Humanos (SARH).

Como parte del SARH, el Ministerio de Hacienda desarrolló en 1995, gracias a préstamos internacionales y con la ayuda de consultorías internacionales, el Sistema de Información de Recursos Humanos (SIRH) [G. Rivas, comunicación personal, 24 de marzo de 2007], con el objetivo de contar con una plataforma informática común para la gestión del recurso humano de todas las entidades estatales.

El sistema constaba de una base central denominada SIRH Central (SIRH-C) y un conjunto de bases periféricas en cada institución, denominadas SIRH Institucionales (SIRH-I) [Ministerio de Hacienda, 1997]. La base central tenía como función integrar los datos propios que definían a cada institución y que eran almacenados en cada SIRH-I. Estos datos incluían estructuras organizativas e información de cada empleado, entre otros.

El sistema se implementó en dependencias como el Ministerio de Educación, el Ministerio de Economía [J. Sandoval, 2004], el Ministerio

de Salud [Ministerio de Salud, 2004], el Ministerio de Agricultura y Ganadería [Ministerio de Agricultura y Ganadería, 2005], la Procuraduría General de la República [Procuraduría General de la República, 2002] y el Instituto Salvadoreño del Seguro Social (ISSS) [Instituto Salvadoreño del Seguro Social, 2005], entre otras. También fue adoptado por la Universidad de El Salvador (UES), instalándose bases periféricas (SIRH-I) en cada Facultad. La versión 1.0 del SIRH-I fue implementada en la Facultad Multidisciplinaria de Occidente (FMO) en 1997, quedando a cargo de la Unidad de Recursos Humanos (RRHH). En la actualidad, se encuentra en funcionamiento en la FMO la versión 5.2004.

El SIRH fue desarrollado en FoxPro 2.6 para DOS. Puede funcionar en una red local Novell 2.2 o superior, y necesita las aplicaciones de compresión/descompresión de archivos PKZIP y PKUNZIP. Entre sus requerimientos de hardware se encuentran computadoras 486 con por lo menos 8 MB de memoria RAM [Ministerio de Hacienda, 1997].

El sistema cuenta con los siguientes módulos:

- Administración de Estructura: permite la administración de la jerarquía organizativa en unidades y puestos de trabajo.
- Administración de Personal: permite administrar la ficha laboral de cada trabajador de la institución, ubicar a los trabajadores en puestos específicos y realizar movimientos de personal entre unidades.
- Emisión de Planillas: permite preparar las planillas salariales de la institución y administrar conceptos de pago.

- Administración de Clases Ocupacionales: permite administrar el sistema de escalafón.
- Estructura Salarial: permite diseñar o rediseñar la estructura de remuneración en base a políticas definidas.
- Presupuesto de Recursos Humanos: permite generar información para control presupuestario.
- Control de Asistencia: permite generar informes sobre llegadas tardías y solicitudes de permisos.

RRHH actualmente sólo utiliza los primeros tres módulos, siendo el más empleado el módulo de Emisión de Planillas. El sistema también cuenta con un módulo de administrador que permite la creación de cuentas y privilegios de los usuarios del sistema y la habilitación o deshabilitación de módulos. Cada usuario ingresa al sistema identificándose con cuatro iniciales y una clave personal.

Periódicamente RRHH genera distintos tipos de planillas salariales, cuyos datos son llevados a la Subgerencia de Personal de la UES, en donde, mediante el SIRH-C, se consolidan los datos de todas las Facultades y luego se entregan al Ministerio de Hacienda [G. Rivas, comunicación personal, 24 de marzo de 2007].

El SIRH-I es una herramienta debidamente autorizada por el Ministerio de Hacienda que sirve en gran medida para que RRHH atienda las necesidades administrativas de forma automatizada. Sin embargo en el desempeño de sus labores cotidianas, RRHH ha detectado que existen

muchas oportunidades de mejora en el sistema que podrían beneficiar a los mismos empleados de la unidad y al resto de empleados de la Facultad brindándoles un mejor servicio.

Lo que origina el presente trabajo de graduación es la necesidad latente de un sistema moderno que corrija las deficiencias del sistema actual, algunas claramente identificadas debido a los problemas que ocasiona en el desarrollo de las funciones, y otras mas sutiles en el desempeño de las labores pero de gran incidencia tecnológica. Estas deficiencias tienen su origen en la obsolescencia del software [M. Flores, 2005].

La mayoría de problemas identificados surgen de las sencillas reglas de obsolescencia del software; RRHH, obedeciendo las teorías de la Evolución de las Organizaciones, está cambiando constantemente en el tiempo por las influencias externas de los organismos gubernamentales, y por mejorar constantemente según los grupos gerenciales y las tecnologías por lo que sus sistemas de cómputo se tienen que modificar para poderse adaptar a los cambios externos. Además el SIRH-I fue desarrollado para el sistema operativo DOS y actualmente los sistemas operativos han evolucionado por lo que el software corre el riesgo de no funcionar correctamente. Finalmente, fue desarrollado bajo el concepto de ciclo de vida clásico en el que según la Ingeniería de software se termina el tiempo de vida de un sistema.

Lo que justifica el desarrollo de este trabajo de graduación es el supuesto incluido en la Normativa General para la Modernización del Órgano Ejecutivo [Secretaría Técnica de la Presidencia, 1999] que

establece que los sistemas de información que se desarrollen para uso estatal deben ser capaces de migrar a nuevas plataformas tecnológicas, requisito que el actual sistema no cumple.

Mediante el uso de tecnologías como la Internet y los sistemas gestores de bases de datos, este trabajo pretende desarrollar software que pueda servir como base para futuras extensiones y que pueda integrarse con otros sistemas desarrollados para la Universidad, dando mayor eficiencia a los procesos administrativos y aprovechando los beneficios que las nuevas plataformas tecnológicas ofrecen como la consulta en línea de los datos. Además es muy probable que la unidad de Recursos Humanos evolucione e implemente nuevas metodologías de trabajo y éstas presenten nuevos requerimientos, pero el esquema de la base de datos podrá ajustarse a tales necesidades.

Los objetivos que se pretenden alcanzar con el presente trabajo de grado son:

General

- Desarrollar software para la administración de la planilla salarial del personal de la FMO

Específicos

- Dar mayor eficiencia al proceso de generación de la planilla salarial
- Minimizar el margen de error en la generación de reportes de historiales de descuentos
- Automatizar los procesos manuales actuales relacionados con la administración de la planilla
- Proponer una alternativa para un futuro remplazo del SIRH
- Desarrollar software que pueda integrarse con otros sistemas administrativos
- Implementar como mínimo la funcionalidad que se utiliza actualmente en el SIRH-I
- Establecer la base para una posible extensión del proyecto

1. Anteproyecto

1.1 Antecedentes

A mediados de los 90's el Gobierno de El Salvador dio inicio al Programa de Modernización del Sector Público [Banco Mundial, 2004], el cual tenía entre sus objetivos hacer frente a la ineficiencia funcional causada por la débil administración del recurso financiero y humano del estado. Uno de los componentes fundamentales de dicho programa era el manejo de los recursos humanos. Esto dio como resultado la creación del Sistema de Administración de Recursos Humanos (SARH).

Como parte del SARH, el Ministerio de Hacienda desarrolló en 1995, gracias a préstamos internacionales y con la ayuda de consultorías internacionales, el Sistema de Información de Recursos Humanos (SIRH) [G. Rivas, comunicación personal, 24 de marzo de 2007], con el objetivo de contar con una plataforma informática común para la gestión del recurso humano de todas las entidades estatales.

El sistema constaba de una base central denominada SIRH Central (SIRH-C) y un conjunto de bases periféricas en cada institución, denominadas SIRH Institucionales (SIRH-I) [Ministerio de Hacienda, 1997]. La base central tenía como función integrar los datos propios que definían a cada institución y que eran almacenados en cada SIRH-I. Estos datos incluían estructuras organizativas e información de cada empleado, entre otros.

El sistema se implementó en dependencias como el Ministerio de Educación, el Ministerio de Economía [J. Sandoval, 2004], el Ministerio de Salud [Ministerio de Salud, 2004], el Ministerio de Agricultura y Ganadería [Ministerio de Agricultura y Ganadería, 2005], la Procuraduría General de la República [Procuraduría General de la República, 2002] y el Instituto Salvadoreño del Seguro Social (ISSS) [Instituto Salvadoreño del Seguro Social, 2005], entre otras. También fue adoptado por la Universidad de El Salvador (UES), instalándose bases periféricas (SIRH-I) en cada Facultad. La versión 1.0 del SIRH-I fue implementada en la Facultad Multidisciplinaria de Occidente (FMO) en 1997, quedando a cargo de la Unidad de Recursos Humanos (RRHH). En la actualidad, se encuentra en funcionamiento en la FMO la versión 5.2004.

El SIRH fue desarrollado en FoxPro 2.6 para DOS. Puede funcionar en una red local Novell 2.2 o superior, y necesita las aplicaciones de compresión/descompresión de archivos PKZIP y PKUNZIP. Entre sus requerimientos de hardware se encuentran computadoras 486 con por lo menos 8 MB de memoria RAM [Ministerio de Hacienda, 1997].

El sistema cuenta con los siguientes módulos:

- Administración de Estructura: permite la administración de la jerarquía organizativa en unidades y puestos de trabajo.
- Administración de Personal: permite administrar la ficha laboral de cada trabajador de la institución, ubicar a los trabajadores en puestos específicos y realizar movimientos de personal entre unidades.

- Emisión de Planillas: permite preparar las planillas salariales de la institución y administrar conceptos de pago.
- Administración de Clases Ocupacionales: permite administrar el sistema de escalafón.
- Estructura Salarial: permite diseñar o rediseñar la estructura de remuneración en base a políticas definidas.
- Presupuesto de Recursos Humanos: permite generar información para control presupuestario.
- Control de Asistencia: permite generar informes sobre llegadas tardías y solicitudes de permisos.

RRHH actualmente sólo utiliza los primeros tres módulos, siendo el más utilizado el módulo de Emisión de Planillas. El sistema también cuenta con un módulo de administrador que permite la creación de cuentas y privilegios de los usuarios del sistema y la habilitación o deshabilitación de módulos. Cada usuario ingresa al sistema identificándose con cuatro iniciales y una clave personal.

Periódicamente RRHH genera distintos tipos de planillas salariales, cuyos datos son llevados a la Subgerencia de Personal de la UES, en donde, mediante el SIRH-C, se consolidan los datos de todas las Facultades y luego se entregan al Ministerio de Hacienda [G. Rivas, comunicación personal, 24 de marzo de 2007].

1.1.1 La Unidad de Recursos Humanos de la FMO

Como se mencionó anteriormente, RRHH es la encargada de la administración del SIRH-I en la FMO. A continuación se explica el proceso de contratación o nombramiento de personal, y cuál es el papel de RRHH en dicho proceso [G. Rivas, comunicación personal, 24 de marzo de 2007]:

- En el caso del personal académico, cada Departamento establece una comisión interna encargada de llevar a cabo el proceso de selección del nuevo docente. Una comisión institucional da seguimiento a dicho proceso y notifica el resultado del mismo a la Junta Directiva de la Facultad, quien emite un acuerdo donde contrata o nombra al nuevo docente.

- En el caso del personal administrativo, RRHH lleva a cabo el proceso de selección y envía los resultados del proceso de selección a Junta Directiva quien emite un acuerdo de contratación o al Decano quien emite un acuerdo de nombramiento.

RRHH recibe el acuerdo emitido en cada caso, registra el número del mismo y solicita al nuevo trabajador presentar los documentos requeridos, los cuales son utilizados para darle de alta en el SIRH-I.

En cuanto a su organización interna, RRHH cuenta con cuatro integrantes: un Jefe de unidad y tres auxiliares administrativos, y entre sus responsabilidades se encuentran:

- Controlar y registrar la asistencia del personal de la Facultad
- Actualizar los expedientes del personal de la Facultad
- Llevar el registro de permisos con y sin goce de sueldo de todo el personal
- Coordinar trámites de servicios de prestación, de bienestar social y manejo de la documentación correspondiente
- Preparar informe mensual sobre el costo financiero del personal por unidades docentes y administrativas
- Elaborar planillas mensuales y otros documentos requeridos por la unidad de Administración Financiera
- Preparar informes requeridos por el Decanato y la Junta Directiva dentro del área de su competencia

En lo que respecta a la administración del SIRH-I, los miembros de RRHH realizan las siguientes tareas:

- Control de las altas y bajas en la planilla de salarios mensual

- Control de movimiento por traslado de categoría del personal docente y administrativo

- Revisar las firmas en planilla de salarios

- Liquidar las planillas de pago mensual con los sistemas de la Subgerencia de Personal de la Universidad

- Elaborar el informe de liquidación de los aportes patronales y de empleados de las AFPs e ISSS a la Subgerencia de Personal

- Elaboración de constancias de sueldo y de tiempo al personal

- Elaborar el informe resumen de las retenciones efectuadas a los empleados para enviarlas a la Subgerencia de Personal

En lo que respecta al equipo informático, la unidad cuenta con dos computadoras de escritorio: 1 Dell Optiplex GX620 y 1 IBM NetVista M. La primera está asignada al Jefe de la unidad y la otra a sus auxiliares. El

sistema operativo privativo instalado en ambas computadoras es Microsoft Windows XP. Éstas cuentan con navegadores web actualizados así como conexión permanente a Internet. El SIRH-I se encuentra instalado en la computadora asignada a la Jefatura de la Unidad.

La unidad también posee cuatro impresoras: 1 HP Business Inkjet 1200 de inyección, 1 Kyocera KM-1815LA Láser, 1 Panasonic KX-P3696 matricial y 1 Epson DFX-5000+ matricial la cual es utilizada para imprimir las planillas mensualmente.

1.2 Planteamiento del Problema

El SIRH-I es una herramienta debidamente autorizada por el Ministerio de Hacienda que sirve en gran medida para que RRHH atienda las necesidades administrativas de forma automatizada.

Sin embargo en el desempeño de sus labores cotidianas, RRHH ha detectado que existen muchas oportunidades de mejora en el sistema que podrían beneficiar a los mismos empleados de la unidad y al resto de empleados de la Facultad brindándoles un mejor servicio.

Lo que origina el presente trabajo de graduación es la necesidad latente de un sistema moderno que corrija las deficiencias del sistema actual, algunas claramente identificadas debido a los problemas que ocasiona en el desarrollo de las funciones, y otras mas sutiles en el desempeño de las labores pero de gran incidencia tecnológica. Estas deficiencias tienen su origen en la obsolescencia del software [M. Flores, 2005].

La mayoría de problemas identificados surgen de las sencillas reglas de obsolescencia del software; RRHH, obedeciendo las teorías de la Evolución de las Organizaciones, está cambiando constantemente en el tiempo por las influencias externas de los organismos gubernamentales, y por mejorar constantemente según los grupos gerenciales y las tecnologías por lo que sus sistemas de cómputo se tienen que modificar para poderse adaptar a los cambios externos. Además el SIRH-I fue

desarrollado para el sistema operativo DOS y actualmente los sistemas operativos han evolucionado por lo que el software corre el riesgo de no funcionar correctamente. Finalmente, fue desarrollado bajo el concepto de ciclo de vida clásico en el que según la Ingeniería de software se termina el tiempo de vida de un sistema.

Como resultado directo se puede mencionar algunas deficiencias puntuales que el sistema adolece:

- Debido a que la tecnología en la que fue desarrollado ya no es popular, los profesionales en desarrollo de software no incluyen en su educación el conocimiento necesario para mantener el software. Y debido a que no se cuenta con el código fuente, esta tarea se vuelve casi imposible.

- No cuenta con una base de datos relacional lo que lo convierte en un sistema inseguro, frágil, vulnerable e ineficiente [J. Sandoval, 2004].

- Desaprovecha las ventajas de las tecnologías vigentes como el Internet o las bases de datos relacionales.

- No puede integrarse fácilmente con el resto de sistemas de la Universidad, lo que genera las famosas islas de información.

- Está limitado al uso exclusivo de RRHH, los trabajadores de la Facultad son otra clase de usuarios que no está contemplada en el SIRH-I y que deberían poder consulta su información.

- No satisface todos los requerimientos de RRHH, obligándolos a realizar algunos procesos rutinarios de forma manual, como por ejemplo el reporte de los descuentos aplicados a un trabajador en un período determinado.

- Al ser un sistema genérico su estructura incluye datos que nunca han sido usados en la Facultad.

1.3 Objetivos

1.3.1 General

- Desarrollar software para la administración de la planilla salarial del personal de la FMO

1.3.2 Específicos

- Dar mayor eficiencia al proceso de generación de la planilla salarial

- Minimizar el margen de error en la generación de reportes de historiales de descuentos

- Automatizar los procesos manuales actuales relacionados con la administración de la planilla

- Proponer una alternativa para un futuro remplazo del SIRH

- Desarrollar software que pueda integrarse con otros sistemas administrativos

- Implementar como mínimo la funcionalidad que se utiliza actualmente en el SIRH-I

- Establecer la base para una posible extensión del proyecto

1.4 Justificaciones

En la Normativa General para la Modernización del Órgano Ejecutivo [Secretaría Técnica de la Presidencia, 1999] se establece que los sistemas de información que se desarrollen para uso estatal deben ser capaces de migrar a nuevas plataformas tecnológicas, requisito que el actual sistema no cumple.

Mediante el uso de tecnologías como la Internet y los sistemas gestores de bases de datos, este trabajo pretende desarrollar software que pueda servir como base para futuras extensiones y que pueda integrarse con otros sistemas desarrollados para la Universidad, dando mayor eficiencia a los procesos administrativos y aprovechando los beneficios que las nuevas plataformas tecnológicas ofrecen como la consulta en línea de los datos. Además es muy probable que la unidad de Recursos Humanos evolucione e implemente nuevas metodologías de trabajo y éstas presenten nuevos requerimientos, pero el esquema de la base de datos podrá ajustarse a tales necesidades.

Además se beneficiará a todos los trabajadores de la Facultad, pues el remplazo del expediente físico por uno electrónico, algo que fue requerido por la Corte de Cuentas, les permitirá conocer qué información se administra en RRHH. Pero llevar a cabo un proceso completo de captura de datos en un sistema obsoleto que presenta las deficiencias del SIRH-I, constituiría un desperdicio de recursos, ya que en un futuro

los datos deberían ser migrados. Mediante el uso de estándares como el SQL se pretende que el software a desarrollar solvente esta problemática.

En resumen, y como se planteó anteriormente, el sistema actual cayó en la obsolescencia y ya no cumple con las exigencias propias y externas de RRHH, ni tampoco las exigencias de la tecnología que actualmente está en vigencia. Se encuentra todavía en uso debido a que la UES no ha encontrado una alternativa conveniente para su remplazo.

1.5 Alcances

El presente proyecto busca brindar una alternativa al SIRH-I y su desarrollo estará delimitado a solventar las necesidades propias de RRHH, siguiendo los requerimientos que se recopilen de esta unidad y considerando también las disposiciones pertinentes para el manejo de sistemas de este tipo.

El nuevo sistema mantendrá la siguiente funcionalidad provista por algunos módulos del sistema actual:

- Administración de Estructura: en lo que se refiere a la definición del organigrama de la Facultad.

- Administración de Personal: en lo que se refiere a la definición de puestos dentro del organigrama, altas, bajas, movimientos del personal en el sistema.

- Emisión de Planillas: contemplará los diferentes tipos de planillas como las planillas normales, complementarias y de aguinaldo, además del manejo de fuentes de financiamiento y los diferentes conceptos de descuentos.

El nuevo sistema proporcionará la siguiente funcionalidad que el sistema actual no provee:

- Expediente Electrónico del Personal: el personal de la Facultad podrá hacer uso de los servicios de consulta en línea de sus datos.

- Historial de descuento: permitirá a la unidad de Recursos Humanos obtener el reporte historial de descuentos de un trabajador, proceso que actualmente se realiza manualmente.

Entre los aspectos que el desarrollo de este proyecto no tomará en cuenta están:

- No se desarrollará la funcionalidad del módulo de Control de Asistencia.

- El sistema no se pondrá en producción de forma oficial. Aunque se desarrollarán las pruebas funcionales correspondientes para comprobar que es una alternativa eficiente y eficaz al sistema actual, la decisión de implementar el software que se desarrolle como un sistema de uso institucional corresponde a la Subgerencia de Personal, y está fuera del alcance de este proyecto influir en dicha decisión.

- La carga o introducción de datos al sistema y la migración de los datos existentes de forma mecanizada quedará bajo responsabilidad de RRHH.

1.6 Limitaciones

- Debido a que la persona “enlace” entre el grupo de trabajo y RRHH es el Jefe de la unidad, el planeamiento y control de cada iteración estarán sujetos al tiempo disponible que sus actividades laborales le permitan.

- La fecha límite para finalizar la ejecución del proyecto es el 30 de noviembre del presente año.

1.7 Metodología a Utilizar

Para desarrollar el software propuesto se utilizarán principios y prácticas de los denominados “métodos ágiles” de desarrollo [“Principles behind the Agile Manifesto”, s.f.], específicamente de la Programación Extrema [D. Wells, 1999].

Tradicionalmente, los proyectos de desarrollo de software se ejecutan siguiendo un modelo de desarrollo conocido como “modelo en cascada” [Wikipedia, 2007]. Es uno de los más difundidos a nivel académico. Dicho modelo establece la secuencia de etapas que debe seguir el “ciclo de vida” del software, de forma que una etapa debe completarse antes de pasar a la siguiente. Comúnmente se definen las siguientes etapas:

1. Análisis de requerimientos
2. Diseño del software
3. Codificación
4. Pruebas
5. Implementación
6. Mantenimiento

El modelo se caracteriza porque se centra en el producto de cada etapa. El progreso generalmente se representa mediante documentos:

especificaciones de requerimientos, documento de diseño, planes de prueba, revisiones de código, etc.

Pero quizá la principal desventaja del modelo es que da especial énfasis a las etapas de análisis de requerimientos y diseño del software. Dicha tendencia fue inspirada por otras ramas de la ingeniería, en donde existe una clara separación entre el proceso de diseño y el proceso de construcción – por ejemplo la ingeniería civil, en donde se dibujan planos que luego son pasados a los encargados de la construcción [Fowler, 1998]. Esto hace que los proyectos de software tarden en llegar a la fase de implementación. Existen casos en donde los imprevistos en las primeras etapas hacen que se acorte el tiempo de las últimas etapas, lo que resulta en software poco probado (a veces sin probar) siendo puesto en producción.

Un problema asociado a esta separación es que realizar cambios al diseño del software es una tarea difícil. Si el diseño creado en las primeras etapas es incorrecto, ya sea porque los requerimientos no fueron entendidos (o el cliente no supo expresarlos), o si el diseño es muy difícil de implementar, esto se sabe hasta que el mismo se trata de codificar o cuando el software es puesto a prueba. El término “ciclo de vida” trata de solventar estos problemas y una vez el software es puesto en producción, la fase de mantenimiento en realidad se vuelve una fase de “rediseño y reparación”.

Los métodos ágiles de desarrollo de software se enfocan en producir software funcional y altamente probado en períodos de tiempo relativamente cortos (iteraciones). El problema no se resuelve como un todo, sino en pequeños subconjuntos que son desarrollados iterativamente. Algunos métodos ágiles utilizan un modelo en cascada en pequeña escala, repitiendo un ciclo en cada iteración.

La metodología de la Programación Extrema, lleva “al extremo” muchas técnicas de desarrollo que han sido utilizadas por años. Da especial énfasis a la validación del software, produciendo las pruebas (de unidad y funcionales) antes de producir el código que será probado [Ambler, 2007].

Los requerimientos son extraídos mediante historias de usuario, cuyo valor radica en la conversación establecida entre desarrollador y cliente antes de implementar la historia y en que el cliente también debe decidir cómo probar funcionalmente la historia que se va a implementar.

El programador debe definir honestamente cuánto tiempo estima que tomará implementar la historia y hacer que las pruebas pasen. Conociendo estos “estimados”, el cliente tiene la oportunidad de elegir la funcionalidad que más valor represente según su necesidad y solicitar que dicha funcionalidad sea incluida en la presente iteración. Además el cliente debe considerar la “velocidad” de desarrollo, es decir cuantas historias pueden implementarse en cada iteración.

Al acortar el tiempo del ciclo a la duración de una iteración, el cliente puede ver si el software que se va produciendo es el que necesita. Si no lo es, tiene la oportunidad de proponer los cambios requeridos en la siguiente iteración. Esto ayuda a que el software finalmente sea lo que el cliente necesita.

En este proyecto se pretende realizar una sesión con el todo el personal de la unidad de Recursos Humanos en donde se pueda recolectar la mayor cantidad de historias de usuario. Luego se establecerá un tiempo estimado de duración para cada iteración, en base a la disponibilidad de tiempo de los integrantes del grupo de trabajo y tomando en cuenta la fecha para la cual se debe concluir con el desarrollo del software. Se estimarán las historias y se planeará la primera iteración con aquellas historias que resulten de mayor prioridad para Recursos Humanos.

Para las iteraciones siguientes, se harán reuniones de planeamiento de iteraciones, en donde se seleccionarán las historias restantes. Se podrán incluir nuevas historias durante el desarrollo de una iteración, pero negociando el cambio por alguna que todavía no se haya implementado.

1.8 Cronograma de Actividades

#	Nombre de la actividad	Duración	mayo				junio				julio				agosto				septiembre				octubre				noviembre			
			1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
1	Evaluación del Anteproyecto	1 día																												
2	Recopilación de Información para el Capítulo 1	10 días																												
3	Organización y Redacción del Capítulo 1	4 días																												
4	Entrega y Revisión del Capítulo 1 al Docente Director	4 días																												
5	Recopilación de Información para el Capítulo 2	10 días																												
6	Organización y Redacción del Capítulo 2	4 días																												
7	Entrega y Revisión del Capítulo 2 al Docente Director	4 días																												
8	Reunión para recolectar Historias de Usuario	1 día																												
9	Reunión para Planeamiento de Primera Release	1 día																												
10	Desarrollo e Instalación de Primera Release	25 días																												
11	Reunión para Planeamiento de Segunda Release	1 día																												
12	Desarrollo e Instalación de Segunda Release	25 días																												
13	Reunión para Planeamiento de Tercera Release	1 día																												
14	Desarrollo e Instalación de Tercera Release	25 días																												
15	Reunión para Planeamiento de Cuarta Release	1 día																												
16	Desarrollo e Instalación de Cuarta Release	25 días																												
17	Entrega Final del Software	1 día																												
18	Redacción del Capítulo 3	4 días																												
19	Entrega y Revisión del Capítulo 3 al Docente Director	4 días																												
20	Redacción del Capítulo 4	4 días																												
21	Entrega y Revisión del Capítulo 4 al Docente Director	4 días																												
22	Evaluación del Documento Final (Capítulos 1, 2, 3 y 4)	1 día																												
23	Evaluación Pública Final	1 día																												

1.9 Planificación de Recursos a Utilizar

1.9.1 Recurso Material

- Servidor Web del Departamento de Física

  El proyecto “Programa de Apoyo a la Educación en Ciencias” ejecutado por el Departamento de Física de la FMO cuenta con un servidor web cuyas características de hardware y software se detallan a continuación:

  - Principales especificaciones de hardware [Dell, 2004]
    - Marca y modelo: Dell PowerEdge 700
    - Microprocesador: Intel(R) Pentium(R) 4 CPU 2.80GHz FSB 800 Mhz 1MB L2 cache
    - Memoria RAM: 512 MB DDR400MHz SDRAM
    - Disco duro: 160 GB 7200 RPM SATA
    - Tarjetas de red: 2 Intel Gigabit NIC Intel PRO/1000 MT
  - Especificaciones relevantes de Software Libre [Fundación del Software Libre, 2006]:
    - Sistema operativo: Debian GNU/Linux Testing Kernel 2.6.18
    - Lenguaje de programación a utilizar: Python 2.4
    - Framework para desarrollo a utilizar: Zope 3.3
    - Base de datos: ZODB 3.7 o PostgreSQL 8.1

- Control de versiones: Subversion 1.4

Además el servidor cuenta con una dirección IP pública (168.243.33.5) y con un nombre de dominio de Internet (fisica.uesocc.edu.sv) asignado por el administrador de la red de la Facultad.

Se cuenta con autorización para utilizar este servidor durante la etapa de desarrollo del software en este trabajo de graduación.

La Facultad cuenta con otros servidores con características de hardware y software similares que podrían ser utilizados para una futura implementación del software que se desarrollará.

- Equipo informático de RRHH

  Se cuenta con autorización para utilizar las dos computadoras de la unidad para estudiar el SIRH-I, así como para realizar las pruebas funcionales del software que se desarrollará.

- Computadoras propiedad de los responsables del proyecto

  Cada integrante del grupo de trabajo cuenta con su propia computadora personal: 1 computadora de escritorio con microprocesador Intel Pentium 4 de 3.2 GHz, memoria RAM de 1 GB y disco duro de 74 GB; 1 computadora portátil VAIO PCG-V505

con microprocesador Intel Centrino de 1.5 GHz, memoria RAM de 512 MB y disco duro de 80 GB.

La computadora de escritorio tiene como sistema operativo Debian GNU/Linux Testing y la computadora portatil usa Microsoft Windows XP.

Entre el Software Libre multiplataforma que será utilizado para el desarrollo de este trabajo y que está instalado en ambas computadoras, se puede destacar: OpenOffice.org 2, Emacs 21, Apache Web Server 2, Python 2.4, Zope 3.3, Putty 0.58, cliente para Subversion y GIMP 2, entre otros.

Ambas computadoras cuentan con conexión permanente a Internet.

1.9.2 Recurso Humano

- Dos integrantes del grupo de trabajo

La tarea principal de los integrantes del grupo será desarrollar el software requerido en este trabajo de graduación.

Debido a los compromisos laborales de ambos integrantes, se programarán con suficiente anticipación las reuniones con el personal de RRHH, así como con el docente director del trabajo.

- Personal de RRHH

  Se organizarán reuniones periódicas con el personal vinculado con la emisión de planillas. Su tarea principal será ayudar en el planeamiento del alcance del software a desarrollar, mediante la creación y selección de historias de usuario y la determinación de pruebas de aceptación para el software.

  Se organizará un horario de trabajo que interfiera lo menos posible en las actividades diarias de la unidad, pero que permita a los involucrados brindar sus aportes y retroalimentación con bastante frecuencia.

1.9.3 Recurso Financiero

Rubro	Costo por Unidad	Unidades	Costo total
Salarios	$3.00 / Hora	1288	$3,864.00
Depreciación de equipos			
Equipo 1	$16.67 / Mes *	8	$66.64
Equipo 2	$12.50 / Mes *	8	$66.64
Transporte	$0.04 / Km	4096	$163.84
Material de papelería			
Papel	$3.50 / Resma	5	$17.50
Impresiones	$0.15/Hoja	3000	$450.00
Útiles de oficina (empastados, bolígrafos, fichas, CDs, etc)			$200.00
Comunicaciones			
Internet	$0.13 / Hora	2400	$312.00
Telefonía	$0.15 / Minuto	800	$120.00
Defensas	$25.00 / Sesión	3	$75.00
Costo Total del Proyecto			$5,335.62

* Calculado mediante el método de depreciación de línea recta [G. Baca, 2001]

** Haciendo énfasis en que “el software libre **es un asunto de libertad y no de precio**”, se aclara que en este proyecto se usará Software Libre, por lo que no se incurrirá en costos de licencias de uso

1.10 Propuesta de Contenido Capitular

I. **Generalidades**: En este capítulo se brindarán los fundamentos y conceptos que familiaricen al lector con el ámbito de desarrollo del proyecto. Para ello se incluirán los resultados de la investigación bibliográfica concerniente a los sistemas de planillas de pago por un lado, así como las disposiciones que deben respetarse en El Salvador cuando se desarrolla este tipo de aplicaciones. Se incluirá también la investigación de campo que documente la historia y características del SIRH, investigación de otras iniciativas de reemplazo dentro de la UES, trabajos de graduación previos relacionados con sistemas de planillas creados dentro de la UES. Este capítulo además permitirá predecir los hechos y problemas que se abordarán en el desarrollo del proyecto.

II. **Marco Metodológico**: El contenido de este capítulo estará orientado a definir, describir y analizar los procedimientos y metodologías que se emplearán para el desarrollo del proyecto. Explicará la metodología ágil de desarrollo de software conocida como Programación Extrema. Se detallará la filosofía, procesos, métodos, técnicas y herramientas que estén recomendados por la Programación Extrema y que servirán de guía.

III. **Desarrollo (resultados del proyecto)**: Este capítulo pretende documentar los resultados de los principales procesos efectuados

que hayan brindado información o que hayan desencadenado en un producto. Como apoyo se incluirán la documentación de las herramientas usadas en los anexos del trabajo.

IV. **Conclusiones y Recomendaciones**: Este capítulo incluirá el resumen ordenado de los resultados y contribuciones más importantes del proyecto. Además, incluirá las sugerencias u orientaciones que beneficien el ámbito de desarrollo del mismo.

Anexos: En esta parte se incluirá la información secundaria que contribuya a la argumentación o que sea de especial importancia para reforzar aún más el conocimiento que se quiere transmitir.

Bibliografía: Se incluirán las referencias de los documentos usados como apoyo en la investigación, como entrevistas, libros, tesis, recursos en línea, etc.

2. Generalidades de los Sistema de Planillas

Una parte fundamental en el funcionamiento de cualquier empresa, organismo o institución que cuenta con un grupo de empleados es el calculo, registro y pago de los salarios de los mismos. Este registro financiero es conocido como planillas de pago.

Puede afirmarse que la complejidad de estas actividades reside principalmente en la cantidad de empleados a las que hay que administrarle los salarios y descuentos, así a mayor número de empleados mayor complejidad y viceversa.

La planilla puede administrarse de varias formas: una de ellas es formar un departamento con especialistas de planillas encargados dentro de la empresa, aunque existe la opción de contratar servicios de terceros, empresas dedicadas a llevar el registro de las planillas, quienes instalan toda la infraestructura necesaria dentro de las oficinas del interesado y son los responsables del mantenimiento y funcionamiento de los sistemas de planillas. Y finalmente delegar estas actividades a un departamento generalmente el de contabilidad o recursos humano, tal es el caso de la Universidad de El Salvador. Además pueden contar con un sistema informático que les permita generar la planilla automáticamente o hacerlo de forma completamente manual; aunque una combinación de ambos es también común. Por ejemplo en la Universidad de El Salvador se genera la planilla utilizando el sistema denominado SIRH (Sistema Institucional de Recursos Humanos)

desarrrollado por el Ministerio de Hacienda

Las actividades que cubren la elaboración de la planilla son mas o menos las mismas en todas las empresas y podemos definir así un proceso general que se detalla a continuación:

1. Alta a los empleados nuevos: Tanto si es un sistema manual o computarizado los empleados nuevos deben brindar información que es luego registrada para calcularle su salario y deducciones.

2. Registro de los horas laboradas por los empleados: Normalmente los empleados asalariados no se ven afectados en el cálculo de su salario para cada planilla, pero el empleador normalmente lleva un control de las horas que sus empleados laboran, esto lo puede hacer a través de una hoja de control manual, un reloj marcador de tarjetas o un reloj de control computarizado.

3. Recolección, verificación y autorización de las horas laboradas: El personal de planilla debe condensar para cada empleado las horas laboradas, verificar que se haya registrado la cantidad de horas correctas, y la autorización de los supervisores de los empleados en cuanto a horas extra se refiere.

4. Cálculo del salario: Básicamente es el resultado de multiplicar la cantidad de horas trabajadas por una tasa salarial estandar para cada empleado. Sin embargo se puede ver afectada por los salarios de horas extra, bonos, o un cambio parcial de salario en el período que se está reportando.

5. Actualización de los datos de los empleados: Los empleados pueden solicitar cambios en su nómina de pagos, generados por diversas razones como el cambio en los impuestos o en las deducciones.

6. Cálculo de impuestos aplicables: Calcular el monto de los impuestos concernientes que indica la ley.

7. Cálculo de deducciones aplicables al salario: Las deducciones son los descuentos que se hacen sobre el salario base. La naturaleza de las deducciones puede ser de dos tipos voluntaria e involuntarias. Las voluntarias pueden incluir rubros como fondo para la pensión y seguros médicos; las involuntarias incluyen deudas de préstamos o descuentos legales. La regularidad y modalidad de descuento (por adelantado o con retraso) puede variar así como el período de descuento, estos son factores que el personal que lleva las planillas debe considerar y tener registrado.

8. Registro de los pagos entre planillas: En ocasiones deben extenderse pagos entre planillas para corregir algún pago erróneo, una indemnización o un adelanto de salario. Cualquiera que sea la razón, este pago debe incluirse en la planilla regular para que pueda ser reportada en los libros contables.

9. Elaborar el registro de la planilla: El cálculo resultante de sumas de salarios, bonos y de las deducciones de cada empleado es registrado oficialmente en la planilla. Normalmente si se usa un software, éste se encarga de calcularlo.

10. Verificación de los salarios e impuestos: Normalmente se realiza comparando la nómina de pago actual del empleado con las de períodos anteriores para detectar montos que están fuera de lo usual.

11. Impresión de las nóminas: Lo usual es que se imprima una notificación de pago para cada empleado con el detalle de sus salarios y deducciones usando un formato estándar desde un sistema de planilla.

12. Ingreso de las planilla en los libros contables: Se transfiere la información de planilla a contabilidad para ingreso en los libros contables o sistemas contables.

13. Realizar depósitos de los salarios en las cuentas de los empleados. Por lo general las empresas tienen convenios con los bancos para realizar los depósitos en las cuentas de los empleados solo notificándole las cuentas y el monto a depositar en cada cuenta, es muy común realizar esta notificación a través de internet.

14. Realizar el pago de los impuestos retenidos: Por lo general el empleador deposita todas las retenciones y deducciones en un banco debidamente autorizado para recibir estos montos.

15. Emisión de reportes de planillas para el gobierno: De forma regular el gobierno exige ciertos reportes relacionados con la planilla.

El proceso de elaboración de las planillas es realmente una actividad repetitiva y común que se ha visto mejorada por la incursión la tecnología que se ha prestado para automatizar muchas tareas y así pueden encontrarse instalados relojes computarizados para la recolección de las horas trabajadas, la utilización de sistemas informáticos de planillas que recolectan la información de los empleados y calculan la planilla, emiten reportes, alimentan otros sistemas, etc, hasta el depósito de los empleados en los bancos via internet y el pago de impuestos también via internet.

2.1 Procedimientos y políticas de planilla

Un procedimiento o política de planilla es un documento escrito que detalla la razón de ser de una actividad, quiénes son los responsables, y describe cómo ésta debe ser llevada a cabo. Esta técnica de documentación es altamente aplicable a la planilla, debido a que es una función llena de actividades que deben ser realizadas siempre de la misma forma.

Para documentar los procedimientos es recomendable crear un diagrama de flujo del proceso general, de forma que sea fácil identificar todas las actividades que serán descritas posteriormente. Los cuadros del diagrama de flujo deben llevar un número identificador de proceso, éste mismo se escribirá en el encabezado del proceso correspondiente. Luego de crear el diagrama se procede a escribir un proceso para cada una de las actividades o cuadros del diagrama. Cada proceso es un documento independiente, pero todos deben respetar el mismo formato:

- Cabecera: En la parte derecha de la cabecera se incluyen 4 datos: El número identificador de proceso que sirve para ubicar el proceso en el diagrama, el número de página, la fecha en que fue escrito el procedimiento, y el número de proceso que reemplaza si éste fuera el caso.

- Cuerpo: Está compuesto por tres secciones: “Propósito y alcance”, “Responsabilidades” y “Procedimientos”. La primera sección explica de que trata el proceso en forma resumida; la segunda, quienes son los responsables involucrados de llevar a cabo las tareas y la tercera, lista los pasos que deben seguirse.

Es recomendable que esta documentación se realice dentro de la Unidad encargada de la planilla de forma continua para ampliarla, corregirla, modificarla, pues es de mayor provecho si se actualiza a medida que los procesos cambian. Para iniciar esta investigación se hizo una primera documentación de los procesos generales desde el punto de vista de Guillermo Hernández, el encargado de planilla de la SubGerencia de Personal. Las ventajas de elaborar esta documentación es que permite en primer lugar, entrenar a cualquier persona involucrada en el proceso entregándole lineamientos en los que se pueda apoyar, ya sea un empleado nuevo o antiguo. En segundo lugar unificar y uniformar procesos en todas las instancias en que se ejecute el mismo proceso y en último lugar brindar un vistazo general que permita identificar puntos de control o puntos débiles que puedan mejorarse.

En la Universidad de El Salvador se elaboran diferentes planillas: planilla general, planilla de aguinaldo, planilla de horas extras, planilla de bonificaciones, etc.

2.2 Legislación salvadoreña sobre los salarios y descuentos

Existe una serie de leyes que afectan el salario que los empleados devengan: impuestos, retenciones, pensiones alimenticias, etc. A continuación se detallan los principales descuentos y deducciones que cualquier sistema de planillas debe considerar.

2.2.1 Ley de Seguro Social

La Seguridad Social puede definirse como la protección que la sociedad proporciona a sus miembros o la cobertura a los mismos en necesidades socialmente reconocidas como la pobreza, enfermedad, maternidad, accidentes de trabajo, o enfermedad laboral, desempleo, invalidez, vejez y muerte; y la protección en forma de asistencia médica y de ayuda a las familias con hijos. Solorio (2001).

En El Salvador la seguridad social es un servicio de carácter público y obligatorio, así establecido en la constitución de la República 1983 (artículo 50) donde además se especifica que el organismo que está a cargo del planteamiento, dirección y administración del Seguro es el Instituto Salvadoreño del Seguro Social (ISSS) y para regir este servicio la Corte Suprema de Justicia decretó la Ley del Seguro Social.

La ley del seguro social especifica que la fuente de financiamiento para dicho servicio son las cotizaciones que aportan todos los empleados, patronos y el Estado.

Esta ley cuenta con una sección denominada “Reglamento para la aplicación del régimen del seguro social” donde especifica que el pago de las aportaciones de los empleados y patronos están a cargo del empleador y que éste tiene la obligación de descontar del salario del empleado la cotización correspondiente al momento de efectuar el pago de sus remuneraciones, es bajo esta ley que se ampara la deducción de Seguro Social que se realiza en el cálculo de la planilla.

Algunas consideraciones que habrá que tener en cuenta al momento de calcular la planilla y que se contemplan en el reglamento anteriormente mencionado son:

1. Los ingresos del empleado que se ven afectos por esta aportación son las remuneraciones totales que recibe por los servicios prestados, a diferencia de los ingresos por viáticos, aguinaldos y gratificaciones extraordinarias que son exentos al Seguro y no deben considerarse para la aportación. DL 1263 Ley del Seguro Social, artículo 2.

2. El monto a pagar por las cotizaciones se divide entre empleado, patrono y Estado, correspondiéndole al empleado aportar un 3%

de su salario y al patrono un 7.5% de la remuneración. DL 1263 Ley del Seguro Social, artículo 46.

3. El ISSS facilitará al empleador formularios especiales en los que se deberá remitir la planilla de forma mensual y el pago de las cotizaciones también debe hacerse de forma mensual. Existen dos formas de recaudación: “Planilla Elaborada por el Patrono” y “Planilla Pre-elaborada con Facturación Directa”. En ambas el pago de las cotizaciones debe hacerse dentro de los primeros ocho días hábiles del mes siguiente al que se refiere la planilla, y la remisión de la planilla se deberá realizar en los primeros ocho días para la primera forma y en los primeros cinco para la segunda. A su vez el atraso en esta actividad hace que el patrono sea sancionado con una multa cuyo monto asciende a un porcentaje de las cotizaciones asi: el 25% si el atraso es en la entrega de la planilla; y si el atraso es en el pago el monto asciende al 5% si el atraso es menor a quince días, y asciende 10% si el atraso es mayor de quince días.

2.2.2 Ley del Sistema de Ahorro para Pensiones

Las pensiones pueden definirse como el mecanismo que el estado define para crear un plan de jubilación obligatorio que se hace efectivo en caso de invalidez común, vejez y muerte ofreciéndole a sus empleados ingresos seguros de por vida.

En El Salvador actualmente el sistema de Pensiones está conformado por el Sistema de Ahorro para Pensiones y el Sistema de Pensiones Público. El sistema de Ahorro para Pensiones de reciente implementación, se caracteriza porque la administración de los fondos se le otorga a empresas privadas, denominadas Instituciones Administradoras o Administradora de Fondos de Pensiones (AFP); mientras que en el Sistema Público, la administración está a cargo del Instituto Salvadoreño del Seguro Social (ISSS) y el Instituto Nacional de Pensiones de los Empleados Públicos (INPEP). Este último sistema dejará de existir cuando su último afiliado fallezca.

De igual manera que en la Ley del Seguro Social, los fondos para las pensiones provienen de las cotizaciones que obligatoriamente aportan los empleados en conjunto con sus patronos y que estos últimos envían a las Instituciones Administradoras.

Todos los empleados públicos y privados tienen la obligación irrevocable de afiliarse al Sistema cuando son contratados por primera vez, su afiliación es de caracer permanente, persiste aún cuando no esté trabajando y termina en el momento de su muerte. Además tiene la libertad de escoger la Institución Administradora a la cual afiliarse y puede cambiarse incluso de Institución, sin embargo no puede cotizar a más de una Institución Administradora a la vez.

El empleador debe descontar del salario del empleado la cotización correspondiente al momento de pagarle, además de una cuota

voluntaria si el empleado así lo autoriza, preparar la planilla y remitirla a la Institución Administradora a la que está afiliado el empleado, así como realizar el pago de las cotizaciones a dichas Instituciones de forma mensual. Esto para cada uno de los empleados. El empleador dejará de hacer esto hasta que el afiliado cumple con el requisito de edad para pensionarse por vejez. Sin embargo si el empleado así lo desea y continúa laborando puede continuar aportando cotizaciones de mutuo acuerdo con el patrono.

El ingreso base sobre el cual el empleador tiene que realizar el cálculo y descuento es el salario mensual que incluye retribuciones por servicios, sobresueldos, período de vacaciones, comisiones y porcentajes de ventas. El monto de las cotizaciones es del 13% del salario del empleado si cotiza bajo el Sistema de Ahorro para Pensiones, del 14% del salario si lo hace bajo el Sistema de Pensiones Público y del 12% si lo hace con el IPSFA. La distribución de las cotizaciones para el Sistema de Ahorro para Pensiones es del 6.75% al empleador y 6.25% al empleado; para el Sistema de Pensiones Público el porcentaje es del 7% para ambas partes; y para el IPSFA es del 6% para ambas partes. Además, como se menciona anteriormente el empleado puede aportar una cuota voluntaria adicional si asi lo desea de forma periódica u ocasional. A continuación se presenta resumido los diferentes porcentajes de cotizaciones en los diferentes sistemas.

Tabla 1: Sistema de Ahorro para pensiones (en las AFP)

TIPO	COTIZACIONES EN PORCENTAJES AL SAP (AFP)			
	Empleado	Empleador		Total
		Cot. Cta.	Comisión máxima	
Empleado Público Administrativo y Empleado Empresa Privada	6.25	4.05	2.70	13.00
Empleado Público Docente	6.25	4.05	2.70	13.00

Tabla 2: Sistema de pensiones público (en el INPEP y en el ISSS)

TIPO DE EMPLEADO	COTIZACIONES EN PORCENTAJES AL SPP (ISSS E INPEP)		
	Empleador	Empleado	Total
Empleado Público Administrativo (cotiza al INPEP)	7.00	7.00	14.00
Empleado Público Docente (cotiza al INPEP)	7.00	7.00	14.00
Empleado Empresa Privada (cotiza al ISSS)	7.00	7.00	14.00

Tabla 3: Sistema de pensiones de la Fuerza Armada

TIPO DE EMPLEADO	COTIZACIONES EN PORCENTAJES AL IPSFA		
	Empleado	**Empleador**	**Total**
Empleado Público Administrativo y Empleado Empresa Privada	6.00	6.00	12.00
Empleado Público Docente	6.00	6.00	12.00

La obligación de trasladar estas cotizaciones, según la ley, debe realizarse dentro de los diez primeros días hábiles del mes siguiente a aquél en que se devengaron los ingresos afectos. En caso de no cumplir con este plazo al empleador se le impondrá un sanción.

Hay dos tipos de sanciones: la que se incure cuando no se presenta la declaración de cotizaciones y en la que se incurre cuando no se pagan las cotizaciones. Para la primera genera una multa equivalente al 5% de las cotizaciones durante los siguientes veinte dias de la fecha límite y genera una multa del 10% si el atraso es mayor de veinte dias.

En el segundo tipo, si el empleador no paga absolutamente nada de las cotizaciones será sancionado con una multa del 20% de la cotización no pagada mas un recargo moratorio del 2% por cada mes o fracción de atraso, además de las rentabilidades dejadas de percibir en las cuentas de los afiliados afectados. Si por otra parte, el empleador paga una suma inferior a la cotización que

corresponde será sancionado con una multa del 10% de las cotizaciones dejadas de pagar más un recargo moratorio del 5% de dichas cotizaciones por cada mes o fracción de atraso además de las rentabilidades dejadas de percibir en las cuentas de los afectados.

Las condiciones para que los empleados tengan derecho a su pensión de vejez son:

1. Cuando el saldo de su cuenta individual de ahorro para pensiones sea suficiente para financiar una pensión igual o superior al sesenta por ciento del Salario Básico Regulador.

2. Cuando el empleado haya cotizado durante 30 años, continuos o discontinuos independientemente de la edad.

3. Cuando haya cumplido 60 años de edad si es hombre o 55 años de edad si es mujer y un mínimo de 25 años de cotizaciones, contínuas o discontinuas.

2.2.3 Ley de Impuesto sobre la Renta

La deducción de la renta se ampara en el artículo 2 del reglamento de ley de impuesto sobre la renta que declara que todas las personas que realizan actos de contenido económico están sujetos a las obligaciones tributarias y los ingresos obtenidos en el período que comprende el ejercicio, que dura un año, constituyen la renta obtenida, sobre la cual se realiza el cálculo del impuesto sobre la renta.

Para los empleados asalariados, la remuneración es afecta a este impuesto y el empleador deberá descontarle el monto correspondiente al momento de pagarle, este concepto se llama retención de renta y es el método empleado para recaudar impuestos, multas, etc. Los descuentos debe realizarlos mensualmente el empleador de acuerdo a la siguiente tabla. DL 75 Tablas de retención del impuesto sobre la renta, artículo 1.

Tabla 4: Tabla de retención mensual del impuesto sobre la renta

Si la remuneración mensual es:		
Desde	**Hasta**	**El impuesto a retener será de**
$0.0	$316.67	Sin retención
$316.67	$469.05	$4.77 más el 10% sobre exceso de $316.67
$469.05	$761.91	$4.77 más el 10% sobre exceso de $228.57
$761.91	$1,904.69	$60 más el 20% sobre exceso de $761.91
$1,904.69	En adelante	$228.57 más el 30% sobre exceso de $1,904.69

Al igual que en las deducciones anteriores, la ley determina que el agente de retención, en este caso el empleador, debe enterar la suma retenida al encargado de la percepción del impuesto, dentro de los diez días hábiles que inmediatamente sigan al vencimiento del período en que se efectúe la retención.

Los ingresos afectos por esta ley son aquellos ingresos percibidos en concepto de salarios, sueldos, sobresueldos, horas extras, primas, comisiones, gratificaciones, aguinaldos y cualquier otra compensación por servicios personales, ya sea que éstos se paguen en efectivo o especie y en donde la prestación de servicio es por tiempo indefinido o bien cuando dichos servicios se contraten por un plazo determinado bien sea a tiempo completo, medio tiempo o tiempo parcial, con carácter de subordinación o dependencia.

2.2.4 Retenciones para pensiones alimenticias

Otra deducción que no es para todos los empleados pero que puede presentarse es la destinada para pensiones alimenticias y si es el caso, ésta goza de preferencia en el sistema de retención, sin tomar en cuenta las restricciones que establezcan otras leyes. La retención ordenada deberá acatarse inmediatamente por la persona encargada de hacer los pagos, de no ser así será responsable de igual forma que el obligado al pago de las cuotas alimenticias no retenidas. El envío de estas retenciones debe realizarse en los primeros 3 días hábiles siguientes del

pago del respectivo salario. DL 677 Código de Familia, artículo 264.

2.2.5 Otras retenciones

La Universidad también aplica otros descuentos a todos los empleados.

- Fondo Universitario de Protección (FUP): Servicios de salud para el empleado y su familia. El porcentaje a descontar es del 1% del salario.

- Pensión: La Universidad prepara un plan de ahorro para el retiro del empleado. El aporte del empleado y de la Universidad es del 1.5% sobre el salario.

2.3 El sistema de planillas en la Universidad de El Salvador

A mediados de los 90's el Gobierno de El Salvador dio inicio al Programa de Modernización del Sector Público [Banco Mundial, 2004], el cual tenía entre sus objetivos hacer frente a la ineficiencia funcional causada por ladébil administración del recurso financiero y humano del estado. Uno de los componentes fundamentales de dicho programa era el manejo de los recursos humanos. Esto dio como resultado la creación del Sistema de Administración de Recursos Humanos (SARH).

Como parte del SARH, el Ministerio de Hacienda desarrolló en 1995, gracias a préstamos internacionales y con la ayuda de consultorías internacionales, el Sistema de Información de Recursos Humanos (SIRH) [G. Rivas, comunicación personal, 24 de marzo de 2007], con el objetivo de contar con una plataforma informática común para la gestión del recurso humano de todas las entidades estatales.

El sistema constaba de una base central denominada SIRH Central (SIRHC) y un conjunto de bases periféricas en cada institución, denominadas SIRH Institucionales (SIRH-I) [Ministerio de Hacienda, 1997]. La base central tenía como función integrar los datos propios que definían a cada institución y que eran almacenados en cada SIRH-I. Estos datos incluían estructuras organizativas e información de cada empleado, entre otros.

El sistema se implementó en dependencias como el Ministerio de Educación, el Ministerio de Economía [J. Sandoval, 2004], el Ministerio de Salud [Ministerio de Salud, 2004], el Ministerio de Agricultura y Ganadería [Ministerio de Agricultura y Ganadería, 2005], la Procuraduría General de la República [Procuraduría General de la República, 2002] y el Instituto Salvadoreño del Seguro Social (ISSS) [Instituto Salvadoreño del Seguro Social, 2005], entre otras. También fue adoptado por la Universidad de El Salvador (UES), instalándose bases periféricas (SIRH-I) en cada Facultad.

La versión 1.0 del SIRH-I fue implementada en la Facultad Multidisciplinaria de Occidente (FMO) en 1997, quedando a cargo de la Unidad de Recursos Humanos (RRHH). En la actualidad, se encuentra en funcionamiento en la FMO la versión 5.2004.

El SIRH fue desarrollado en FoxPro 2.6 para DOS. Puede funcionar en una red local Novell 2.2 o superior, y necesita las aplicaciones de compresión/descompresión de archivos PKZIP y PKUNZIP. Entre sus requerimientos de hardware se encuentran computadoras 486 con por lo menos 8 MB de memoria RAM [Ministerio de Hacienda, 1997].

El sistema cuenta con los siguientes módulos:

- Administración de Estructura: permite la administración de la jerarquía organizativa en unidades y puestos de trabajo.

- Administración de Personal: permite administrar la ficha laboral de cada trabajador de la institución, ubicar a los trabajadores en puestos específicos y realizar movimientos de personal entre unidades.

- Emisión de Planillas: permite preparar las planillas salariales de la institución y administrar conceptos de pago.

- Administración de Clases Ocupacionales: permite administrar el sistema de escalafón.

- Estructura Salarial: permite diseñar o rediseñar la estructura de remuneración en base a políticas definidas.

- Presupuesto de Recursos Humanos: permite generar información para control presupuestario.

- Control de Asistencia: permite generar informes sobre llegadas tardías y solicitudes de permisos.

RRHH actualmente sólo utiliza los primeros tres módulos, siendo el más utilizado el módulo de Emisión de Planillas. El sistema también cuenta con un módulo de administrador que permite la creación de cuentas y privilegios de los usuarios del sistema y la habilitación o deshabilitación de módulos. Cada usuario ingresa al sistema identificándose con cuatro iniciales y una clave personal.

Periódicamente RRHH genera distintos tipos de planillas salariales, cuyos datos son llevados a la Subgerencia de Personal de la UES, en donde, mediante el SIRH-C, se consolidan los datos de todas las Facultades y luego se entregan al Ministerio de Hacienda [G. Rivas, comunicación personal, 24 de marzo de 2007].

2.4 La Unidad de Recursos Humanos de la FMO

Como se mencionó anteriormente, RRHH es la encargada de la administración del SIRH-I en la FMO. A continuación se explica el proceso de contratación o nombramiento de personal, y cuál es el papel de RRHH en dicho proceso [G. Rivas, comunicación personal, 24 de marzo de 2007]:

- En el caso del personal académico, cada Departamento establece una comisión interna encargada de llevar a cabo el proceso de selección del nuevo docente. Una comisión institucional da seguimiento a dicho proceso y notifica el resultado del mismo a la Junta Directiva de la Facultad, quien emite un acuerdo donde contrata o nombra al nuevo docente.

- En el caso del personal administrativo, RRHH lleva a cabo el proceso de selección y envía los resultados del proceso de selección a Junta Directiva quien emite un acuerdo de contratación o al Decano quien emite un acuerdo de nombramiento.

RRHH recibe el acuerdo emitido en cada caso, registra el número del mismo y solicita al nuevo trabajador presentar los documentos requeridos, los cuales son utilizados para darle de alta en el SIRH-I.

En cuanto a su organización interna, RRHH cuenta con cuatro integrantes: un Jefe de unidad y tres auxiliares administrativos, y entre sus responsabilidades se encuentran:

- Controlar y registrar la asistencia del personal de la Facultad
- Actualizar los expedientes del personal de la Facultad
- Llevar el registro de permisos con y sin goce de sueldo de todo el personal
- Coordinar trámites de servicios de prestación, de bienestar social y manejo de la documentación correspondiente
- Preparar informe mensual sobre el costo financiero del personal por unidades docentes y administrativas
- Elaborar planillas mensuales y otros documentos requeridos por la unidad de Administración Financiera
- Preparar informes requeridos por el Decanato y la Junta Directiva dentro del área de su competencia

En lo que respecta a la administración del SIRH-I, los miembros de RRHH realizan las siguientes tareas:

- Control de las altas y bajas en la planilla de salarios mensual

- Control de movimiento por traslado de categoría del personal docente y administrativo

- Revisar las firmas en planilla de salarios

- Liquidar las planillas de pago mensual con los sistemas de la Subgerencia de Personal de la Universidad

- Elaborar el informe de liquidación de los aportes patronales y de empleados de las AFPs e ISSS a la Subgerencia de Personal

- Elaboración de constancias de sueldo y de tiempo al personal

- Elaborar el informe resumen de las retenciones efectuadas a los empleados para enviarlas a la Subgerencia de Personal

En lo que respecta al equipo informático, la unidad cuenta con dos computadoras de escritorio: 1 Dell Optiplex GX620 y 1 IBM NetVista M. La primera está asignada al Jefe de la unidad y la otra a sus auxiliares. El sistema operativo privativo instalado en ambas computadoras es Microsoft Windows XP. Éstas cuentan con navegadores web actualizados así como conexión permanente a Internet. El SIRH-I se encuentra instalado en la computadora asignada a la Jefatura de la Unidad.

La unidad también posee cuatro impresoras: 1 HP Business Inkjet 1200 de inyección, 1 Kyocera KM-1815LA Láser, 1 Panasonic KX-P3696 matricial y 1 Epson DFX-5000+ matricial la cual es utilizada para imprimir las planillas mensualmente.

3. Metodología de Desarrollo

Del primer capítulo, en que se definieron los procesos y conceptos involucrados en la planillla, logró concluirse que debido a su naturaleza repetitiva, la planilla permite la introducción de sistemas informáticos que automaticen casi todas las etapas, siendo la etapa más significativa de automatización la de la generación y cálculo de la planilla, que es el objetivo de este trabajo.

Una vez establecidos los conceptos teóricos y legales que consideran la Facultad Multidisciplinaria de Occidente para este proceso, es necesario proceder a la producción en sí del sistema que automatice la generación de planilla. Antes de comenzar es necesario aclarar que la producción de cualquier tipo de software está regida por un sinúmero de técnicas que se recogen en la disciplina denominada Ingeniería de software. La aplicación de la Ingeniería de Software no es una novedad, sin embargo es una disciplina que ha evolucionado y que desarrolló nuevas propuestas. Las metodologías comunes denominadas ahora metodologías tradicionales, hacen hincapie en la etapa de planificación sin embargo no consideran que un cambio en los requerimientos pueda ocurrir en el proyecto fuera de la fase de recolección. Sin embargo este escenario es parte de la realidad en cualquier proyecto de programación, por eso surgen las denominadas metodologías ágiles que establecen técnicas que hacen hincapie en los cambios de los requerimientos. Para el desarrollo de este proyecto se ha seleccionado las metodologías ágiles.

En el curso de este capítulo se definirá a profundidad en que consisten estas metodologías y las ventajas que presentan por sobre las metodologías tradicionales.

3.1 Metodologías ágiles

Las metodologías ágiles son un paradigma de desarrollo de software basado en procesos ágiles que agrupan las metodologías que surgen como alternativa a las tradicionales.

3.2 ¿Que es la programación extrema?

La programación extrema, XP por sus siglas en inglés, puede ser definida como una nueva disciplina para el desarrollo de software que fue desarrollada a principios de los 90's por Kent Beck y en colaboración con Wark Cunningham. XP es una de las metodologías ágiles más populares y exitosas aplicables a cualquier proyecto de desarrollo de software. Está orientado a romper con los viejos esquemas de desarrollo, resolver los problemas que en las metodologías tradicionales siempre se presentan.

El éxito del XP se basa en el énfasis que se le da a la satisfacción del cliente. La metodología está diseñada para darle al cliente el software que necesita cuando lo necesita. XP le da a los programadores la

capacidad de responder a los requerimientos de los clientes, aun cuando el proyecto se encuentre en sus etapas finales.

XP está conformado por

- Una filosofía de desarrollo de software basada en los valores de comunicación, retroalimentación, simplicidad, valentía y respeto.

- Un cuerpo de prácticas probadas y útiles en mejorar el desarrollo de software. Las prácticas se complementan unas con otras, de manera que se amplifican sus efectos. Se escogen como una expresión de los valores.

- Un conjunto de principios complementarios, técnicas intelectuales para trasladar los valores en prácticas, utiles cuando no exista una práctica aplicable para un problema en particular.

- Una comunidad que comparte estos valores y muchas de las mismas prácticas.

3.2.1 Valores, Principios y Prácticas

Se mencionaba anteriormente los pilares que conforman la Programación Extrema. Para profundizar un poco en el tema se aclara a

continuación la relación entre ellos.

Los **valores** que pueden definirse como el criterio que se usa para juzgar lo que vemos, pensamos y hacemos; los valores son universales y son aplicables a cualquier ámbito de la vida. Las **prácticas** que son las cosas que se hacen a diario en el desempeño de las labores como buenos hábitos y difieren completamente de un ámbito a otro de la vida, éstas pueden definirse como una expresión de los valores, es decir que nacen con el propósito de practicar un valor. Y finalmente los **principios** son los lineamientos específicos podría decirse de carácter científico que respaldan a las prácticas.

La programación extrema se enfoca en cinco valores que guían el desarrollo: comunicación, simplicidad, retroalimentación, coraje y respeto.

La comunicación es un valor útil en cualquier equipo de trabajo, ayuda a desarrolar el sentido de equipo de trabajo y de cooperación. Pero su verdadero valor se ve refle ado en la resolución de problemas, pues el conocimiento, la información y la experiencia se comparte entre los integrantes para ayudarlos a resolver problemas o para evitar que ocurran nuevamente.

La simplicidad es un valor relativo al contexto de desarrollo. Así lo que sea simple para un equipo puede ser lo más complejo para otro equipo

de trabajo; lo que hoy es una solución simple, mañana será catalogado como algo complejo. Pero básicamente se basa en desarrollar un sistema lo suficientemente simple que permita resolver los problemas de la actualidad.

En XP el valor de la retroalimentación tiene mucha importancia porque es el valor que da inicio a la adaptación a los cambios. Se trata de obtener retroalimentación de varias formas y de la manera más rápida. Incluso es necesario que se modifiquen los tiempos programados si se hace en pos de obtener y tomar en cuenta toda la retroalimentación.

El coraje es simplemente el valor de hacer. Tomar una decisión, probar una nueva técnica, resolver un problema, etc, requieren de parte del equipo iniciativa y valor de actuar.

Finalmente para que toda la metodología XP funcione, debe existir y practicarse el respeto. El respeto por lo que los miembros del equipo hacen, el respeto por el proyecto, por los usuarios.

Los equipos pueden definir además otros valores que rijan el comportamiento del grupo, por ejemplo, la seguridad, calidad de vida, predictabilidad, etc. Los valores por sí solos no le dicen al equipo cómo desarrollar el software, eso lo hacen las prácticas, y para llegar a las prácticas a partir de los valores es necesario tender un puente. Este puente son los Principios: humanidad, economía, beneficio mutuo, auto

similaridad, mejora, diversidad, reflexión, circulación, oportunidad, redundancia, fracaso, calidad, pequeños pasos, responsabilidad aceptada.

A continuación un resumen de lo que implica poner en práctica los principios:

1. Mostrar respeto por la ***Humanidad*** del equipo.

2. ***Economía***: Optimizar el valor del negocio.

3. Convertir todas las actividades de forma que sean de M***utuo Beneficio***para todos.

4. ***Auto-similaridad*** es tratar de copiar la ESTRUCTURA de una solución en un nuevo contexto, incluso a diferentes escalas.

5. Luchar por la ***Mejora*** contínua en todos los aspectos.

6. Buscar la ***Diversidad*** de perspectivas y destrezas en el equipo.

7. ***Reflexionar*** en el “cómo y porqué” del trabajo.

8. Asegurar un ***Flujo*** continuo del valor del negocio.

9. Visualizar un problema como una ***Oportunidad*** de cambio.

10. Usar ***Redundancia*** de soluciones diferentes.

11. Para tener éxito hay que arriesgarse a ***Fracasar.***

12. Maximizar la ***Calidad***, pero no intentar la perfección.

13. Dar ***Pequeños pasos*** en vez de grandes cambios explosivos.

14. Los miembros del equipo deben ***Aceptar responsabilidades*** para trabajar.

3.3 Prácticas

Las prácticas son el día a día de los programadores. Son vías, modos, métodos, costumbres, estilos aplicados con el fin de llegar a un estado de desarrollo efectivo. Las prácticas dejan de ser una mera costumbre cuando se hacen con el objetivo de darle vida a un valor. La aplicación de las prácticas dependen de la situación, de acuerdo a las condiciones los programadores pueden escoger una u otra práctica para darle vida a

un valor. Los valores por su parte son absolutos indiferentemente de las condiciones. Los programadores pueden no solo escoger una práctica sino crear otras con el fin de obtener un valor para el cual no exista una práctica adecuada.

La mejor forma de aplicar XP a un proyecto parece ser desde el inicio de un proyecto nuevo. Pero también puede aplicarsele XP a un proyecto ya iniciado que está en crisis. La clave en este caso es analizar la metodología de software que se está empleando e identificar la fase que está incidiendo negativamente en el proyecto y aplicarle XP a este problema en primer lugar. Y así sucesivamente ir aplicando paulatinamente a cada problema identificado. Inicialmente existen dos divisiones en las prácticas [Beck, 2004]: primarias y corolarias. Dícese primarias de aquellas prácticas que pueden aplicarse de forma independiente y corolarias aquellas más complejas que requieren un dominio de las primarias.

También puede clasificarse las prácticas de acuerdo al área de desarrollo en que se aplican [http://www.extremeprogramming.org/rules.html] como puede apreciarse en la siguiente tabla donde se han resaltado en negrita los nombres de las prácticas.

Tabla 5: Las reglas y prácticas de la programación extrema

Fase	Reglas y Prácticas
Planificación	• Escribir **Historias de Usuario** • Crear un calendario de trabajo basado en **La Planificación de liberación de versiones** • Medir la **Velocidad del proyecto** • Dividir el proyecto en **iteraciones** • Iniciar cada iteración con **planeamiento de iteraciones** • **Rotar al personal** • Iniciar cada día con una **reunión parados** • **Corregir XP** cuando se venga abajo
Diseño	• **Simplicidad** • Escoger una **metáfora del sistema** • Usar **tarjetas CRC** para las sesiones de diseño • Crear **soluciones específicas** para reducir los riesgos • Nunca agregue **funcionalidad antes de tiempo** • Aplicar **Refactoreo** siempre y en todo
Codificación	• El cliente debe estar **siempre disponible** • La codificación debe hacerse siempre bajo un **estándar** • Codificar la **prueba de unidad** primero • La producción de código debe hacerse bajo **programación de parejas** • **Integrar el código** con el todo de forma

Fase	Reglas y Prácticas
	secuencial • **Integrar frecuentemente** • Implementar la **propiedad colectiva del código** • Dejar la **optimización** para el final • No trabajar **Tiempo extra**
Pruebas	• Contar con **pruebas de unidad** para todo el código • Todo el código debe superar las **pruebas de unidad** antes de ser liberado. • Crear pruebas cuando se **encuentre un error** • Correr a menudo **tests de aceptación** y publicar los puntajes

El presente trabajo se enfocará en las prácticas primarias según Beck [2004].

A continuación se detallan las prácticas primarias que son aplicables a este proyecto:

3.3.1 Programación en pareja

Esta práctica requiere que dos programadores combinen sus esfuerzos y destrezas en la programación trabajando ambos en una sola

computadora. Dos programadores trabajando en el mismo problema tiene muchas ventajas tanto para la calidad del producto como para los participantes. Puede ser más gratificante para los programadores encontrar apoyo cuando no encuentran solución a un problema, aprender técnicas de los compañeros, compartir logros, vivir el espíritu de equipo, participar en el desarrollo de todo el código. Eso por mencionar algunos puntos que beneficien socialmente a los participantes. Por el lado del producto podría mencinarse que se produce código de mayor calidad, en menos tiempo, y con menos recursos pues se utiliza una estación de trabajo menos.

Hay dos roles principales en esta práctica: el conductor, es la persona que tiene el control del teclado, y el navegador, es la persona que está guiando. Es recomendable que ambas personas roten de papel cada cierto tiempo e incluso que se rote de pareja también. [http://en.wikipedia.org/wiki/Pair_programming].

Después de analizar la programación en pareja, es aplicable al proyecto por la cantidad de integrantes del equipo y porque pone de manifiesto los valores de la comunicación, el coraje y la retroalimentación.

3.3.2 Historias de Usuario

Puede decirse que las historias de usuario tienen la misma finalidad que los casos de uso en la programación, pero hay que dejar claro que no

son lo mismo.

Las historias de usuario son escenarios que el cliente o usuario final escribe en pequeñas tarjetas donde explica con lenguaje común sin tecnicismos una funcionalidad en particular que necesita. Debido al dinamismo de XP esta práctica sustituye lo que en el modelo tradicional constituye el análisis de requerimientos.

Estas historias una vez escritas son ponderadas en tiempo de acuerdo a la complejidad, esta ponderación es utilizada como criterio para tomar decisiones como dividir las historia, combinarlas o extender su alcance. Estas historias se organizan en una pared de fáci acceso para los integrantes del equipo. Como las historias son breves sin detalles de interfaces, ni algoritmos, ni detalles de una tecnología específicas, es necesario que cuando llega la hora de abordar el desarrollo de la historia de usuario, el desarrollador se reunirá con el cliente para recibir una descripción detallada de los requerimientos.

Las historias a diferencia de los “requerimientos” no son algo definitivo ni obligatorio, pues los requerimientos tienden a cambiar durante el desarrollo del proyecto.

Esta técnica permitirá poner en práctica los valores de la comunicación con el cliente, la simplicidad y la retroalimentación.

Es fácilmente aplicable a cualquier situación, no requiere muchos recursos, pero si exige la disponibilidad del cliente. Estas condiciones están presentes en el proyecto y por ello puede aplicarse la práctica.

3.3.3 Ciclo Semanal

Se trata de planificar el trabajo de una semana. Comenzando la semana con una reunión de trabajo que sirve para revisar el progreso y si se alcanzaron las metas de la semana anterior, presentar las historias de usuario que el cliente seleccionó para la semana, y que los miembros del grupo acepten las tareas que implementarán en la semana. La planificación semanal sustenta los valores de comunicación, simplicidad, retroalimentación y también es aplicable al presente proyecto.

3.3.4 Desarrollo Iterativo

El desarrollo iterativo añade agilidad al proceso de desarrollo. Dividira su agenda de desarrollo en cerca de una docena de iteraciones de una a tres semanas de longitud. Mantenga la longitud de la iteración constante a través del proyecto. Este es el corazón de su proyecto. Es esta constante que hace que la medición de progreso y la planificación sean simples y confiables en Programación Extrema.

No planificar las tareas de programación por adelantado. En lugar de ello

tenga una reunión de planificación de iteraciones al principio de cada iteración para planear lo que se hará. La planificación justo a tiempo es una forma fácil de mantenerse a la delantera de los requerimientos de usuario.

También va contra las reglas adelantarse añadiendo funcionalidad que creemos que se necesitará mas adelante y que no esté programado para esta iteración.

Habrá mucho tiempo para implementar esa funcionalidad cuando se convierta en la historia de usuario más importante en el plan de lanzamiento. El plan de lanzamiento es la calendarizacion de todas las versiones que se lanzarán y en que fechas, incluye la selección de cuáles historias de usuario serán incluidas en que versión, la selección se hace en reuniones junto con el cliente. Después de escoger las historias cada una de ellas se traslada a tareas específicas de programación que serán implementadas en cada iteración hasta completar la historia.

Se toma el plazo de la iteración con toda la seriedad. Mida su progreso durante una iteración. Si parece que no terminará todas sus tareas convoque otra reunión de planificación de iteraciones, re calcule, y quita algunas de las tareas.

Concentre su esfuerzo en completar las tareas más importantes a

medida sean escogidas por su cliente, en lugar de tener varias tareas sin terminar escogidas por los desarrolladores.

Puede parecer tonto si sus iteraciones son de solo de una semana de duración hacer un nuevo plan, pero se recompensa al final. Planear cada iteración como si fuera la última permite al desarrollador fijarse un tiempo de entrega puntual del producto. Planteamiento del Problema

3.3.5 Tests de aceptación

Los tests de aceptación son creados a partir de las historias de usuario. Durante una iteración las historias de usuario seleccionadas serán trasladadas en tests de aceptación. El cliente especifica escenarios para probar cuando una historia de usuario ha sido implementada correctamente. Una historia puede tener uno o más tests de aceptación, con el fin de asegurar que la funcionalidad sea correcta.

Los tests de aceptación son tests de sistema al estilo caja negra. Cada test de aceptación representa algún resultado esperado del sistema. Los clientes son responsables de verificar que los tests de aceptación sean correctos y revisar los puntajes de los tests para decidir cuales tests fallados son de alta prioridad. Los tests de aceptación también son usados como tests de regresión previo a una version de producción del software.

Una historia de usuario no está considerada completa hasta que ha pasado su test de aceptación. Esto significa que nuevos tests de aceptación deben ser creados en cada iteración o el equipo de desarrollo no reportará ningún progreso.

Los tests de aceptación deben automatizarse para que puedan ser ejecutados frecuentemente. Es responsabilidad del equipo programar un tiempo destinado para corregir cualquier test que falle.

El nombre inicial de los test de aceptación era test funcional. Fue cambiado porque refleja mejor la intención de uso, la cual es garantizar que los requerimientos de un cliente se hayan alcanzado y el sistema sea aceptable.

4. Desarrollo

Con todo la investigación preliminar efectuada y requerida se procedió a desarrollar el sistema de administración de planilla. Este capítulo resume en esencia los procesos, herramientas y técnicas que se utilizaron en el desarrollo del producto final el software para manejo de planilla de la facultad multidisciplinaria de occidente.

El desarrollo se llevó a cabo en tres etapas:

1. Entrevistas

2. Análisis funcional del software actual

3. Desarrollo y pruebas del software

4.1 Entrevistas:

El levantamiento de requerimientos fue cubierto en su mayoría usando este recurso.

Se realizaron entrevistas al Jefe de Recursos Humanos de la Facultad Multidisciplinaria de Occidente con el objetivo de indagar no sólo cuáles

son los procedimientos que se llevan a cabo en esta unidad para generar las planillas de pago sino también las limitantes y reglas de negocio que deben respetarse y tomarse en cuenta, la relación y dependencia con otros departamentos y con la Facultad Central.

Para examinar el ciclo completo de la generación de la planilla se rastreó con Guillermo Hernandez, técnico encargado en la Subgerencia de Personal de la UES de la generaciòn de planilla, quien explicó y profundizó el proceso de integración y verificación de planillas.

El producto concreto de las entrevistas es la documentación de las políticas de planilla siguiendo el formato descrito en el capítulo I.

4.1.1 Politicas de planilla

A continuación se documenta el proceso para la planilla general.

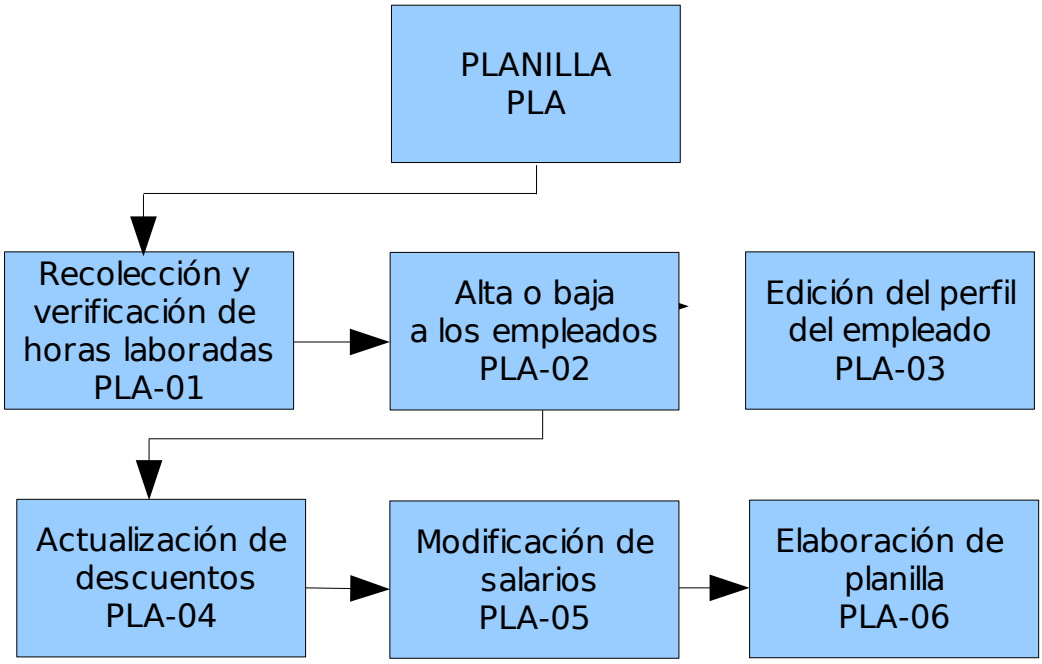


Figura 1: Diagrama de flujo y procedimientos de la planilla general de la Universidad de El Salvador

Política/Procedimiento

No. PLA-01
Pagina: 1 de 2
Fecha: 12/09/07
Suplanta: N/A

Tema: Recolección de horas laboradas

1. PROPÓSITO Y ALCANCE

   Este procedimiento es usado por las Facultades para poder aplicar el descuento por llegadas tardías en los empleados en que aplica y revisar si laboró las horas pactadas.

2. RESPONSABILIDADES

   Encargado de Sistema de cada facultad ESF

3. PROCEDIMIENTOS

   3.1 ESF C**ontabilización de horas laboradas**

   1. El encargado de operar el sistema de planillas de cada Facultad, recibe un archivo proveniente del reloj marcador o recibe las hojas de control de asistencia de la Facultad.
   2. Totaliza las horas laboradas para cada empleado
   3. Totaliza la cantidad de minutos por llegadas tardías para los empleados a los que se les aplica descuento por llegadas tardías

4. Alimenta el sistema de planillas con las horas laboradas para cada empleado en el período
5. Alimenta el sistema de planillas con los minutos de llegadas tardías para cada empleado.
6. Calcula y aplica el descuento por llegadas tardías en la planilla

Política/Procedimiento

No. PLA-02
Pagina: 1 de 2
Fecha: 12/09/07
Suplanta: N/A

Tema: Alta y Bajas de empleados

1. PROPÓSITO Y ALCANCE

   Este procedimiento es usado por el operador del sistema de planilla de cada Facultad para añadir o eliminar empleados del sistema de planillas.

2. RESPONSABILIDADES

   Encargado de Sistema de cada facultad ESF

3. PROCEDIMIENTOS

   3.1 ESF **Recepción de documentación**

   1. El encargado de operar el sistema de planillas recibe de Recursos Humanos la solicitd de añadir o eliminar de la base de datos de empleados de la Facultad. Revisa la documentación pertinente (acuerdos de contratación, acuerdos de nombramiento, fichas de recolección de datos que el empleado completó, documentos del empleado) que esté debidamente autorizada y firmada.
   2. Si falta alguna información la devuelve a Recursos Humanos para su corrección.

3.2 ESF **Actualizar la base de datos de empleados**

1. Ingresar al SIRH-I y accede al menú de Altas del empleado, ingresar información del nuevo empleado tal como: datos propios de la persona, documentos, capacitaciones recibidas, experiencia docente, experiencia laboral, formación académica, grupo familiar, habilidades y destrezas, idiomas que habla. Luego ubicarlo en uno o varios puestos específicos.

2. Impresión de la ficha de empleado para revisar que los cambios se han llevado a cabo en la base de datos del sistema.
3. Archivar los documentos físicos del empleado

Política/Procedimiento

No. PLA-03
Pagina: 1 de 2
Fecha: 12/09/07
Suplanta: N/A

Tema: Modificaciones al perfil del empleado

1. PROPÓSITO Y ALCANCE

   Mantener actualizada la base de datos de los empleados.

2. RESPONSABILIDADES

   Encargado de Sistema de cada facultad ESF

3. PROCEDIMIENTOS

   3.1 ESF **Recepción de datos a modificar**

   1. El encargado recibe del empleado la ficha con la información personal a modificar (estado civil, formación académica, capacitaciones, etc) o recibe del jefe la Facultad la orden del movimiento de empleados en la estructura jerárquica.
   2. Si falta alguna documentación la solicita al jefe de la Facultad

3.2 ESF **Modificación de la base de datos de personal**

3. Previa autorización de las ediciones, el encargado ingresa al módulo de administración de personal y modifica el dato personal del empleado o lo mueve de puesto o lo agrega a un nuevo puesto.
4. Imprime el reporte de actualizaciones y lo entrega al solicitante de la modificación para que revise si se aplicó la edición al perfil del empleado.

Política/Procedimiento

No. PLA-04
Pagina: 1 de 2
Fecha: 12/09/07
Suplanta: N/A

Tema: Modificación en las deducciones del empleado

1. PROPÓSITO Y ALCANCE

   Este procedimiento es usado por las Facultades para actualizar el perfil de los empleados y agregarle al empleado una deducción, eliminar o modificar una existente.

2. RESPONSABILIDADES

   Tesorería de la Facultad Central

   Encargado de Sistema de cada facultad

3. PROCEDIMIENTOS

   3.1 Tesorería a **Recepción de datos para dar de baja y alta las deducciones**

   1. Tesorería recibe de las instituciones financieras, superintendencia, secretaría de la famlilia, AFP las órdenes de descuento o finalización de descuento
   2. Verifica que el descuento efectivamente puede ser aplicado, eliminado o modificado en el perfil del empleado

3.2 Tesorería N**otificación a las facultades de las modificaciones en los descuentos**

3. Genera un listado de los empleados y las modificaciones a los descuentos
4. Enviar listados de modificaciones de descuentos a las Facultades correspondientes

3.3 Facultades **Actualización del perfil de empleado en el sistema de planillas**

1. El encargado de operar el sistema de planillas actualiza las altas y bajas de descuentos de acuerdo al listado recibido de tesorería
2. En caso de que el empleado haya cumplido la edad para pensionarse por vejez pero continuará laborando, cada Facultad modifica el porcentaje de descuento para el empleado.
3. Se imprime una versión preliminar con los cambios que revisa el encargado de sueldos y salarios de la Facultad; si hay algún error éste hace la observación, sino hay error se autoriza la modificación.

Política/Procedimiento

No. PLA-05
Pagina: 1 de 2
Fecha: 12/09/07
Suplanta: N/A

Tema: Edición de salarios

1. PROPÓSITO Y ALCANCE

   Las facultades pueden registrar con la edición de salarios ya sea aumentos de sueldo así como modificaciones temporales en el sueldo de un empleado.

2. RESPONSABILIDADES

   Encargado de Sistema de cada facultad ESF

3. PROCEDIMIENTOS

   3.1 ESF **Recepción de documentos para editar salarios.**

   1. ESF recibe la solicitud debidamente autorizada del jefe de la Facultad de modificar el salario del empleado ya sea generado por un aumento de salario o permiso sin goce de sueldo u otra razón.
   2. Si es un permiso sin goce de sueldo, el encargado de Sueldos y Salarios de la Facultad hace el cálculo del nuevo salario a aplicar.

3. El encargado de Sueldos y Salarios puede solicitar también la edición de un salario originada por la revisión de la planilla preliminar.
4. El operador del sistema de la Facultad edita manualmente el salario del empleado.

Política/Procedimiento

No. PLA-06
Pagina: 1 de 4
Fecha: 12/09/07
Suplanta: N/A

Tema: Elaboración de planilla

1. PROPÓSITO Y ALCANCE

   Este procedimiento es usado por las Facultades para generar la planilla de pagos general

2. RESPONSABILIDADES

Encargado de Sistema de cada facultad	ESF
SubGerencia de Personal	SGP
Tesorería	
Presupuestos	PPTO

3. PROCEDIMIENTOS

   3.1 ESF **Emisión de planilla**

   1. El encargado recolecta datos de los movimientos de empleado para actualizar las estructuras y puestos.
   2. Recolecta información de empleados que han terminado, calcula sus pagos finales y los introduce enel sistema si aun no se le han cancelado con cheques.

3. Se genera la planilla usando el sistema, el cual calcula todas las deducciones pertinentes.
4. Imprimen una versión preliminar que la revisa el encargado de sueldos y salarios de la facultad.
5. Si hay algún error se procede a corregir inmediatamente.
6. Se imprime las planillas junto con las boletas de pago y se llama al personal a firmarlas.

3.2 EMPLEADOS F**irma de planilla**

7. Los empleados se presentan a firmar la planilla y retiran su boleta de pago

3.3 ESF **Depuración de planilla**

8. Los empleados que no se presentaron a firmar son eliminados de la planilla
9. Se generan los archivos digitales e impresiones de la nueva planilla y se envian los digitales a la SubGerencia de Personal y las impresiones a Tesorería y una impresión extra se archiva en la Facultad.

3. 3.4 PPTO **Recepción de planillas de las Facultades**

10. Presupuestos verifica y autoriza que los desembolsos de la planilla estén debidamente documentados. Que las altas de empleados tengan su respectivo contrato, que las deducciones se hayan aplicado, etc.

3.5 SGP **Recepción de planillas de las Facultades**

11. Sub Gerencia de personal consolida los archivos usando el sistema y genera un reporte con todos los devengos de todas las facultades el cual compara contra el reporte de Prespuestos.

12. Si ambos reportes coincidieron se continúa el proceso de lo contrario se procede a buscar las diferencias y corregirlas.

13. SubGerencia de Personal genera un archivo con el detalle de las cuentas de banco de los empleados, monto a depositarles y cuenta de donde se tomará el dinero para el empleado. Este reporte se envía a Tesorería quien se encargará de gestionar el pago con el Banco.

14. SGP genera simultáneamente las planillas del Seguro Social y de las AFP. Genera un archivo con la información de la deducciones del Seguro y la envía a Tesorería para que efectúe los pagos.

15. SGP genera reportes para las instituciones financieras, ministerio de hacienda, cooperativas, bancos, etc usando el Sistema de planillas central y un aplicativo intermedio. Envía estos reportes a Tesorería quien se encarga de realizar los pagos correspondientes.

4.2 Análisis funcional del software actual

En la siguiente etapa se recreó el proceso de generación de planilla con una copia del software SIRHI y se analizó la funcionlidad que éste ofrece. Esta etapa ayudo a establecer los cálculos necesarios para la generación de planilla y además un parámetro de comparación de resultados con el software desarrollado.Por esta razón fue una herramienta de verificación de gran importancia.

Además permitió establecer los puntos de mejora mas significativos que el software a desarrollar debía incluir.

3.3 Desarrollo y pruebas del software

Finalmente la etapa de desarrollo incluyó la codificación y creación del software en sí, así como su puesta en funcionamiento en una red doméstica para la comprobación.

Los software más significativos que se utilizaron en esta etapa fueron:

- El software fue programado en Python. Python es un lenguaje de programación dinámico, orientado a objetos y de alto nivel. También cuenta con la característica de ser multiplataforma y

estar instalado en la mayoría de distribuciones de GNU/Linux. Pero su mayor ventaja es que tanto su intérprete, como su librería estandar son libres.

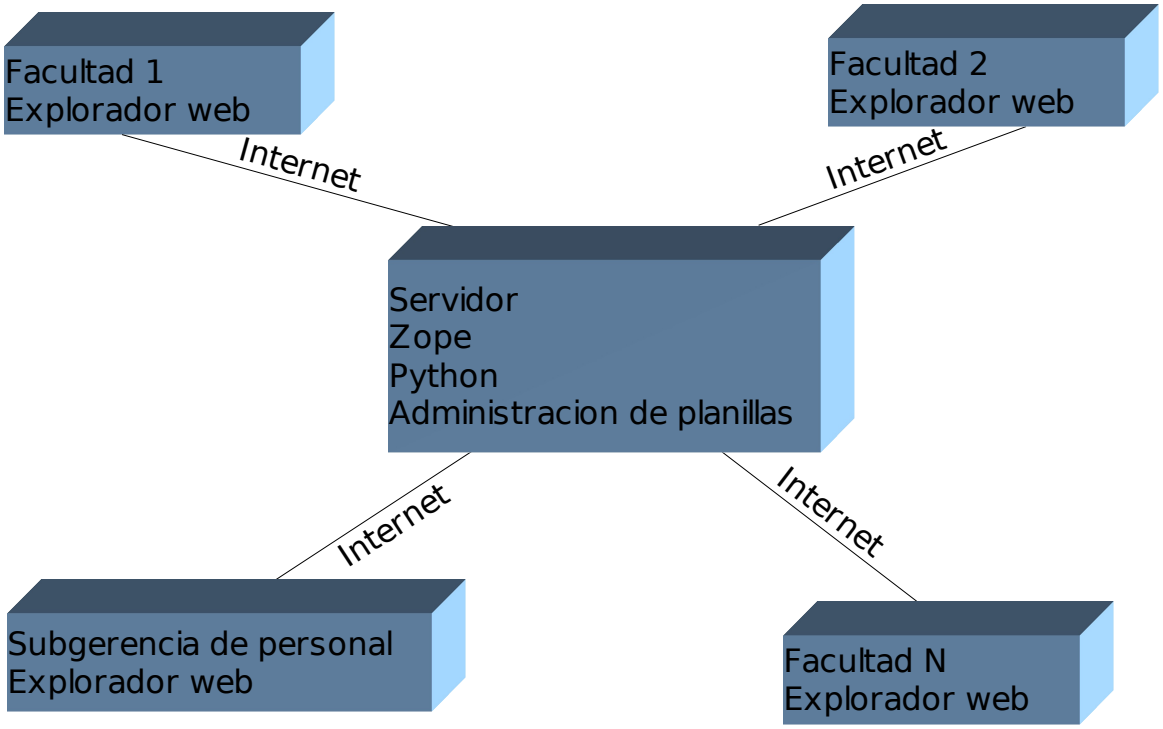
Para ahorrar esfuerzo y con el fin que el software fuera más robusto, se decidió utilizar un framework de desarrollo multiplataforma llamado Zope 3 que proporciona servicios necesarios al construir aplicaciones (por lo general aplicaciones web). Entre sus características se puede mencionar su arquitectura orientada a componentes, donde cada objeto tiene una responsabilidad específica y las aplicaciones son formadas al agrupar diferentes componentes. Zope 3 también cuenta con una base de datos orientada a objetos llamada ZODB (Zope Object Database) que permite almacenar los datos como objetos y evita que el programador deba realizar tareas de mapeo objeto relacional, que son necesarias al utilizar programación orientada a objetos junto con gestores de bases de datos relacionales.

- El código de este proyecto fue administrado utilizando un sistema multiplataforma de control de versiones llamado Subversion. Subversion es software libre y permite conocer las modificaciones que ha tenido un archivo o directorio a través del tiempo. Esto permite recuperar versiones anteriores de los datos, comparar el mismo archivo entre diferentes versiones y saber cómo el código ha evolucionado en el transcurso del proyecto.

- La aplicación desarrollada fue sometida a pruebas de funcionalidad desarrolladas con Selenium IDE. Selenium IDE es una extensión para el navegador web Firefox/Iceweasel que permite registrar los clicks, la digitación y otras acciones que un usuario realiza con una aplicación web. Dichas acciones pueden ser posteriormente reproducidas, lo que permite crear pruebas funcionales de regresión.

3.4 Plan de implementación

3.4.1 Diagrama de distribución



3.4.2 Requerimientos de instalación

El sistema aquí desarrollado requiere que esté instalado la última version más estable de python y sobre esta base debe estar instalado zope 3. El paquete del sistema aquí desarrollado y distribuido en el cd debe colocarse en la ruta de búsqueda de python path.

5. Conclusiones y Recomendaciones

5.1 Conclusiones

La presente investigación comprobó que el software desarrollado por el Ministerio de Hacienda SIRH en foxpro 2.6 se encuentra obsoleto principalmente porque fue desarrollado bajo los paradigmas de programación que dominaban en su momento y por eso adolece de las limitantes tecnológicas de la época en que se desarrolló.

A pesar que el Ministerio de Hacienda ha migrado su software hacia nuevas plataformas, la Universidad de El Salvador ha rehusado actualizar el sistema ya que estima que la nueva versión irrespetaría su autonomía.

Al analizar el sistema actual pudo apreciarse que el uso contínuo de un software de naturaleza genérica afecta, entre otras cosas, la calidad de los datos almacenados dando una impresión de confusión y descuido.

La evolución de los procedimientos de generación de planillas se ha visto afectada ya que éstos tienen que ajustarse a las limitantes del software de administración de planilla, cuando lo correcto es que el software esté al servicio de los procedimientos.

Las metodologías ágiles de desarrollo, a diferencia de las tradicionales, proporcionan la flexibilidad para que el equipo de desarrollo tome aquellas prácticas, valores y principios que mejor se apeguen a un proyecto y aplicarlas de la manera que más le convenga de acuerdo a su realidad. Por ejemplo debido a la dinámica de trabajo y situaciones que afrontó el grupo, no se pudo realizar otras prácticas ágiles como la programación en parejas o el desarrollo en iteraciones.

Hasta este proyecto, Python y Zope 3 no habían sido utilizados para desarrollar trabajos de graduación en la FMO. Python proporcionó la ventaja de ser un lenguaje dinámico permitiendo realizar tareas complejas en muchas menos líneas de código que las requeridas en un lenguaje estático. La arquitectura de componentes no es un concepto nuevo de Zope 3, sin embargo su dependencia en los conceptos fundamentales de interfaces, adaptadores y utilidades permitieron que la mayoría de componentes del sistema puedieron ser personalizados y extendidos fácilmente.

El uso de herramientas de automatización de pruebas, como el testrunner de Zope 3 o Selenium IDE garantizó que durante el desarrollo del software la fallas fueran encontradas y corregidas rápidamente. Para cada característica que se agregó al software se crearon pruebas que garantizaron que la modificación al código no afectaba negativamente al resto del sistema.

5.2 Recomendaciones

A partir de los problemas enfrentados en el desarrollo de una alternativa de reemplazo de este proyecto surgen las siguientes recomendaciones:

- Debido a la existencia en el mercado de una nueva versión de SIRH que probablemente ponga en riesgo la privacidad se recomienda seleccionar un software de administracion de planillas que garantice la privacidad y confidencialidad de la información así como la independencia, ya sea desarrollado internamente o adquirido comercialmente.

- Debido a que el sistema maneja datos de carácter personal se recomienda elaborar una política o reglamento de protección de datos que garantice el uso correcto de los datos y la privacidad de la base de datos de información personal.

- Elaborar la documentación de los procesos y políticas de planilla a nivel institucional y mantenerla actualizada porque esto ayudará a unificar y uniformar los procesos en todas las facultades y departamentos en que se ejecutan y además permitirá a los encargados identificar puntos de control o puntos débiles que puedan mejorarse.

- Aplicar reingeniería a los procedimientos de generación de planillas para ofrecer una solución integral que incluya procesos administratvos más eficientes e interconexión con los sistemas existentes que requieran o brinden información de recursos humanos y con los sistemas externos como administradoras de pensiones, Instituto Salvadoreño del Seguro Social, bancos, Ministerio de Hacienda, etc.

- Realizar una auditoría de datos con el fin de revelar errores, inconsistencias, redundancias y datos incompletos almacenados en el sistema de planillas con el fin de corregirlos, validarlos y normalizarlos.

- Elaborar manuales de procedimiento para el proceso de generación de planilla que garantice la calidad de la información en la entrada, gestión y salida de los datos, divulgarlos y capacitar al personal encargado de las tareas de generación de planillas.

- El software producto de los trabajos de graduación en la FMO debería depender exclusivamente en software libre. Esto permitiría a los estudiantes seleccionar y conseguir las herramientas por su propia cuenta y no tener restricciones en cuanto a la forma de uso ni a la distribución de las mismas, aprovechando así la ventaja social propia del software libre.

6. Glosario

- **AFP**: Administradora de Fondo de Pensiones
- **Base de Datos Relacional**: Conjunto de dos o más tablas estructuradas en registros y campos que se vinculan entre sí
- **CPU**: Central Processing Unit (Unidad Central de Procesamiento)
- **Dirección IP**: Número que identifica a una interfaz de un dispositivo dentro de una red que utilice el protocolo de Internet
- **Dominio de Internet**: Nombre de equipo que es más fácil de recordar que una dirección IP
- **DOS**: Disk Operating System (Sistema Operativo de Disco)
- **FMO**: Facultad Multidisciplinaria de Occidente
- **FoxPro**: FoxBASE Professional. Lenguaje de programación que integra un gestor de base de datos para el sistema operativo DOS
- **GHz**: GigaHertz
- **GNU/Linux**: Sistema operativo que utiliza el kernel Linux en conjunto con aplicaciones creadas por el proyecto GNU
- **Hardware**: Los componentes físicos de un sistema computacional, incluyendo cualquier equipo periférico tales como impresor, módems y ratón
- **ISSS**: Instituto Salvadoreño del Seguro Social
- **Kernel**: Núcleo, parte fundamental de un sistema operativo para gestión de recursos

- **MB**: Megabyte
- **MHz**: Megahertz
- **PKUNZIP**: Herramienta de descompresión de archivos creada por PKWARE
- **PKZIP**: Herramienta de compresión de archivos creada por PKWARE
- **PostgreSQL**: Motor de base de datos relacional libre
- **Python**: Lenguaje de programación dinámico interpretado
- **RAM**: Random Access Memory (Memoria de Acceso Aleatorio)
- **RRHH**: Unidad de Recursos Humanos de la Facultad Multidisciplinaria de Occidente
- **SARH**: Sistema de Administración de Recursos Humanos
- **SATA**: Serial Advanced Technology Attachment
- **SDRAM**: Synchronous Dynamic Random Access Memory (Memoria Dinámica de Acceso Síncrono Aleatorio)
- **SIRH-C**: Sistema de Información de Recursos Humanos Central
- **SIRH-I**: Sistema de Información de Recursos Humanos Institucional
- **SIRH**: Sistema de Información de Recursos Humanos
- **SQL**: Standard Query Language (Lenguaje de Consulta Estándar)
- **UES**: Universidad de El Salvador
- **ZODB**: Zope Object Database (Base de Datos de Objetos de Zope)
- **Zope**: Servidor de aplicaciones escrito en Python

7. Referencias Bibliográficas

Beck, Kent ***Extreme Programming Explained: Embrace Change.*** United States of America: Addison Wesley Professional, 2004. 224 p.

Bragg, Steven M. ***Essentials of payroll management and accounting.*** United States of America: John Wiley & Sons, Inc, 2003. 299 p.

Banco Mundial (2004) ***El Salvador Public Expenditure Review*** **Report** No.32856-SV, preparado por el Central America Department, LAC Region. Fecha de consulta 25 de abril de 2007 en http://siteresources.worldbank.org/INTELSALVADOR/Resources/El_Salvador_PER_2004.pdf

DL 677 ***Código de Familia.***

DL 927 ***Ley del Sistema de Ahorro para Pensiones***

DL 1263 ***Ley del seguro social.***

DL 2940 ***Reglamento de la ley de impuesto sobre la renta***

Hernandez, Guillermo. **Entrevista Personal**. Universidad de El Salvador, 12/09/2007

Instituto Salvadoreño del Seguro Social (2005) ***Suministro, Desarrollo, Instalación e Implementación de un Sistema de Información Administrativo Financiero para el ISSS (SAFISSS)***. *Fecha de consulta 25* de abril de 2007 en http://www.isss.gob.sv/uaci/uploaded/LPG0352005.pdf

Ministerio de Agricultura y Ganadería (2005) ***Normativa Principal Relacionada con la Gestión Financiera***. Fecha de consulta 25 de abril de 2007 en http://intranet.mag.gob.sv/asp/finanzas/normativainterna/docs/Lista_normativa_agosto2005.pdf

Ministerio de Hacienda (1997) ***Manual del Sistema de Información de Recursos Humanos***. San Salvador

Procuraduría General de la República (2002) ***Reglamento Interno de Trabajo de la Procuraduría General de la República***. Fecha de consulta 25 de abril de 2007 en http://www.csj.gob.sv/leyes.nsf/ef438004d40bd5dd862564520073ab15/69cb8c31f486d33006256ba0005ef40a

Sandoval, José Jesús (2004) ***Los casos de los ministerios de educación y de economía en El Salvador.*** Fecha de consulta 25 de

abril de 2007 en http://unpan1.un.org/intradoc/groups/public/documents/CLAD/UNPAN001172.pdf

Solorio, C. ***Administración de la Seguridad Social.*** Ginebra: OIT, 2001

Wells, J. Donnovan ***The rules and Practices of Extreme Programming.*** Fecha de consulta: 25 de abril de 2007 en http://www.extremeprogramming.org/rules.html

Wikipedia. ***Pair Programming.*** Fecha de consulta: 25 de abril de 2007 en http://en.wikipedia.org/wiki/Pair_programming

8. Anexos

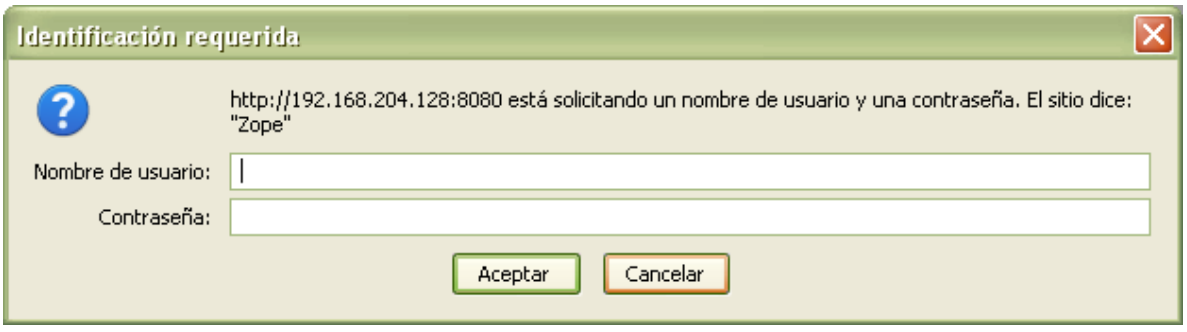
8.1 Guia del usuario

8.1.1 Introducción

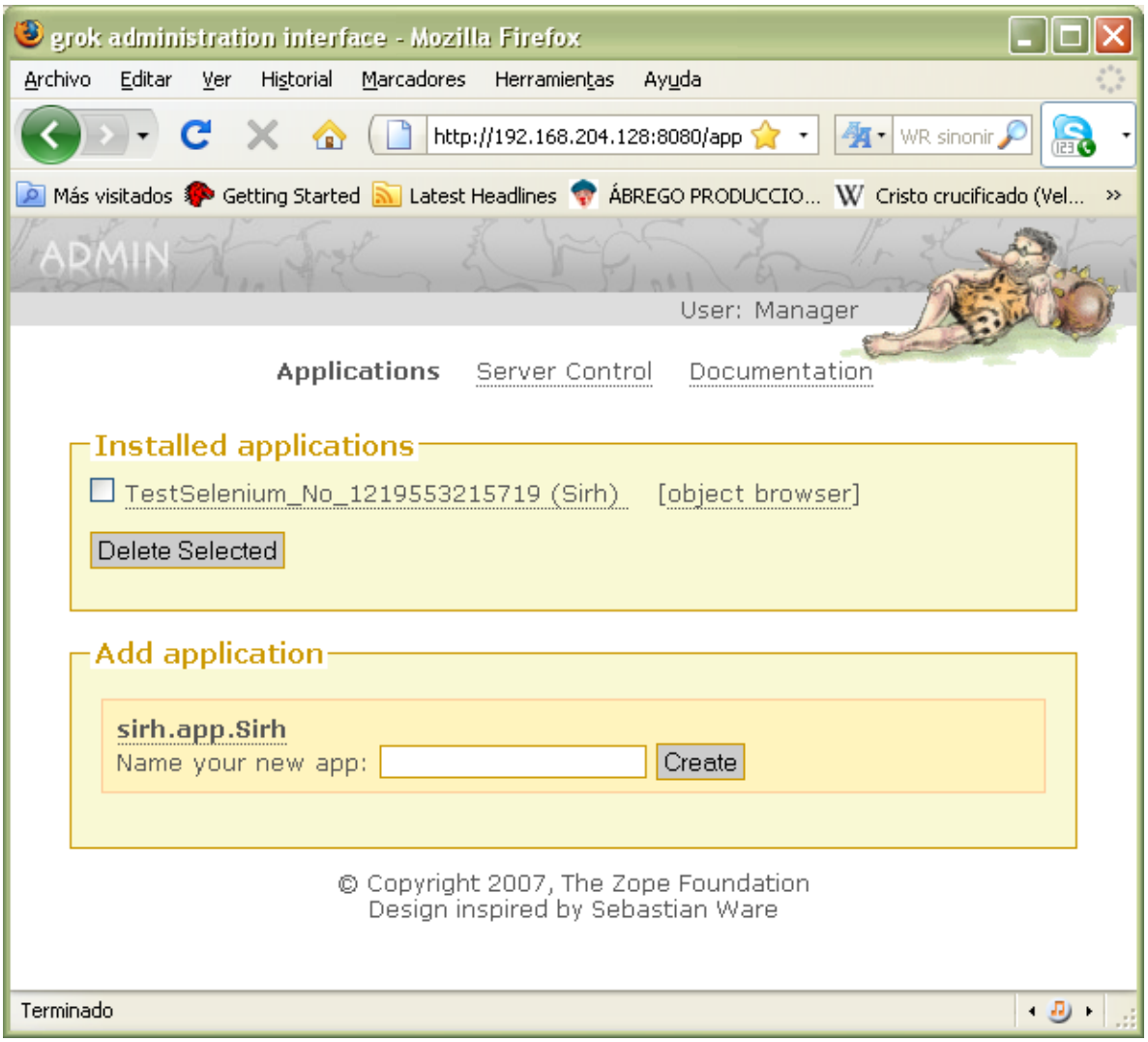
El sistema de administración de planillas es un sistema que almacena en una base central una conjunto de “aplicaciones” con el fin de mantener la independencia de cada facultad o unidad pero sin la necesidad de realizar un proceso de consolidación.

8.1.2 Preparación de entorno

Ingresar a la aplicación para preparar el entorno es una tarea administrativa que está protegida y solo puede realizarla un usuario con privilegios. El primer paso es ingresar a la aplicación e introducir usuario y contraseña, en caso de no ingresarla el sistema no permitrá el acceso a la administración.

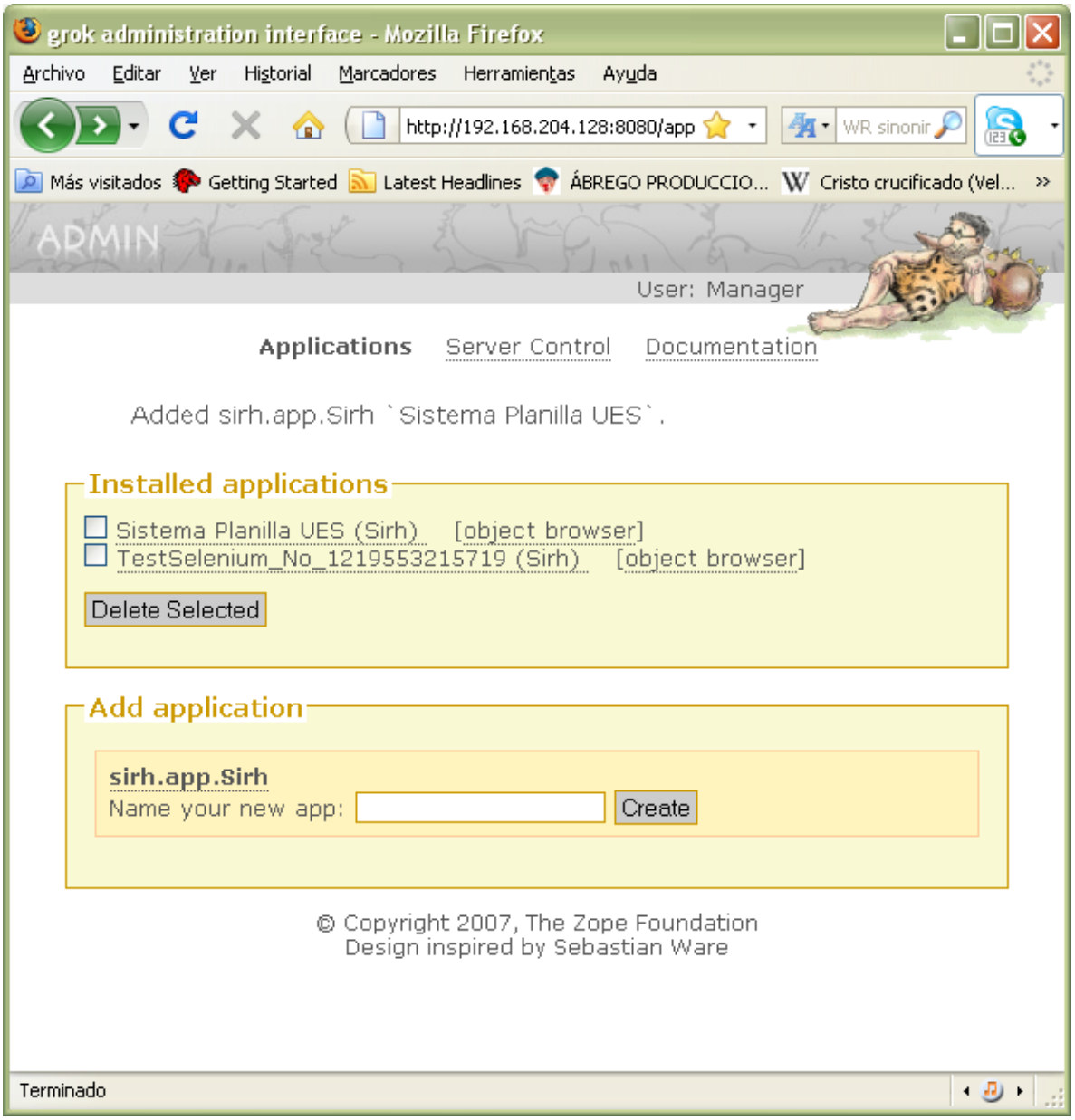


Una vez ingresada la constraseña el usuario accede al administrador de aplicaciones que muestra las aplicaciones creadas y permite crear aplicaciones nuevas “vacías”.

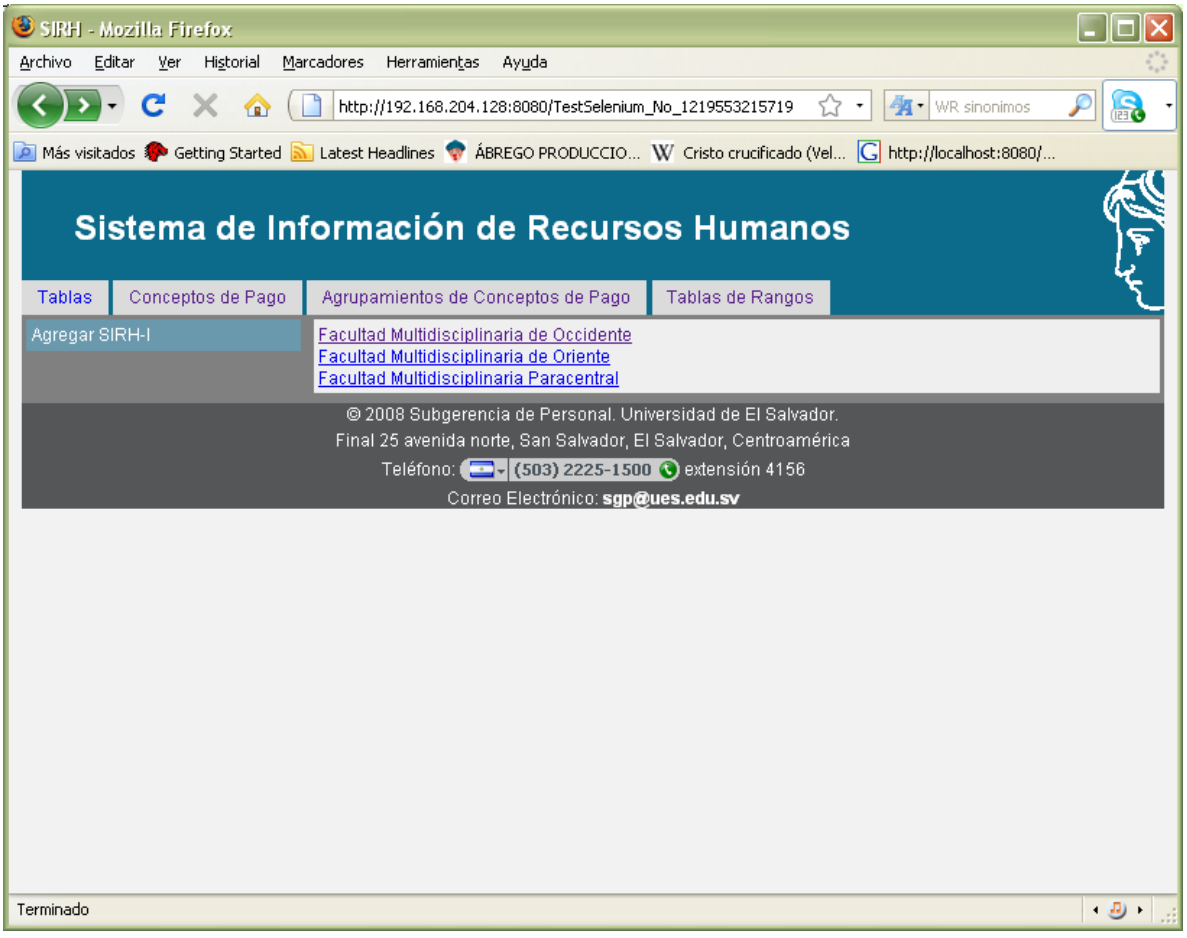


Para crear una aplicación el usuario debe ingresar el nombre de la unidad por ejemplo: Sistema de planillas UES en la caja de texto Name

your new app y hacer clic en create. En la siguiente imagen puede verse que se han creado 2 aplicaciones una llamada Sistema Planilla UES



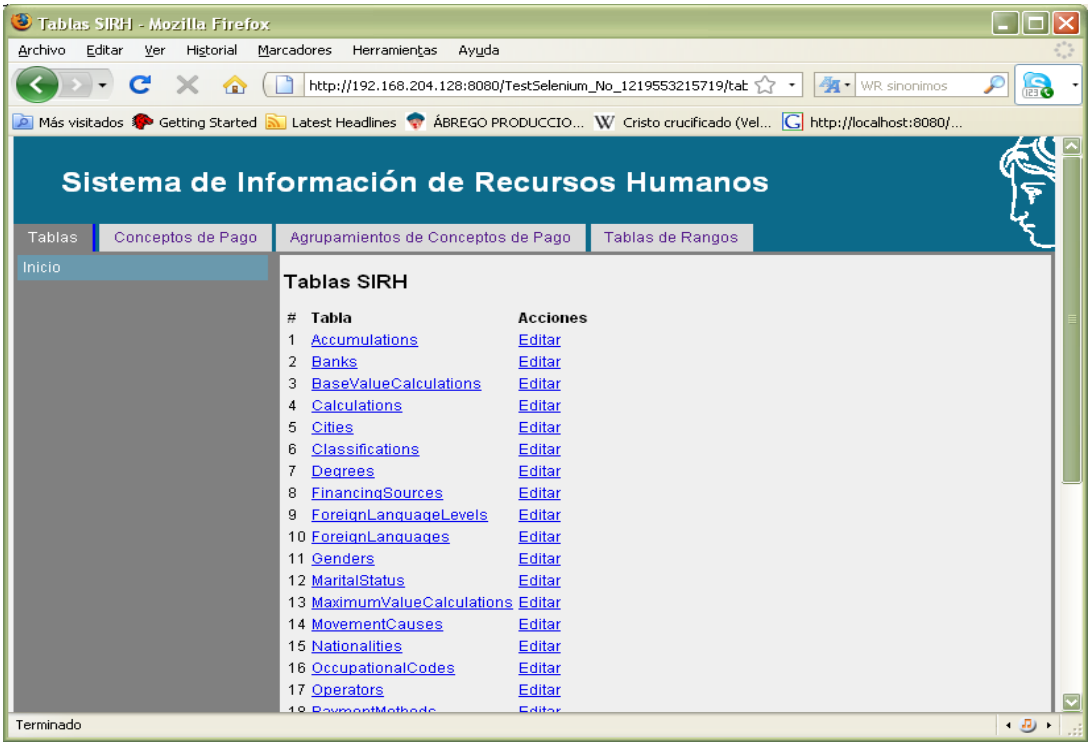
Cada una de las aplicaciones es un repositorio vacío, listo para usarse que respeta la misma estructura. Al ingresar a una aplicación se nos permite administrar aquellas opciones que son generales para todas las facultades. Así como una lista de las facultades creadas. Cada una de ellas es un Sistema de Información de Recursos Humanos Institucional. En el ejemplo se han creado SIRH-I para Facultad Multidisciplinaria de Occidente, Facultad Multidisciplinaria de Oriente y Facultad Multidisciplinaria Paracentral.



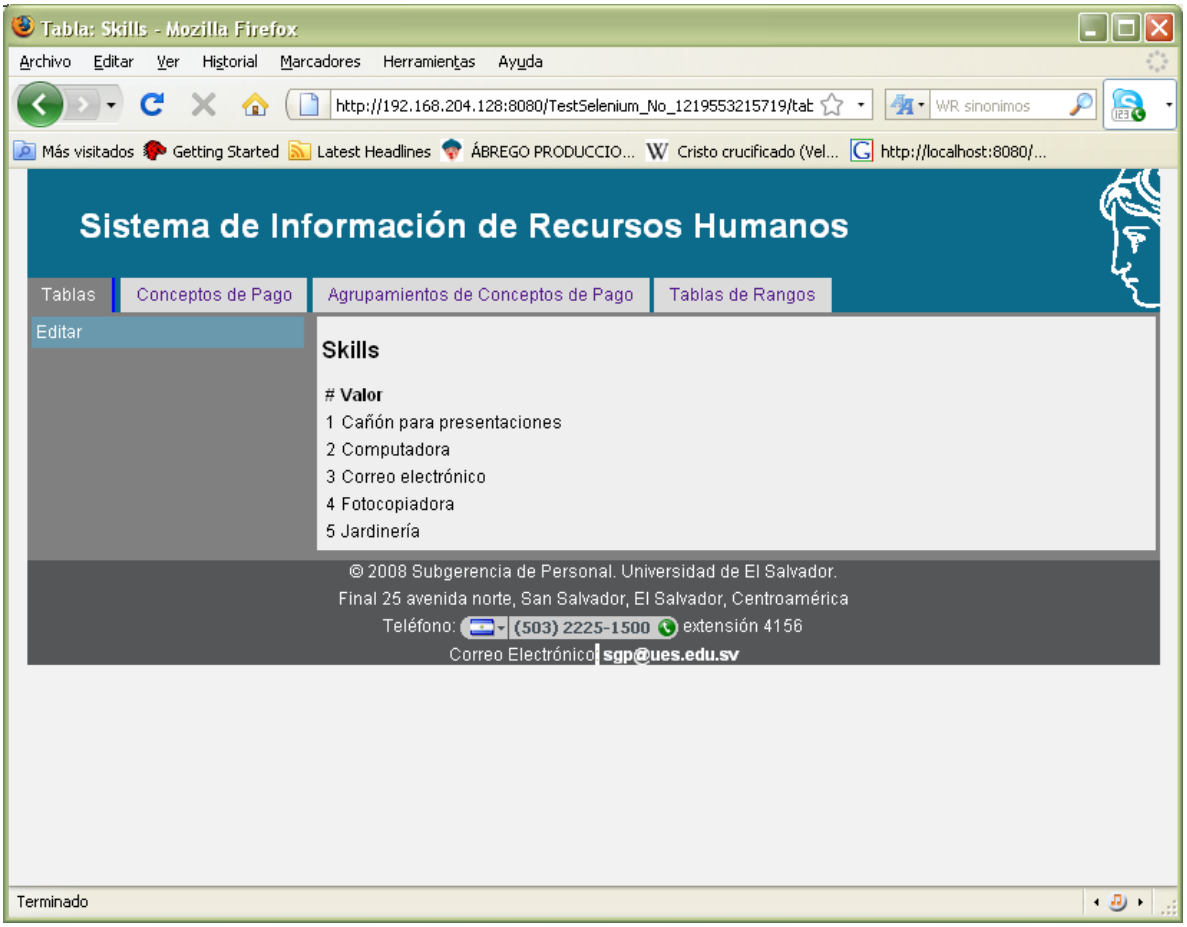
El sistema está diseñado de tal manera que en la parte superior se encuentra un menú de navegación con las secciones que pueden accederse y a la izquierda un menú con las acciones que pueden realizarse en la sección en que el usuario se encuentra.

A continuación se revisan las secciones de administración comunes para los SIRH-I

Tablas: Presenta un listado de todos los vocabularios que el sistema utiliza y cuyas definiciones pueden modificarse.

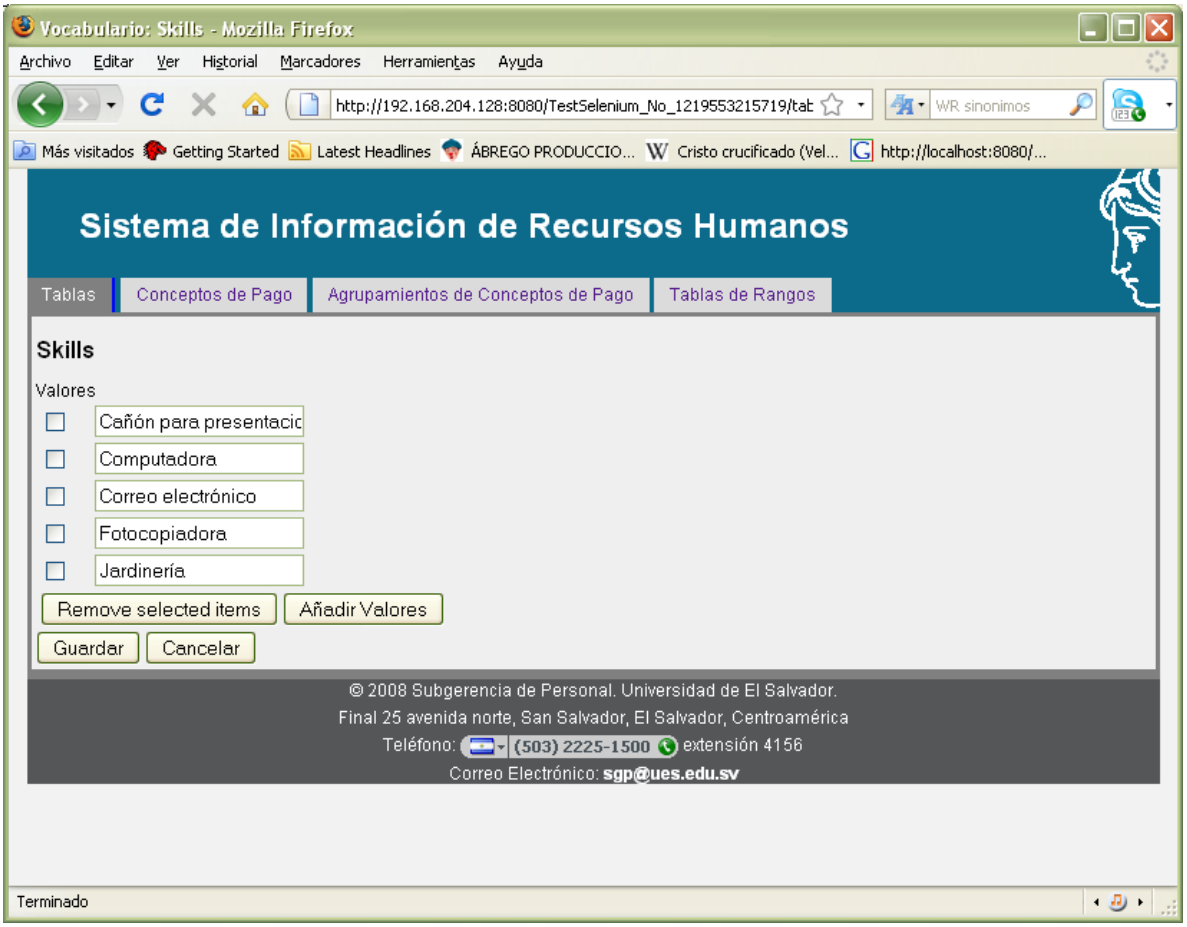


Acá se encuentran las tablas con la lista de bancos, de ciudades, de nacionalidades, de habilidades, etc entre otros que son usadas dentro del sistema. El administrador podría enriquecer por ejemplo la tabla de habilidades (skills) haciendo clic en el nombre de la tabla podrá acceder a los valores almacenados en ella. Las habilidades son aquellas que un empleado puede poseer en su perfil de empleado.

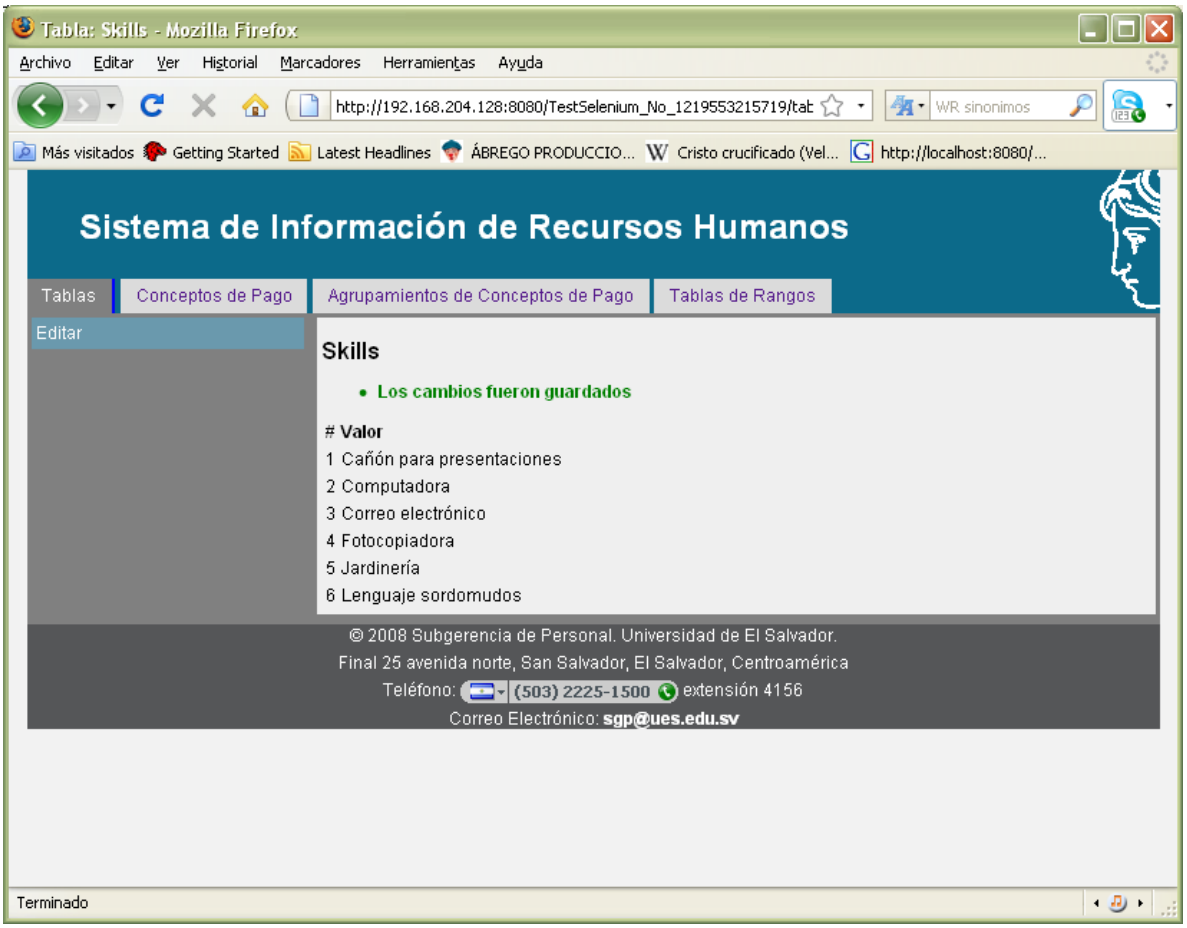


Por ejemplo queremos registrar una habilidad nueva el lenguaje para sordomudos. En el menú de la izquierda hacer clic en el link editar veremos un formulario que permite modificar los valores existentes, eliminar y/o agregar nuevos valores todo en un mismo formulario.

Nuevamente todas las acciones de editar responden a este mismo formato.



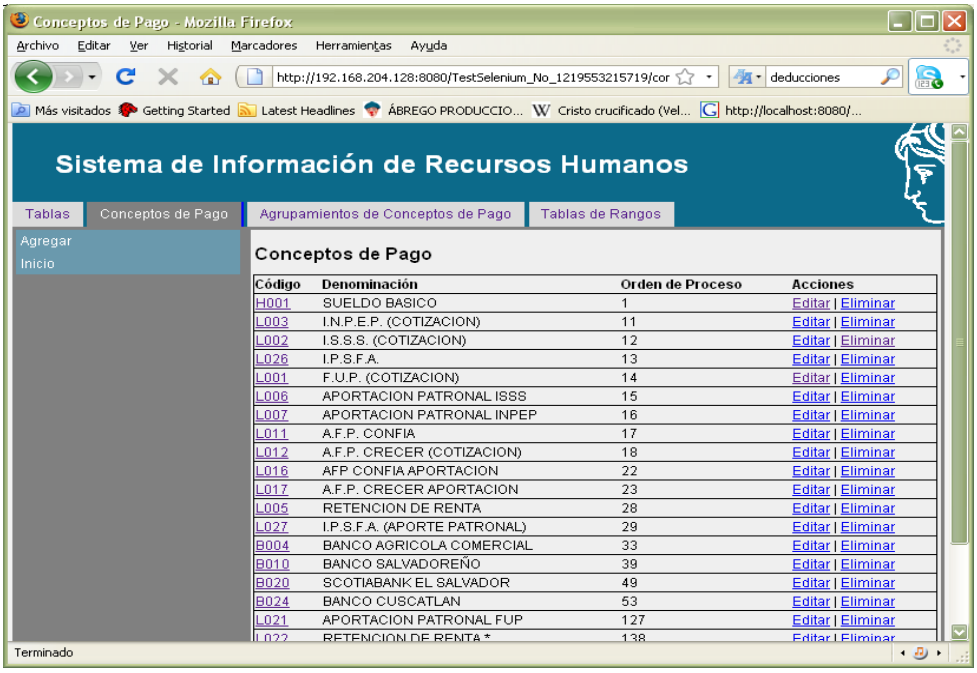
Una vez que se haya terminado la edición hacer clic en guardar o cancelar dependiendo si el usuario desea hacer permanentes o no los cambios.



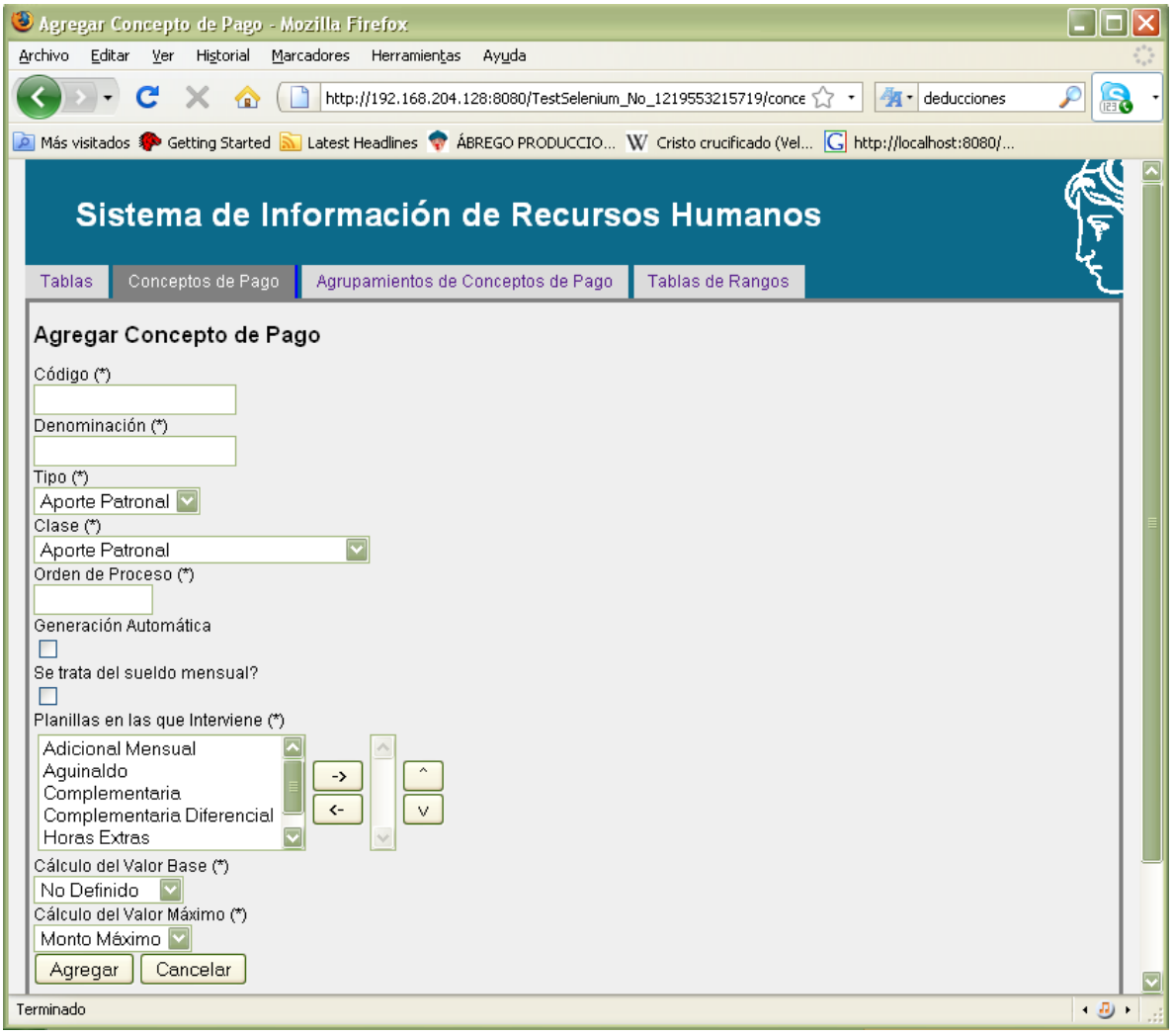
Unmensaje al inicio de la tabla mostrará un mensaje inidicándonos que la actualización se realizó con éxito. Todos los mensajes de error, confirmación que el sistema envía al usuario se hacen de esta forma también.

Conceptos de pago: El monto final que se paga a una persona está conformada por una serie de aportaciones y deducciones a las que se llamará conceptos de pago. Ejemplo de aportaciones son: el salario, ingreso por horas extra, aguinaldos, bonificaciones, etc. Ejemplo de deducciones son: impuestos, seguridad social, aportaciones al sistema de previsiones, descuentos por préstamos.

Los conceptos de pago son una plantilla que modela la lógica de estas aportaciones y deducciones que pueden ser aplicadas a todos los empleados (como el salario) o solo a ciertos empleados (como los préstamos bancarios). Este es el momento de modelar los conceptos de pago que los encargados de cada planilla de cada facultad tendrán a su disposición.



Inicialmente el sistema despliega la tabla de los conceptos de pago definidos. A continuación se explica a detalle como se conforma un concepto de pago.



Código y denominación son identificadores del concepto. El concepto puede clasificarse en tres Tipos: Haber, descuento y aporte patronal que especifica si el concepto debe sumarse, restarse o simplemente

calcularse a la hora de calcular la planilla.

Clase: permite agrupar los conceptos para su totalización y exhibicion de haberes y descuentos. Pueden ser Haber sujeto a retenciones de ley, Haber no sujeto a retenciones de ley, Retención de ley, Otras retenciones y Aportes patronales.

Orden de proceso: Es un número que indica el orden a seguir a la hora de calcular la planilla de un empleado. Tomar en cuenta que los haberes deben tener un orden de procesamiento menor al de los descuentos y aportes patronales.

Generación automática: significa que este concepto siempre será calculado para todos los empleados por ejemplo el sueldo mensual, las retenciones de ley, bonificaciones masivas, etc. En caso de que sea un concepto como descuento por dias no trabajados en que se aplica solo a unos trabajadores se debe dejar sin chequear esta opción

Se trata del sueldo mensual: debe especificarse además que este concepto se refiere al sueldo mensual o no.

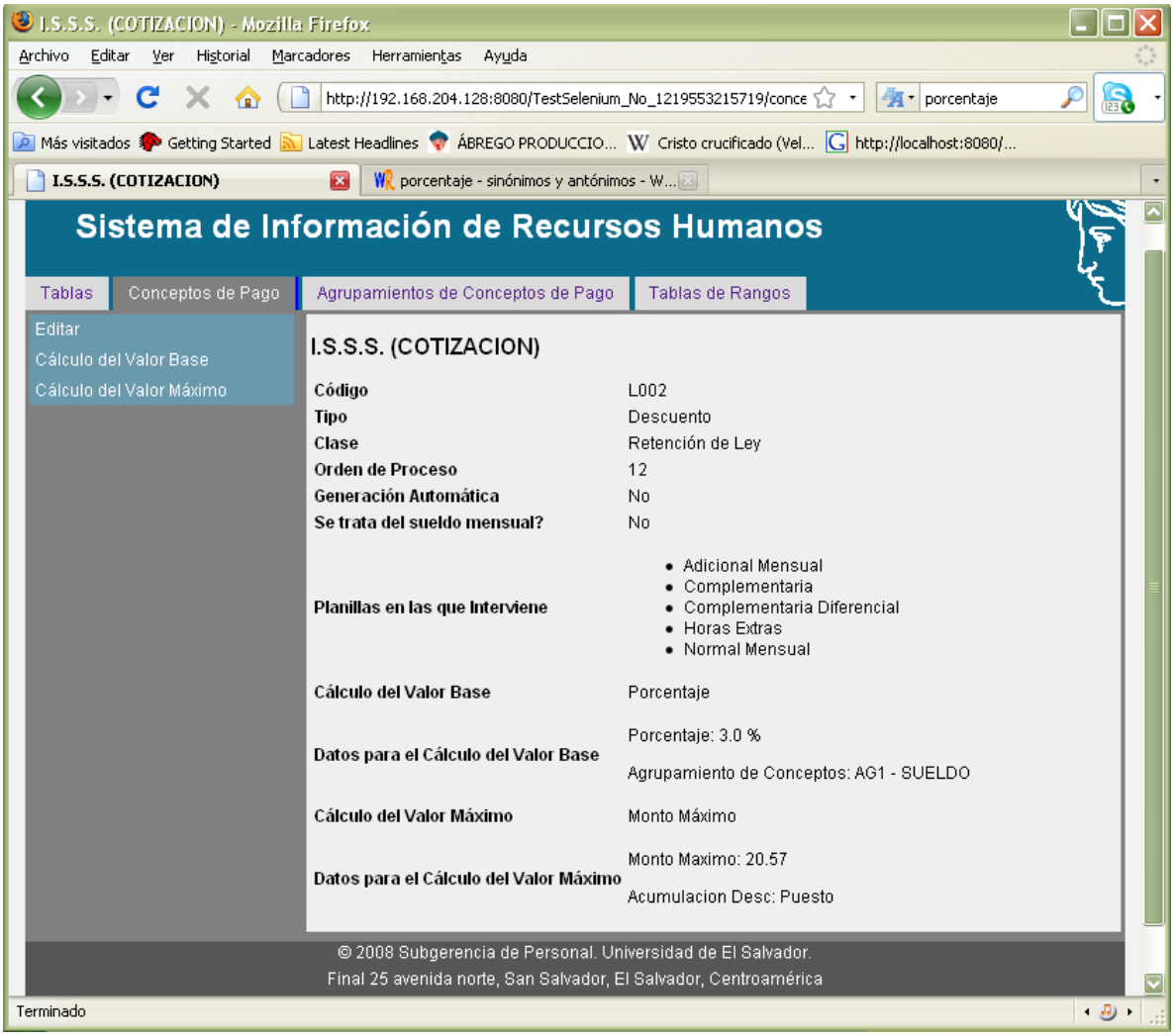
Planillas en las que interviene: Debido a que pueden existir diversos tipos de planilla aquí se especifica en que planillas se considerará este concepto. Por ejemplo en una planilla de aguinaldos el concepto de descuento por seguridad social no debe aplicarse, mientras que el

concepto de retención de renta si debe intervenir.

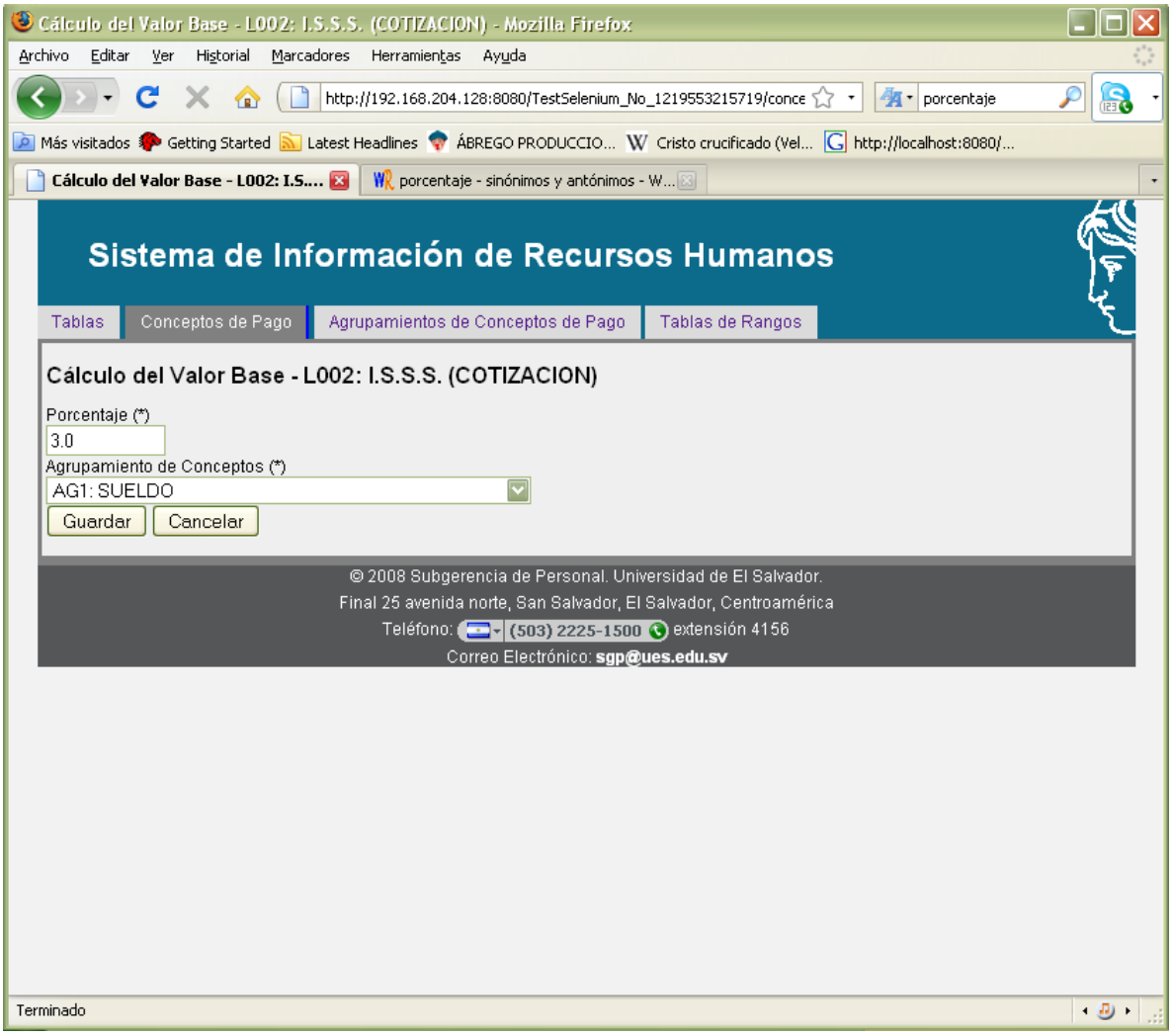
Cálculo del valor base: Acá se define la forma en que se calcula el valor monetario de el concepto. Basicamente son tres: porcentaje, tabla o valor unitario. Porcentaje se refiere a la proporción sobre la suma de lo liquidado en un grupo de conceptos, Tabla permite seleccionar una tabla de rangos también sobre el valor de otro grupo de conceptos y finalmente valor unitario permite definir un valor fijo para el concepto.

Cálculo del valor máximo: En ocasiones el valor de un concepto no puede exceder un límite. Acá se define cómo se calcula ese límite si aplica esta restricción, en caso contrario debe selecconarse No Definido. Conceptos como descuento por seguridad social tienen un límite por ley.

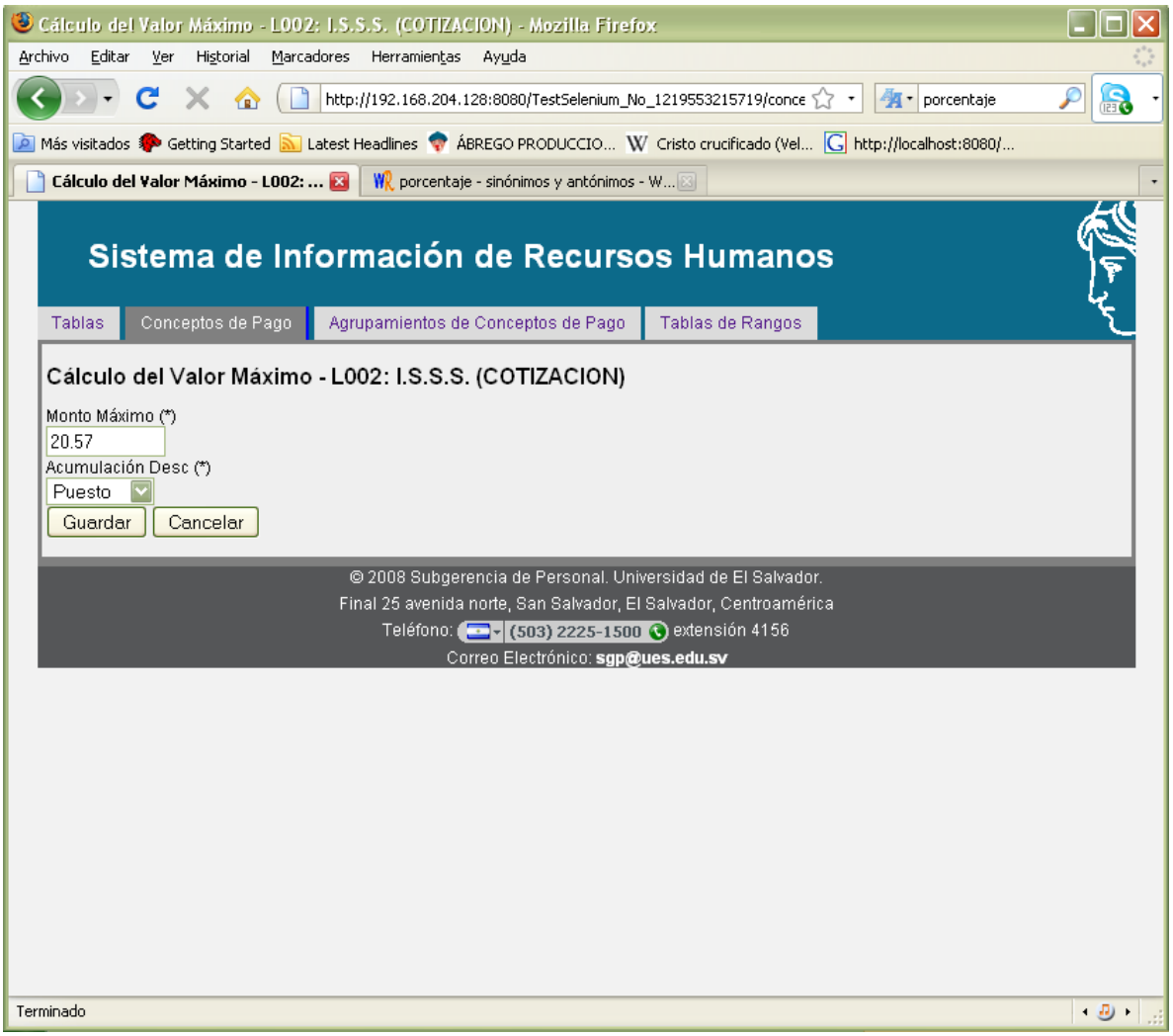
Una vez definido el concepto se procede a especificar el cálculo del valor base y máximo si aplica. Por ejemplo el concepto I.S.S.S. (cotización)



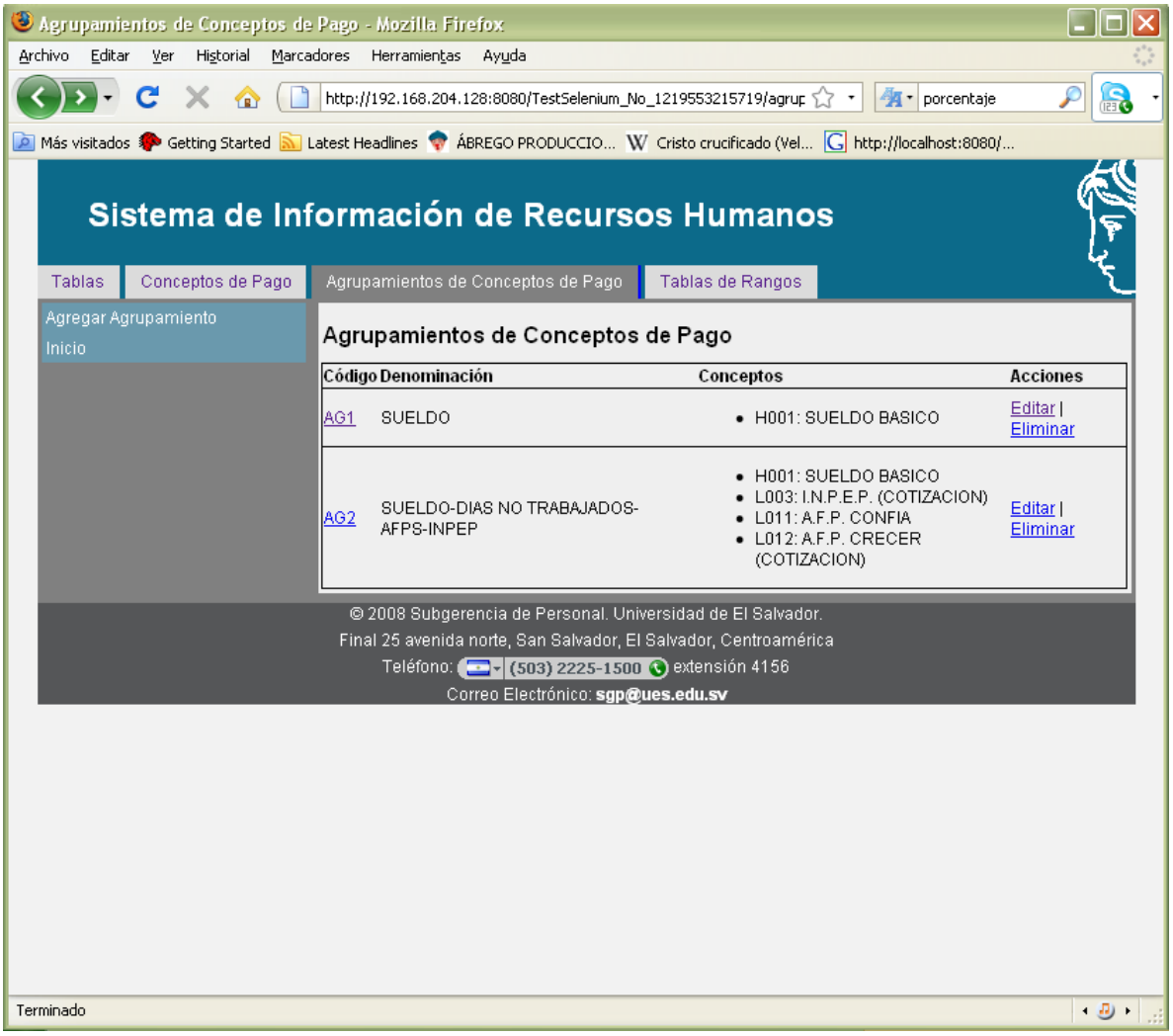
Al hacer clic en la acción cálculo del valor base permite definirsele un valor porcentual sobre el total de un grupo de conceptos. Para el ejemplo, de acuerdo con la ley, se descontará al empleado un 3% de su salario



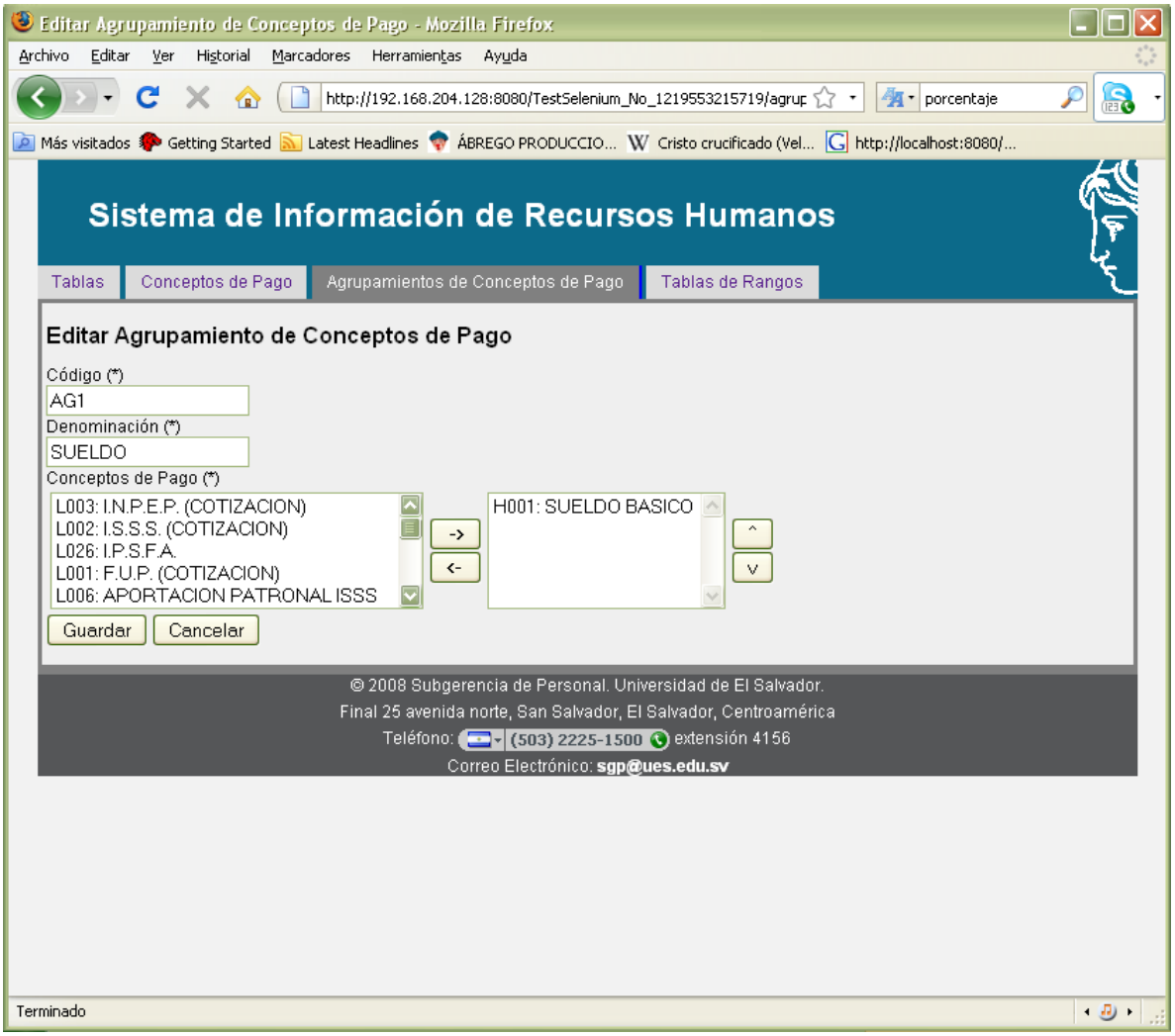
Si se hace clic la acción cálculo del valor máximo podrá especificarse que el descuento nunca podrá exceder de 20.57 de acuerdo con la ley de seguridad social.



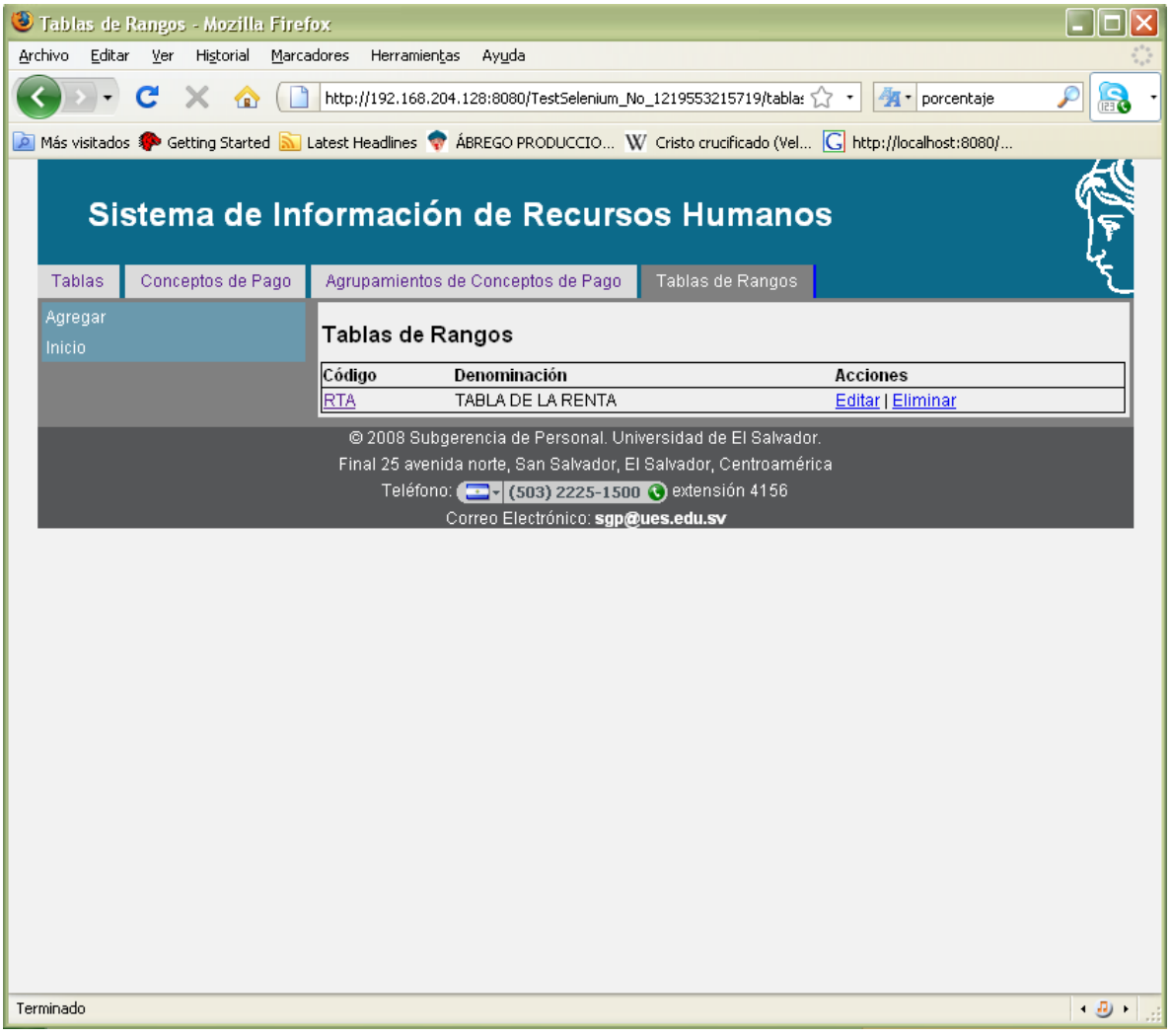
Agrupamientos de conceptos de pago: Se deben definir todos los conceptos que deben agruparse a la hora de calcularse. Se utiliza para facilitar el cálculo de conceptos que dependen del totali de varios conceptos.



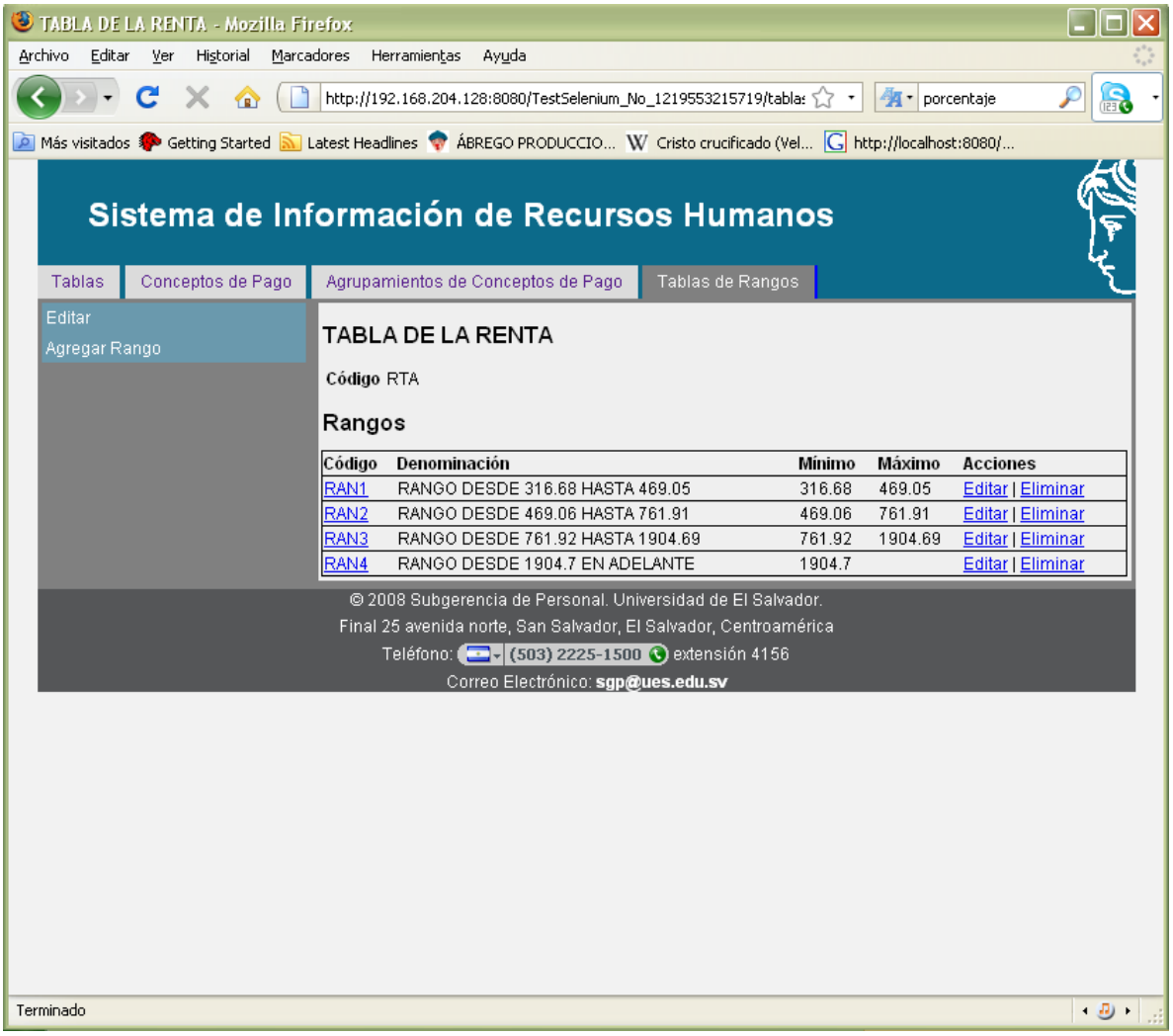
El usuario puede editar el nombre de los conceptos, eliminar y crear nuevos. Asi como modificar la lista de conceptos q integran el grupo, como puede verse en la siguente imagen.



Tablas de Rangos: Existen conceptos cuyo algoritmo de cálculo se relaciona con una tabla de valores, tal es el caso de la renta, que evalúa el total del salario en varios rangos y dependiendo del rango en que cae se aplicará una fórmula definida en el rango. Aunque aquí pueden definirse más tablas para efectos de ejemplo se materializa la tabla de renta.

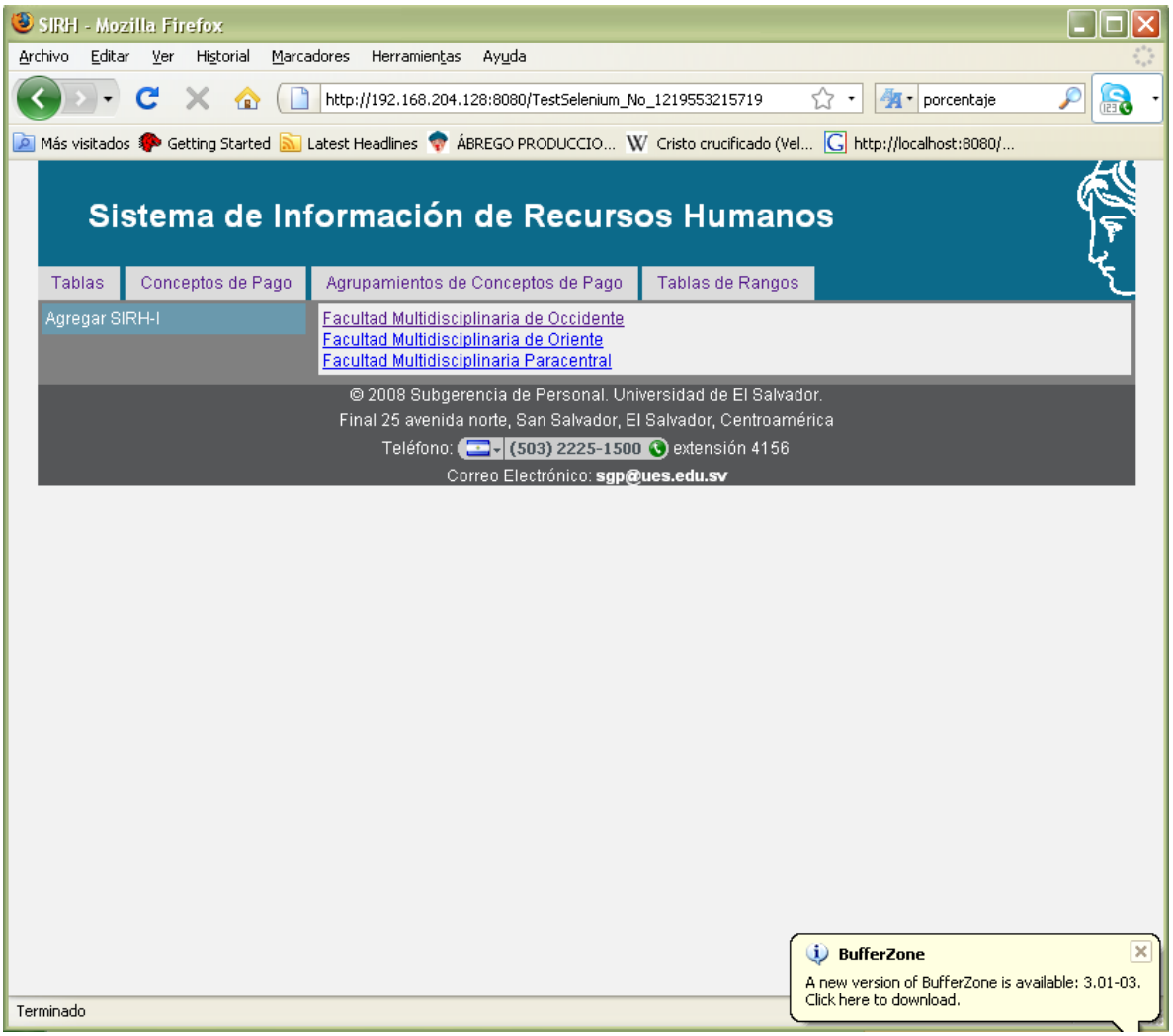


En primer término al ingresar a la tabla veremos los rangos que la conforman definidas por un valor mínmio y máximo:



Además dentro de cada rango se especifican los parámetros de cálculo

Ahora se han definido los parámetros globales y se procederá a explicar cada una de las secciones de los Sirh institucionales. Para ello volvemos a la página de inicio:



y selecionamos una facultad para el ejemplo Facultad Multidisciplinaria de Occidente.

8.1.3 Sistema de información de recursos humanos

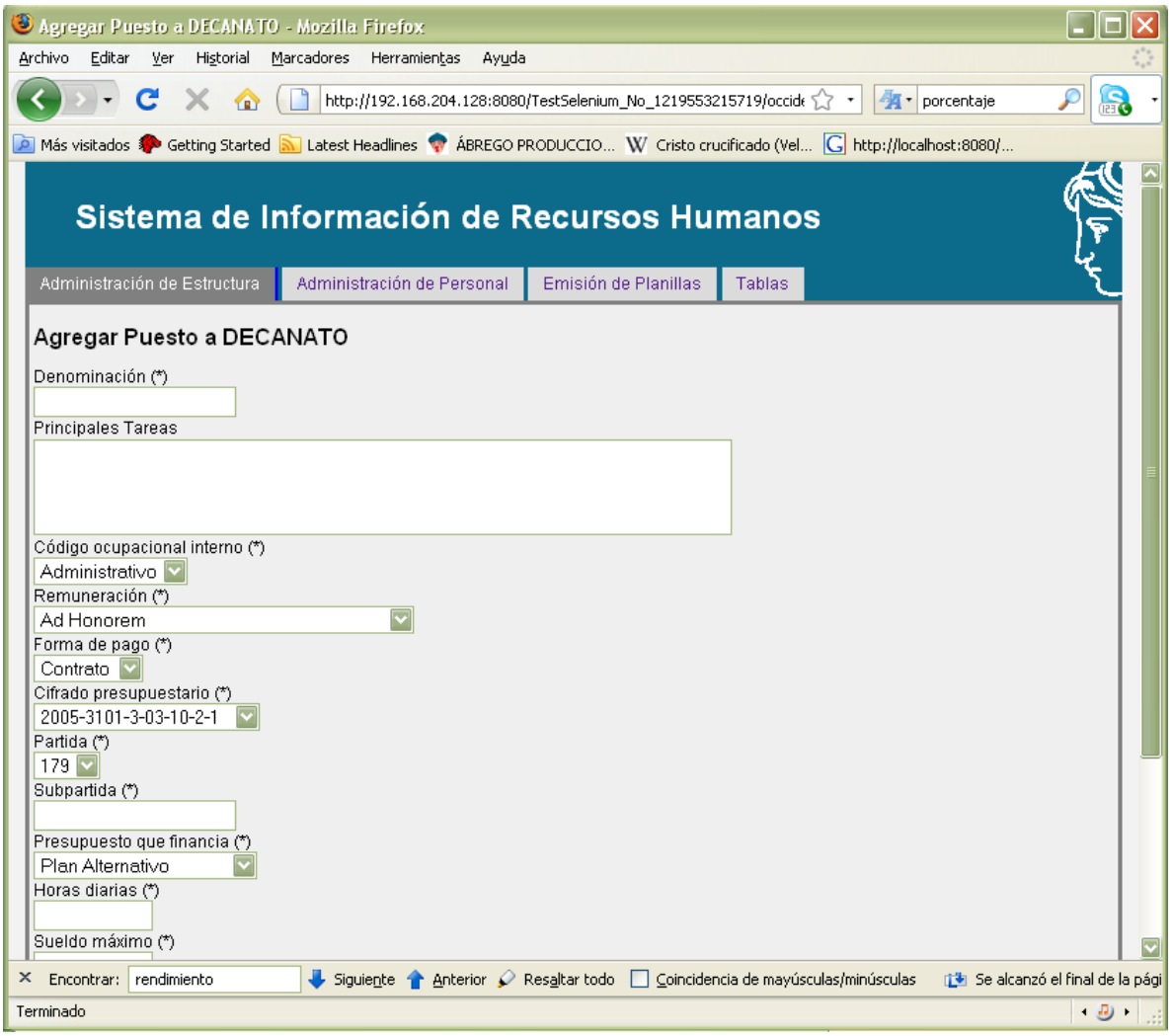
Dentro de la Facultad Multidisciplinaria de Occidente hay 4 secciones las cuales están en todas las facultades: Administración de estructura, Administración de personal, Emisión de planillas y tablas.

Tablas: Permite ingresar las partidas de trabajo y cifrados presupuestarios usados para los puestos de trabajo.

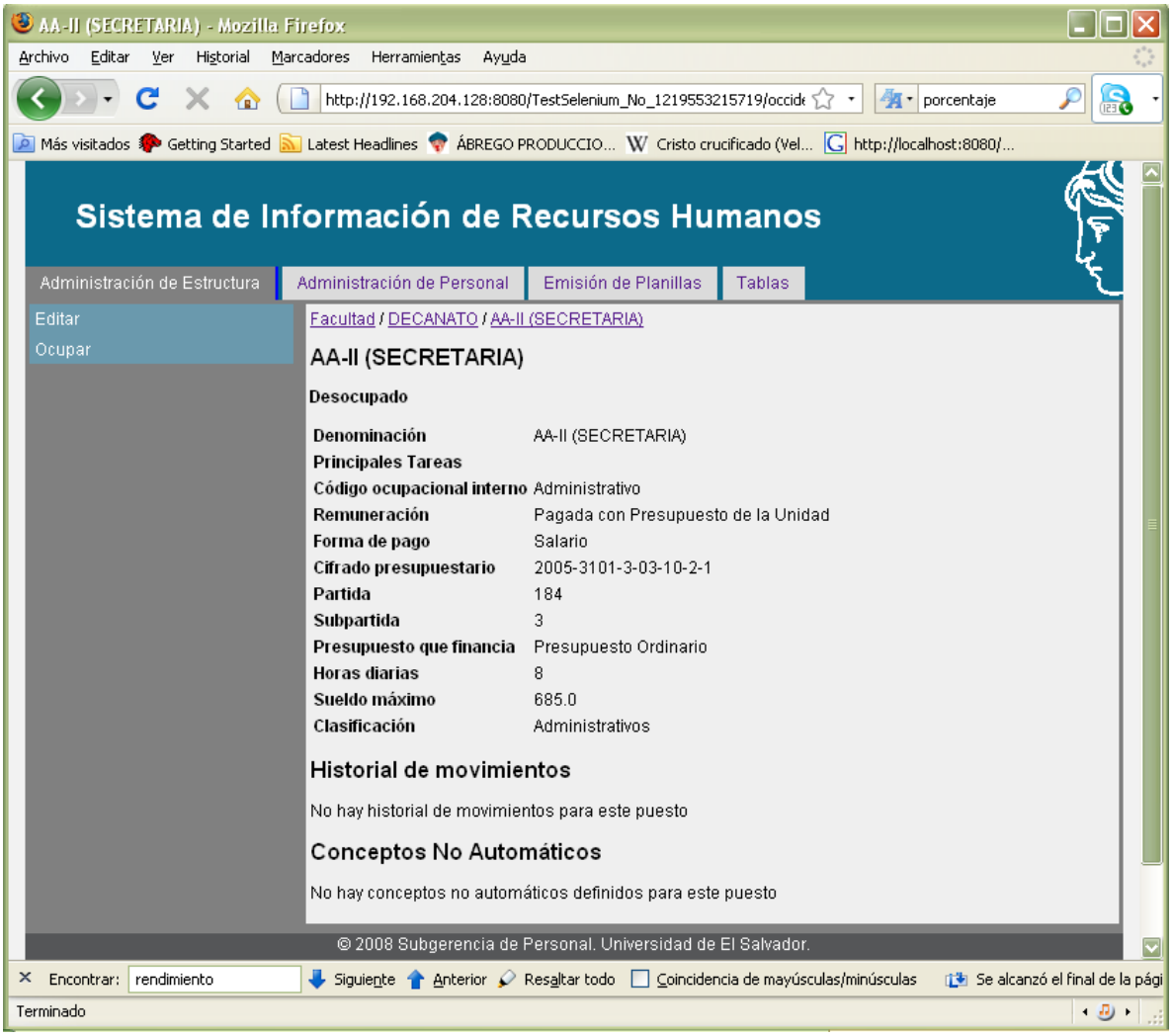
Administración de estructura: permite la administración de la jerarquía organizativa en unidades y puestos de trabajo.

Permite agregar subunidades y puestos así como obtener un reporte de inventario de unidad (ver menún de acciones a la izquierda)

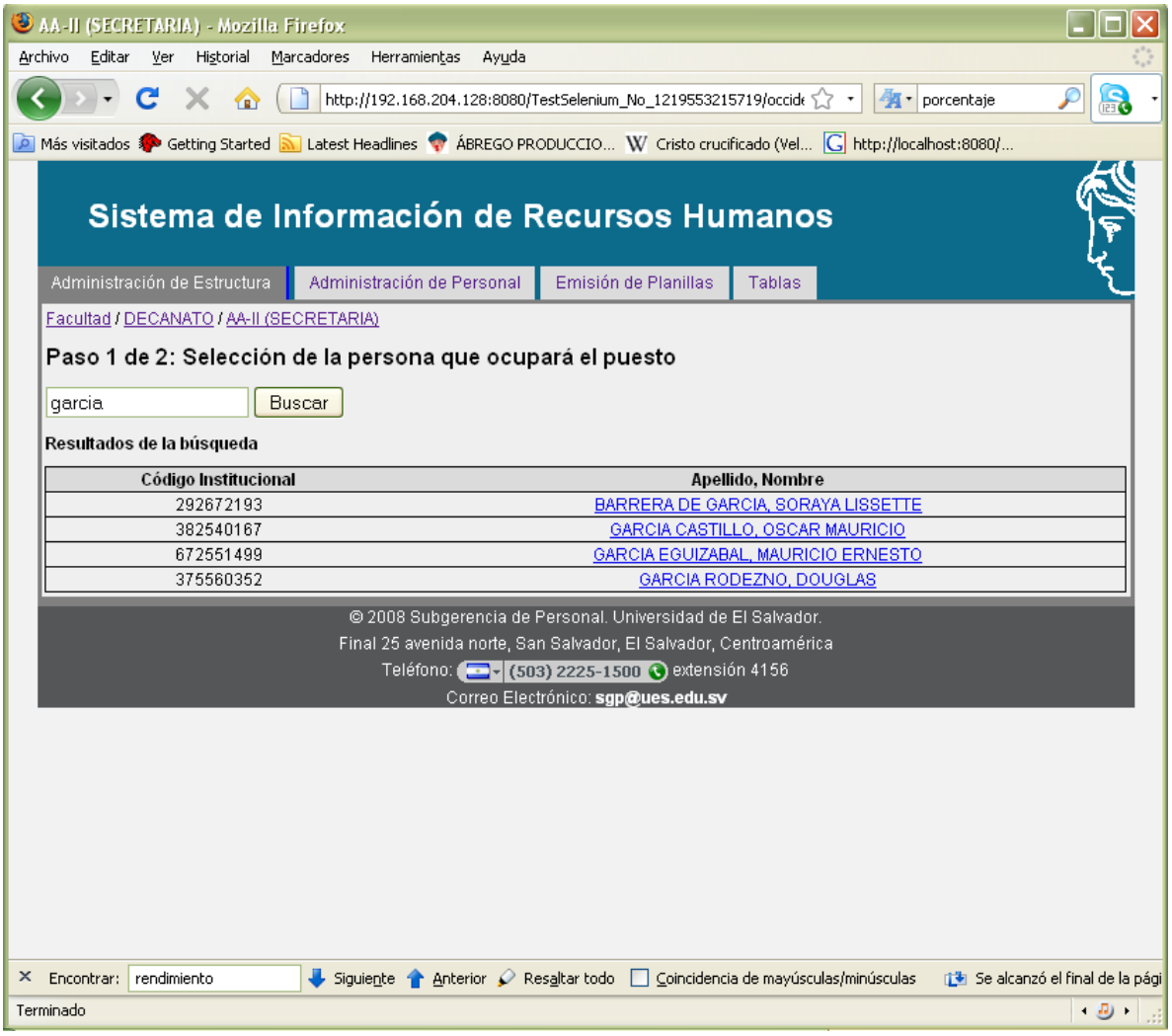
Para agregar puesto solo debe navegar hasta la unidad deseada y clic en el enlace agregar puesto que lo llevará al formulario para llenar los datos del nuevo puesto


Agregar Puesto a DECANATO - Mozilla Firefox
Archivo Editar Ver Historial Marcadores Herramientas Ayuda
http://192.168.204.128:8080/TestSelenium_No_1219553215719/occid
porcentaje
Más visitados Getting Started Latest Headlines ÁBREGO PRODUCCIO... Cristo crucificado (Vel... http://localhost:8080/...
Sistema de Información de Recursos Humanos
Administración de Estructura
Administración de Personal
Emisión de Planillas
Tablas
Agregar Puesto a DECANATO
Denominación (*)
Principales Tareas
Código ocupacional interno (*)
Administrativo
Remuneración (*)
Ad Honorem
Forma de pago (*)
Contrato
Cifrado presupuestario (*)
2005-3101-3-03-10-2-1
Partida (*)
179
Subpartida (*)
Presupuesto que financia (*)
Plan Alternativo
Horas diarias (*)
Sueldo máximo (*)
Encontrar: rendimiento
Siguiente
Anterior
Resaltar todo
Coincidencia de mayúsculas/minúsculas
Se alcanzó el final de la pági
Terminado

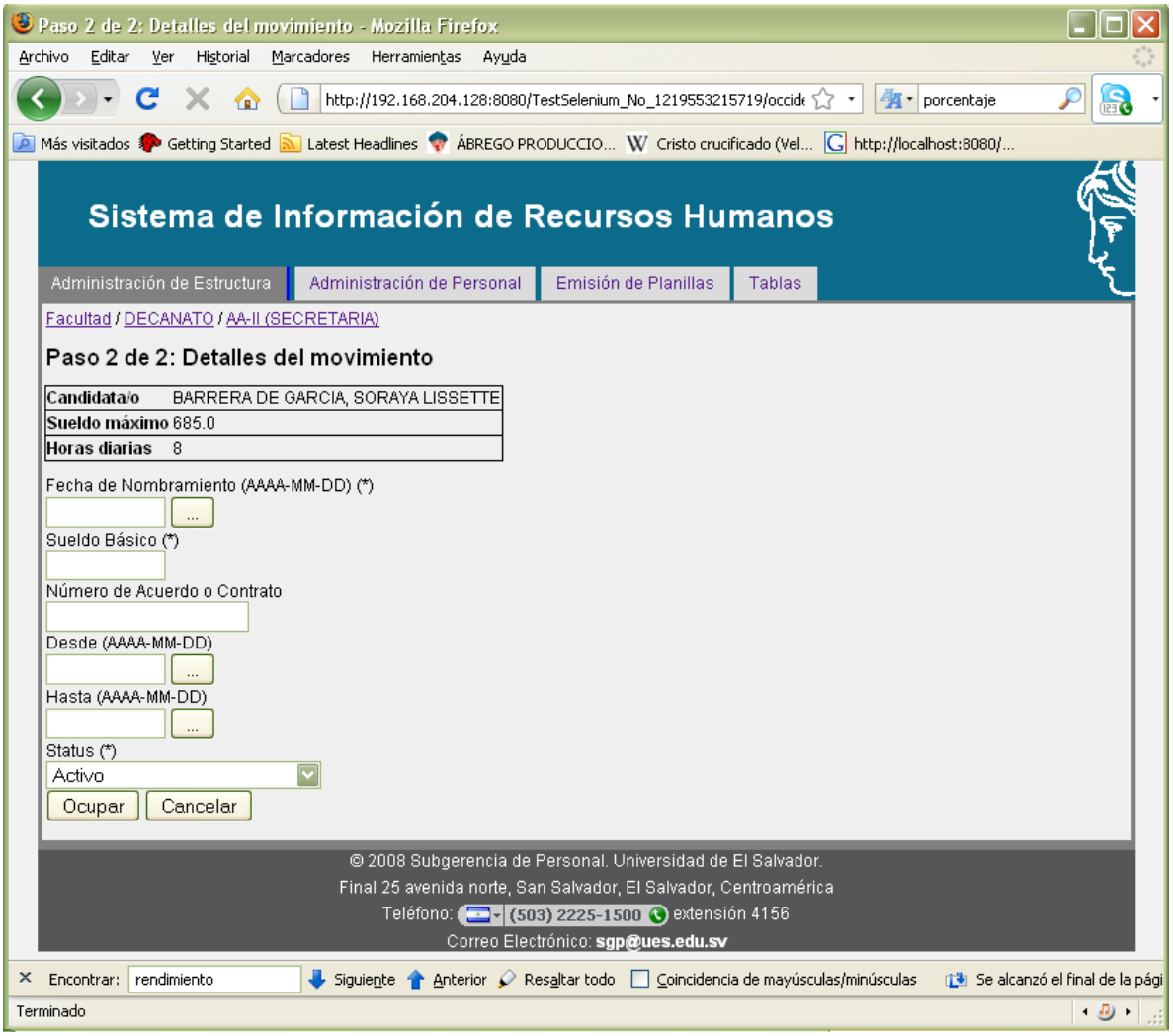
Una vez agregados los puestos estos pueden eliminarse o editarse y ser ocupados por un empleado. Para esto último ingresamos al puesto



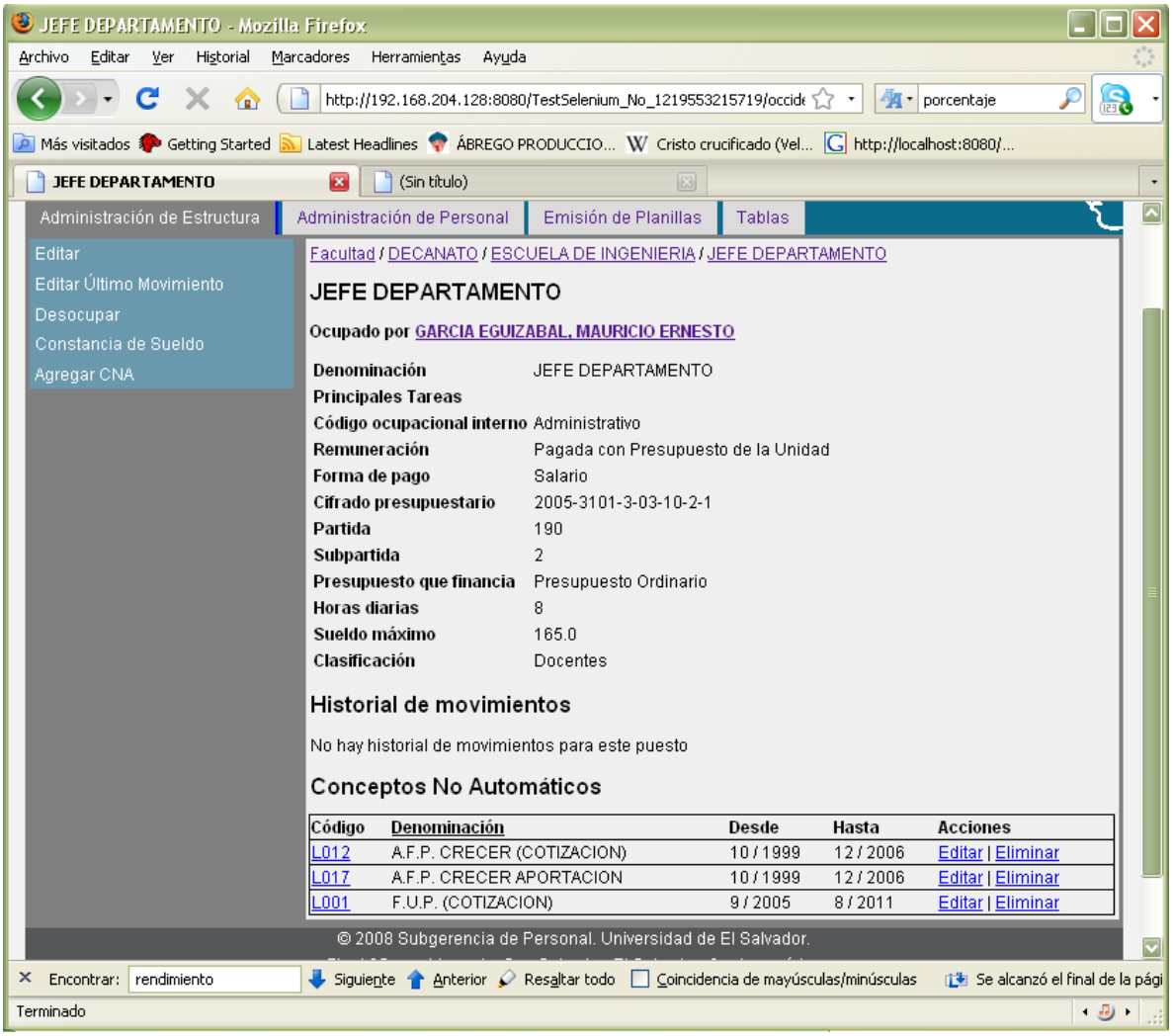
Y usamos el menu de acciones ocupar lo que nos permitirá buscar un empleado por su nombre o apellido y seleccionarlo para ocupar el puesto:



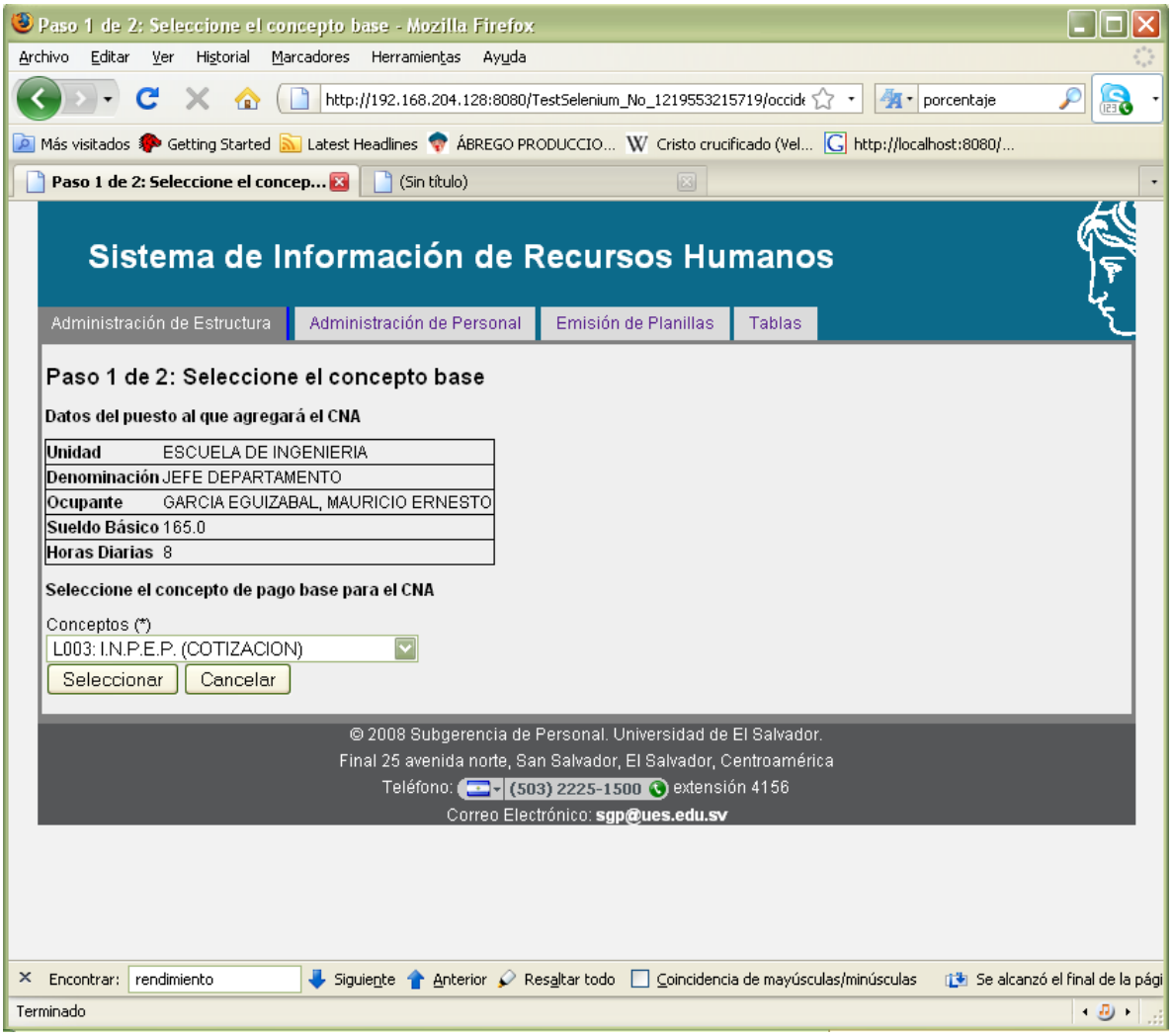
Una vez seleccionado el empleado se llenarán los datos de ocupación de puesto tal como fecha, nombramiento, etc



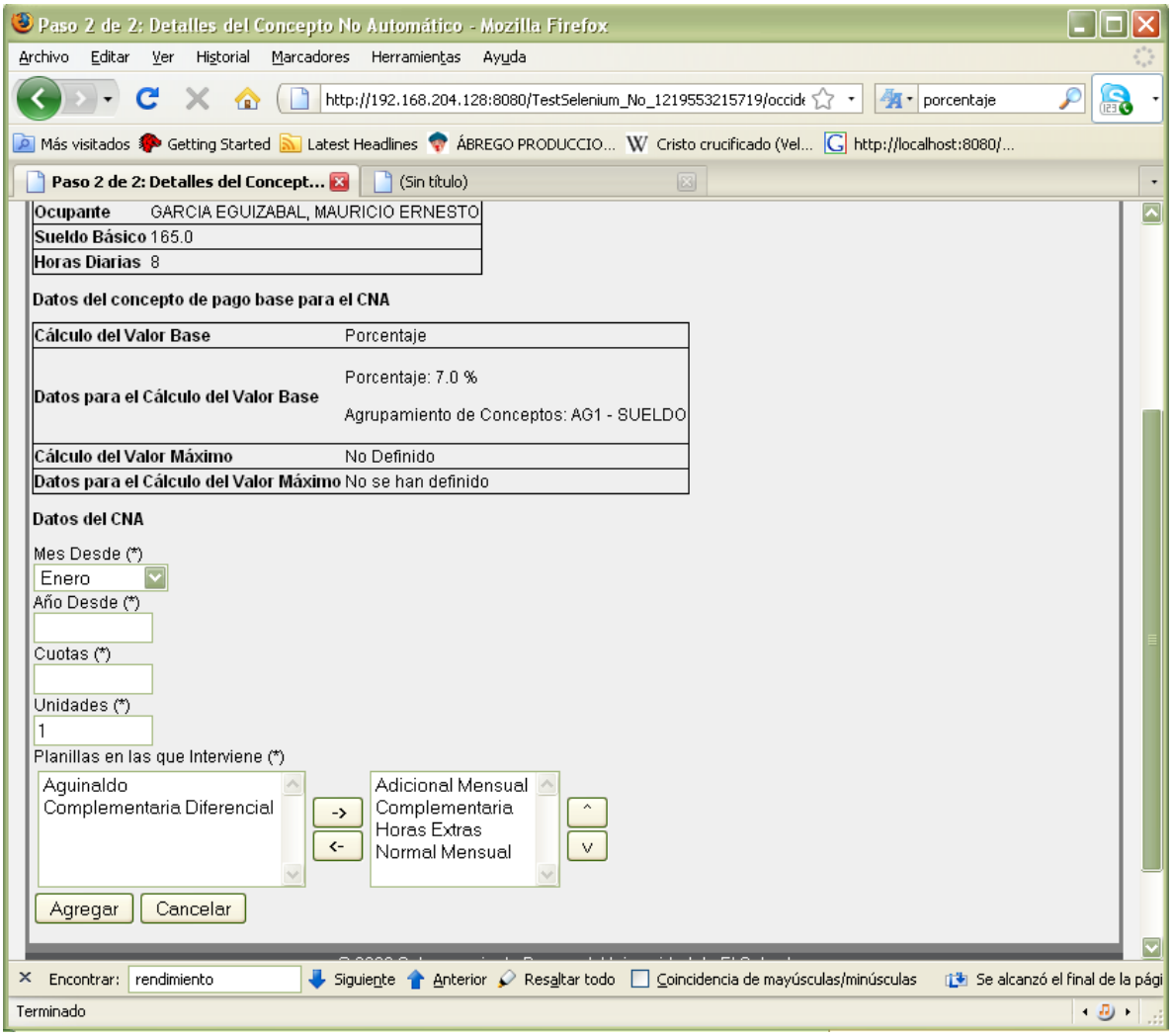
Además acá se pueden agregar los conceptos de pago que no son automáticos a cada puesto



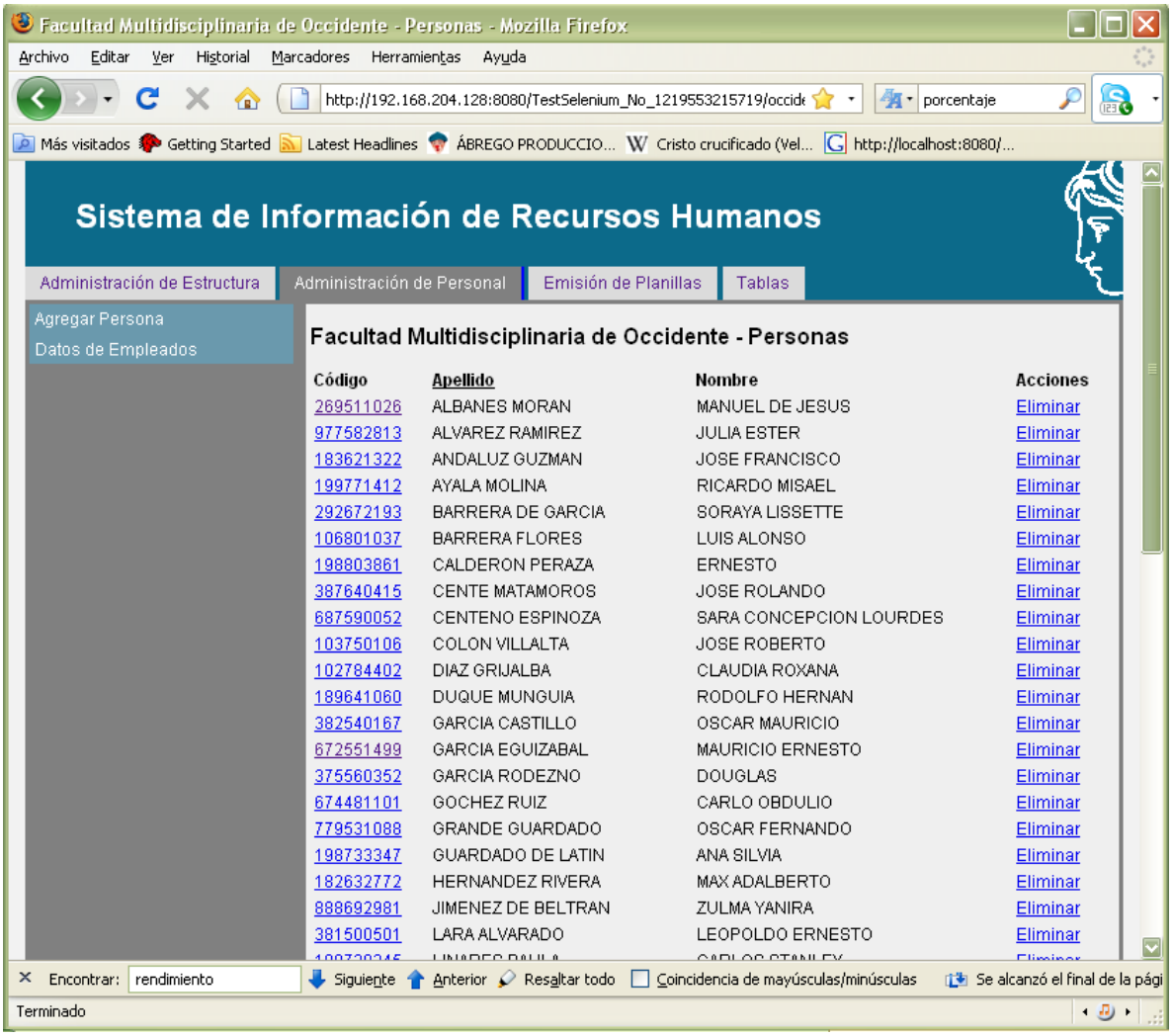
Usando el menu de acciones Agregar CNA en dos pasos: el primero seleccionar el concepto



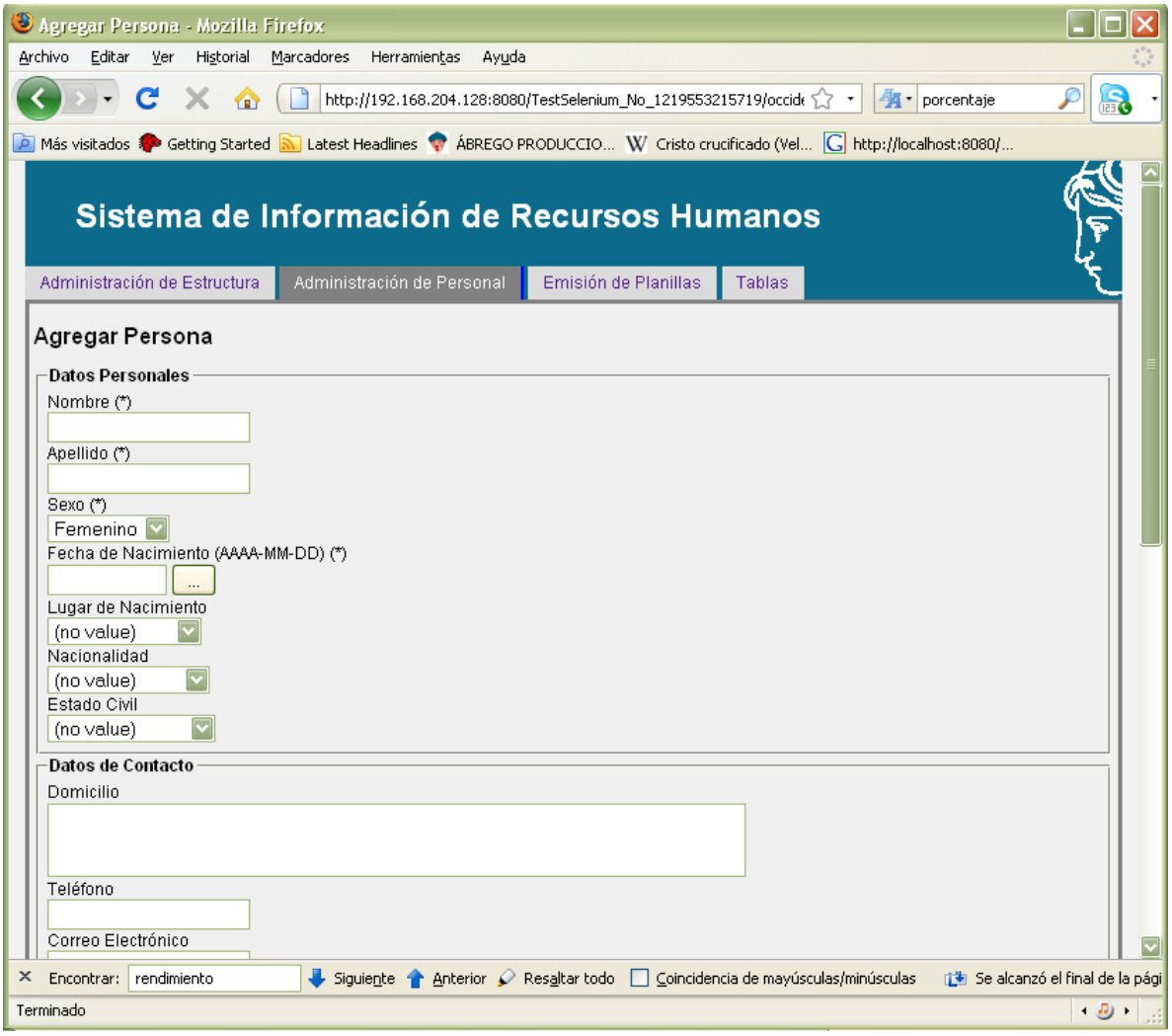
El segundo paso permite personalizar el tiempo y las planillas en las que interviene



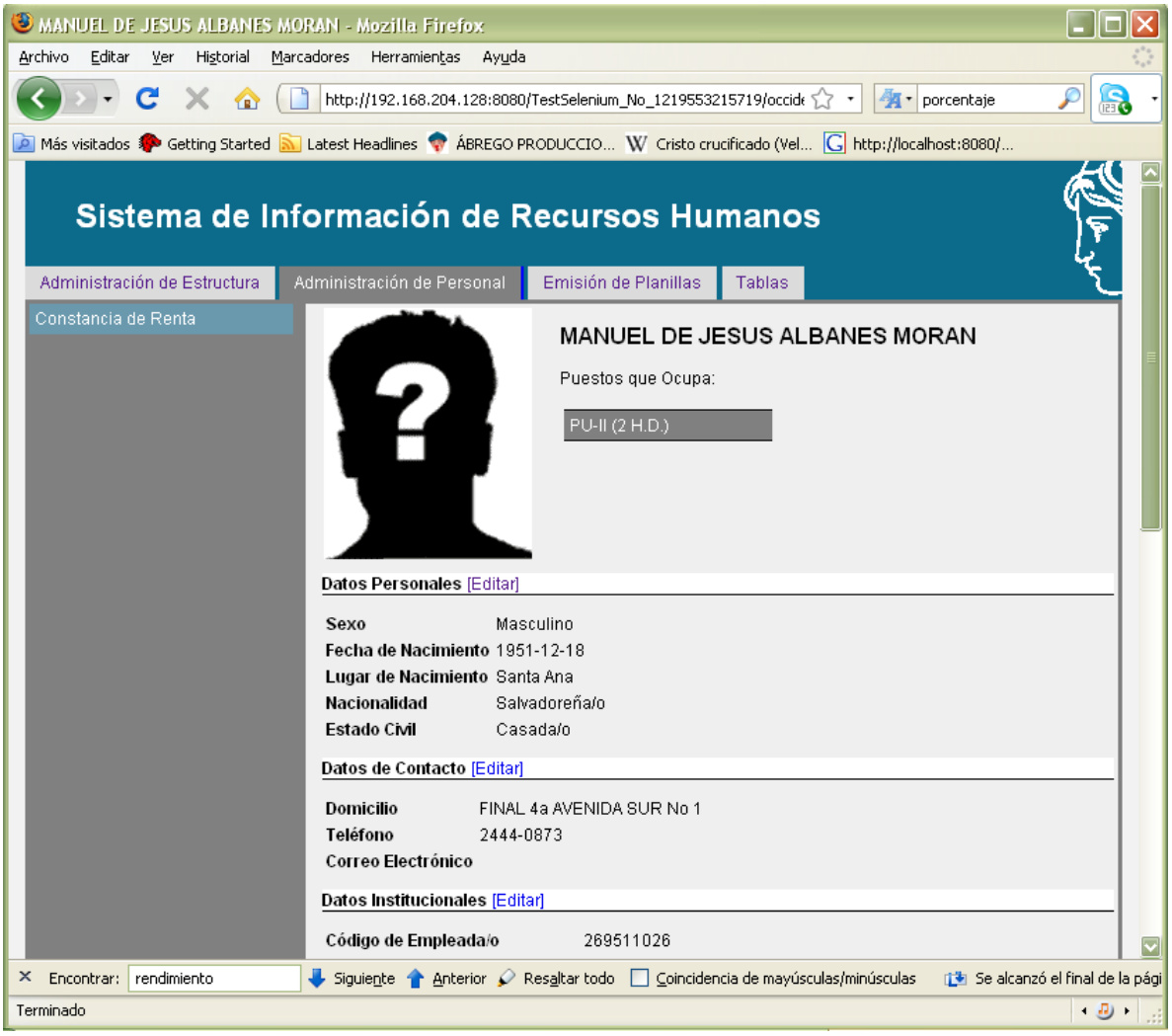
Administración de Personal: permite administrar la ficha laboral de cada trabajador de la institución.



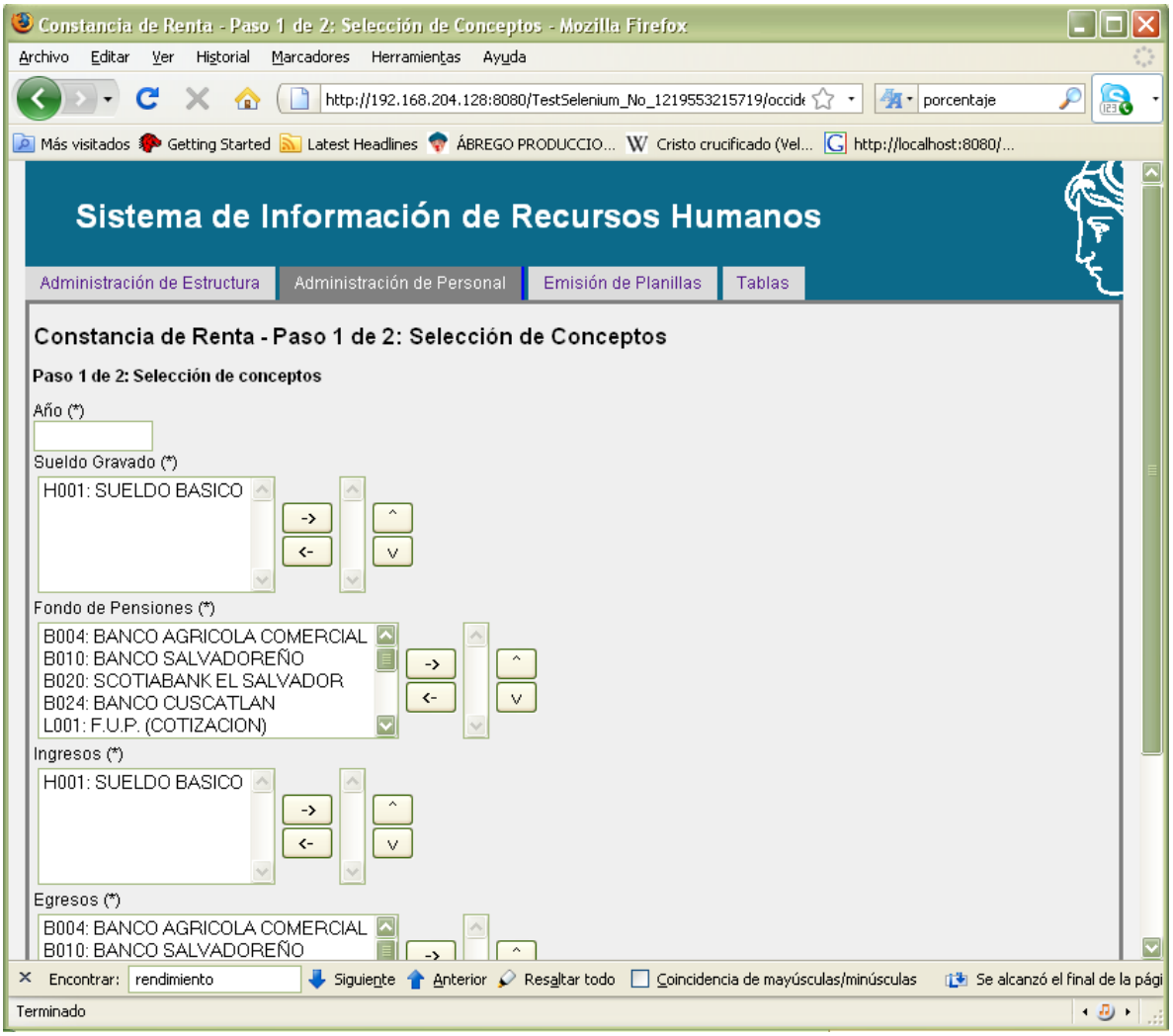
El menú de acción agregar persona muestra el formulario correspondiente con los datos básicos de la persona a la que se dará de alta



Una vez creado, haciendo clic en la persona puede completarse la ficha de empleado con datos como la fotografía, datos bancarios, grupo familiar y dependientes, etc

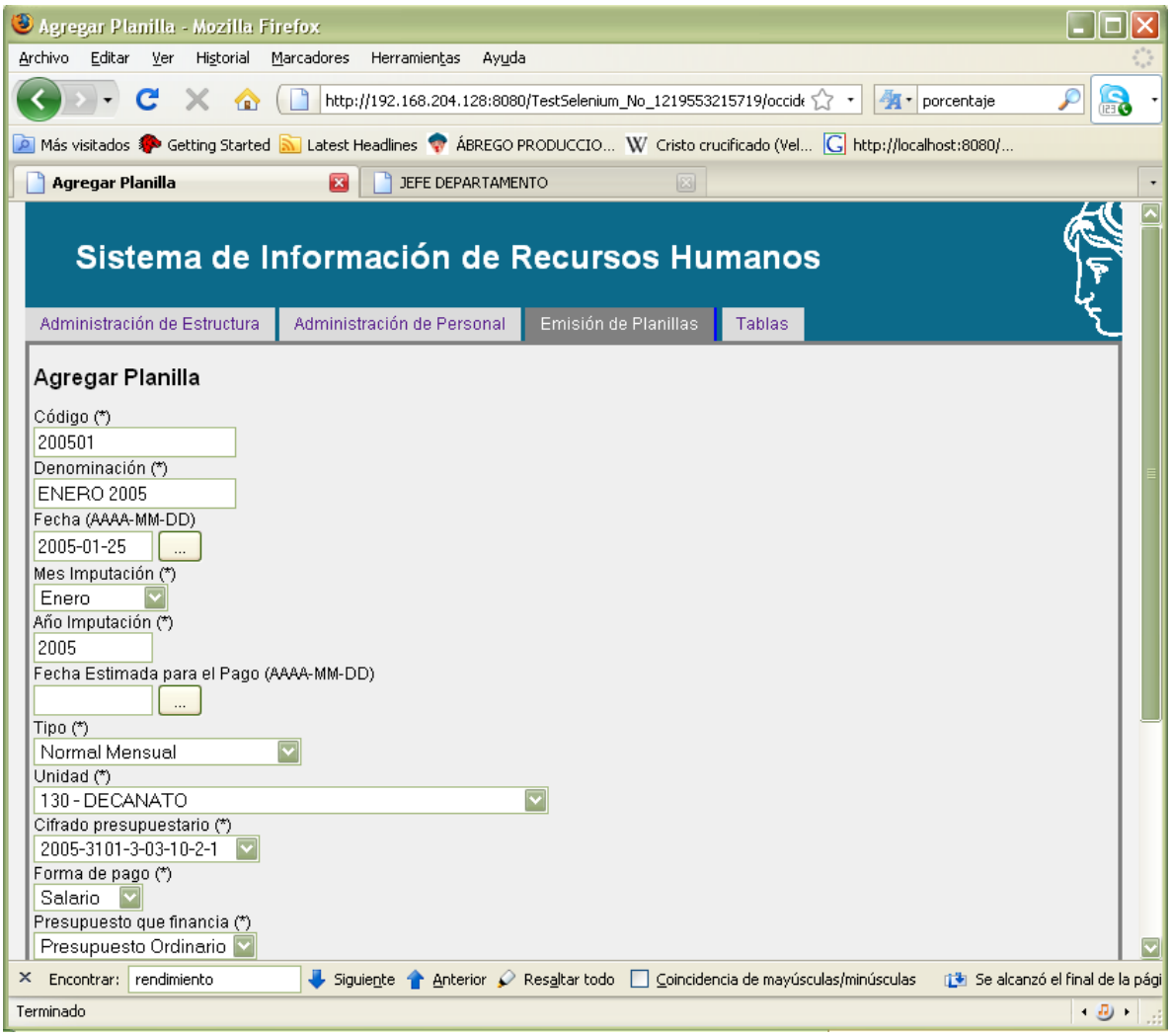


En las acciones de la persona puede además imprimirse la constancia de renta

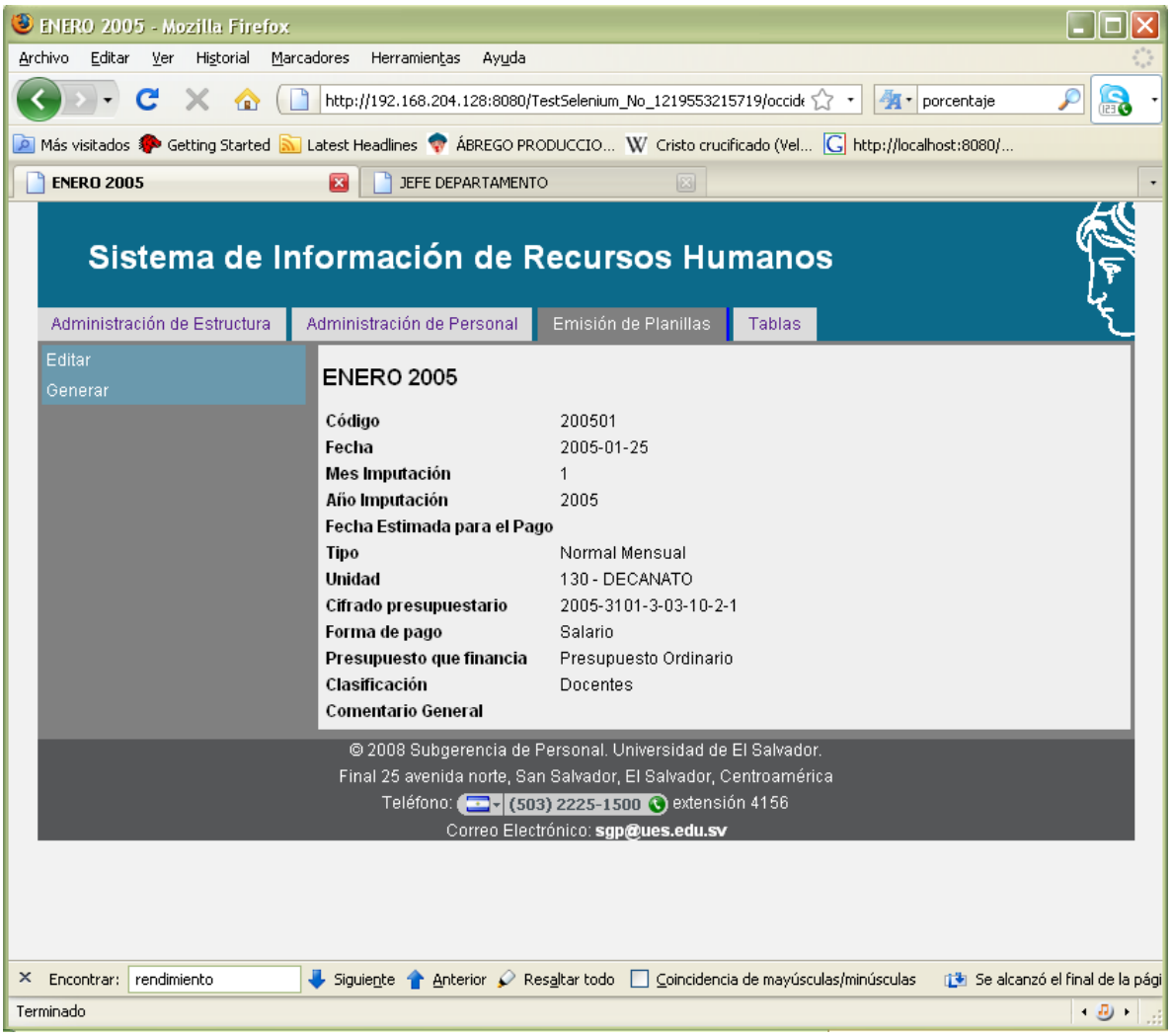


Emisión de planillas: permite preparar las planillas salarialaes de la institución y obtener reportes como constancias de pago personales, reportes sumarios de liquidaciones, etc.

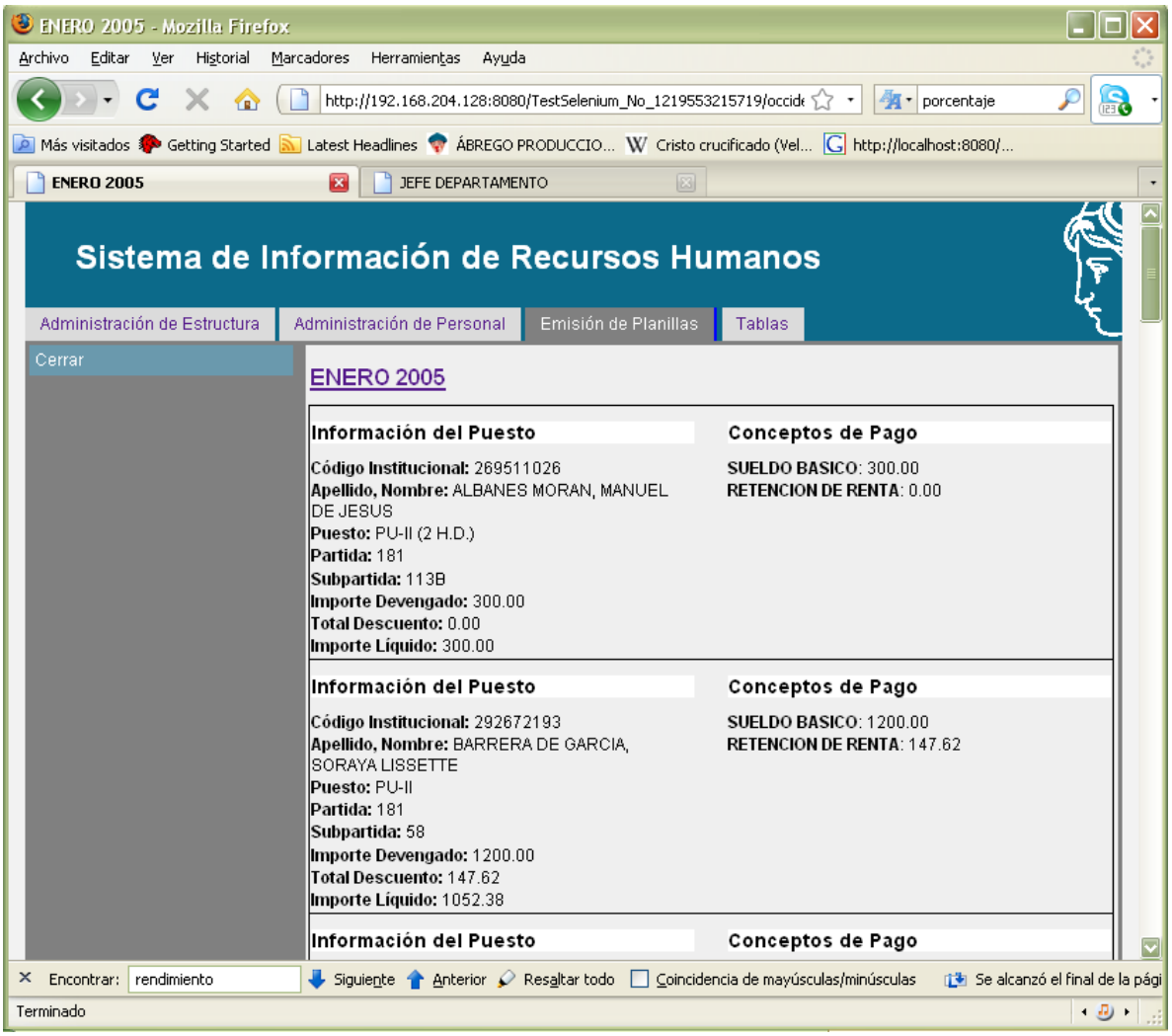
Para agregar una planilla se usa el menu de accion agregar planilla lo que desplegara el formulario para llenar los datos de la nueva planilla



Una vez creada la planilla usar el menu de acciones de generar

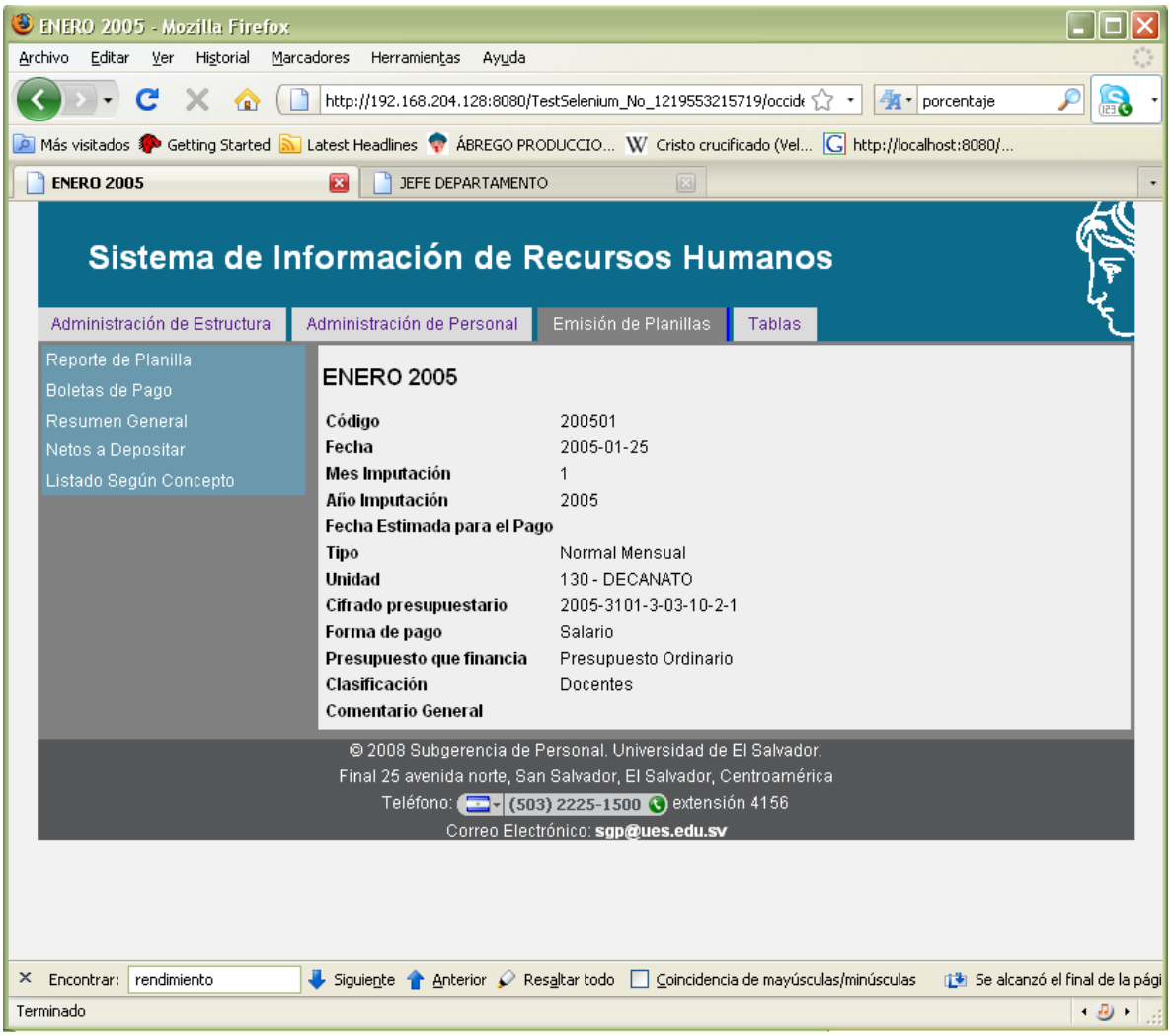


lo que desplegará los registros de la planilla



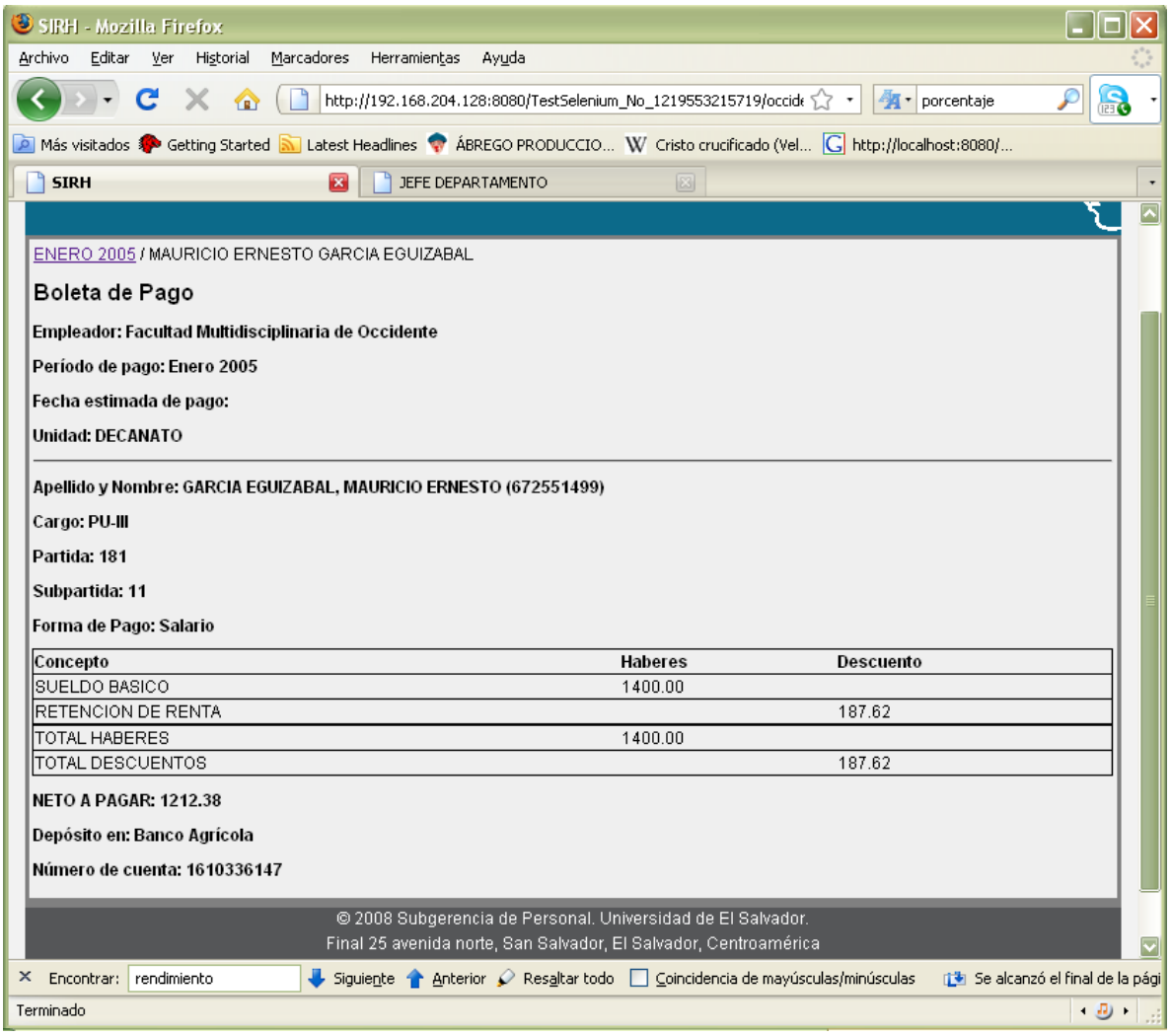
Puede revisarse y hacerse modificaciones en la estructura organizativa o en los datos del personal y volver a generarse cuantas veces sea necesario, una vez que se acepte la planilla como definitiva deberá cerrarse y no podrá modificarse mas.

Para cerrar usar el menu de acciones cerrar

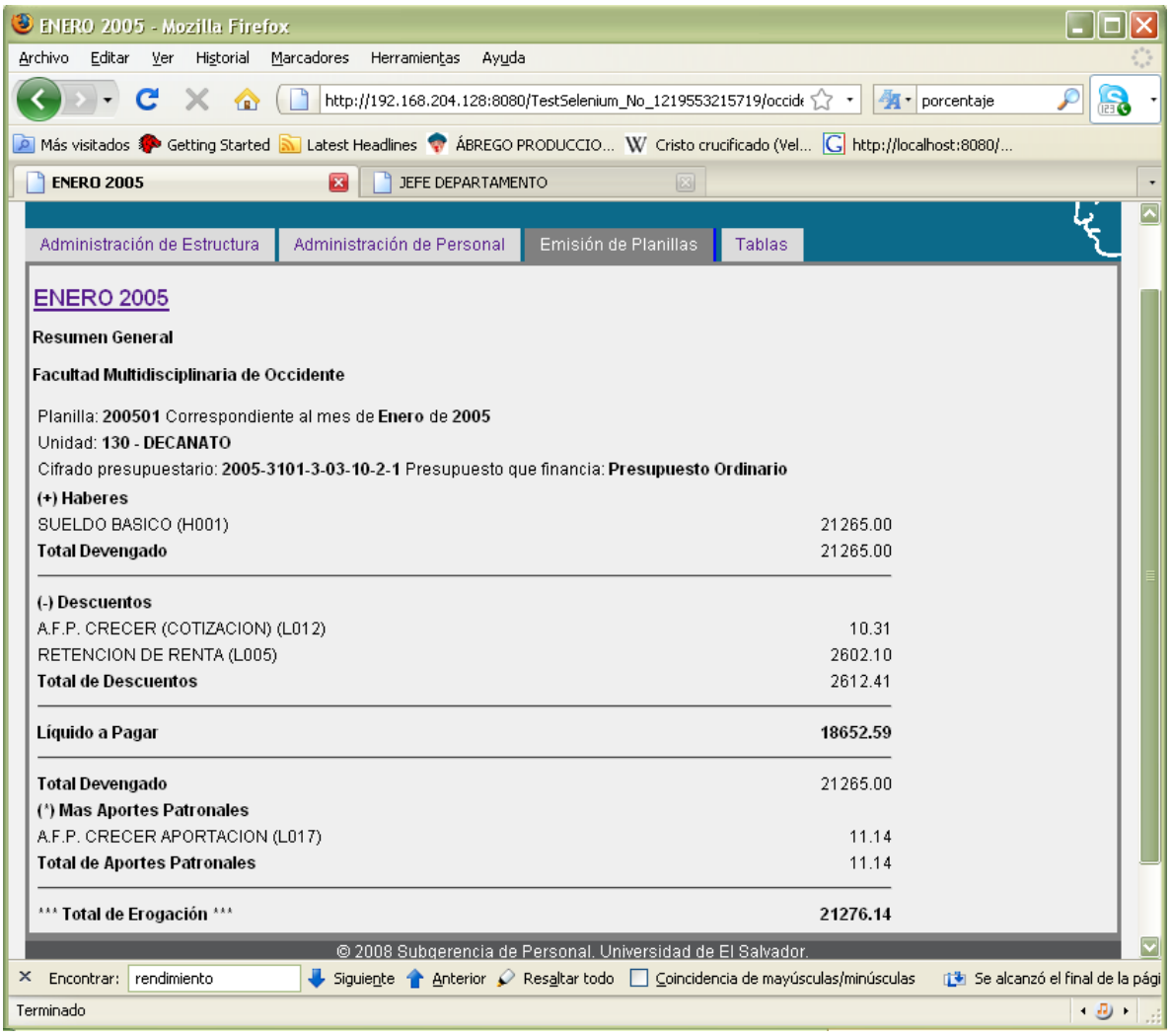


Ahora el menú de acciones presenta los reportes que es posible obtener.

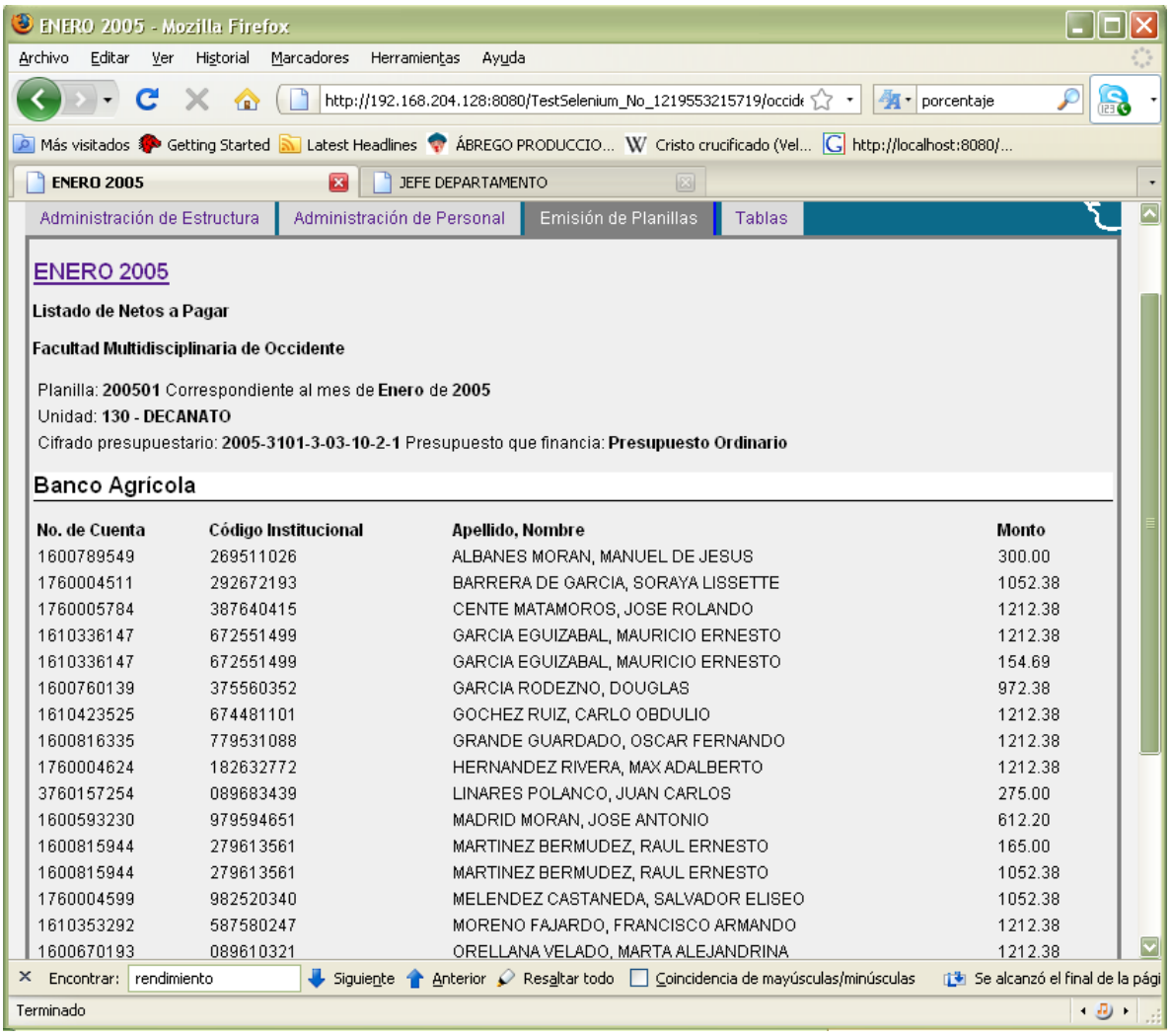
Boletas de pago:



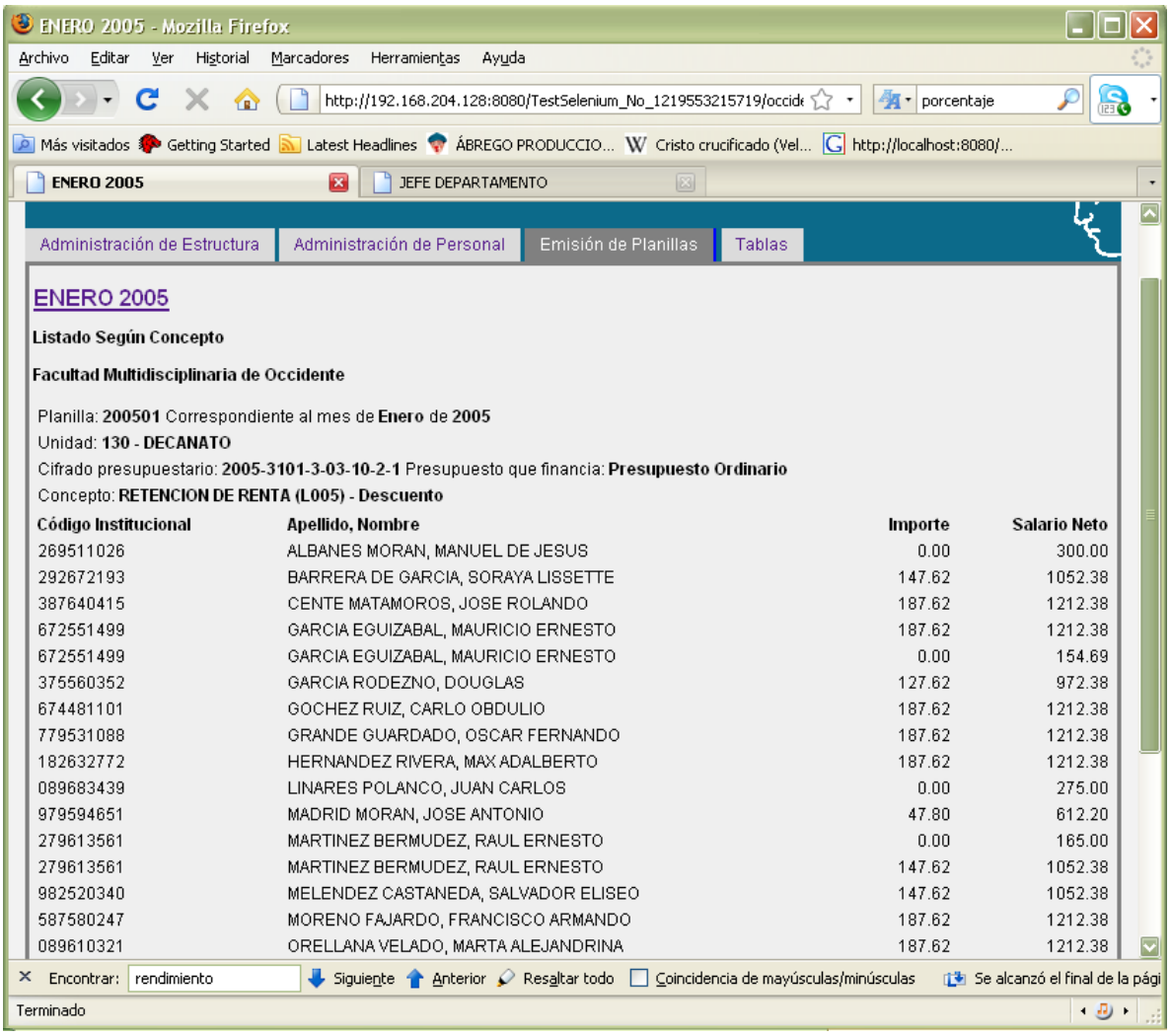
Resumen general:



Netos a depositar:



Listado según concepto:


ENERO 2005 - Mozilla Firefox
Archivo Editar Ver Historial Marcadores Herramientas Ayuda
http://192.168.204.128:8080/TestSelenium_No_1219553215719/occid
porcentaje
Más visitados Getting Started Latest Headlines ÁBREGO PRODUCCIO... Cristo crucificado (Vel... http://localhost:8080/...
ENERO 2005
JEFE DEPARTAMENTO
Administración de Estructura
Administración de Personal
Emisión de Planillas
Tablas
ENERO 2005
Listado Según Concepto
Facultad Multidisciplinaria de Occidente
Planilla: 200501 Correspondiente al mes de Enero de 2005
Unidad: 130 - DECANATO
Cifrado presupuestario: 2005-3101-3-03-10-2-1 Presupuesto que financia: Presupuesto Ordinario
Concepto: RETENCION DE RENTA (L005) - Descuento
Código Institucional Apellido, Nombre Importe Salario Neto
269511026 ALBANES MORAN, MANUEL DE JESUS 0.00 300.00
292672193 BARRERA DE GARCIA, SORAYA LISSETTE 147.62 1052.38
387640415 CENTE MATAMOROS, JOSE ROLANDO 187.62 1212.38
672551499 GARCIA EGUIZABAL, MAURICIO ERNESTO 187.62 1212.38
672551499 GARCIA EGUIZABAL, MAURICIO ERNESTO 0.00 154.69
375560352 GARCIA RODEZNO, DOUGLAS 127.62 972.38
674481101 GOCHEZ RUIZ, CARLO OBDULIO 187.62 1212.38
779531088 GRANDE GUARDADO, OSCAR FERNANDO 187.62 1212.38
182632772 HERNANDEZ RIVERA, MAX ADALBERTO 187.62 1212.38
089683439 LINARES POLANCO, JUAN CARLOS 0.00 275.00
979594651 MADRID MORAN, JOSE ANTONIO 47.80 612.20
279613561 MARTINEZ BERMUDEZ, RAUL ERNESTO 0.00 165.00
279613561 MARTINEZ BERMUDEZ, RAUL ERNESTO 147.62 1052.38
982520340 MELENDEZ CASTANEDA, SALVADOR ELISEO 147.62 1052.38
587580247 MORENO FAJARDO, FRANCISCO ARMANDO 187.62 1212.38
089610321 ORELLANA VELADO, MARTA ALEJANDRINA 187.62 1212.38
Encontrar: rendimiento
Siguiente Anterior Resaltar todo Coincidencia de mayúsculas/minúsculas
Se alcanzó el final de la pági
Terminado

8.2 Manual del programador

Esta sección describe los elementos técnicos que componen la aplicación. Algunos conceptos son propios de Zope 3, que es el framework de desarrollo utilizado para esta aplicación.

Componentes Principales:

- Sirh: la aplicación principal. En ella se encuentran registradas 33 utilidades, que son componentes globales que realizan tareas específicas dentro de la aplicación. La aplicación contiene dos tipos de componentes:
  - Conceptos de Pago
  - Componentes Sirhi
- Conceptos de Pago: son componentes que comprenden la lógica necesaria para la generación de planillas. Almacenan datos como el tipo de concepto (haber, descuento o aporte patronal), el orden de procesamiento, las planilals en que interviene, y los criterios de cálculo de su valor base y de su valor máximo.
- Sirhi: un componente que representa un Sistema de Información de Recursos Humanos Institucional. Almacena datos específicos

para cada Facultad, como partidas y subpartidas presupuestarias. Cada componente Sirhi, contiene otros tres componentes, los cuales se explican posteriormente:

- Estructura Organizacional

- Personas

- Planillas

- Estructura Organizacional: este componente almacena la estructura jerárquica de las unidades y puestos propios de cada Facultad. Puede contener dos componentes:
  - Unidades: almacenan el nombre de la unidad y pueden contener subunidades y puestos.

  - Puestos: almacena datos específicos de un puesto como su denominación, forma de pago, partida y subpartida presupuestaria, la fuente de financiamiento del puesto, el número de horas de trabajo y el sueldo máximo. También es capaz de almacenar el historial de movimientos de personal en el puesto y los conceptos no automáticos que son propios de cada trabajador.

- Personas: es un contenedor específico para componentes Persona, los cuales almacenan todos los datos personales de un trabajador. Mediante el uso de Adaptadores, también se almacenan los datos de interés institucional del trabajador como su núcleo familiar, los idiomas que maneja, sus habilidades, las capacitaciones recibidas, los títulos obtenidos y el historial de empleos. A toda esta información se hace referencia como el perfil electrónico del trabajador.

- Planillas: es un contenedor para componentes Planilla. Las planillas almacenan registros de planilla para cada trabajador que se toma en cuenta al generar una planilla. La lógica de generación de una planilla se encuentra codificada en el componente Generador de Planillas, el cual discrimina puestos basado en diferentes criterios:
  - Que el puesto no está vacío
  - Que el estado del empleado se encuentre activo
  - Que el código presupuestario del puesto sea igual que el de la planilla que se esá generando
  - Que la forma de pago (salario, contrato, jornal) del puesto sea igual que la de la planilla que se está generando

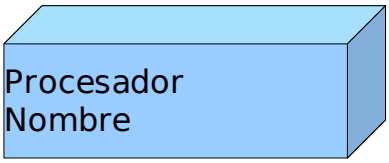
- Que la fuente de financiamiento del puesto sea igual que la de la planilla que se está generando

- Que la clasificación del puesto sea la requerida por la planilla que se está generando

8.3 Simbología Diagramas de distribución UML

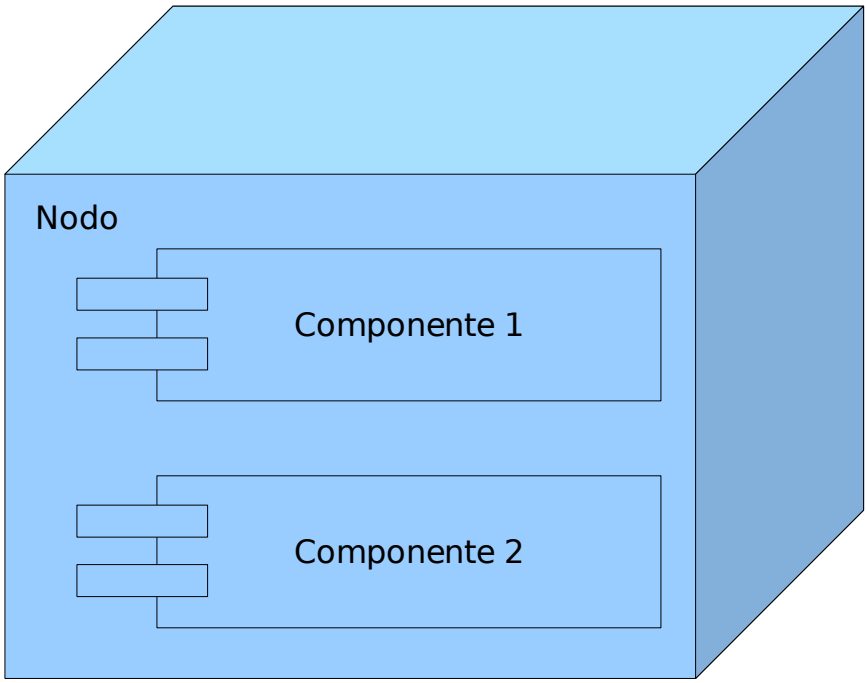
El diagrama de distribción UML muestra la arquitectura física de un sistema informático. Puede representar a los equipos y a los dispositivos, y también mostrar sus interconexiones y el software que se encontrará en cada máquina.

Nodo: un nodo es un recurso físico capaz de ejecutar componentes de código (procesador).

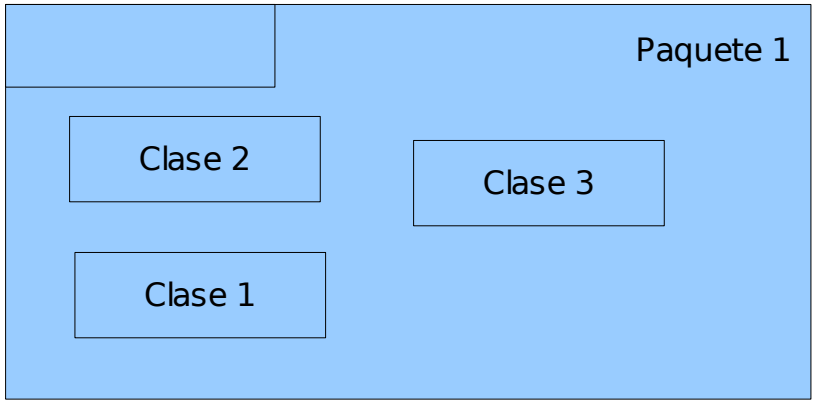


Asociación: la asociación se refiere a la conexión física entre los nodos, como por ejemplo Ethernet.

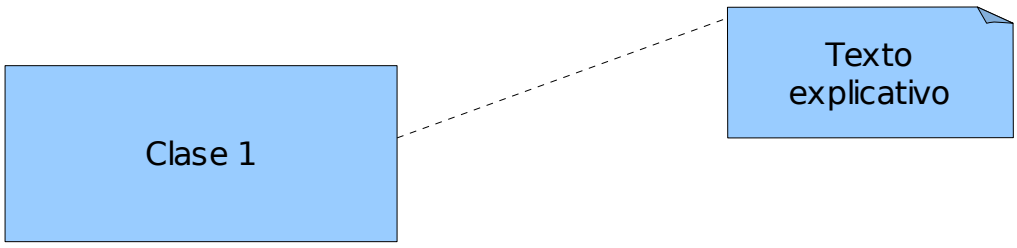
Componentes y Nodos



Paquetes: en algunas ocasiones se encontrarä con la necesidad de organizar los elementos de un diagrama en un grupo. Tal vez quiera mostrar que cieras clases o componentes son parte de un subsistema en particular. Para ello, se pueden agrupar en un paquete, que se representa como una carpeta tabular.



Notas: es frecuente que alguna parte del diagrama no presente una clara explicación del porqué está allí o la manera en qué trabaja. Cuando éste sea el caso, la nota UML será útil. La nota tiene una esquina doblada y se adjunta al elemento del diagrama conectándolo mediante una línea punteada.



8.4 Código Fuente Sistema

App.y

```
# -*- coding: utf8 -*-#


import grok

import os

from zope import interface, app, component, traversing, formlib, security

from zope.publisher.interfaces import INotFound

from zope.app.container.interfaces import INameChooser


from sirh import sirhi

from sirh import vocabulary

from sirh import interfaces

from sirh import default_data

from sirh import payment

from sirh import group

from sirh import table


# Models
```

```
class Sirh(grok.Application, grok.Container):
    pass


# Utilities


grok.global_utility(vocabulary.UnicodeVocabularyFactory(name=
"OccupationalCodes),                name="OccupationalCodes",
direct=True)
grok.global_utility(vocabulary.UnicodeVocabularyFactory(name=
"Remunerations"),
name="Remunerations", direct=True)
grok.global_utility(vocabulary.UnicodeVocabularyFactory(name=
"FinancingSources")
, name="FinancingSources", direct=True)
grok.global_utility(vocabulary.UnicodeVocabularyFactory(name=
"PaymentMethods"),
name="PaymentMethods",direct=True)
grok.global_utility(vocabulary.UnicodeVocabularyFactory(name=
"Classifications"),
name="Classifications",direct=True)
grok.global_utility(vocabulary.UnicodeVocabularyFactory(name=
"Genders"),name=
```

```
"Genders",direct=True)

grok.global_utility(vocabulary.UnicodeVocabularyFactory(name=
"Cities"),name=

"Cities",direct=True)

grok.global_utility(vocabulary.UnicodeVocabularyFactory(name=
"Nationalities"),

name="Nationalities",direct=True)

grok.global_utility(vocabulary.UnicodeVocabularyFactory(name=
"MaritalStatus"),

name="MaritalStatus",direct=True)

grok.global_utility(vocabulary.UnicodeVocabularyFactory(name=
"Banks"),name=

"Banks",direct=True)

grok.global_utility(vocabulary.UnicodeVocabularyFactory(name=
"PensionManagers"),

name="PensionManagers",direct=True)

grok.global_utility(vocabulary.UnicodeVocabularyFactory(name=
"Relationships"),

name="Relationships",direct=True)

grok.global_utility(vocabulary.UnicodeVocabularyFactory(name=
"ForeignLanguages")

,name="ForeignLanguages",direct=True)

grok.global_utility(vocabulary.UnicodeVocabularyFactory(name=

"ForeignLanguageLevels"),name="ForeignLanguageLevels",direct=
```

```
True)

grok.global_utility(vocabulary.UnicodeVocabularyFactory(name=
"Skills"),name=

"Skills",direct=True)

grok.global_utility(vocabulary.UnicodeVocabularyFactory(name=
"TrainingTypes"),

name="TrainingTypes",direct=True)

grok.global_utility(vocabulary.UnicodeVocabularyFactory(name=
"TrainingAreas"),

name="TrainingAreas",direct=True)

grok.global_utility(vocabulary.UnicodeVocabularyFactory(name=
"Degrees"),name=

"Degrees",direct=True)

grok.global_utility(vocabulary.UnicodeVocabularyFactory(name=
"Sectors"),name=

"Sectors",direct=True)

grok.global_utility(vocabulary.UnicodeVocabularyFactory(name=
"PositionStatus"),

name="PositionStatus",direct=True)

grok.global_utility(vocabulary.UnicodeVocabularyFactory(name=
"MovementCauses"),

name="MovementCauses",direct=True)

grok.global_utility(vocabulary.UnicodeVocabularyFactory(name=
"PayrollTypes"),
```

```
name="PayrollTypes",direct=True)
grok.global_utility(vocabulary.UnicodeVocabularyFactory(name=
"PaymentReferenceTypes"),name="PaymentReferenceTypes",direct=
True)
grok.global_utility(vocabulary.UnicodeVocabularyFactory(name=
"PaymentReferenceClasses"),name="PaymentReferenceClasses",dir
ect=True)
grok.global_utility(vocabulary.UnicodeVocabularyFactory(name=
"BaseValueCalculations"),name="BaseValueCalculations",direct=
True)
grok.global_utility(vocabulary.UnicodeVocabularyFactory(name=
"MaximumValueCalculations"),name="MaximumValueCalculations",d
irect=True)
grok.global_utility(payment.PaymentReferenceVocabularyFactory
(),name=
"PaymentReferences",direct=True)
grok.global_utility(group.PaymentReferenceGroupVocabularyFact
ory(),name=
"PaymentReferenceGroups",direct=True)
grok.global_utility(vocabulary.UnicodeVocabularyFactory(name=
"Accumulations"),
name="Accumulations",direct=True)
grok.global_utility(table.TableVocabularyFactory(),name="Tabl
es",direct=True)
```

```
grok.global_utility(vocabulary.UnicodeVocabularyFactory(name=
"Operators"),name=

"Operators",direct=True)

grok.global_utility(vocabulary.UnicodeVocabularyFactory(name=
"Calculations"),

name="Calculations",direct=True)

grok.global_utility(payment.NoAutomaticPaymentReferenceVocabu
laryFactory(),name=

"NoAutomaticPaymentReferences",direct=True)



# Views


class Index(grok.View):


    grok.context(Sirh)

    grok.require("sirh.Admin")


    def update(self):

                self.sirhis = sorted([item for item in
self.context.values()

                                                        if
interfaces.ISirhi.providedBy(item)],
```

```
                                key=lambda x:x.name.lower())


class Master2(grok.View):

    grok.context(interface.Interface)


class Master1(grok.View):

    grok.context(interface.Interface)


class NotFound(grok.View):

    grok.context(INotFound)
    grok.name("index.html")

    def update(self):
        self.home = self.url(grok.getSite())

class Unauthorized(grok.View):
```

```
    grok.context(security.interfaces.IUnauthorized)
    grok.name("index.html")

    def update(self):
        self.request.response.setStatus(403)
        self.request.response.setHeader('Expires',
        'Mon, 26 Jul 1997 05:00:00 GMT')
        self.request.response.setHeader('Cache-Control',
        'no-store, no-cache, must-revalidate')
        self.request.response.setHeader('Pragma', 'no-cache')
        principal = self.request.principal
                                                auth      =
component.getUtility(app.security.interfaces.IAuthentication)
        auth.unauthorized(principal.id, self.request)


class TabUrl(grok.View):

    grok.context(interface.Interface)
    grok.name("tab_url")

    def hasParentProviding(self, iface):
```

```
        obj = self.context
        while obj is not None:
            if iface.providedBy(obj):
                return True
            obj = obj.__parent__
        return False


    def update(self):
            # XXX: y si los nombres por defecto de los
contenedores cambian o
        # se necesita cambiar el contenido de los enlaces?
        self.data = [
             ("estructura", u"Administración de Estructura"),
("personas",
                          u"Administración  de  Personal"),
("planillas", u"Emisión de
               Planillas"),          ("tablas", u"Tablas"),
("conceptos", u
             "Conceptos de Pago"),         ("agrupamientos",
u"Agrupamientos de
              Conceptos de Pago"),         ("tablas_rangos",
u"Tablas de Rangos"),
            ]
```

```
        self.options = {}
        for container, label in self.data:
                    self.options[container] = {"show": False,
"current": False}
            self.options[container]["display"] = label
        # Tablas siempre debe desplegarse
        self.options["tablas"]["show"] = True
        if self.hasParentProviding(interfaces.ISirhi):
            for option in self.options.values():
                option["show"] = True
            self.options["conceptos"]["show"] = False
            self.options["agrupamientos"]["show"] = False
            self.options["tablas_rangos"]["show"] = False
                                                        if
self.hasParentProviding(grok.interfaces.IApplication) and \
           not self.hasParentProviding(interfaces.ISirhi):
            self.options["conceptos"]["show"] = True
            self.options["agrupamientos"]["show"] = True
            self.options["tablas_rangos"]["show"] = True
                                                        if
self.hasParentProviding(interfaces.IUnitContainer):
            self.options["estructura"]["current"] = True
                                                        if
```

```
self.hasParentProviding(interfaces.IPersonContainer):
            self.options["personas"]["current"] = True
                                                                if
self.hasParentProviding(interfaces.IPayrollContainer):
            self.options["planillas"]["current"] = True
                                                                if
self.hasParentProviding(interfaces.IVocabularyDataContainer):
            self.options["tablas"]["current"] = True
                                                                if
self.hasParentProviding(interfaces.IPaymentReferenceContainer
):
            self.options["conceptos"]["current"] = True
                                                                if
self.hasParentProviding(interfaces.IPaymentReferenceGroupCont
ainer):
            self.options["agrupamientos"]["current"] = True
                                                                if
self.hasParentProviding(interfaces.ITableContainer):
            self.options["tablas_rangos"]["current"] = True



    def render(self):
        result = "<ul>"
        for name, label in self.data:
```

```
            if self.options[name]["show"]:
                result += self.options[name]["current"] and \
                          '<li class="current">' or '<li>'
                result += '<a href="%s">%s</a></li>' \
                                      % (self.tab_url(name),
self.options[name]["display"])
        result += "</ul>"
        return result

    def tab_url(self, name=None):
        obj = self.context
        while obj is not None:
            if interfaces.ISirhi.providedBy(obj) or \
               grok.interfaces.IApplication.providedBy(obj):
                return self.url(obj, name)
            obj = obj.__parent__
        raise ValueError("No Sirhi or Sirh found.")


class DeleteUrl(grok.View):

    grok.context(interface.Interface)
```

```
    grok.name("delete_url")

    def update(self):
        parent = self.context.__parent__
        name = self.context.__name__
          self.delete_url = "%s?name=%s" % (self.url(parent,
"delete"), name)

    def render(self):
        return self.delete_url


class Delete(grok.View):

    grok.context(grok.Container)
    grok.name("delete")
    grok.require("sirh.Admin")

    def update(self, name):
        if name in self.context:
            del(self.context[name])
```

```
    def render(self):
        self.redirect(self.url(self.context))


class Add(grok.Form):

    grok.context(Sirh)
    grok.require("sirh.Admin")

    form_fields = grok.Fields(interfaces.ISirhi)
                                          template          =
grok.PageTemplateFile(os.path.join(os.path.dirname(__file__),
    "custom_templates",
    "addsirhi.pt"))
    label = u"Agregar SIRH-I"

    @grok.action("Agregar")
    def add(self, **data):
        sirh_i = sirhi.Sirhi(**data)
            name  =  INameChooser(self.context).chooseName(u"",
sirh_i)
        self.context[name] = sirh_i
```

```
        self.redirect(self.url(sirh_i))

    @formlib.form.action("Cancelar", validator=lambda *a: ())
    def cancel(self, action, data):
        self.redirect(self.url(self.context))


# Subscribers

@grok.subscribe(Sirh, grok.IObjectAddedEvent)
def createSirhis(app, event):
    names = {}
    names["09"] = u"Facultad Multidisciplinaria de Oriente"
    names["10"] = u"Facultad Multidisciplinaria de Occidente"
    names["11"] = u"Facultad Multidisciplinaria Paracentral"
    app["oriente"] = sirhi.Sirhi(**{"name": names["09"]})
    app["occidente"] = sirhi.Sirhi(**{"name": names["10"]})
     app["occidente"].financial_mgr = u"Licda. Yecenia Flores
de Umaña"
    app["occidente"].rrhh_mgr = u"Ing. Alexander Calderón"
    app["paracentral"] = sirhi.Sirhi(**{"name": names["11"]})
    app["tablas"] = vocabulary.VocabularyDataContainer()
```

```
    app["conceptos"] = payment.PaymentReferenceContainer()
                                    app["agrupamientos"]        =
group.PaymentReferenceGroupContainer()
    app["tablas_rangos"] = table.TableContainer()


@grok.subscribe(vocabulary.VocabularyDataContainer,
grok.IObjectAddedEvent)
def fillVocabularyData(container, event):
    # XXX: to fix vdcontainer creation on individual sirhis
    if isinstance(container.__parent__, Sirh):
          # XXX: what if the container isn't directly in the
application?
        sm = container.__parent__.getSiteManager()
                                                chooser      =
app.container.interfaces.INameChooser(container)
        for data in default_data.global_vocabulary_data:
            vd = vocabulary.VocabularyData(**data)
            name = chooser.chooseName(vd.name, vd)
            container[name] = vd
                                        sm.registerUtility(vd,
interfaces.IVocabularyData, name=vd.name)
```

Criteria.py

```
# -*- coding: utf8 -*-

import grok
import os
from zope.formlib import form
from zope.app.form.interfaces import WidgetInputError

from sirh import interfaces
from sirh import payment



class ValueCriteria(grok.Model):
    grok.implements(interfaces.IValueCriteria)
    def __init__(self, **data):
        self.value = data.get("value")
    def calculate(self):
        return self.value
class ValueCriteriaIndex(grok.View):
    grok.context(ValueCriteria)
    grok.name("render")
```

```
    def render(self):
                    return "<p>Valor Unitario: %s</p>" %
(self.context.value)
class PercentageCriteria(grok.Model):
    grok.implements(interfaces.IPercentageCriteria)
    def __init__(self, **data):
        self.percentage = data.get("percentage")
        self.group = data.get("group")
    def calculate(self, group_total):
        return group_total * (self.percentage / 100.0)
class PercentageCriteriaIndex(grok.View):
    grok.context(PercentageCriteria)
    grok.name("render")
    def render(self):
        return "<p>Porcentaje: %s %%</p>" \
              "<p>Agrupamiento de Conceptos: %s - %s</p>" % \
               (self.context.percentage,
                self.context.group.code,
                self.context.group.name)
class TableCriteria(grok.Model):
    grok.implements(interfaces.ITableCriteria)
    def __init__(self, **data):
```

```
        self.table = data.get("table")
        self.group = data.get("group")
    def calculate(self, group_total):
        return self.table.evaluate(group_total)
class TableCriteriaIndex(grok.View):
    grok.context(TableCriteria)
    grok.name("render")
    def render(self):
        return "<p>Tabla: %s - %s</p>" \
               "<p>Agrupamiento de Conceptos: %s - %s</p>" % \
                (self.context.table.code,
                 self.context.table.name,
                 self.context.group.code,
                 self.context.group.name)
class ValueCriteriaForm(grok.Form):
    grok.context(interfaces.IPaymentReference)
    grok.name("value")
    grok.require("sirh.Admin")
    def __init__(self, *args, **kw):
        super(ValueCriteriaForm, self).__init__(*args, **kw)
          self.label = u"Cálculo del Valor Base - %s: %s" %
(self.context.code,
```

```
self.context.name)
    form_fields = grok.Fields(interfaces.IValueCriteria)
template=grok.PageTemplateFile(os.path.join(os.path.dirname(_
_file__),"custom_templates","criteriaform.pt"))

    def setUpWidgets(self, *args, **kw):
            super(ValueCriteriaForm, self).setUpWidgets(*args,
**kw)
if self.context.criteria and getattr(self.context.criteria,
"value")
self.widgets["value"].setRenderedValue(self.context.criteria.
value)

    @grok.action("Guardar")
    def save(self, **data):
        criteria = ValueCriteria(**data)
        self.context.criteria = criteria
        self.redirect(self.url(self.context))

    @form.action("Cancelar", validator=lambda *a: ())
    def cancel(self, action, data):
        self.redirect(self.url(self.context))
```

```
class PercentageCriteriaForm(grok.Form):
    grok.context(interfaces.IPaymentReference)
    grok.name("percentage")
    grok.require("sirh.Admin")

form_fields = grok.Fields(interfaces.IPercentageCriteria)
template =
grok.PageTemplateFile(os.path.join(os.path.dirname(__file__),
"custom_templates","criteriaform.pt"))

    def __init__(self, *args, **kw):
  super(PercentageCriteriaForm, self).__init__(*args, **kw)
          self.label = u"Cálculo del Valor Base - %s: %s" %
(self.context.code,self.context.name)

    def setUpWidgets(self, *args, **kw):
super(PercentageCriteriaForm, self).setUpWidgets(*args, **kw)
        if self.context.criteria:
            if getattr(self.context.criteria, "percentage"):
                self.widgets["percentage"].setRenderedValue(s
elf.context.criteria.percentage)
            if getattr(self.context.criteria, "group"):
```

```
                self.widgets["group"].setRenderedValue(self.c
ontext.criteria.group)


    @grok.action("Guardar")
    def save(self, **data):
        criteria = PercentageCriteria(**data)
        self.context.criteria = criteria
        self.redirect(self.url(self.context))


    def validate(self, action, data):
                    errors   =   super(PercentageCriteriaForm,
self).validate(action, data)
        if errors: return errors


        references = data.get("group").references
           max_order = max([reference.order for reference in
references])


        if self.context.order <= max_order:
            error = WidgetInputError("group",
u"Agrupamiento de Conceptos",u"El orden de proceso del
concepto actual debe ser mayor al de los conceptos contenidos
en el agrupamiento seleccionado")
```

```
            errors.append(error)
            return errors
        return errors

    @form.action("Cancelar", validator=lambda *a: ())
    def cancel(self, action, data):
        self.redirect(self.url(self.context))
class TableCriteriaForm(grok.Form):
    grok.context(interfaces.IPaymentReference)
    grok.name("table")
    grok.require("sirh.Admin")
    form_fields = grok.Fields(interfaces.ITableCriteria)
template                                                    =
grok.PageTemplateFile(os.path.join(os.path.dirname(__file__),
"custom_templates","criteriaform.pt"))

    def __init__(self, *args, **kw):
        super(TableCriteriaForm, self).__init__(*args, **kw)
          self.label = u"Cálculo del Valor Base - %s: %s" %
(self.context.code,self.context.name)

    def setUpWidgets(self, *args, **kw):
```

```
            super(TableCriteriaForm,  self).setUpWidgets(*args,
**kw)
        if self.context.criteria:
            if getattr(self.context.criteria, "table"):
                self.widgets["table"].setRenderedValue(self.c
ontext.criteria.table)
            if getattr(self.context.criteria, "group"):
                self.widgets["group"].setRenderedValue(self.c
ontext.criteria.group)


    @grok.action("Guardar")
    def save(self, **data):
        criteria = TableCriteria(**data)
        self.context.criteria = criteria
        self.redirect(self.url(self.context))
    def validate(self, action, data):
errors   =   super(TableCriteriaForm,   self).validate(action,
data)
        if errors: return errors
        references = data.get("group").references
          max_order = max([reference.order for reference in
references])
```

```
        if self.context.order <= max_order:
              error = WidgetInputError("group",u"Agrupamiento
de Conceptos",u"El orden de proceso del concepto actual debe
ser mayor al de los conceptos contenidos en el agrupamiento
seleccionado")
            errors.append(error)
            return errors
        return errors

    @form.action("Cancelar", validator=lambda *a: ())
    def cancel(self, action, data):
        self.redirect(self.url(self.context))

class BaseValue(grok.View):

    grok.context(interfaces.IPaymentReference)
    grok.name("base")

    def render(self):
        if self.context.base_value == u"Valor Unitario":
            self.redirect(self.url("value"))
            return
        if self.context.base_value == u"Porcentaje":
```

```
            self.redirect(self.url("percentage"))
            return
        if self.context.base_value == u"Tabla":
            self.redirect(self.url("table"))
            return
class MaximumTotalCriteria(grok.Model):
    grok.implements(interfaces.IMaximumTotalCriteria)
    def __init__(self, **data):
        self.accumulation = data.get("accumulation")
        self.value = data.get("value")
class MaximumTotalCriteriaIndex(grok.View):
    grok.context(MaximumTotalCriteria)
    grok.name("render")
    def render(self):
        return "<p>Monto Maximo: %s</p>" \
               "<p>Acumulacion Desc: %s</p>" % \
              (self.context.value, self.context.accumulation)
class MaximumTotalCriteriaForm(grok.Form):
    grok.context(interfaces.IPaymentReference)
    grok.name("maximum_value")
    grok.require("sirh.Admin")
  form_fields = grok.Fields(interfaces.IMaximumTotalCriteria)
```

```
 form_fields = form_fields.select("value", "accumulation")
template=
grok.PageTemplateFile(os.path.join(os.path.dirname(__file__),
"custom_templates","criteriaform.pt"))

    def __init__(self, *args, **kw):
        super(MaximumTotalCriteriaForm, self).__init__(*args,
**kw)
         self.label = u"Cálculo del Valor Máximo - %s: %s" %
(self.context.code, self.context.name)

    def setUpWidgets(self, *args, **kw):
        super(MaximumTotalCriteriaForm,self).setUpWidgets(*ar
gs, **kw)
        if self.context.maximum_criteria:
            if getattr(self.context.maximum_criteria,
"value"):
                self.widgets["value"].setRenderedValue(self.c
ontext.maximum_criteria.value)

            if getattr(self.context.maximum_criteria,
"accumulation"):
                self.widgets["accumulation"].setRenderedValue
(self.context.maximum_criteria.accumulation)
```

```
    @grok.action("Guardar")
    def save(self, **data):
        criteria = MaximumTotalCriteria(**data)
        self.context.maximum_criteria = criteria
        self.redirect(self.url(self.context))
    @form.action("Cancelar", validator=lambda *a: ())
    def cancel(self, action, data):
        self.redirect(self.url(self.context))
class MaximumPercentageCriteria(grok.Model):
    grok.implements(interfaces.IMaximumPercentageCriteria)
    def __init__(self, **data):
        self.accumulation = data.get("accumulation")
        self.percentage = data.get("percentage")
        self.group = data.get("group")
class MaximumPercentageCriteriaIndex(grok.View):
    grok.context(MaximumPercentageCriteria)
    grok.name("render")
    def render(self):
        return "<p>Porcentaje: %s %%</p>" \
               "<p>Agrupamiento de Conceptos: %s - %s</p>" \
               "<p>Acumulacion Desc: %s</p>" % \
               (self.context.percentage,
```

```
                self.context.group.code,
                self.context.group.name,
                self.context.accumulation)
class MaximumPercentageCriteriaForm(grok.Form):
    grok.context(interfaces.IPaymentReference)
    grok.name("maximum_percentage")
    grok.require("sirh.Admin")
    template =
grok.PageTemplateFile(os.path.join(os.path.dirname(__file__),
"custom_templates", "criteriaform.pt"))


form_fields
=grok.Fields(interfaces.IMaximumPercentageCriteria)
    def __init__(self, *args,
**kw):super(MaximumPercentageCriteriaForm,self).__init__(*arg
s, **kw)
        self.label = u"Cálculo del Valor Máximo - %s: %s" %
(self.context.code,self.context.name)
    def setUpWidgets(self, *args, **kw):
        super(MaximumPercentageCriteriaForm,
self).setUpWidgets(*args, **kw)
        if self.context.maximum_criteria:
            if getattr(self.context.maximum_criteria,
"percentage"):
```

```
                self.widgets["percentage"].setRenderedValue(s
elf.context.maximum_criteria.percentage)
if getattr(self.context.maximum_criteria, "group"):
                self.widgets["group"].setRenderedValue(self.c
ontext.maximum_criteria.group)
            if getattr(self.context.maximum_criteria,
"accumulation"):
                self.widgets["accumulation"].setRenderedValue
(self.context.maximum_criteria.accumulation)

    @grok.action("Guardar")
    def save(self, **data):
        criteria = MaximumPercentageCriteria(**data)
        self.context.maximum_criteria = criteria
        self.redirect(self.url(self.context))

    @form.action("Cancelar", validator=lambda *a: ())
    def cancel(self, action, data):
        self.redirect(self.url(self.context))

class MaximumValue(grok.View):
    grok.context(interfaces.IPaymentReference)
    grok.name("maximum")
```

```
def render(self):
    if self.context.maximum_value == u"Monto Máximo":
        self.redirect(self.url("maximum_value"))
        return
    if self.context.maximum_value == u"Porcentaje":
        self.redirect(self.url("maximum_percentage"))
        return
```

Group.py

```
# -*- coding: utf8 -*-


import grok

import os

from persistent.list import PersistentList

from zope.app.container.interfaces import INameChooser

from zope.formlib import form

from zope.traversing.api import getName

from zope import schema


from sirh import interfaces

from sirh import vocabulary

class PaymentReferenceGroupContainer(grok.Container):

grok.implements(interfaces.IPaymentReferenceGroupContainer)

class PaymentReferenceGroup(grok.Model):

    grok.implements(interfaces.IPaymentReferenceGroup)

    def __init__(self, **data):

        self.code = data.get("code")

        self.name = data.get("name")

     self.references = PersistentList(data.get("references"))

    def total(self):
```

```
        for reference in references:
            pass
class Add(grok.AddForm):
    grok.context(interfaces.IPaymentReferenceGroupContainer)
    grok.require("sirh.Admin")
 form_fields = grok.Fields(interfaces.IPaymentReferenceGroup)
template =
grok.PageTemplateFile(os.path.join(os.path.dirname(__file__),
    "custom_templates","paymentreferencegroupform.pt"))
    label = u"Agregar Agrupamiento de Conceptos de Pago"

    @grok.action("Agregar")
    def add(self, **data):
        group = PaymentReferenceGroup(**data)
name = INameChooser(self.context).chooseName(group.code,
group)
        self.context[name] = group
        self.redirect(self.url(self.context))

    @form.action("Cancelar", validator=lambda *a: ())
    def cancel(self, action, data):
        self.redirect(self.url(self.context))
```

```
class Index(grok.View):
    grok.context(interfaces.IPaymentReferenceGroup)
    grok.require("sirh.Admin")
class GroupReferences(grok.View):
    grok.context(interfaces.IPaymentReferenceGroup)
    grok.name("references")
    def render(self):
        if not self.context.references:
    return "No se han agregado conceptos a este agrupamiento"
        return "<ul>%s</ul>" % \
("".join(["<li>%s: %s</li>" % (reference.code,reference.name)
    for reference in self.context.references]))

class Edit(grok.EditForm):
    grok.context(interfaces.IPaymentReferenceGroup)
    grok.require("sirh.Admin")
form_fields = grok.Fields(interfaces.IPaymentReferenceGroup)
template =
grok.PageTemplateFile(os.path.join(os.path.dirname(__file__),
"custom_templates","paymentreferencegroupform.pt"))
    label = u"Editar Agrupamiento de Conceptos de Pago"
    @grok.action("Guardar")
```

```
    def edit(self, **data):
        changes = self.applyData(self.context, **data)
        if changes:
            self.flash("Los cambios fueron guardados")
        self.redirect(self.url(self.context))

    @form.action("Cancelar", validator=lambda *a: ())
    def cancel(self, action, data):
        self.redirect(self.url(self.context))
class ContainerIndex(grok.View):
    grok.context(interfaces.IPaymentReferenceGroupContainer)
    grok.name("index")
    grok.require("sirh.Admin")
    def update(self):
        self.groups = self.context.values()
class
PaymentReferenceGroupVocabulary(vocabulary.BaseVocabulary):
    def titleValue(self, value):
        return "%s: %s" % (value.code, value.name)
class PaymentReferenceGroupVocabularyFactory(object):
    grok.provides(schema.interfaces.IVocabularyFactory)
    def getPaymentReferenceGroupContainer(self, context):
```

```
        obj = context
        while obj is not None:
if grok.interfaces.IApplication.providedBy(obj):
                container = [value for value in obj.values()
if
interfaces.IPaymentReferenceGroupContainer.providedBy(value)]
[0]
                return container
            obj = obj.__parent__
        raise ValueError("No Sirh found.")


    def __call__(self, context):
        container =
self.getPaymentReferenceGroupContainer(context)
values = sorted([ref for ref in
container.values()],key=lambda x:x.code)
        return PaymentReferenceGroupVocabulary(values)
```

interfaces.py

```
# -*- coding: utf-8 -*-


from zope import interface, schema
import re
phone_regex = r"^[0-9]{4}-[0-9]{4}$"
check_phone = re.compile(phone_regex).match
class NotAValidPhone(schema.ValidationError):
 __doc__ = u"Por favor utilice el formato ####-#### en este campo"
def validate_phone(phone):
    if not check_phone(phone):
        raise NotAValidPhone(phone)
    return True
code_regex = r"^[0-9]{9}$"
check_code = re.compile(code_regex).match
class NotAValidCode(schema.ValidationError):
 __doc__ = u"Por favor utilice el número de seguro social en este campo"
def validate_code(code):
    if not check_code(code):
        raise NotAValidCode(code)
    return True
```

```
bank_account_regex = r"^[0-9-]*$"
check_bank_account = re.compile(bank_account_regex).match
class NotAValidBankAccount(schema.ValidationError):
    __doc__ = u"Por favor utilice sólo números y guiones en este campo"
def validate_bank_account(bank_account):
    if not check_bank_account(bank_account):
        raise NotAValidBankAccount(bank_account)
    return True
dui_regex = r"^[0-9]{8}-[0-9]{1}$"
check_dui = re.compile(dui_regex).match
class NotAValidDui(schema.ValidationError):
 __doc__ = u"Por favor utilice el formato ########-# en este campo"
def validate_dui(dui):
    if not check_dui(dui):
        raise NotAValidDui(dui)
    return True
nit_regex = r"^[0-9]{4}-[0-9]{6}-[0-9]{3}-[0-9]{1}$"
check_nit = re.compile(nit_regex).match
class NotAValidNit(schema.ValidationError):
__doc__ = u"Por favor utilice el formato ####-######-###-# en este campo"
def validate_nit(nit):
```

```
    if not check_nit(nit):
        raise NotAValidNit(nit)
    return True
nup_regex = r"^[0-9]{12}$"
check_nup = re.compile(nup_regex).match
class NotAValidNup(schema.ValidationError):
__doc__ = u"Por favor introduzca los doce dígitos requeridos en este campo"
def validate_nup(nup):
    if not check_nup(nup):
        raise NotAValidNup(nup)
    return True
class ISirhi(interface.Interface):
    name = schema.TextLine( title=u"Nombre" )
    financial_mgr = schema.TextLine(title=u"Nombre del Administrador/a Financiero/a" )
rrhh_mgr = schema.TextLine(title=u"Nombre del Jefe/a de la Unidad de RRHH" )
class IUnitContainer(interface.Interface): pass
class IUnit(interface.Interface):
 code = schema.TextLine( title=u"Código")
name = schema.TextLine( title=u"Denominación")
```

```
class IVocabularyDataContainer(interface.Interface): pass

class IVocabularyData(interface.Interface):

 name = schema.TextLine(title=u"Nombre")

values = schema.List(title=u"Valores",
value_type=schema.TextLine(title=u"Valor"),min_length=1,
default=[u""])

display_name = schema.TextLine(title=u"Nombre de Despliegue")

class IMovement(interface.Interface):

starting_date = schema.Date(title=u"Fecha de Nombramiento (AAAA-MM-DD)")

ending_date = schema.Date( title=u"Fecha de Retiro (AAAA-MM-DD)" )

salary = schema.Float(title=u"Sueldo Básico",min=0.0)

reference = schema.TextLine(title=u"Número de Acuerdo o Contrato", required=False)

date_from = schema.Date( title=u"Desde (AAAA-MM-DD)", required=False )

date_to = schema.Date( title=u"Hasta (AAAA-MM-DD)",required=False )

status = schema.Choice(title=u"Status",vocabulary="PositionStatus")

cause = schema.Choice(title=u"Causa",
vocabulary="MovementCauses")

    # worker data

id = schema.Int(title=u"IntId de IPerson",)
```

```
class IPersonContainer(interface.Interface): pass

class IPerson(interface.Interface):
    # Personal Data
first_name = schema.TextLine(title=u"Nombre")
    last_name = schema.TextLine(title=u"Apellido")
    gender = schema.Choice(title=u"Sexo",vocabulary="Genders")
date_of_birth = schema.Date(title=u"Fecha de Nacimiento (AAAA-MM-DD)")
birthplace = schema.Choice( title=u"Lugar de Nacimiento",
        vocabulary="Cities", required=False)
    nationality = schema.Choice( title=u"Nacionalidad",
        vocabulary="Nationalities", required=False )
    marital_status = schema.Choice( title=u"Estado Civil",
        vocabulary="MaritalStatus", required=False )
    # Documents
    dui = schema.TextLine( title=u"Documento Único de Identidad",
        constraint=validate_dui)
    nit = schema.TextLine( title=u"Número de Identificación Tributaria",
        constraint=validate_nit)
    nup = schema.TextLine(title=u"Número Único Previsional",
        constraint=validate_nup)
    # Bank Account
```

```
    bank = schema.Choice(title=u"Banco",vocabulary="Banks")
        bank_account = schema.TextLine(title=u"Número de Cuenta Bancaria",constraint=validate_bank_account)
    # Institutional Data
code = schema.TextLine(title=u"Código de Empleada/o",
        constraint=validate_code)
name_isss = schema.TextLine( title=u"Nombre según ISSS",
        required=False)
name_pension = schema.TextLine(title=u"Nombre según AFP/INPEP/IPSFA", required=False )
pension_manager = schema.Choice(title=u"Administradorde Pensiones",
vocabulary="PensionManagers")
photo = schema.Bytes(title=u"Fotografía (JPG)",required=False )
    # Contact Data
address = schema.Text(title=u"Domicilio",required=False)
phone = schema.TextLine(title=u"Teléfono",
constraint=validate_phone, required=False, )
email = schema.TextLine(title=u"Correo Electrónico",
required=False)
class IRelative(interface.Interface):
relationship = schema.Choice(title=u"Parentesco",
vocabulary="Relationships",)
first_name = schema.TextLine(title=u"Nombre", )
```

```
last_name = schema.TextLine( title=u"Apellido")

gender = schema.Choice(title=u"Gender",vocabulary="Genders")

date_of_birth = schema.Date(title=u"Fecha de Nacimiento (AAAA-MM-DD)",required=False)

dependent = schema.Bool(title=u"Dependiente",default=False,

        required=False)

deceased = schema.Bool

 title=u"Fallecida/o",default=False, required=False)

class IForeignLanguage(interface.Interface):

    name = schema.Choice(  title=u"Idioma",
vocabulary="ForeignLanguages")

speaking = schema.Choice(title=u"Nivel de Habla",

        vocabulary="ForeignLanguageLevels")

 reading = schema.Choice(title=u"Nivel de Lectura",

        vocabulary="ForeignLanguageLevels")

writing = schema.Choice(title=u"Nivel de Escritura",

        vocabulary="ForeignLanguageLevels")

translating = schema.Choice(title=u"Nivel de Traducción",

        vocabulary="ForeignLanguageLevels")

class ISkill(interface.Interface):

name =
schema.Choice( title=u"Habilidad/Destreza",vocabulary="Skills")
```

```
class ITraining(interface.Interface):
    date = schema.Date( title=u"Fecha (AAAA-MM-DD)",)
type = schema.Choice(title=u"Tipo de Evento",
        vocabulary="TrainingTypes")
area = schema.Choice(title=u"Área",vocabulary="TrainingAreas")
giver = schema.TextLine(title=u"Dada por")
duration = schema.Int(title=u"Duración (Hrs)",min=1)
place = schema.TextLine(title=u"Lugar")
class IDegree(interface.Interface):
degree = schema.Choice( title=u"Grado
Obtenido",vocabulary="Degrees")
    institution = schema.TextLine(title=u"Centro Educativo")
place = schema.TextLine(title=u"Lugar")
starting_year = schema.Int(title=u"Año de Inicio",min=1900,
max=2100,required=False)
ending_year = schema.Int(title=u"Año de Fin",min=1900, max=2100,
        required=False )
    @interface.invariant
    def check_years(obj):
        if obj.ending_year and obj.ending_year < obj.starting_year:
            raise interface.Invalid(u"Año de Fin debe ser mayor que "
                                    u"Año de Inicio")
```

```
class IEmployment(interface.Interface):
    institution = schema.TextLine(title=u"Institución")
    sector = schema.Choice( title=u"Sector",vocabulary="Sectors")
    department = schema.TextLine(title=u"Unidad",required=False)
    position = schema.TextLine(title=u"Puesto")
starting_date = schema.Date(title=u"Fecha de Inicio (AAAA-MM-DD)",  )
ending_date = schema.Date( title=u"Fecha de Fin (AAAA-MM-DD)",
        required=False )
class IPayrollContainer(interface.Interface): pass
class IPayroll(interface.Interface):
code = schema.TextLine(title=u"Código")
name = schema.TextLine(title=u"Denominación")
date = schema.Date(title=u"Fecha (AAAA-MM-DD)",required=False)
month = schema.Choice(title=u"Mes Imputación",vocabulary="months")
year = schema.Int(title=u"Año Imputación",min=2000)
payment_date = schema.Date(title=u"Fecha Estimada para el Pago
(AAAA-MM-DD)",required=False)
type = schema.Choice(title=u"Tipo",vocabulary="PayrollTypes")
closed = schema.Bool(title=u"Cerrada?")
calculated = schema.Bool( title=u"Calculada?")
unit = schema.Choice( title=u"Unidad", vocabulary="Units")
budgetary_code = schema.Choice(title=u"Cifrado presupuestario",
```

```
vocabulary="BudgetaryCodes")
payment_method = schema.Choice(title=u"Forma de pago",
        vocabulary="PaymentMethods")
financing_source = schema.Choice(title=u"Presupuesto que financia",
        vocabulary="FinancingSources")
classification =
schema.Choice(title=u"Clasificación",vocabulary="Classifications")
comments = schema.Text(title=u"Comentario General", required=False)
class ICriteria(interface.Interface):
    pass
class IMaximumCriteria(interface.Interface):
 accumulation = schema.Choice( title=u"Acumulación Desc",
        vocabulary="Accumulations")
class IValueCriteria(ICriteria):
    value = schema.Float( title=u"Valor Unitario")
class IPercentageCriteria(ICriteria):
    percentage = schema.Float(title=u"Porcentaje")
group = schema.Choice(title=u"Agrupamiento de
Conceptos",vocabulary="PaymentReferenceGroups")
class ITableCriteria(ICriteria):
    table = schema.Choice(title=u"Tabla",vocabulary="Tables")
                group     =     schema.Choice(title=u"Agrupamiento     de
```

```
Conceptos",vocabulary="PaymentReferenceGroups")

class IMaximumTotalCriteria(IMaximumCriteria):

    value = schema.Float(title=u"Monto Máximo")

class IMaximumPercentageCriteria(IMaximumCriteria):

    percentage = schema.Float(title=u"Porcentaje")

group = schema.Choice(title=u"Agrupamiento de
Conceptos",vocabulary="PaymentReferenceGroups")

class ITableContainer(interface.Interface): pass

class ITable(interface.Interface):

    code = schema.TextLine(title=u"Código")

    name = schema.TextLine(title=u"Denominación")

class IRange(interface.Interface):

    code = schema.TextLine(title=u"Código")

    name = schema.TextLine(title=u"Denominación")

    minimum = schema.Float(title=u"Mínimo")

    maximum = schema.Float(title=u"Máximo",required=False)

    value = schema.Float(title=u"Valor")

 operator = schema.Choice(title=u"Operador",vocabulary="Operators")

base_value = schema.Float(title=u"Valor Base (%)")calculation =
schema.Choice(title=u"Cálculo",vocabulary="Calculations")

base_dif = schema.Float(title=u"Base Dif",default=0.0)

    @interface.invariant
```

```
    def checkBaseDif(range):
        if range.calculation == u"Base Dif" and range.base_dif == 0:
            raise interface.Invalid(u"Debe introducir un valor mayor que 0.0 "
            u"en Base Dif")
class IPaymentReferenceContainer(interface.Interface): pass
class IPaymentReference(interface.Interface):
    code = schema.TextLine(title=u"Código")
    name = schema.TextLine(title=u"Denominación")
type
=schema.Choice(title=u"Tipo",vocabulary="PaymentReferenceTypes")
klass=
schema.Choice(title=u"Clase",vocabulary="PaymentReferenceClasses")
order = schema.Int(title=u"Orden de Proceso")
automatic = schema.Bool(title=u"Generación Automática")
montly_salary = schema.Bool(title=u"Se trata del sueldo mensual?")
payrolls = schema.List(title=u"Planillas en las que
Interviene",value_type=schema.Choice(title=u"Planilla",vocabulary="Pa
yrollTypes"),unique=True,min_length=1)
base_value = schema.Choice(title=u"Cálculo del Valor
Base",vocabulary="BaseValueCalculations")
criteria = schema.Object(title=u"Datos para el Cálculo del Valor
Base",schema=ICriteria,required=False)
maximum_value = schema.Choice(title=u"Cálculo del Valor
```

Máximo",vocabulary="MaximumValueCalculations")

maximum_criteria = schema.Object(title=u"Datos para el Cálculo del Valor Máximo",schema=IMaximumCriteria,required=False)

class INoAutomaticPaymentReference(interface.Interface):

reference = schema.Object(title=u"Concepto de Pago",schema=IPaymentReference)

origin = schema.Choice(title=u"Origen",vocabulary="Origins")

month_from = schema.Choice(title=u"MesDesde",vocabulary="months")

year_from = schema.Int(title=u"Año Desde")

month_to = schema.Choice(title=u"Mes Hasta",vocabulary="months",required=False)

year_to = schema.Int(title=u"Año Hasta")

instalments = schema.Int(title=u"Cuotas",)

unit_price = schema.Float(title=u"Valor Unitario",required=False)

units = schema.Int(title=u"Unidades",default=1,)

amount = schema.Float(title=u"Monto",required=False)

payrolls = schema.List(title=u"Planillas en las que Interviene",value_type=schema.Choice(title=u"Planilla",vocabulary="PayrollTypes"),unique=True,min_length=1)

class IPaymentReferenceGroupContainer(interface.Interface): pass

class IPaymentReferenceGroup(interface.Interface):

code = schema.TextLine(title=u"Código")

```
name = schema.TextLine(title=u"Denominación")

references = schema.List(title=u"Conceptos de Pago",value_type=schema.Choice(title=u"Concepto",vocabulary="PaymentReferences"),unique=True,min_length=1)

class IPayrollPaymentReference(interface.Interface):

    code = schema.TextLine(title=u"Código")

    name = schema.TextLine(title=u"Denominación")

type= schema.Choice(title=u"Tipo",vocabulary="PaymentReferenceTypes")

order = schema.Int(title=u"Orden de Proceso")

result = schema.Float(title=u"Resultado",required=False)

class IPayrollRecord(interface.Interface):

name = schema.TextLine(title=u"Denominación")

first_name = schema.TextLine(title=u"Nombre")

last_name = schema.TextLine(title=u"Apellido")

code = schema.TextLine(title=u"Código Institucional",constraint=validate_code)

position_name = schema.TextLine(title=u"Nombre del Puesto")

budget_item = schema.Choice(title=u"Partida",vocabulary="BudgetItems")

budget_subitem = schema.TextLine(title=u"Subpartida")

wages = schema.Float(title=u"Importe Devengado")

deductions = schema.Float(title=u"Total Descuento")
```

```
net_salary = schema.Float(title=u"Importe Líquido")

references = schema.Dict(title=u"Conceptos de
Pago",key_type=schema.Int(title=u"Orden"),value_type=schema.Object
(title=u"Concepto",schema=IPayrollPaymentReference))

class IPayrollGenerator(interface.Interface):

def generate(payroll):

      """Generate a payroll"""

class IPositionsGetter(interface.Interface):

   def values(unit):

      """Return the positions of a unit"""

class IPosition(interface.Interface):

   name = schema.TextLine(title=u"Denominación")

   tasks = schema.Text(title=u"Principales Tareas",required=False)

occupational_code = schema.Choice(title=u"Código ocupacional
interno",vocabulary="OccupationalCodes")

remuneration =
schema.Choice(title=u"Remuneración",vocabulary="Remunerations")

payment_method = schema.Choice(title=u"Forma de
pago",vocabulary="PaymentMethods")

budgetary_code = schema.Choice(title=u"Cifrado
presupuestario",vocabulary="BudgetaryCodes")

budget_item =
schema.Choice(title=u"Partida",vocabulary="BudgetItems")
```

```
budget_subitem = schema.TextLine(title=u"Subpartida")

financing_source = schema.Choice(title=u"Presupuesto que
financia",vocabulary="FinancingSources")

hours = schema.Int(title=u"Horas diarias",min=1)

salary = schema.Float(title=u"Sueldo máximo",min=0.0)

classification =
schema.Choice(title=u"Clasificación",vocabulary="Classifications")

history =
schema.List(title=u"Historial",value_type=schema.Tuple(title=u"Nombre
Completo del Trabajador/Id de Movimiento"),default=[],required=False)

noautomatic = schema.List(title=u"Conceptos No
Automáticos",value_type=schema.Object(title=u"Concepto",schema=IN
oAutomaticPaymentReference),default=[],required=False)
```

payment.py

```
# -*- coding: utf8 -*-

import grok

import os

from persistent.list import PersistentList

from zope.app.container.interfaces import INameChooser

from zope.formlib import form

from zope.app.form.interfaces import WidgetInputError

from zope.traversing.api import getName

from zope import schema

from zope.component import getUtility

from sirh import interfaces

from sirh import vocabulary

class PaymentReferenceContainer(grok.Container):

grok.implements(interfaces.IPaymentReferenceContainer)

class PaymentReference(grok.Model):

grok.implements(interfaces.IPaymentReference)

def __init__(self, **data):

self.code = data.get("code")

self.name = data.get("name")

self.type = data.get("type")
```

```
self.klass = data.get("klass")
self.order = data.get("order")
self.automatic = data.get("automatic")
self.montly_salary = data.get("montly_salary")
self.payrolls = PersistentList(data.get("payrolls"))
self.base_value = data.get("base_value")
self.criteria = data.get("criteria")
self.maximum_value = data.get("maximum_value")
self.maximum_criteria = data.get("maximum_criteria")
class NoAutomaticPaymentReference(grok.Model):
grok.implements(interfaces.INoAutomaticPaymentReference)
def __init__(self, **data):
self.reference = data.get("reference")
self.origin = data.get("origin")
self.month_from = data.get("month_from")
self.year_from = data.get("year_from")
self.month_to = data.get("month_to")
self.year_to = data.get("year_to")
self.instalments = data.get("instalments")
self.unit_price = data.get("unit_price")
self.units = data.get("units")
```

self.amount = data.get("amount")

self.payrolls = PersistentList(data.get("payrolls"))

class ContainerView(grok.View):

grok.context(interfaces.IPaymentReferenceContainer)

grok.name("index")

grok.require("sirh.Admin")

def update(self):

self.references = sorted([ref for ref in self.context.values()],key=lambda x:x.order)

class AddPaymentReference(grok.Form):

grok.context(interfaces.IPaymentReferenceContainer)

grok.name("add")

grok.require("sirh.Admin")

form_fields = grok.Fields(interfaces.IPaymentReference).omit("criteria", "maximum_criteria")

template =
grok.PageTemplateFile(os.path.join(os.path.dirname(__file__),

"custom_templates",

"paymentreferenceform.pt"))

label = u"Agregar Concepto de Pago"

@grok.action("Agregar")

def add(self, **data):

reference = PaymentReference(**data)

name = INameChooser(self.context).chooseName(str(reference.order), reference)

self.context[name] = reference

self.redirect(self.url(reference))

@form.action("Cancelar", validator=lambda *a: ())

def cancel(self, action, data):

self.redirect(self.url(self.context))

def validate(self, action, data):

errors = super(AddPaymentReference, self).validate(action, data)

if errors: return errors

orders = [reference.order for reference in self.context.values()]

if data.get("order") in orders:

error = WidgetInputError("order", u"Orden de Proceso",

u"El orden de proceso introducido ya corresponde a otro concepto de pago. El orden de proceso debe ser único para cada concepto.")

errors.append(error)

return errors

codes = [reference.code for reference in self.context.values()]

if data.get("code") in codes:error = WidgetInputError("code", u"Código", u"El código introducido ya corresponde a otro concepto de pago. El código debe ser único para cada concepto.")

```
errors.append(error)
return errors
return errors
class PaymentReferenceView(grok.View):
grok.context(interfaces.IPaymentReference)
grok.name("index")
grok.require("sirh.Admin")
def update(self):
self.automatic = self.context.automatic and u"Si" or u"No"
self.montly_salary = self.context.montly_salary and u"Si" or u"No"
self.payrolls = "<ul>%s</ul>" % \("".join(["<li>%s</li>" % (payroll)
for payroll in self.context.payrolls]))
def criteria(self):
return self.context.criteria is not None
def base_value_calculable(self):
return self.context.base_value != u"No Definido"
def maximum_criteria(self):
return self.context.maximum_criteria is not None
def maximum_value_calculable(self):
if self.base_value_calculable() and self.context.base_value != u"Valor
Unitario" and self.context.maximum_value != u"No Definido":
return True
```

```
return False
class EditPaymentReference(grok.EditForm):
grok.context(interfaces.IPaymentReference)
grok.name("edit")
grok.require("sirh.Admin")
form_fields = grok.Fields(interfaces.IPaymentReference)
form_fields = form_fields.omit("criteria", "maximum_criteria")
template =
grok.PageTemplateFile(os.path.join(os.path.dirname(__file__),
"custom_templates",
"paymentreferenceform.pt"))
label = u"Editar Concepto de Pago"
@grok.action("Guardar")
def edit(self, **data):
old_base_value = self.context.base_value
old_maximum_value = self.context.maximum_value
self.applyData(self.context, **data)
new_base_value = self.context.base_value
new_maximum_value = self.context.maximum_value
if old_base_value != new_base_value:
self.context.criteria = None
if old_maximum_value != new_maximum_value:
```

```
self.context.maximum_criteria = None

self.redirect(self.url(self.context))

def validate(self, action, data):

errors = super(EditPaymentReference, self).validate(action, data)

if errors: return errors

orders = [reference.order

for reference in self.context.__parent__.values()

if reference is not self.context]

if data.get("order") in orders:

error = WidgetInputError("order",u"Orden de Proceso",u"El orden de
proceso introducido ya corresponde a otro concepto de pago. El orden
de proceso debe ser unico para cada concepto.")

errors.append(error)

return errors

codes = [reference.code for reference in self.context.__parent__.values()

 if reference is not self.context]

if data.get("code") in codes:

error = WidgetInputError("code",

 u"Código",

 u"El código introducido ya corresponde a otro concepto de pago. El
código debe ser único para cada concepto.")

errors.append(error)
```

```
return errors

old_base_value = self.context.base_value

new_base_value = data.get("base_value")

if old_base_value == new_base_value and (new_base_value ==
u"Porcentaje" or new_base_value == u"Tabla"):

order = data.get("order")

if self.context.criteria is not None:

references = self.context.criteria.group.references

max_order = max([reference.order for reference in references])

if order <= max_order:

error = WidgetInputError("order",u"Orden de Proceso",u"El orden de
proceso del concepto actual debe ser mayor al de los conceptos
contenidos en el agrupamiento seleccionado")

errors.append(error)

return errors

return errors

@form.action("Cancelar", validator=lambda *a: ())

def cancel(self, action, data):

self.redirect(self.url(self.context))

class PaymentReferenceVocabulary(vocabulary.BaseVocabulary):

def titleValue(self, value):

return "%s: %s" % (value.code, value.name)
```

```
class PaymentReferenceVocabularyFactory(object):
grok.provides(schema.interfaces.IVocabularyFactory)
def getPaymentReferenceContainer(self, context):
obj = context
while obj is not None:
if grok.interfaces.IApplication.providedBy(obj):
container = [value for value in obj.values()
if interfaces.IPaymentReferenceContainer.providedBy(value)][0]
return container
obj = obj.__parent__
raise ValueError("No Sirh found.")
def __call__(self, context):
container = self.getPaymentReferenceContainer(context)
values = sorted([ref for ref in container.values()],
key=lambda x:x.order)
return PaymentReferenceVocabulary(values)
class NoAutomaticPaymentReferenceVocabularyFactory(object):
grok.provides(schema.interfaces.IVocabularyFactory)
def __call__(self, context):
vocabulary = getUtility(schema.interfaces.IVocabularyFactory,
u"PaymentReferences")(context)
```

```
values = [term.value for term in vocabulary
if not term.value.automatic]
return PaymentReferenceVocabulary(values)
```

payroll.py

```
# -*- coding: utf8 -*-

import grok

from grok import index

import os

from decimal import Decimal

from zope import schema

from zope.app.container.interfaces import INameChooser

from zope.formlib import form

from zope.app.form.browser.textwidgets import escape

from zope.app.form.interfaces import WidgetInputError

from tempfile import TemporaryFile

from zope.component import getMultiAdapter, getUtility

from zope.app.form.browser.interfaces import IWidgetInputErrorView

from hurry.query import query

from zope.app.intid.interfaces import IIntIds

from zope.index.text.parsetree import ParseError

from persistent.dict import PersistentDict

from sirh import interfaces

# Models

class PayrollContainer(grok.Container):
```

```
grok.implements(interfaces.IPayrollContainer)
class Payroll(grok.Container):
grok.implements(interfaces.IPayroll)
def __init__(self, **data):
super(Payroll, self).__init__()
self.code = data.get("code")
self.name = data.get("name")
self.type = data.get("type")
self.date = data.get("date")
self.month = data.get("month")
self.year = data.get("year")
self.payment_date = data.get("payment_date")
self.comments = data.get("comments")
self.closed = data.get("closed")
self.calculated = data.get("calculated")
self.unit = data.get("unit")
self.budgetary_code = data.get("budgetary_code")
self.payment_method = data.get("payment_method")
self.financing_source = data.get("financing_source")
self.classification = data.get("classification")
# Views
```

```
class ContainerView(grok.View):
grok.context(interfaces.IPayrollContainer)
grok.name("index")
grok.require("sirh.Edit")
def update(self):
# XXX: ordernar despliegue de planillas
self.payrolls = self.context.values()
def closed(self, payroll):
return payroll.closed and u"Sí" or u"No"
class PayrollView(grok.View):
grok.context(interfaces.IPayroll)
grok.name("index")
grok.require("sirh.Edit")
class AddPayroll(grok.Form):
grok.context(interfaces.IPayrollContainer)
grok.name("add")
grok.require("sirh.Edit")
label = u"Agregar Planilla"
form_fields = grok.Fields(interfaces.IPayroll).omit("closed", "calculated")
template =
grok.PageTemplateFile(os.path.join(os.path.dirname(__file__),"custom_te
mplates","payrollform.pt"))
```

```
def setUpWidgets(self, *args, **kw):

super(AddPayroll, self).setUpWidgets(*args, **kw)

self.widgets["comments"].height = 2

if not self.widgets["budgetary_code"].hasValidInput():

if len(self.widgets["budgetary_code"].vocabulary) < 1:

self.widgets["budgetary_code"].vocabulary =
schema.vocabulary.SimpleVocabulary.fromValues([u"Introduzca cifrados
presupuestarios propios para esta Facultad"])

@grok.action("Agregar")

def add(self, **data):

payroll = Payroll(**data)

name = u"%s-%s" % (payroll.year, payroll.month)

name = INameChooser(self.context).chooseName(name, payroll)

self.context[name] = payroll

self.redirect(self.url(payroll))

@form.action("Cancelar", validator=lambda *a: ())

def cancel(self, action, data):

self.redirect(self.url(self.context))

def validate(self, action, data):

errors = super(AddPayroll, self).validate(action, data)

if errors: return errors

codes = [payroll.code for payroll in self.context.values()]
```

```
if data.get("code") in codes:
error = WidgetInputError("code",u"Código",u"El codigo de planilla ya fue
creado")
errors.append(error)
message = getMultiAdapter((error, self.request),
IWidgetInputErrorView).snippet()
self.widgets["code"].error = message
return errors
class EditPayroll(grok.EditForm):
grok.context(interfaces.IPayroll)
grok.name("edit")
grok.require("sirh.Edit")
label = u"Editar Planilla"
form_fields = grok.Fields(interfaces.IPayroll).omit("closed", "calculated")
template =
grok.PageTemplateFile(os.path.join(os.path.dirname(__file__),
"custom_templates","payrollform.pt"))
def setUpWidgets(self, *args, **kw):
super(EditPayroll, self).setUpWidgets(*args, **kw)
self.widgets["comments"].height = 2
@grok.action("Guardar")
def edit(self, **data):
```

```
changes = self.applyData(self.context, **data)
if changes:
self.flash(u"Los cambios fueron guardados")
self.redirect(self.url(self.context))
@form.action("Cancelar", validator=lambda *a: ())
def cancel(self, action, data):
self.redirect(self.url(self.context))
class PayrollRecord(grok.Model):
grok.implements(interfaces.IPayrollRecord)
def __init__(self, **data):
self.name = data.get("name")
self.first_name = data.get("first_name")
self.last_name = data.get("last_name")
self.code = data.get("code")
self.position_name = data.get("position_name")
self.budget_item = data.get("budget_item")
self.budget_subitem = data.get("budget_subitem")
self.wages = data.get("wages")
self.deductions = data.get("deductions")
self.net_salary = data.get("net_salary")
self.references = PersistentDict()
```

```
class PayrollPaymentReference(grok.Model):
grok.implements(interfaces.IPayrollPaymentReference)
def __init__(self, **data):
self.code = data.get("code")
self.name = data.get("name")
self.type = data.get("type")
self.order = data.get("order")
self.result = data.get("result", 0.0)
class Generate(grok.View):
grok.context(interfaces.IPayroll)
grok.require("sirh.Edit")
def update(self):
generator =
getUtility(interfaces.IPayrollGenerator,name=u"PayrollGenerator")
records = [(rec.last_name, rec)for rec in
generator.generate(self.context,self.request)]
self.records = [rec for k, rec in sorted(records)]
class Close(grok.View):
grok.context(interfaces.IPayroll)
grok.require("sirh.Edit")
def update(self):
if not self.context.closed:
```

```
generator =
getUtility(interfaces.IPayrollGenerator,name=u"PayrollGenerator")
for record in generator.generate(self.context,self.request):
name = INameChooser(self.context).chooseName(u"", record)
self.context[name] = record
self.context.closed = True
def render(self):
self.redirect(self.url(self.context))
class View(grok.View):
grok.context(interfaces.IPayroll)
grok.require("sirh.Edit")
def update(self):
self.records = sorted(self.context.values(),
key=lambda x:x.last_name)
months_vocab = getUtility(schema.interfaces.IVocabularyFactory,
name="months")()
self.month =
months_vocab.getTermByToken(str(self.context.month)).title
class Summary(grok.View):
grok.context(interfaces.IPayroll)
grok.require("sirh.Edit")
def faculty_name(self):
```

```
parent = self.context.unit
while not interfaces.ISirhi.providedBy(parent):
parent = parent.__parent__
if grok.interfaces.IApplication.providedBy(parent):
raise ValueError("No Sirhi or Sirh found.")
return parent.name
def payroll_records(self):
view = getMultiAdapter((self.context, self.request), name="view")
view.update()
return view.records
def payroll_month_name(self):
view = getMultiAdapter((self.context, self.request), name="view")
view.update()
return view.month
def update(self):
self.faculty = self.faculty_name()
self.records = self.payroll_records()
self.month = self.payroll_month_name()
wages = []
deductions = []
contributions = []
```

```
references = []
for record in self.records:
references.extend(record.references.values())
for reference in references:
if reference.type == u"Haber":
wages.append(reference)
if reference.type == u"Descuento":
deductions.append(reference)
if reference.type == u"Aporte Patronal":
contributions.append(reference)
w = {}
for reference in wages:
if reference.name not in w:
w[reference.name] = [Decimal("%.2f" % reference.result)]
else:
w[reference.name].append(Decimal("%.2f" % reference.result))
self.wages = [{"code": [w for w in wages if w.name == k][0].code,
"name": k, "total": sum(v)}
for k,v in sorted(w.items())]
d = {}
for reference in deductions:
if reference.name not in d:
```

```
d[reference.name] = [Decimal("%.2f" % reference.result)]

else:

d[reference.name].append(Decimal("%.2f" % reference.result))

self.deductions = [{"code": [d for d in deductions if d.name == k]
[0].code, "name": k, "total": abs(sum(v))}for k,v in sorted(d.items())]

c = {}

for reference in contributions:

if reference.name not in c:

c[reference.name] = [Decimal("%.2f" % reference.result)]

else:

c[reference.name].append(Decimal("%.2f" % reference.result))

self.contributions = [{"code": [c for c in contributions if c.name == k]
[0].code, "name": k, "total": sum(v)}

for k,v in sorted(c.items())]


self.wages_total = sum([Decimal("%.2f" % r.result) for r in references

if r.type == u"Haber"])

self.deductions_total = sum([Decimal("%.2f" % abs(r.result)) for r in
referencesif r.type == u"Descuento"])

self.contributions_total = sum([Decimal("%.2f" % r.result) for r in
references

if r.type == u"Aporte Patronal"])
```

```
class NetSalaries(grok.View):

grok.context(interfaces.IPayroll)

grok.require("sirh.Edit")

def faculty_name(self):

parent = self.context.unit

while not interfaces.ISirhi.providedBy(parent):

parent = parent.__parent__

if grok.interfaces.IApplication.providedBy(parent):

raise ValueError("No Sirhi or Sirh found.")

return parent.name

def payroll_records(self):

view = getMultiAdapter((self.context, self.request), name="view")

view.update()

return view.records

def payroll_month_name(self):

view = getMultiAdapter((self.context, self.request), name="view")

view.update()

return view.month

def update(self):

self.faculty = self.faculty_name()

self.records = self.payroll_records()
```

```
self.month = self.payroll_month_name()
self.banks = [{"name": u"","accounts": ({"bank_account": u"","code":
u"","full_name": u"","net_salary": u""},),"total": 0,},
]accounts_info = []
for record in self.records:
info = {}
person = grok.getSite()["personas"][record.code]
info["bank"] = person.bank
info["bank_account"] = person.bank_account
info["code"] = record.code
info["full_name"] = "%s, %s" % (person.last_name,
person.first_name,)
info["net_salary"] = Decimal("%.2f" % record.net_salary)
accounts_info.append(info)
banks = {}
for account in accounts_info:
if account["bank"] not in banks:
banks[account["bank"]] = {"name": account["bank"]}
banks[account["bank"]]["accounts"] = [account]
banks[account["bank"]]["total"] = [account["net_salary"]]
else:
banks[account["bank"]]["total"].append(account["net_salary"])
```

```
banks[account["bank"]]["accounts"].append(account)
for bank in banks.values():
bank["total"] = sum(bank["total"])
self.banks = sorted(banks.values(), key=lambda x:x["name"])
class ReferenceSearch(grok.View):
grok.context(interfaces.IPayroll)
grok.require("sirh.Edit")
def update(self, search=u""):
if not search:
self.flash(u"Especifique el nombre o código del concepto "u"de pago a
buscar")
else:
references = grok.getSite().__parent__["conceptos"]
self.result = sorted([r for r in references.values()
if search.lower() in r.name.lower() or
search.lower() in r.code.lower()],key=lambda x:x.name)
if not self.result:
self.flash(u"No se encontraron resultados para este "u"término de
búsqueda", "error")
self.search = search
# XXX: copied from PositionView
def breadcrumbs(self):
```

```
parents = []
parent = self.context.__parent__
while interfaces.IUnit.providedBy(parent):
parents.insert(0, {"url": self.url(parent),"name": escape(parent.name)})
parent = parent.__parent__
payroll_container = parent
faculty_name = payroll_container.__parent__.name
parents.insert(0, {"url": self.url(parent), "name": escape(faculty_name)})
parents.append({"url": self.url(self.context),
"name": escape(self.context.name)})
result = []
for parent in parents:
result.append('<a href="%(url)s">%(name)s</a>' % (parent))
return " / ".join(result)
class References(grok.View):
grok.context(interfaces.IPayroll)
grok.require("sirh.Edit")
def faculty_name(self):
parent = self.context.unit
while not interfaces.ISirhi.providedBy(parent):
parent = parent.__parent__
```

```
if grok.interfaces.IApplication.providedBy(parent):
raise ValueError("No Sirhi or Sirh found.")
return parent.name
def payroll_records(self):
view = getMultiAdapter((self.context, self.request), name="view")
view.update()
return view.records
def payroll_month_name(self):
view = getMultiAdapter((self.context, self.request), name="view")
view.update()
return view.month
def update(self):
self.faculty = self.faculty_name()
self.records = self.payroll_records()
self.month = self.payroll_month_name()
reference_name = self.request.get("id")
self.reference = None
if reference_name:
references = grok.getSite().__parent__["conceptos"]
if reference_name in references:
self.reference = references[reference_name]
```

```
if self.reference is None:
self.flash(u"Se especificó un concepto de pago inexistente","error")
self.redirect(self.url(self.context, "referencesearch"))
return
self.records = [record for record in self.records
if self.reference.order in record.references.keys()]
if not self.records:
self.flash(u"El concepto de pago seleccionado no fue aplicado "u"en esta planilla", "error")
self.redirect(self.url(self.context, "referencesearch"))
return
self.net_salary_total = sum([Decimal("%.2f" % record.net_salary)for record in self.records])
self.reference_result_total = sum([Decimal(self.reference_result(record))for record in self.records])
def reference_result(self, record):
return "%.2f" % abs(record.references[self.reference.order].result)
# XXX: copied from PositionView
def breadcrumbs(self):
parents = []
parent = self.context.__parent__
while interfaces.IUnit.providedBy(parent):
```

```
parents.insert(0, {"url": self.url(parent),"name": escape(parent.name)})
parent = parent.__parent__
payroll_container = parent
faculty_name = payroll_container.__parent__.name
parents.insert(0, {"url": self.url(parent), "name": escape(faculty_name)})
result = []
for parent in parents:
result.append('<a href="%(url)s">%(name)s</a>' % (parent))
result.append(escape(self.context.name))
return " / ".join(result)
class PayrollGenerator(object):
grok.implements(interfaces.IPayrollGenerator)
def getPaymentReferenceContainer(self, context):
obj = context
while obj is not None:
if grok.interfaces.IApplication.providedBy(obj):
container = [value for value in obj.values()if
interfaces.IPaymentReferenceContainer.providedBy(value)][0]
return container
obj = obj.__parent__
raise ValueError("No Sirhi or Sirh found.")
def generate(self, payroll, request):
```

```
self.payroll = payroll

utility = getUtility(interfaces.IPositionsGetter, name=u"Positions")

positions = utility.values(payroll.unit)

references = self.getPaymentReferenceContainer(payroll).values()

self.automatic = [(reference.order, reference)

for reference in references

if reference.automatic == True and

payroll.type in reference.payrolls]

records = [self.createRecord(position, request) for position in positionsif
not getMultiAdapter((position, request), name=u"is_empty")()
and(getattr(getMultiAdapter((position, request),
name="last_movement")(), "status", None) == u"Activo" or

getattr(getMultiAdapter((position, request), name="last_movement")(),
"status", None) == u"Licencia con Goce de Sueldo")
andposition.budgetary_code == payroll.budgetary_code
andposition.payment_method == payroll.payment_method
andposition.financing_source == payroll.financing_source
andposition.classification == payroll.classification]

return records

def createRecord(self, position, request):

result = {}

person = getMultiAdapter((position, request), name=u"get_person")()

result["name"] = position.name
```

```
result["first_name"] = person.first_name
result["last_name"] = person.last_name
result["code"] = person.code
result["budget_item"] = position.budget_item
result["budget_subitem"] = position.budget_subitem
result["position_name"] = position.name
record = PayrollRecord(**result)
cnas = [(cna.reference.order, cna)
for cna in position.noautomatic
if self.includeCNA(cna)]
cnas.extend(self.automatic)
references = sorted(cnas)
for k, reference in references:
result = 0
if interfaces.INoAutomaticPaymentReference.providedBy(reference):
if reference.reference.base_value == u"No Definido":
result = reference.amount
if reference.reference.type == u"Descuento":
result = result * -1
info = {}
info["code"] = reference.reference.code
```

```
info["name"] = reference.reference.name
info["type"] = reference.reference.type
info["order"] = reference.reference.order
info["result"] = result
ppr = PayrollPaymentReference(**info)
record.references[ppr.order] = ppr
continue
else:
reference = reference.reference
if reference.montly_salary:
result = getMultiAdapter((position, request), name=u"last_movement")
().salary
if reference.base_value == u"Valor Unitario":
result = reference.criteria.value
if reference.type == u"Descuento":
result = result * -1
if reference.base_value == u"Porcentaje":
group_result = 0
group_refs = []
if reference.criteria is not None:
group_refs = [ref for ref in reference.criteria.group.references]
for ref in group_refs:
```

```
if ref.order in record.references:
group_result += record.references[ref.order].result
if reference.criteria is not None:
result = reference.criteria.calculate(group_result)
if reference.maximum_value == u"Monto Máximo" and
\reference.maximum_criteria is not None:
if result and reference.maximum_criteria.value < result:
result = reference.maximum_criteria.value
if reference.maximum_value == u"Porcentaje":
group_result = 0
group_refs = []
if reference.maximum_criteria is not None:
group_refs = [ref for ref in
reference.maximum_criteria.group.references]
for ref in group_refs:
if ref.order in record.references:
group_result += record.references[ref.order].result
if reference.maximum_criteria is not None:
max_result = reference.maximum_criteria.calculate(group_result)
if result and max_result < result:
result = max_result
if reference.type == u"Descuento":
```

```
result = result * -1
if reference.base_value == u"Tabla":
group_result = 0
group_refs = []
if reference.criteria is not None:
group_refs = [ref for ref in reference.criteria.group.references]
for ref in group_refs:
if ref.order in record.references:
group_result += record.references[ref.order].result
if reference.criteria is not None:
result = reference.criteria.calculate(group_result)
if reference.maximum_value == u"Monto Máximo" and
\reference.maximum_criteria is not None:
if result and reference.maximum_criteria.value < result:
result = reference.maximum_criteria.value
if reference.maximum_value == u"Porcentaje":
group_result = 0
group_refs = []
if reference.maximum_criteria is not None:
group_refs = [ref for ref in
reference.maximum_criteria.group.references]
for ref in group_refs:
```

```
if ref.order in record.references:
group_result += record.references[ref.order].result
if reference.maximum_criteria is not None:
max_result = reference.maximum_criteria.calculate(group_result)
if result and max_result < result:
result = max_result
if reference.type == u"Descuento":
result = result * -1
info = {}
info["code"] = reference.code
info["name"] = reference.name
info["type"] = reference.type
info["order"] = reference.order
info["result"] = result
ppr = PayrollPaymentReference(**info)
record.references[ppr.order] = ppr
record.wages = sum([ref.result for ref in record.references.values() if
ref.type == u"Haber"])
record.deductions = abs(sum([ref.result for ref in
record.references.values() if ref.type == u"Descuento"]))
record.net_salary = sum([ref.result for ref in record.references.values() if
ref.type != u"Aporte Patronal"])
```

```
return record


def includeCNA(self, cna):
if self.payroll.type in cna.payrolls:
date_from = int("%d%02d" % (cna.year_from, cna.month_from))
date_to = int("%d%02d" % (cna.year_to, cna.month_to))
date_payroll = int("%d%02d" % (self.payroll.year,self.payroll.month))
if date_from <= date_payroll <= date_to:
return True
return False


class PayslipSearch(grok.View):
grok.context(interfaces.IPayroll)
grok.name("payslipsearch")
grok.require("sirh.Edit")
def update(self, search=u""):
if not search:
self.flash(u"Especifique un código institucional, nombre o "u"apellido
para buscar")
else:
try:
results = query.Query().searchResults(
```

```
query.Eq((u"", "code"), search) |
query.Text((u"","text_code"), search) |
query.Eq((u"", "first_name"), search) |
query.Text((u"", "text_first_name"), search) |
query.Eq((u"", "last_name"), search) |
query.Text((u"", "text_last_name"), search)
)
self.result = sorted(list(results), key=lambda x:x.last_name)
except (TypeError, ParseError):
pass
if not hasattr(self, "result") or not self.result:
self.flash(u"No se encontraron resultados para este "u"término de
búsqueda", "error")
self.search = search
def payslip_url(self, person):
intid = getUtility(IIntIds).queryId(person)
return "%s?id=%s" % (self.url("payslip"), intid)
# XXX: copied from PositionView
def breadcrumbs(self):
parents = []
parent = self.context.__parent__
while interfaces.IUnit.providedBy(parent):
```

```
parents.insert(0, {"url": self.url(parent),"name": escape(parent.name)})
parent = parent.__parent__
payroll_container = parent
faculty_name = payroll_container.__parent__.name
parents.insert(0, {"url": self.url(parent), "name": escape(faculty_name)})
parents.append({"url": self.url(self.context),
"name": escape(self.context.name)})
result = []
for parent in parents:
result.append('<a href="%(url)s">%(name)s</a>' % (parent))
return " / ".join(result)

class Payslip(grok.View):
grok.context(interfaces.IPayroll)
grok.require("sirh.Edit")
def update(self):
intids = getUtility(IIntIds)
try:
person = intids.queryObject(int(self.request.get("id", "")))
self.records = [record for record in self.context.values()
if record.code == person.code]
```

except (ValueError, TypeError, IndexError,):

self.flash(u"No se puede generar la boleta de pago para "u"esta persona. No fue incluida en la generación "u"de esta planilla", "error")

self.redirect(self.url(self.context, "payslipsearch"))

return

self.result = [getMultiAdapter((record, self.request), name="index")()for record in self.records]

self.result = "".join(self.result)

self.payroll = self.context

net_salary = 0

for record in self.records:

references = [r for r in record.references.values()if r.type != u"Aporte Patronal"]

wages = [r.result for r in references if r.result >= 0.0]

deductions = [r.result for r in references if r.result < 0]

net_salary += Decimal("%.2f" % (sum(wages) + sum(deductions)))

self.net_salary = net_salary

self.employeer = self.payroll.__parent__.__parent__

months_vocab = getUtility(schema.interfaces.IVocabularyFactory,

name="months")()

self.month = months_vocab.getTermByToken(str(self.payroll.month)).title

```
self.person = person
# XXX: copied from PositionView
## def breadcrumbs(self):
## parents = []
## parent = self.context.__parent__
## while interfaces.IUnit.providedBy(parent):
## parents.insert(0, {"url": self.url(parent),
##"name": escape(parent.name)})
## parent = parent.__parent__
## payroll_container = parent
## faculty_name = payroll_container.__parent__.name
## parents.insert(0, {"url": self.url(parent), "name":
escape(faculty_name)})
## result = []
## for parent in parents:
## result.append('<a href="%(url)s">%(name)s</a>' % (parent))
## result.append(escape(self.context.name))
## return " / ".join(result)
def breadcrumbs(self):
parents = []
parents.append({"url": self.url(self.context),
"name": escape(self.context.name)})
```

```
result = []
for parent in parents:
result.append('<a href="%(url)s">%(name)s</a>' % (parent))
result.append(escape(self.person.first_name + " " +
self.person.last_name))
return " / ".join(result)

class PayrollRecordView(grok.View):
grok.context(interfaces.IPayrollRecord)
grok.name("index")
grok.require("sirh.Edit")
def update(self):
references = [r for r in self.context.references.values()if r.type !=
u"Aporte Patronal"]
self.wages = [r for r in references if r.result > 0.0]
self.deductions = [r for r in references if r.result < 0.0]
```

permissions.py

```
# -*- coding: utf-8 -*-

import grok



class Edit(grok.Permission):
    grok.name("sirh.Edit")



class Admin(grok.Permission):
    grok.name("sirh.Admin")
```

person.py

```
# -*- coding: utf8 -*-


import grok

import datetime

from grok import index

import os

from zope.app.container.interfaces import INameChooser

from zope.formlib import form

from zope.app.form.browser.textwidgets import escape

from zope.app.form.interfaces import WidgetInputError

from tempfile import TemporaryFile

from zope.component import getMultiAdapter, getUtility

from zope.app.form.browser.interfaces import IWidgetInputErrorView

from hurry.query import query

from zope.app.intid.interfaces import IIntIds

from zope import schema

from z3c.table import table, column

import z3c.table

from zope import interface
```

```
from sirh import interfaces
from sirh.num2word.num2word_ES import to_card
from sirh import vocabulary

# Models

class PersonContainer(grok.Container):

    grok.implements(interfaces.IPersonContainer)



class Person(grok.Container):

    grok.implements(interfaces.IPerson)

    def __init__(self, **data):
        super(Person, self).__init__()
        # personal data
        self.first_name = data.get("first_name")
        self.last_name = data.get("last_name")
        self.gender = data.get("gender")
```

```
self.date_of_birth = data.get("date_of_birth")
self.birthplace = data.get("birthplace")
self.nationality = data.get("nationality")
self.marital_status = data.get("marital_status")
# documents
self.dui = data.get("dui")
self.nit = data.get("nit")
self.nup = data.get("nup")
# bank info
self.bank = data.get("bank")
self.bank_account = data.get("bank_account")
# institutional info
self.code = data.get("code")
self.name_isss = data.get("name_isss")
self.name_pension = data.get("name_pension")
self.pension_manager = data.get("pension_manager")
self.photo = data.get("photo")
# contact info
self.address = data.get("address")
self.phone = data.get("phone")
self.email = data.get("email")
```

```
    def __eq__(self, other):
        return self.dui == other.dui or self.nit == other.nit or \
                self.nup == other.nup or self.code == other.code


class Relative(grok.Model):

    grok.implements(interfaces.IRelative)

    def __init__(self, **data):
        self.relationship = data.get("relationship")
        self.first_name = data.get("first_name")
        self.last_name = data.get("last_name")
        self.gender = data.get("gender")
        self.date_of_birth = data.get("date_of_birth")
        self.dependent = data.get("dependent")
        self.deceased = data.get("deceased")


class ForeignLanguage(grok.Model):
```

```
    grok.implements(interfaces.IForeignLanguage)

    def __init__(self, **data):
        self.name = data.get("name")
        self.speaking = data.get("speaking")
        self.reading = data.get("reading")
        self.writing = data.get("writing")
        self.translating = data.get("translating")

    def __eq__(self, other):
        return self.name == other.name


class Skill(grok.Model):

    grok.implements(interfaces.ISkill)

    def __init__(self, **data):
        self.name = data.get("name")
```

```
    def __eq__(self, other):
        return self.name == other.name


class Training(grok.Model):

    grok.implements(interfaces.ITraining)

    def __init__(self, **data):
        self.type = data.get("type")
        self.area = data.get("area")
        self.date = data.get("date")
        self.duration = data.get("duration")
        self.place = data.get("place")
        self.giver = data.get("giver")


class Degree(grok.Model):

    grok.implements(interfaces.IDegree)
```

```
    def __init__(self, **data):
        self.degree = data.get("degree")
        self.institution = data.get("institution")
        self.place = data.get("place")
        self.starting_year = data.get("starting_year")
        self.ending_year = data.get("ending_year")


class Employment(grok.Model):

    grok.implements(interfaces.IEmployment)

    def __init__(self, **data):
        self.institution = data.get("institution")
        self.sector = data.get("sector")
        self.department = data.get("department")
        self.position = data.get("position")
        self.starting_date = data.get("starting_date")
        self.ending_date = data.get("ending_date")
```

```
class PersonIndexes(grok.Indexes):

    grok.site(interfaces.ISirhi)
    grok.context(interfaces.IPerson)

    code = index.Field()
    text_code = index.Text(attribute="code")
    first_name = index.Field()
    text_first_name = index.Text(attribute="first_name")
    last_name = index.Field()
    text_last_name = index.Text(attribute="last_name")
    dui = index.Field()
    nit = index.Field()
    nup = index.Field()


# Views

personal = ["first_name",
            "last_name",
            "gender",
```

```
            "date_of_birth",
            "birthplace",
            "nationality",
            "marital_status"]
documents = ["dui",
             "nit",
             "nup"]
bank = ["bank",
        "bank_account"]
institutional = ["code",
                 "name_isss",
                 "name_pension",
                 "pension_manager",
                 "photo"]
contact = ["address",
           "phone",
           "email"]


class ContainerView(grok.View):
```

```
    grok.context(interfaces.IPersonContainer)
    grok.name("index")

    def update(self):
        self.values = sorted(self.context.values(),
                             key=lambda x:x.last_name)

class ColumnBase(grok.MultiAdapter, column.GetAttrColumn):
    grok.adapts(interface.Interface,
                interface.Interface,
                z3c.table.interfaces.ITable)
    grok.provides(z3c.table.interfaces.IColumn)
    grok.baseclass()

class CodeColumn(ColumnBase):
    grok.name("codecolumn")

    header = u"Código Institucional"
    attrName = "code"
    weight = 1
```

```
class NameColumn(ColumnBase):
    grok.name("namecolumn")

    header = u"Apellido, Nombre"
    attrName = "last_name"
    weight = 2
    cssClasses = {"th": "underline"}

    def getSortKey(self, item):
        return item.last_name

    def getValue(self, item):
        return "%s, %s" % (item.last_name, item.first_name)

class GenderColumn(ColumnBase):
    grok.name("gendercolumn")

    header = u"Sexo"
    attrName = "gender"
    weight = 3
```

```
    def getValue(self, item):
        if item.gender == u"Masculino": return "M"
        if item.gender == u"Femenino": return "F"

class DateOfBirthColumn(ColumnBase):
    grok.name("dateofbirthcolumn")

    header = u"Fecha de Nacimiento"
    attrName = "date_of_birth"
    weight = 4

class BirthPlaceColumn(ColumnBase):
    grok.name("birthplacecolumn")

    header = u"Lugar de Nacimiento"
    attrName = "birthplace"
    weight = 5

class NationalityColumn(ColumnBase):
    grok.name("nationalitycolumn")
```

```
    header = u"Nacionalidad"
    attrName = "nationality"
    weight = 6

class MaritalStatusColumn(ColumnBase):
    grok.name("maritalstatuscolumn")

    header = u"Estado Civil"
    attrName = "marital_status"
    weight = 7

class StartingDateColumn(ColumnBase):
    grok.name("startingdatecolumn")

    header = u"Fecha de Nombramiento"
    weight = 8

    def renderCell(self, item):
        positions = getMultiAdapter((item, self.request), name="positions")
()
        result = []
        for position in positions:
```

```
            movement = getMultiAdapter((position, self.request),
                                       name="last_movement")()
            if movement.starting_date:
                result.append(str(movement.starting_date))
            else:
                result.append("-")
        return "<br />".join(result)

class DateFromColumn(ColumnBase):
    grok.name("datefromcolumn")

    header = u"Contrato desde"
    weight = 9

    def renderCell(self, item):
        positions = getMultiAdapter((item, self.request), name="positions")
()
        result = []
        for position in positions:
            movement = getMultiAdapter((position, self.request),
                                       name="last_movement")()
            if movement.date_from:
```

```
                result.append(str(movement.date_from))
            else:
                result.append("-")
        return "<br />".join(result)


class DateToColumn(ColumnBase):
    grok.name("datetocolumn")

    header = u"Contrato hasta"
    weight = 10

    def renderCell(self, item):
        positions = getMultiAdapter((item, self.request), name="positions")
()
        result = []
        for position in positions:
            movement = getMultiAdapter((position, self.request),
                                       name="last_movement")()
            if movement.date_to:
                result.append(str(movement.date_to))
            else:
```

```
                result.append("-")
        return "<br />".join(result)

class Employees(grok.View):

    grok.context(interfaces.IPersonContainer)
    grok.name("employees")
    grok.require("sirh.Edit")

    def update(self):
        self.table = table.Table(self.context, self.request)
        self.table.sortOn = u"table-namecolumn-1"
        self.table.cssClasses = {"table": "border"}
        self.table.update()

class PersonView(grok.View):

    grok.context(interfaces.IPerson)
    grok.name("index")

    def getPerson(self):
```

```
    if interfaces.IPerson.providedBy(self.context):
        return self.context
    return self.context.__parent__

def sortValuesProviding(self, iface, key):
    person = self.getPerson()
    return sorted([item for item in person.values()
                   if iface.providedBy(item)],
                  key=key)

def relatives(self):
    return self.sortValuesProviding(interfaces.IRelative,
                                    lambda x:x.last_name)

def languages(self):
    return self.sortValuesProviding(interfaces.IForeignLanguage,
                                    lambda x:x.name)

def trainings(self):
    return self.sortValuesProviding(interfaces.ITraining,
                                    lambda x:x.date)
```

```
    def skills(self):
        return self.sortValuesProviding(interfaces.ISkill,
                                        lambda x:x.name)

    def degrees(self):
        return self.sortValuesProviding(interfaces.IDegree,
                                        lambda x:x.degree)

    def employments(self):
        return self.sortValuesProviding(interfaces.IEmployment,
                                        lambda x:(x.sector,x.institution))


class FieldSets(object):

    def getTheseWidgets(self, names):
        result = []
        for name in names:
            if self.widgets.get(name):
                result.append(self.widgets.get(name))
```

```
        return result

    def fieldsets(self):
        return [
            {"legend": u"Datos Personales",
             "widgets": self.getTheseWidgets(personal)},
            {"legend": u"Datos de Contacto",
             "widgets": self.getTheseWidgets(contact)},
            {"legend": u"Datos Institucionales",
             "widgets": self.getTheseWidgets(institutional)},
            {"legend": u"Datos Bancarios",
             "widgets": self.getTheseWidgets(bank)},
            {"legend": u"Documentos",
             "widgets": self.getTheseWidgets(documents)},
            ]


class AddPerson(grok.Form, FieldSets):

    grok.context(interfaces.IPersonContainer)
    grok.name("add")
```

```
    grok.require("sirh.Edit")


    form_fields = grok.Fields(interfaces.IPerson)
    template =
grok.PageTemplateFile(os.path.join(os.path.dirname(__file__),
                                       "custom_templates",
                                       "personform.pt"))
    label = u"Agregar Persona"


    @grok.action(u"Agregar")
    def add(self, **data):
        person = Person(**data)
        name = INameChooser(self.context).chooseName(person.code,
person)
        self.context[name] = person
        self.redirect(self.url(self.context))


    @form.action("Cancelar", validator=lambda *a: ())
    def cancel(self, action, data):
        self.redirect(self.url(self.context))


    def setUpWidgets(self, *args, **kw):
```

```
    super(AddPerson, self).setUpWidgets(*args, **kw)
    self.widgets["address"].height = 2

def validate(self, action, data):
    errors = super(AddPerson, self).validate(action, data)
    if errors: return errors

    q = query.Query()
    result = q.searchResults(query.Eq((u"", "code"), data.get("code")))
    count = list(result)
    if count:
        error = WidgetInputError("code",
                            u"Código Institucional",
                            u"El número ya existe en otro empleado")
        errors.append(error)
        message = getMultiAdapter((error, self.request),
                             IWidgetInputErrorView).snippet()
        self.widgets["code"].error = message


    result = q.searchResults(query.Eq((u"", "dui"), data.get("dui")))
```

```
count = list(result)
if count:
    error = WidgetInputError("dui",
                             u"DUI",
                             u"El dui corresponde a otro empleado")
    errors.append(error)
    message = getMultiAdapter((error, self.request),
                              IWidgetInputErrorView).snippet()
    self.widgets["dui"].error = message


result = q.searchResults(query.Eq((u"", "nit"), data.get("nit")))
count = list(result)
if count:
    error = WidgetInputError("nit",
                             u"NIT",
                             u"El nit corresponde a otro empleado")
    errors.append(error)
    message = getMultiAdapter((error, self.request),
                              IWidgetInputErrorView).snippet()
    self.widgets["nit"].error = message
```

```
result = q.searchResults(query.Eq((u"", "nup"), data.get("nup")))
count = list(result)
if count:
    error = WidgetInputError("nup",
                             u"NUP",
                             u"El nup corresponde a otro empleado")
    errors.append(error)
    message = getMultiAdapter((error, self.request),
                              IWidgetInputErrorView).snippet()
    self.widgets["nup"].error = message

if errors:

    s = u"Se detectaron datos repetidos en el perfil de otro "
    s += u"trabajador que deberían ser únicos para cada persona"

    self.flash(s, "error")

return errors
```

```
class EditPersonal(grok.EditForm, PersonView):

    grok.context(interfaces.IPerson)
    grok.name("personal")

    form_fields = grok.Fields(interfaces.IPerson).select(*personal)
    template =
grok.PageTemplateFile(os.path.join(os.path.dirname(__file__),
                                       "custom_templates",
                                       "persondataform.pt"))

    @grok.action(u"Guardar")
    def save(self, **data):
        changes = self.applyData(self.context, **data)
        if changes:
            self.flash(u"Los cambios fueron guardados")
        self.redirect(self.url(self.context))

    @form.action("Cancelar", validator=lambda *a: ())
    def cancel(self, action, data):
```

```
        self.redirect(self.url(self.context))

class EditContact(grok.EditForm, PersonView):

    grok.context(interfaces.IPerson)
    grok.name("contact")

    form_fields = grok.Fields(interfaces.IPerson).select(*contact)
    template =
grok.PageTemplateFile(os.path.join(os.path.dirname(__file__),
                                       "custom_templates",
                                       "personcontactform.pt"))

    @grok.action(u"Guardar")
    def save(self, **data):
        changes = self.applyData(self.context, **data)
        if changes:
            self.flash(u"Los cambios fueron guardados")
        self.redirect(self.url(self.context))

    @form.action("Cancelar", validator=lambda *a: ())
    def cancel(self, action, data):
```

```
        self.redirect(self.url(self.context))

    def setUpWidgets(self, *args, **kw):
        super(EditContact, self).setUpWidgets(*args, **kw)
        self.widgets["address"].height = 2



class EditInstitutional(grok.EditForm, PersonView):

    grok.context(interfaces.IPerson)
    grok.name("institutional")

    form_fields = grok.Fields(interfaces.IPerson).select(*institutional)
    template =
grok.PageTemplateFile(os.path.join(os.path.dirname(__file__),
                                       "custom_templates",
                                       "personinstitutionalform.pt"))

    @grok.action(u"Guardar")
    def save(self, **data):
        if data.get("photo") is None:
```

```
        data["photo"] = self.context.photo
    changes = self.applyData(self.context, **data)
    if changes:
        self.flash(u"Los cambios fueron guardados")
    self.redirect(self.url(self.context))

@form.action("Cancelar", validator=lambda *a: ())
def cancel(self, action, data):
    self.redirect(self.url(self.context))

def validate(self, action, data):
    errors = super(EditInstitutional, self).validate(action, data)
    if errors: return errors
    if data.get("code") != self.context.code:
        result = query.Query().searchResults(query.Eq((u"", "code"),
                                             data.get("code")))
        count = list(result)
        if count:
            error = WidgetInputError("code",
                                     u"Código Institucional",
                                     u"El número ya existe en otro empleado")
```

```
            errors.append(error)
            message = getMultiAdapter((error, self.request),
                                      IWidgetInputErrorView).snippet()
            self.widgets["code"].error = message
        return errors

class EditBank(grok.EditForm, PersonView):

    grok.context(interfaces.IPerson)
    grok.name("bank")

    form_fields = grok.Fields(interfaces.IPerson).select(*bank)
    template =
grok.PageTemplateFile(os.path.join(os.path.dirname(__file__),
                                       "custom_templates",
                                       "personbankform.pt"))

    @grok.action(u"Guardar")
    def save(self, **data):
        changes = self.applyData(self.context, **data)
        if changes:
            self.flash(u"Los cambios fueron guardados")
```

```
        self.redirect(self.url(self.context))

    @form.action("Cancelar", validator=lambda *a: ())
    def cancel(self, action, data):
        self.redirect(self.url(self.context))


class EditDocs(grok.EditForm, PersonView):

    grok.context(interfaces.IPerson)
    grok.name("docs")

    form_fields = grok.Fields(interfaces.IPerson).select(*documents)
    template =
grok.PageTemplateFile(os.path.join(os.path.dirname(__file__),
                                   "custom_templates",
                                   "persondocsform.pt"))

    @grok.action(u"Guardar")
    def save(self, **data):
        changes = self.applyData(self.context, **data)
        if changes:
```

```
        self.flash(u"Los cambios fueron guardados")
    self.redirect(self.url(self.context))

@form.action("Cancelar", validator=lambda *a: ())
def cancel(self, action, data):
    self.redirect(self.url(self.context))

def validate(self, action, data):
    errors = super(EditDocs, self).validate(action, data)
    if errors: return errors
    q = query.Query()
    if data.get("dui") != self.context.dui:
        result = q.searchResults(query.Eq((u"", "dui"), data.get("dui")))
        count = list(result)
        if count:
            error = WidgetInputError("dui",
                                     u"DUI",
                                     u"El dui corresponde a otro empleado")
            errors.append(error)
            message = getMultiAdapter((error, self.request),
                                      IWidgetInputErrorView).snippet()
```

```
            self.widgets["dui"].error = message
        if data.get("nit") != self.context.nit:
            result = q.searchResults(query.Eq((u"", "nit"), data.get("nit")))
            count = list(result)
            if count:
                error = WidgetInputError("nit",
                                         u"NIT",
                                         u"El nit corresponde a otro empleado")
                errors.append(error)
                message = getMultiAdapter((error, self.request),
                                          IWidgetInputErrorView).snippet()
                self.widgets["nit"].error = message
        if data.get("nup") != self.context.nup:
            result = q.searchResults(query.Eq((u"", "nup"), data.get("nup")))
            count = list(result)
            if count:
                error = WidgetInputError("nup",
                                         u"NUP",
                                         u"El nup corresponde a otro empleado")
                errors.append(error)
                message = getMultiAdapter((error, self.request),
```

```
                                IWidgetInputErrorView).snippet()
            self.widgets["nup"].error = message
        return errors


class Photo(grok.View):

    grok.context(interfaces.IPerson)
    grok.name("photo")

    def render(self):
        self.response.setHeader("Content-Type", "image/jpeg")
        photo = TemporaryFile()
        photo.write(self.context.photo)
        return photo


class AddRelative(grok.Form, PersonView):

    grok.context(interfaces.IPerson)
    grok.name("relative")
```

```
    form_fields = grok.Fields(interfaces.IRelative)
    template =
grok.PageTemplateFile(os.path.join(os.path.dirname(__file__),
                                   "custom_templates",
                                   "personrelativeform.pt"))

    @grok.action(u"Agregar")
    def add(self, **data):
        relative = Relative(**data)
        name = INameChooser(self.context).chooseName(u"", relative)
        self.context[name] = relative
        self.redirect(self.url(self.context))

    @form.action("Cancelar", validator=lambda *a: ())
    def cancel(self, action, data):
        self.redirect(self.url(self.context))


class EditRelative(grok.EditForm, PersonView):

    grok.context(interfaces.IRelative)
```

```
    grok.name("edit")

    form_fields = grok.Fields(interfaces.IRelative)
    template =
grok.PageTemplateFile(os.path.join(os.path.dirname(__file__),
                                       "custom_templates",
                                       "personrelativeeditform.pt"))

    @grok.action(u"Guardar")
    def save(self, **data):
        changes = self.applyData(self.context, **data)
        if changes:
            self.flash(u"Los cambios fueron guardados")
        self.redirect(self.url(self.context.__parent__))

    @form.action("Cancelar", validator=lambda *a: ())
    def cancel(self, action, data):
        self.redirect(self.url(self.context.__parent__))

class AddLanguage(grok.Form, PersonView):

    grok.context(interfaces.IPerson)
```

```
    grok.name("language")

    form_fields = grok.Fields(interfaces.IForeignLanguage)
    template =
grok.PageTemplateFile(os.path.join(os.path.dirname(__file__),
                                       "custom_templates",
                                       "personlanguageform.pt"))

    @grok.action(u"Agregar")
    def add(self, **data):
        language = ForeignLanguage(**data)
        name = INameChooser(self.context).chooseName(u"", language)
        self.context[name] = language
        self.redirect(self.url(self.context))

    @form.action("Cancelar", validator=lambda *a: ())
    def cancel(self, action, data):
        self.redirect(self.url(self.context))

    def validate(self, action, data):
        errors = super(AddLanguage, self).validate(action, data)
        if errors: return errors
```

```
        if data.get("name") in [l.name for l in self.languages()]:
            error = WidgetInputError("name",
                                     u"Idioma",
                                     u"Este idioma ya se agregó al perfil")
            errors.append(error)
            message = getMultiAdapter((error, self.request),
                                      IWidgetInputErrorView).snippet()
            self.widgets["name"].error = message
            return errors
        return errors


class EditLanguage(grok.EditForm, PersonView):

    grok.context(interfaces.IForeignLanguage)
    grok.name("edit")

    form_fields = grok.Fields(interfaces.IForeignLanguage["name"],
for_display=True) +
grok.Fields(interfaces.IForeignLanguage).omit("name")
    template =
grok.PageTemplateFile(os.path.join(os.path.dirname(__file__),
```

```
                                    "custom_templates",
                                    "personlanguageeditform.pt"))

    @grok.action(u"Guardar")
    def save(self, **data):
        changes = self.applyData(self.context, **data)
        if changes:
            self.flash(u"Los cambios fueron guardados")
        self.redirect(self.url(self.context.__parent__))

    @form.action("Cancelar", validator=lambda *a: ())
    def cancel(self, action, data):
        self.redirect(self.url(self.context.__parent__))

class AddSkill(grok.Form, PersonView):

    grok.context(interfaces.IPerson)
    grok.name("skill")

    form_fields = grok.Fields(interfaces.ISkill)
    template =
grok.PageTemplateFile(os.path.join(os.path.dirname(__file__),
```

```
                                    "custom_templates",
                                    "personskillform.pt"))

@grok.action(u"Agregar")
def add(self, **data):
    skill = Skill(**data)
    name = INameChooser(self.context).chooseName(u"", skill)
    self.context[name] = skill
    self.redirect(self.url(self.context))

@form.action("Cancelar", validator=lambda *a: ())
def cancel(self, action, data):
    self.redirect(self.url(self.context))

def validate(self, action, data):
    errors = super(AddSkill, self).validate(action, data)
    if errors: return errors
    if data.get("name") in [s.name for s in self.skills()]:
        error = WidgetInputError("name",
                                 u"Habilidad/Destreza",
                                 u"Esta habilidad/destreza ya se agregó "
```

```
                                u"al perfil")
            errors.append(error)
            message = getMultiAdapter((error, self.request),
                                      IWidgetInputErrorView).snippet()
            self.widgets["name"].error = message
            return errors
        return errors


class EditSkill(grok.EditForm, PersonView):

    grok.context(interfaces.ISkill)
    grok.name("edit")

    form_fields = grok.Fields(interfaces.ISkill)
    template =
grok.PageTemplateFile(os.path.join(os.path.dirname(__file__),
                                       "custom_templates",
                                       "personskilleditform.pt"))

    @grok.action(u"Guardar")
    def save(self, **data):
```

```
    changes = self.applyData(self.context, **data)
    if changes:
        self.flash(u"Los cambios fueron guardados")
    self.redirect(self.url(self.context.__parent__))

@form.action("Cancelar", validator=lambda *a: ())
def cancel(self, action, data):
    self.redirect(self.url(self.context.__parent__))

def validate(self, action, data):
    errors = super(EditSkill, self).validate(action, data)
    if errors: return errors
    if data.get("name") in [s.name for s in self.skills()]:
        error = WidgetInputError("name",
                                 u"Habilidad/Destreza",
                                 u"Esta habilidad/destreza ya se agregó "
                                 u"al perfil")
        errors.append(error)
        message = getMultiAdapter((error, self.request),
                                  IWidgetInputErrorView).snippet()
        self.widgets["name"].error = message
```

```
    return errors

return errors
```

```
class AddTraining(grok.Form, PersonView):

    grok.context(interfaces.IPerson)
    grok.name("training")

    form_fields = grok.Fields(interfaces.ITraining)
    template =
grok.PageTemplateFile(os.path.join(os.path.dirname(__file__),
                                       "custom_templates",
                                       "persontrainingform.pt"))

    @grok.action(u"Agregar")
    def add(self, **data):
```

```
        training = Training(**data)
        name = INameChooser(self.context).chooseName(u"", training)
        self.context[name] = training
        self.redirect(self.url(self.context))

    @form.action("Cancelar", validator=lambda *a: ())
    def cancel(self, action, data):
        self.redirect(self.url(self.context))



class EditTraining(grok.EditForm, PersonView):

    grok.context(interfaces.ITraining)
    grok.name("edit")

    form_fields = grok.Fields(interfaces.ITraining)
    template =
grok.PageTemplateFile(os.path.join(os.path.dirname(__file__),
                                   "custom_templates",
                                   "persontrainingeditform.pt"))
```

```
@grok.action(u"Guardar")
def save(self, **data):
    changes = self.applyData(self.context, **data)
    if changes:
        self.flash(u"Los cambios fueron guardados")
    self.redirect(self.url(self.context.__parent__))

@form.action("Cancelar", validator=lambda *a: ())
def cancel(self, action, data):
    self.redirect(self.url(self.context.__parent__))
```

```
class AddDegree(grok.Form, PersonView):
```

```
    grok.context(interfaces.IPerson)
    grok.name("degree")

    form_fields = grok.Fields(interfaces.IDegree)
    template =
grok.PageTemplateFile(os.path.join(os.path.dirname(__file__),
                                   "custom_templates",
                                   "persondegreeform.pt"))

    @grok.action(u"Agregar")
    def add(self, **data):
        degree = Degree(**data)
        name = INameChooser(self.context).chooseName(u"", degree)
        self.context[name] = degree
        self.redirect(self.url(self.context))

    @form.action("Cancelar", validator=lambda *a: ())
    def cancel(self, action, data):
        self.redirect(self.url(self.context))
```

```
class EditDegree(grok.EditForm, PersonView):

    grok.context(interfaces.IDegree)
    grok.name("edit")

    form_fields = grok.Fields(interfaces.IDegree)
    template =
grok.PageTemplateFile(os.path.join(os.path.dirname(__file__),
                                       "custom_templates",
                                       "persondegreeeditform.pt"))

    @grok.action(u"Guardar")
    def save(self, **data):
        changes = self.applyData(self.context, **data)
        if changes:
            self.flash(u"Los cambios fueron guardados")
        self.redirect(self.url(self.context.__parent__))

    @form.action("Cancelar", validator=lambda *a: ())
    def cancel(self, action, data):
        self.redirect(self.url(self.context.__parent__))
```

```
class AddEmployment(grok.Form, PersonView):


    grok.context(interfaces.IPerson)
    grok.name("employment")


    form_fields = grok.Fields(interfaces.IEmployment)
    template =
grok.PageTemplateFile(os.path.join(os.path.dirname(__file__),
                                   "custom_templates",
```

```
                                    "personemploymentform.pt"))

    @grok.action(u"Agregar")
    def add(self, **data):
        employment = Employment(**data)
        name = INameChooser(self.context).chooseName(u"",
employment)
        self.context[name] = employment
        self.redirect(self.url(self.context))

    @form.action("Cancelar", validator=lambda *a: ())
    def cancel(self, action, data):
        self.redirect(self.url(self.context))



class EditEmployment(grok.EditForm, PersonView):

    grok.context(interfaces.IEmployment)
    grok.name("edit")

    form_fields = grok.Fields(interfaces.IEmployment)
```

```
    template =
grok.PageTemplateFile(os.path.join(os.path.dirname(__file__),
                                       "custom_templates",
                                       "personemploymenteditform.pt"))

    @grok.action(u"Guardar")
    def save(self, **data):
        changes = self.applyData(self.context, **data)
        if changes:
            self.flash(u"Los cambios fueron guardados")
        self.redirect(self.url(self.context.__parent__))

    @form.action("Cancelar", validator=lambda *a: ())
    def cancel(self, action, data):
        self.redirect(self.url(self.context.__parent__))




class Positions(grok.View):

    grok.context(interfaces.IPerson)
```

```
    grok.name("positions")

    def update(self):
        result = []
        intid = getUtility(IIntIds).queryId(self.context)
        results = query.Query().searchResults(query.Eq((u"", "id"), intid))
        movements = [movement for movement in results
                     if movement.cause is None]
        for movement in movements:
            result.append(movement.__parent__)
        self.positions = set(result)

    def render(self):
        return self.positions



year = schema.Int(
    __name__ = "year",
    title=u"Año",
    min=1900,
    max=2100,
```

```
    )

taxed_salary = schema.List(
    __name__ = "taxed_salary",
    title=u"Sueldo Gravado",
    value_type=schema.Choice(title=u"Concepto",
vocabulary="Haberes"),
    unique=True,
    min_length=1,
    )

pension_deductions = schema.List(
    __name__ = "pension_deductions",
    title=u"Fondo de Pensiones",
    value_type=schema.Choice(title=u"Concepto",
vocabulary="Descuentos"),
    unique=True,
    min_length=1,
    )

income = schema.List(
    __name__ = "income",
```

```
    title=u"Ingresos",
    value_type=schema.Choice(title=u"Concepto",
vocabulary="Haberes"),
    unique=True,
    default=[]
    )

deductions = schema.List(
    __name__ = "deductions",
    title=u"Egresos",
    value_type=schema.Choice(title=u"Concepto",
vocabulary="Descuentos"),
    unique=True,
    min_length=1,
    )


class IncomeVocabulary(vocabulary.BaseVocabulary):

    def titleValue(self, value):
        return "%s: %s" % (value.code, value.name)
```

```
class IncomeVocabularyFactory(grok.GlobalUtility):
    grok.name("Haberes")
    grok.provides(schema.interfaces.IVocabularyFactory)

    def getPaymentReferenceContainer(self, context):
        obj = context
        while obj is not None:
            if grok.interfaces.IApplication.providedBy(obj):
                container = [value for value in obj.values()
                             if
interfaces.IPaymentReferenceContainer.providedBy(value)][0]
                return container
            obj = obj.__parent__
        raise ValueError("No Sirh found.")

    def __call__(self, context):
        container = self.getPaymentReferenceContainer(context)
        values = sorted([ref for ref in container.values()
                         if ref.type == u"Haber"],
                        key=lambda x:x.code)
        return IncomeVocabulary(values)
```

```
class DeductionVocabulary(vocabulary.BaseVocabulary):

    def titleValue(self, value):
        return "%s: %s" % (value.code, value.name)



class DeductionVocabularyFactory(grok.GlobalUtility):
    grok.name("Descuentos")
    grok.provides(schema.interfaces.IVocabularyFactory)

    def getPaymentReferenceContainer(self, context):
        obj = context
        while obj is not None:
            if grok.interfaces.IApplication.providedBy(obj):
                container = [value for value in obj.values()
                             if
interfaces.IPaymentReferenceContainer.providedBy(value)][0]
                return container
            obj = obj.__parent__
        raise ValueError("No Sirh found.")
```

```
    def __call__(self, context):
        container = self.getPaymentReferenceContainer(context)
        values = sorted([ref for ref in container.values()
                         if ref.type == u"Descuento"],
                        key=lambda x:x.code)
        return DeductionVocabulary(values)


class IncomeTaxForm(grok.Form):

    grok.context(interfaces.IPerson)

    form_fields = grok.Fields(year, taxed_salary, pension_deductions,
                              income, deductions)

    template =
grok.PageTemplateFile(os.path.join(os.path.dirname(__file__),
                                           "custom_templates",
                                           "incometaxform.pt"))
    label = u"Constancia de Renta - Paso 1 de 2: Selección de Conceptos"
```

```
    @grok.action(u"Generar")
    def generate(self, **data):
        self.redirect(self.url(self.context, "incometax", data={"year":
data["year"], "taxed_salary": [s.code for s in data["taxed_salary"]],
"pension_deductions": [d.code for d in data["pension_deductions"]],
"income": [s.code for s in data["income"]], "deductions": [d.code for d in
data["deductions"]]}))

    @form.action("Cancelar", validator=lambda *a: ())
    def cancel(self, action, data):
        self.redirect(self.url(self.context))


class IncomeTax(grok.View):

    grok.context(interfaces.IPerson)

    def faculty_name(self):
        parent = self.context
        while not interfaces.ISirhi.providedBy(parent):
            parent = parent.__parent__
            if grok.interfaces.IApplication.providedBy(parent):
```

```
                raise ValueError("No Sirhi or Sirh found.")
        return parent.name


    def getPaymentReferenceContainer(self, context):
        obj = context
        while obj is not None:
            if grok.interfaces.IApplication.providedBy(obj):
                container = [value for value in obj.values()
                             if
interfaces.IPaymentReferenceContainer.providedBy(value)][0]
                return container
            obj = obj.__parent__
        raise ValueError("No Sirh found.")


    def getPayrollContainer(self, context):
        obj = context
        while obj is not None:
            if interfaces.ISirhi.providedBy(obj):
                container = [value for value in obj.values()
                             if interfaces.IPayrollContainer.providedBy(value)][0]
                return container
            obj = obj.__parent__
```

```
        raise ValueError("No Sirh found.")


    def update(self):
        references_codes = [reference.code for reference in
self.getPaymentReferenceContainer(self.context).values()]
        try:
            year = int(self.request.get("year"))
        except (ValueError, TypeError,):
            self.flash(u"El año introducido no es válido", "error")
            self.redirect(self.url(self.context))
            return
        self.year = to_card(year)
        taxed_salary = self.request.get("taxed_salary")
        deductions = self.request.get("deductions")
        pension_deductions = self.request.get("pension_deductions")
        income = self.request.get("income")
        if not taxed_salary:
            self.flash(u"No se seleccionaron conceptos para el sueldo
gravado",
                       "error")
            self.redirect(self.url(self.context))
            return
```

```
        else:
            if not isinstance(taxed_salary, list):
                taxed_salary = [taxed_salary,]
            taxed_salary = [ref for ref in taxed_salary
                            if ref in references_codes]
            if not taxed_salary:
                self.flash(u"No se seleccionaron conceptos para el sueldo
gravado",
                           "error")
                self.redirect(self.url(self.context))
                return
        if not deductions:
            self.flash(u"No se seleccionaron conceptos para los egresos",
                       "error")
            self.redirect(self.url(self.context))
            return
        else:
            if not isinstance(deductions, list):
                deductions = [deductions,]
            deductions = [ref for ref in deductions
                          if ref in references_codes]
            if not deductions:
```

```
            self.flash(u"No se seleccionaron conceptos para los egresos",
                       "error")
            self.redirect(self.url(self.context))
            return
        if not pension_deductions:
            self.flash(u"No se seleccionaron conceptos para el fondo de
pensiones",
                       "error")
            self.redirect(self.url(self.context))
            return
        else:
            if not isinstance(pension_deductions, list):
                pension_deductions = [pension_deductions,]
            pension_deductions = [ref for ref in pension_deductions
                                  if ref in references_codes]
            if not pension_deductions:
                self.flash(u"No se seleccionaron conceptos para el fondo de
pensiones",
                           "error")
                self.redirect(self.url(self.context))
                return
        if income:
```

```
            if not isinstance(income, list):
                income = [income]
            income = [ref for ref in income
                      if ref in references_codes]
        else:
            income = []


        records = self.getPayrollRecords(year)
        if not records:
            self.flash(u"No se encontraron registros de planilla para esta
persona en el año especificado", "error")
            self.redirect(self.url(self.context))
            return
        self.total_taxed_salary = 0
        self.total_income = 0
        self.total_pension_deductions = 0
        self.total_deductions = 0
        self.deductions = {}
        for record in records:
            for reference in record.references.values():
                if reference.code in taxed_salary:
                    self.total_taxed_salary += abs(reference.result)
```

```
            if reference.code in income:
                self.total_income += abs(reference.result)
            if reference.code in pension_deductions:
                self.total_pension_deductions += abs(reference.result)
            if reference.code in deductions:
                if reference.name not in self.deductions:
                    self.deductions[reference.name] = [abs(reference.result)]
                else:
                    self.deductions[reference.name].append(abs(reference.re
sult))
                self.total_deductions += abs(reference.result)
        self.total_income = self.total_taxed_salary + self.total_income -
self.total_pension_deductions
        self.deductions = [{"name": k, "result": "%.2f" % sum(v)}
                           for k,v in self.deductions.items()]
        self.date = datetime.date.today()
        months_vocab = getUtility(schema.interfaces.IVocabularyFactory,
                                  name="months")()
        month = months_vocab.getTermByToken(str(self.date.month)).title
        self.date_string = u"%s días del mes de %s de %s" %
(to_card(self.date.day), month.lower(), to_card(self.date.year))
```

```
def getPayrollRecords(self, year):
    container = self.getPayrollContainer(self.context)
    payrolls = [payroll for payroll in container.values()
                if payroll.year == year]
    result = []
    for payroll in payrolls:
        result.extend([record for record in payroll.values()
                       if record.code == self.context.code])
    return result
```

position.py

```
# -*- coding: utf-8 -*-


import grok

from datetime import datetime, date

import time

from grok import index

import os

from zope.formlib import form

from zope.app.container.interfaces import INameChooser

from zope.app.form.browser.textwidgets import escape

from zope.component import getUtility, getMultiAdapter

from zope.app.catalog.interfaces import ICatalog

from zope.index.text.parsetree import ParseError

from hurry.query import query

from zope.session.interfaces import ISession

from zope.app.intid.interfaces import IIntIds

from zope.app.form.interfaces import WidgetInputError

from zope.app.form.browser.interfaces import IWidgetInputErrorView

from persistent.list import PersistentList

from zope import schema, i18n
```

```
from zope.traversing.api import getName

from sirh import interfaces
from sirh import vocabulary
from sirh import payment
from sirh.num2word.num2word_ES import to_card
from sirh.payroll import PayrollRecord, PayrollPaymentReference

# Models

class Position(grok.Container):

    grok.implements(interfaces.IPosition)

    def __init__(self, **data):
        super(Position, self).__init__()
        self.name = data.get("name")
        self.tasks = data.get("tasks")
        self.occupational_code = data.get("occupational_code")
        self.remuneration = data.get("remuneration")
        self.payment_method = data.get("payment_method")
```

```
        self.budgetary_code = data.get("budgetary_code")
        self.budget_item = data.get("budget_item")
        self.budget_subitem = data.get("budget_subitem")
        self.financing_source = data.get("financing_source")
        self.hours = data.get("hours")
        self.salary = data.get("salary")
        self.classification = data.get("classification")
        self.history = PersistentList(data.get("history"))
        self.noautomatic = PersistentList(data.get("noautomatic"))

    def __eq__(self, other):
        if self.__parent__ is other.__parent__ and self.name == other.name:
            return True
        return False


class Movement(grok.Model):

    grok.implements(interfaces.IMovement)

    def __init__(self, **data):
```

```
        self.starting_date = data.get("starting_date")
        self.ending_date = data.get("ending_date")
        self.salary = data.get("salary")
        self.reference = data.get("reference")
        self.date_from = data.get("date_from")
        self.date_to = data.get("date_to")
        self.status = data.get("status")
        self.cause = data.get("cause")
        self.id = data.get("id")


# Indexes

class MovementIndexes(grok.Indexes):

    grok.site(interfaces.ISirhi)
    grok.context(interfaces.IMovement)

    id = index.Field()
```

```
# Views

class PositionView(grok.View):

    grok.context(Position)
    grok.name("index")

    def breadcrumbs(self):
        parents = []
        parent = self.context.__parent__
        while interfaces.IUnit.providedBy(parent):
            parents.insert(0, {"url": self.url(parent),
                               "name": escape(parent.name)})
            parent = parent.__parent__
        parents.insert(0, {"url": self.url(parent), "name": u"Facultad"})
        result = []
        parents.append({"url": self.url(self.context), "name":
escape(self.context.name)})
        for parent in parents:
            result.append('<a href="%(url)s">%(name)s</a>' % (parent))
        return " / ".join(result)
```

```
def empty(self):
    movements = [movement for movement in self.context.values()
                 if movement.cause is None]
    if not movements:
        return True
    assert(len(movements) == 1)
    return False

def person(self):
    movements = [movement for movement in self.context.values()
                 if movement.cause is None]
    intids = getUtility(IIntIds)
    return intids.queryObject(movements[0].id, None)

def movements(self):
    result = []
    intids = getUtility(IIntIds)
    for person_name, movement_id in self.context.history:
        movement = intids.queryObject(movement_id, None)
        result.append({"full_name": person_name,
                       "starting_date": movement.starting_date,
```

```
                    "ending_date": movement.ending_date,
                    "cause": movement.cause})
        return result


    def noautomatic(self):
        cnas = [(cna.reference.name, cna) for cna in
self.context.noautomatic]
        return [cna for k, cna in sorted(cnas)]

    def cna_url(self, cna):
        index = self.context.noautomatic.index(cna)
        return "%s/cna?index=%s" % (self.url(self.context), index)

    def edit_url(self, cna):
        index = self.context.noautomatic.index(cna)
        return "%s/editcna?index=%s" % (self.url(self.context), index)

    def delete_url(self, cna):
        index = self.context.noautomatic.index(cna)
        return "%s/delete?index=%s" % (self.url(self.context), index)
```

```
class AddPosition(grok.Form):

    grok.context(interfaces.IUnit)
    grok.name("position")
    grok.require("sirh.Edit")

    form_fields = grok.Fields(interfaces.IPosition).omit("history",
"noautomatic")
    template =
grok.PageTemplateFile(os.path.join(os.path.dirname(__file__),
                                       "custom_templates",
                                       "positionform.pt"))

    def __init__(self, *args, **kw):
        super(AddPosition, self).__init__(*args, **kw)
        self.label = u"Agregar Puesto a %s" % (self.context.name)

    @grok.action("Agregar")
    def add(self, **data):
        position = Position(**data)
```

```
        name = INameChooser(self.context).chooseName(u"", position)
        self.context[name] = position
        self.redirect(self.url(self.context))

    def setUpWidgets(self, *args, **kw):
        super(AddPosition, self).setUpWidgets(*args, **kw)
        if not self.widgets["budgetary_code"].hasValidInput():
            if len(self.widgets["budgetary_code"].vocabulary) < 1:
                self.widgets["budgetary_code"].vocabulary =
schema.vocabulary.SimpleVocabulary.fromValues([u"Introduzca cifrados
presupuestarios propios para esta Facultad"])
        if not self.widgets["budget_item"].hasValidInput():
            if len(self.widgets["budget_item"].vocabulary) < 1:
                self.widgets["budget_item"].vocabulary =
schema.vocabulary.SimpleVocabulary.fromValues([u"Introduzca partidas
presupuestarias propias para esta Facultad"])
        self.widgets["tasks"].height = 3

    @form.action("Cancelar", validator=lambda *a: ())
    def cancel(self, action, data):
        self.redirect(self.url(self.context))
```

```
class EditPosition(grok.EditForm):

    grok.context(interfaces.IPosition)
    grok.name("edit")
    grok.require("sirh.Edit")

    form_fields = grok.Fields(interfaces.IPosition).omit("history",
"noautomatic")
    template =
grok.PageTemplateFile(os.path.join(os.path.dirname(__file__),
                                       "custom_templates",
                                       "positionform.pt"))

    def __init__(self, *args, **kw):
        super(EditPosition, self).__init__(*args, **kw)
        self.label = u"Editar %s" % (self.context.name)

    @grok.action("Guardar")
    def edit(self, **data):
        changes = self.applyData(self.context, **data)
        if changes:
```

```
            self.flash(u"Los cambios fueron guardados")
        self.redirect(self.url(self.context))

    def setUpWidgets(self, *args, **kw):
        super(EditPosition, self).setUpWidgets(*args, **kw)
        self.widgets["tasks"].height = 3

    @form.action("Cancelar", validator=lambda *a: ())
    def cancel(self, action, data):
        self.redirect(self.url(self.context))

class Filled(grok.View):

    grok.context(interfaces.IPosition)
    grok.name("filled")

    def update(self):
        self.result = bool([movement for movement in self.context.values()
                            if movement.cause is None])

    def render(self):
```

```
        return self.result


class FillSearch(grok.View):

    grok.context(interfaces.IPosition)
    grok.name("fillsearch")
    grok.require("sirh.Edit")

    def update(self, search=u""):
        if not search:
            self.flash(u"Especifique un código institucional, nombre o "
                       u"apellido para buscar")
        else:
            try:
                results = query.Query().searchResults(
                    query.Eq((u"", "code"), search) |
                    query.Text((u"","text_code"), search) |
                    query.Eq((u"", "first_name"), search) |
                    query.Text((u"", "text_first_name"), search) |
                    query.Eq((u"", "last_name"), search) |
```

```
                query.Text((u"", "text_last_name"), search)
                )
            self.result = sorted(list(results), key=lambda x:x.last_name)
        except (TypeError, ParseError):
            pass
        if not hasattr(self, "result") or not self.result:
            self.flash(u"No se encontraron resultados para este "
                       u"término de búsqueda", "error")
    self.search = search

def fill_url(self, person):
    intid = getUtility(IIntIds).queryId(person)
    return "%s?id=%s" % (self.url("fill"), intid)

# XXX: copied from PositionView
def breadcrumbs(self):
    parents = []
    parent = self.context.__parent__
    while interfaces.IUnit.providedBy(parent):
        parents.insert(0, {"url": self.url(parent),
                           "name": escape(parent.name)})
```

```
            parent = parent.__parent__
        parents.insert(0, {"url": self.url(parent), "name": u"Facultad"})
        result = []
        parents.append({"url": self.url(self.context), "name":
escape(self.context.name)})
        for parent in parents:
            result.append('<a href="%(url)s">%(name)s</a>' % (parent))
        return " / ".join(result)



class Fill(grok.Form):


    grok.context(interfaces.IPosition)
    grok.name("fill")
    grok.require("sirh.Edit")


    form_fields = grok.Fields(interfaces.IMovement).omit("ending_date",
                                            "cause", "id")
    template =
grok.PageTemplateFile(os.path.join(os.path.dirname(__file__),
                                       "custom_templates",
                                       "positionfillform.pt"))
```

```
def isValidId(self, id):
    try:
        int(id)
    except (ValueError, TypeError):
        return False
    return True

def getRequestId(self):
    return self.request.get("id")

def isAPerson(self, id):
    intids = getUtility(IIntIds)
    try:
        id = int(id)
        person = intids.queryObject(id, None)
    except (ValueError, TypeError):
        return False
    if person is not None and interfaces.IPerson.providedBy(person):
        return True
    return False
```

```
def setSessionId(self, id):
    session = ISession(self.request)["sirh"]
    session["id"] = int(id)

def resetSession(self):
    session = ISession(self.request)["sirh"]
    if session.get("id"):
        del(session["id"])

def leave(self):
    self.resetSession()
    self.redirect(self.url(self.context, "fillsearch"))

def getPerson(self, id):
    intids = getUtility(IIntIds)
    try:
        return intids.queryObject(int(id))
    except (ValueError, TypeError):
        return
```

```
def getSessionId(self):
    session = ISession(self.request)["sirh"]
    return session.get("id")

def isThereSessionId(self):
    if self.getSessionId() is not None:
        return True
    return False

def update(self, *args, **kw):
    super(Fill, self).update(*args, **kw)
    if self.getRequestId():
        if self.isValidId(self.getRequestId()):
            intid = int(self.getRequestId())
            if self.isAPerson(intid):
                self.setSessionId(intid)
                self.person = self.getPerson(intid)
            else:
                self.leave()
        else:
            self.leave()
```

```
        else:
            if not self.isThereSessionId():
                self.leave()
            intid = self.getSessionId()
            if self.isValidId(intid):
                self.setSessionId(intid)
                self.person = self.getPerson(intid)
            else:
                self.leave()


    # XXX: copied from PositionView
    def breadcrumbs(self):
        parents = []
        parent = self.context.__parent__
        while interfaces.IUnit.providedBy(parent):
            parents.insert(0, {"url": self.url(parent),
                               "name": escape(parent.name)})
            parent = parent.__parent__
        parents.insert(0, {"url": self.url(parent), "name": u"Facultad"})
        result = []
        parents.append({"url": self.url(self.context), "name":
escape(self.context.name)})
```

```
    for parent in parents:
        result.append('<a href="%(url)s">%(name)s</a>' % (parent))
    return " / ".join(result)

@grok.action("Ocupar")
def add(self, **data):
    data["id"] = self.getSessionId()
    movement = Movement(**data)
    name = INameChooser(self.context).chooseName(u"", movement)
    self.context[name] = movement
    self.resetSession()
    self.redirect(self.url(self.context.__parent__))

@form.action("Cancelar", validator=lambda *a: ())
def cancel(self, action, data):
    self.resetSession()
    self.redirect(self.url(self.context.__parent__))

label = u"Paso 2 de 2: Detalles del movimiento"

def validate(self, action, data):
```

```
        errors = super(Fill, self).validate(action, data)
        if errors: return errors

        if float(data.get("salary")) > self.context.salary:

            error = WidgetInputError("salary",
                                     u"Sueldo Básico",
                                     u"El sueldo básico no puede ser mayor al "
                                     u"sueldo máximo para la plaza")
            errors.append(error)
            message = getMultiAdapter((error, self.request),
                                      IWidgetInputErrorView).snippet()
            self.widgets["salary"].error = message
        return errors



class Empty(grok.Form):

    grok.context(interfaces.IPosition)
    grok.name("empty")
```

```
    grok.require("sirh.Edit")


    form_fields = grok.Fields(interfaces.IMovement).select("ending_date",
"cause")
    template =
grok.PageTemplateFile(os.path.join(os.path.dirname(__file__),
                                       "custom_templates",
                                       "positionemptyform.pt"))
    label = u"Desocupar Puesto"


    @grok.action("Desocupar")
    def empty(self, **data):
        # Buscar ultimo movimiento
        latestid = getUtility(IIntIds).queryId(self.latest)
        self.latest.cause = data.get("cause")
        self.latest.ending_date = data.get("ending_date")
        # XXX: se agregan al historial el id del trabajador y del movimiento
        full_name = "%s %s" % (self.person.first_name,
self.person.last_name)
        self.context.history.append((full_name, latestid))
        self.latest.id = None
        self.redirect(self.url(self.context.__parent__))
```

```
def update(self):
    movements = [movement for movement in self.context.values()
                 if movement.cause is None]
    self.latest = movements[0]
    if self.latest.id is None:
        self.redirect(self.url(self.context.__parent__))
    intids = getUtility(IIntIds)
    self.person = intids.queryObject(self.latest.id, None)

@form.action("Cancelar", validator=lambda *a: ())
def cancel(self, action, data):
    self.redirect(self.url(self.context.__parent__))



# XXX: copied from PositionView
def breadcrumbs(self):
    parents = []
    parent = self.context.__parent__
    while interfaces.IUnit.providedBy(parent):
        parents.insert(0, {"url": self.url(parent),
```

```
                        "name": escape(parent.name)})
            parent = parent.__parent__
        parents.insert(0, {"url": self.url(parent), "name": u"Facultad"})
        result = []
        parents.append({"url": self.url(self.context), "name":
escape(self.context.name)})
        for parent in parents:
            result.append('<a href="%(url)s">%(name)s</a>' % (parent))
        return " / ".join(result)

    def validate(self, action, data):
        errors = super(Empty, self).validate(action, data)
        if errors: return errors

        if data.get("ending_date") < self.latest.starting_date:

            error = WidgetInputError("ending_date",
                                     u"Fecga de Retiro",
                                     u"La Fecha de Retiro debe ser mayor "
                                     u"a la Fecha de Nombramiento")
            errors.append(error)
            message = getMultiAdapter((error, self.request),
```

```
                                IWidgetInputErrorView).snippet()
            self.widgets["ending_date"].error = message
        return errors


class EditOpenMovement(grok.View):

    grok.context(interfaces.IPosition)
    grok.name("movement")

    def update(self):
        open_movements = [movement for movement in
self.context.values()
                          if movement.cause is None]
        if not open_movements:
            self.redirect(self.url(self.context))
        self.latest = open_movements[0]

    def render(self):
        self.redirect(self.url(self.latest, "edit"))
```

```
class EditMovement(grok.EditForm):

    grok.context(interfaces.IMovement)
    grok.name("edit")
    grok.require("sirh.Edit")

    form_fields = grok.Fields(interfaces.IMovement).omit("ending_date",
                                            "cause", "id")
    template =
grok.PageTemplateFile(os.path.join(os.path.dirname(__file__),
                                    "custom_templates",
                                    "positionmovementeditform.pt"))

    @grok.action("Guardar")
    def save(self, **data):
        changes = self.applyData(self.context, **data)
        if changes:
            self.flash(u"Los cambios fueron guardados")
        self.redirect(self.url(self.context.__parent__))

    @form.action("Cancelar", validator=lambda *a: ())
    def cancel(self, action, data):
```

```
    self.redirect(self.url(self.context.__parent__))


def validate(self, action, data):
    errors = super(EditMovement, self).validate(action, data)
    if errors: return errors

    if float(data.get("salary")) > self.context.__parent__.salary:

        error = WidgetInputError("salary",
                                 u"Sueldo Básico",
                                 u"El sueldo básico no puede ser mayor al "
                                 u"sueldo máximo para la plaza")
        errors.append(error)
        message = getMultiAdapter((error, self.request),
                                  IWidgetInputErrorView).snippet()
        self.widgets["salary"].error = message
    return errors

def update(self):
    intids = getUtility(IIntIds)
```

```
        self.person = intids.queryObject(self.context.id, None)


class PositionsGetter(object):

    grok.implements(interfaces.IPositionsGetter)

    def values(self, unit):
        return self.getPositions(unit)

    def getPositions(self, unit):
        result = [item for item in unit.values()
                  if interfaces.IPosition.providedBy(item)]
        children = [item for item in unit.values()
                    if interfaces.IUnit.providedBy(item)]
        for child in children:
            result.extend(self.getPositions(child))
        return result


class IsEmpty(grok.View):
```

```
    grok.context(interfaces.IPosition)
    grok.name("is_empty")

    def render(self):
        movements = [movement for movement in self.context.values()
                     if movement.cause is None]
        if not movements:
            return True
        assert(len(movements) == 1)
        return False


class LastMovement(grok.View):

    grok.context(interfaces.IPosition)
    grok.name("last_movement")

    def render(self):
        try:
            return [movement for movement in self.context.values()
```

```
                    if movement.cause is None][0]
        except (IndexError):
            return


class GetPerson(grok.View):

    grok.context(interfaces.IPosition)
    grok.name("get_person")

    def render(self):
        movements = [movement for movement in self.context.values()
                     if movement.cause is None]
        intids = getUtility(IIntIds)
        return intids.getObject(movements[0].id)


cna_field = schema.Choice(
    __name__="cna",
    title=u"Conceptos",
    vocabulary="NoAutomaticPaymentReferences"
```

```
    )


class CNA(grok.Form):

    grok.context(interfaces.IPosition)
    grok.name("select_cna")
    grok.require("sirh.Edit")

    form_fields = grok.Fields(cna_field)
    template =
grok.PageTemplateFile(os.path.join(os.path.dirname(__file__),
                                       "custom_templates",
                                       "selectcnaform.pt"))
    label = u"Paso 1 de 2: Seleccione el concepto base"

    @grok.action("Seleccionar")
    def select(self, **data):
        name = getName(data.get("cna"))
        session = ISession(self.request)["sirh"]
        session["name"] = name
        self.redirect(self.url("addcna"))
```

```
    @form.action("Cancelar", validator=lambda *a: ())
    def cancel(self, action, data):
        self.redirect(self.url(self.context))

    def update(self, *args, **kw):
        super(CNA, self).update(*args, **kw)
        self.person = getMultiAdapter((self.context, self.request),
                                      name="get_person")()
        self.movement = getMultiAdapter((self.context, self.request),
                                        name="last_movement")()


class AddCNA(grok.Form):

    grok.context(interfaces.IPosition)
    grok.require("sirh.Edit")

    template =
grok.PageTemplateFile(os.path.join(os.path.dirname(__file__),
                                           "custom_templates",
                                           "addcnaform.pt"))
    label = u"Paso 2 de 2: Detalles del Concepto No Automático"
```

```
    def __init__(self, *args, **kw):
        super(AddCNA, self).__init__(*args, **kw)
        always = ["month_from", "year_from", "instalments", "units",
"payrolls"]

        session = ISession(self.request)["sirh"]
        name = session.get("name")
        if not name:
            self.redirect(self.url(self.context))
            return
        self.ref = self.getPaymentReferenceContainer().get(name)
        if not self.ref:
            self.redirect(self.url(self.context))
            return
        if self.ref.base_value == u"No Definido":
            always.insert(3, "unit_price")

        fields =
schema.getFields(interfaces.INoAutomaticPaymentReference)
        self.form_fields = grok.Fields()
        for field in always:
```

```
        self.form_fields += grok.Fields(fields[field])


def setUpWidgets(self, *args, **kw):
    super(AddCNA, self).setUpWidgets(*args, **kw)
    w = self.widgets["payrolls"]
    vocab = getUtility(schema.interfaces.IVocabularyFactory,
                               name="PayrollTypes")(None)
    ref_payrolls = [term for term in vocab
                    if term.value in self.ref.payrolls]
    w.choices = [{'text': w.textForValue(term), 'value': term.token}
                 for term in vocab
                 if term not in ref_payrolls]
    w.selected = [{'text': w.textForValue(term), 'value': term.token}
                  for term in ref_payrolls]



def getPaymentReferenceContainer(self):
    obj = self.context
    while obj is not None:
        if grok.interfaces.IApplication.providedBy(obj):
            container = [value for value in obj.values()
```

```
                        if
interfaces.IPaymentReferenceContainer.providedBy(value)][0]
            return container
        obj = obj.__parent__
    raise ValueError("No Sirh found.")



@grok.action("Agregar")
def add(self, **data):
    cna = payment.NoAutomaticPaymentReference(**data)
    cna.reference = self.ref
    if self.ref.base_value == u"No Definido":
        cna.origin = u"Informado"
    else:
        cna.origin = u"Calculado"


    y = cna.year_from + ((cna.instalments - 1) / 12)
    v = int("%.0f" % ((((cna.instalments - 1) / 12.0) - ((cna.instalments -
1) / 12)) * 12)) + cna.month_from
    if v - 12 > 0:
        cna.year_to = y + 1
        cna.month_to = v - 12
```

```
    else:
        cna.year_to = y
        cna.month_to = v

    if cna.unit_price:
        cna.amount = cna.units * cna.unit_price
    self.context.noautomatic.append(cna)
    session = ISession(self.request)["sirh"]
    del(session["name"])
    self.redirect(self.url(self.context))

@form.action("Cancelar", validator=lambda *a: ())
def cancel(self, action, data):
    self.redirect(self.url(self.context))

def update(self, *args, **kw):
    super(AddCNA, self).update(*args, **kw)
    self.person = getMultiAdapter((self.context, self.request),
                                  name="get_person")()
    self.movement = getMultiAdapter((self.context, self.request),
                                    name="last_movement")()
```

```
class CNAIndex(grok.View):

    grok.context(interfaces.IPosition)
    grok.name("cna")
    grok.require("sirh.Edit")

    def update(self, index=None):
        if not index:
            self.redirect(self.url(self.context))
            return
        try:
            index = int(index)
            cna = self.context.noautomatic[index]
        except (ValueError, IndexError):
            self.redirect(self.url(self.context))
            return
        self.cna = cna
        self.title = "%s: %s" % (cna.reference.code,
                                 cna.reference.name)
        self.edit_url = "%s/editcna?index=%s" % (self.url(self.context),
index)
```

```
        # YACK!!!
        self.payrolls = "<ul>%s</ul>" % \
                        ("".join(["<li>%s</li>" % (payroll)
                          for payroll in self.cna.payrolls]))
class EditCNA(grok.Form):

    grok.context(interfaces.IPosition)
    grok.require("sirh.Edit")

    template =
grok.PageTemplateFile(os.path.join(os.path.dirname(__file__),
                                           "custom_templates",
                                           "editcnaform.pt"))

    label = u"Editar Concepto No Automático"

    def __init__(self, *args, **kw):
        super(EditCNA, self).__init__(*args, **kw)

        # The first time the form is rendered the index come from
        # the request
```

```
session = ISession(self.request)["sirh"]

try:
    index = int(self.request.get("index"))
except (ValueError, TypeError):
    # Now let's check the session
    if session.get("index") is None:
        # if we have no index, we get out
        self.redirect(self.url(self.context))
        return
else:
    session["index"] = index

self.index = session.get("index")

try:
    self.cna = self.context.noautomatic[self.index]
except IndexError:
    # the index is too high, get out
    del(session["index"])
```

```
            self.redirect(self.url(self.context))
            return

        always = ["month_from", "year_from", "instalments", "units",
"payrolls"]

        if self.cna.reference.base_value == u"No Definido":
            always.insert(3, "unit_price")

        fields =
schema.getFields(interfaces.INoAutomaticPaymentReference)
        self.form_fields = grok.Fields()
        for field in always:
            self.form_fields += grok.Fields(fields[field])

    def setUpWidgets(self, *args, **kw):
        super(EditCNA, self).setUpWidgets(*args, **kw)
        self.widgets["month_from"].setRenderedValue(self.cna.month_from
)
        self.widgets["year_from"].setRenderedValue(self.cna.year_from)
        self.widgets["instalments"].setRenderedValue(self.cna.instalments)
        self.widgets["units"].setRenderedValue(self.cna.units)
```

```
        if self.cna.reference.base_value == u"No Definido":
            self.widgets["unit_price"].setRenderedValue(self.cna.unit_price)
        w = self.widgets["payrolls"]
        vocab = getUtility(schema.interfaces.IVocabularyFactory,
                           name="PayrollTypes")(None)
        cna_payrolls = [term for term in vocab
                        if term.value in self.cna.payrolls]
        w.choices = [{'text': w.textForValue(term), 'value': term.token}
                     for term in vocab
                     if term not in cna_payrolls]
        w.selected = [{'text': w.textForValue(term), 'value': term.token}
                      for term in cna_payrolls]

    @grok.action("Guardar")
    def edit(self, **data):
        self.applyData(self.cna, **data)



        y = self.cna.year_from + ((self.cna.instalments - 1) / 12)
        v = int("%.0f" % ((((self.cna.instalments - 1) / 12.0) -
((self.cna.instalments - 1) / 12)) * 12)) + self.cna.month_from
        if v - 12 > 0:
```

```
            self.cna.year_to = y + 1
            self.cna.month_to = v - 12
        else:
            self.cna.year_to = y
            self.cna.month_to = v

        if self.cna.unit_price:
            self.cna.amount = self.cna.units * self.cna.unit_price
        self.context.noautomatic[self.index] = self.cna
        session = ISession(self.request)["sirh"]
        del(session["index"])
        self.redirect(self.url(self.context))

    @form.action("Cancelar", validator=lambda *a: ())
    def cancel(self, action, data):
        self.redirect(self.url(self.context))



def update(self, *args, **kw):
    super(EditCNA, self).update(*args, **kw)
    self.person = getMultiAdapter((self.context, self.request),
```

```
                                    name="get_person")()
        self.movement = getMultiAdapter((self.context, self.request),
                                        name="last_movement")()
class Delete(grok.View):

    grok.context(interfaces.IPosition)
    grok.name("delete")
    grok.require("sirh.Edit")

    def update(self, index=None):
        if not index:
            self.redirect(self.url(self.context))
            return
        try:
            index = int(index)
            del(self.context.noautomatic[index])
        except (ValueError, IndexError):
            self.redirect(self.url(self.context))

    def render(self):
        self.redirect(self.url(self.context))
```

```
class Income(grok.View):
    grok.context(interfaces.IPosition)
    def getPaymentReferenceContainer(self):
        obj = self.context
        while obj is not None:
            if grok.interfaces.IApplication.providedBy(obj):
                container = [value for value in obj.values()
             if interfaces.IPaymentReferenceContainer.providedBy(value)][0]
                return container
            obj = obj.__parent__
        raise ValueError("No Sirhi or Sirh found.")

    def update(self):
        self.date = date.today()
        self.person = getMultiAdapter((self.context, self.request),
                                      name="get_person")()
        self.last_movement = getMultiAdapter((self.context, self.request),
                                             name="last_movement")()
        salary = str(self.last_movement.salary)
        salary_dollars = to_card(int(salary.split(".")[0]))
        salary_cents = salary.split(".")[-1] + "/100"
```

```
        self.salary = salary_dollars + " " + salary_cents
        locale = i18n.locales.locales.getLocale(language="es")
        formatter = locale.dates.getFormatter("date", "long")
        self.starting_date =
formatter.format(self.last_movement.starting_date)
        months_vocab = getUtility(schema.interfaces.IVocabularyFactory,
                                  name="months")()
        month = months_vocab.getTermByToken(str(self.date.month)).title
        o = to_card(self.date.day).decode("utf8")
        u = to_card(self.date.year).decode("utf8")
        self.date_string = u"%s días del mes de %s de %s" % (o,
month.lower(), u)
        formatter = locale.numbers.getFormatter("decimal")
        self.salary_number = formatter.format(self.last_movement.salary)
        record = self.createRecord()
        self.references = [ref for ref in record.references.values()
                           if ref.type == u"Descuento"]
        self.total = "%.2f" % sum([abs(ref.result) for ref in self.references])
        sirhi = self.get_sirhi()
        self.human_resources = sirhi.rrhh_mgr
        self.financial_administrator = sirhi.financial_mgr
```

```
    def get_sirhi(self):
        parent = self.context
        while not interfaces.ISirhi.providedBy(parent):
            parent = parent.__parent__
            if grok.interfaces.IApplication.providedBy(parent):
                raise ValueError("No Sirhi or Sirh found.")
        return parent
    def createRecord(self):
        references = self.getPaymentReferenceContainer().values()
        automatic = [(reference.order, reference) for reference in
references
                    if reference.automatic == True and
                    u"Normal Mensual" in reference.payrolls]
        position = self.context
        request = self.request
        person = self.person
        result = {}
        result["name"] = position.name
        result["first_name"] = person.first_name
        result["last_name"] = person.last_name
        result["code"] = person.code
        result["budget_item"] = position.budget_item
```

```
        result["budget_subitem"] = position.budget_subitem
        result["position_name"] = position.name
        record = PayrollRecord(**result)
        cnas = [(cna.reference.order, cna) for cna in position.noautomatic
                if self.includeCNA(cna)]
        cnas.extend(automatic)
        references = sorted(cnas)
        for k, reference in references:
            result = 0
            if
interfaces.INoAutomaticPaymentReference.providedBy(reference):
                if reference.reference.base_value == u"No Definido":
                    result = reference.amount
                    if reference.reference.type == u"Descuento":
                        result = result * -1
                    info = {}
                    info["code"] = reference.reference.code
                    info["name"] = reference.reference.name
                    info["type"] = reference.reference.type
                    info["order"] = reference.reference.order
                    info["result"] = result
                    ppr = PayrollPaymentReference(**info)
```

```
                    record.references[ppr.order] = ppr
                    continue
                else:
                    reference = reference.reference
            if reference.montly_salary:
                result = getMultiAdapter((position, request),
name=u"last_movement")().salary
            if reference.base_value == u"Valor Unitario":
                result = reference.criteria.value
                if reference.type == u"Descuento":
                    result = result * -1
            if reference.base_value == u"Porcentaje":
                group_result = 0
                group_refs = []
                if reference.criteria is not None:
                    group_refs = [ref for ref in
reference.criteria.group.references]
                for ref in group_refs:
                    if ref.order in record.references:
                        group_result += record.references[ref.order].result
                if reference.criteria is not None:
                    result = reference.criteria.calculate(group_result)
```

```
            if reference.maximum_value == u"Monto Máximo" and \
              reference.maximum_criteria is not None:
                if result and reference.maximum_criteria.value < result:
                    result = reference.maximum_criteria.value
            if reference.maximum_value == u"Porcentaje":
                group_result = 0
                group_refs = []
                if reference.maximum_criteria is not None:
                    group_refs = [ref for ref in
reference.maximum_criteria.group.references]
                for ref in group_refs:
                    if ref.order in record.references:
                        group_result += record.references[ref.order].result
                if reference.maximum_criteria is not None:
                    max_result =
reference.maximum_criteria.calculate(group_result)
                    if result and max_result < result:
                        result = max_result
            if reference.type == u"Descuento":
                result = result * -1
        if reference.base_value == u"Tabla":
            group_result = 0
```

```
            group_refs = []
            if reference.criteria is not None:
                group_refs = [ref for ref in
reference.criteria.group.references]
            for ref in group_refs:
                if ref.order in record.references:
                    group_result += record.references[ref.order].result
            if reference.criteria is not None:
                result = reference.criteria.calculate(group_result)
            if reference.maximum_value == u"Monto Máximo" and \
              reference.maximum_criteria is not None:
                if result and reference.maximum_criteria.value < result:
                    result = reference.maximum_criteria.value
            if reference.maximum_value == u"Porcentaje":
                group_result = 0
                group_refs = []
                if reference.maximum_criteria is not None:
                    group_refs = [ref for ref in
reference.maximum_criteria.group.references]
                for ref in group_refs:
                    if ref.order in record.references:
                        group_result += record.references[ref.order].result
```

```
                if reference.maximum_criteria is not None:
                    max_result =
reference.maximum_criteria.calculate(group_result)
                    if result and max_result < result:
                        result = max_result
            if reference.type == u"Descuento":
                result = result * -1
        info = {}
        info["code"] = reference.code
        info["name"] = reference.name
        info["type"] = reference.type
        info["order"] = reference.order
        info["result"] = result
        ppr = PayrollPaymentReference(**info)
        record.references[ppr.order] = ppr

    record.wages = sum([ref.result for ref in record.references.values()
if ref.type == u"Haber"])
    record.deductions = abs(sum([ref.result for ref in
record.references.values() if ref.type == u"Descuento"]))
    record.net_salary = sum([ref.result for ref in
record.references.values() if ref.type != u"Aporte Patronal"])
```

```
        return record


    def includeCNA(self, cna):
        if u"Normal Mensual" in cna.payrolls:
            date_from = int("%d%02d" % (cna.year_from, cna.month_from))
            date_to = int("%d%02d" % (cna.year_to, cna.month_to))
            date_payroll = int("%d%02d" % (self.date.year,
                                           self.date.month))
            if date_from <= date_payroll <= date_to:
                return True
        return False
        records = [self.createRecord(position, request) for position in
positions
                   if not getMultiAdapter((position, request),
name=u"is_empty")() and
                   getattr(getMultiAdapter((position, request),
name="last_movement")(), "status", None) == u"Activo" and
                   position.budgetary_code == payroll.budgetary_code and
                   position.payment_method == payroll.payment_method and
                   position.financing_source == payroll.financing_source and
                   position.classification == payroll.classification]
        return records
```

```
    def breadcrumbs(self):
        parents = []
        parent = self.context.__parent__
        while interfaces.IUnit.providedBy(parent):
            parents.insert(0, {"url": self.url(parent),
                               "name": escape(parent.name)})
            parent = parent.__parent__
        parents.insert(0, {"url": self.url(parent), "name": u"Facultad"})
        result = []
        parents.append({"url": self.url(self.context), "name":
escape(self.context.name)})
        for parent in parents:
            result.append('<a href="%(url)s">%(name)s</a>' % (parent))
        return " / ".join(result)


    def faculty_name(self):
        parent = self.context
        while not interfaces.ISirhi.providedBy(parent):
            parent = parent.__parent__
            if grok.interfaces.IApplication.providedBy(parent):
                raise ValueError("No Sirhi or Sirh found.")
```

```
return parent.name
```

sirhi.py

```
# -*- coding: utf8 -*-

import grok
import os
from zope import app, component, schema
from zope.formlib import form

from sirh import interfaces
from sirh import unit
from sirh import default_data
from sirh import vocabulary
from sirh import person
from sirh import payroll
from sirh import position

# Models

class Sirhi(grok.Container, grok.Site):

    grok.implements(interfaces.ISirhi)
```

```
    def __init__(self, **data):
        super(Sirhi, self).__init__()
        self.name = data.get("name")
        self.financial_mgr = data.get("financial_mgr") or u"NOMBRE DEL
ADMINISTRADOR/A FINANCIERO/A"
        self.rrhh_mgr = data.get("rrhh_mgr") or u"NOMBRE DEL JEFE DE
RRHH"


class Edit(grok.EditForm):


    grok.context(interfaces.ISirhi)
    grok.require("sirh.Edit")


    form_fields = grok.Fields(interfaces.ISirhi).omit("name")
    template =
grok.PageTemplateFile(os.path.join(os.path.dirname(__file__),
                                       "sirhi_templates",
                                       "edit.pt"))


    @grok.action(u"Guardar")
    def save(self, **data):
```

```
        changes = self.applyData(self.context, **data)
        if changes:
            self.flash(u"Los cambios fueron guardados")
        self.redirect(self.url(self.context["planillas"]))

    @form.action("Cancelar", validator=lambda *a: ())
    def cancel(self, action, data):
        self.redirect(self.url(self.context["planillas"]))


# Utilities

class UnitVocabulary(vocabulary.BaseVocabulary):

    def titleValue(self, value):
        if self.isUnitContainer(value):
            return value.__parent__.name
        # Is a regular unit
        return "%s - %s" % (value.code, value.name)

    def isUnitContainer(self, value):
```

```
        if interfaces.IUnitContainer.providedBy(value):
            return True
        return False


class UnitVocabularyFactory(object):

    grok.provides(schema.interfaces.IVocabularyFactory)


    def getSirhiSite(self, context):
        obj = context
        while obj is not None:
            if interfaces.ISirhi.providedBy(obj):
                return obj
            obj = obj.__parent__
        raise ValueError("No Sirhi or Sirh found.")

    def getUnitContainer(self, sirhi):
        try:
            return [container for container in sirhi.values()
```

```
                if interfaces.IUnitContainer.providedBy(container)][0]
        except (IndexError):
            raise ValueError("No UnitContainer found")

    def __call__(self, context):
        sirhi = self.getSirhiSite(context)
        values = self.getChildren(self.getUnitContainer(sirhi))
        return UnitVocabulary(values)

    def getChildren(self, unit):
        result = [item for item in unit.values()
                  if interfaces.IUnit.providedBy(item)]
        children = [item for item in unit.values()
                    if interfaces.IUnit.providedBy(item)]
        for child in children:
            result.extend(self.getChildren(child))
        return result

# Views

class SirhiView(grok.View):
```

```
    grok.context(Sirhi)
    grok.name("index")


# Subscribers


@grok.subscribe(Sirhi, grok.IObjectAddedEvent)
def createUnitContainer(sirhi, event):
    sirhi["estructura"] = unit.UnitContainer()
    sirhi["personas"] = person.PersonContainer()
    sirhi["tablas"] = vocabulary.VocabularyDataContainer()
    sirhi["planillas"] = payroll.PayrollContainer()
    sm = component.getSiteManager(context=sirhi)
    chooser = app.container.interfaces.INameChooser(sirhi["tablas"])
    for data in default_data.local_vocabulary_data:
        vd = vocabulary.VocabularyData(**data)
        name = chooser.chooseName(vd.name, vd)
        sirhi["tablas"][name] = vd
        sm.registerUtility(vd, interfaces.IVocabularyData, name=vd.name)
        sm.registerUtility(vocabulary.UnicodeVocabularyFactory(name=vd.
name),
```

```
                schema.interfaces.IVocabularyFactory,
                name=vd.name)
sm.registerUtility(UnitVocabularyFactory(),
                schema.interfaces.IVocabularyFactory,
                name=u"Units")
sm.registerUtility(position.PositionsGetter(),
                interfaces.IPositionsGetter,
                name=u"Positions")
sm.registerUtility(payroll.PayrollGenerator(),
                interfaces.IPayrollGenerator,
                name=u"PayrollGenerator")
```

table.py

```
# -*- coding: utf8 -*-

import grok
import os
from zope import schema
from zope.app.container.interfaces import INameChooser
from zope.formlib import form
from zope.traversing.api import getName

from sirh import interfaces
from sirh import vocabulary

class TableContainer(grok.Container):

    grok.implements(interfaces.ITableContainer)


class Table(grok.Container):

    grok.implements(interfaces.ITable)
```

```
    def __init__(self, **data):
        super(Table, self).__init__()
        self.code = data.get("code")
        self.name = data.get("name")

    def evaluate(self, value):
        for rango in self.values():
            if value in rango:
                return rango.calculate(value)
        return 0.0

class Range(grok.Model):

    grok.implements(interfaces.IRange)

    def __init__(self, **data):
        self.code = data.get("code")
        self.name = data.get("name")
        self.minimum = data.get("minimum")
        self.maximum = data.get("maximum")
```

```
    self.value = data.get("value")
    self.operator = data.get("operator")
    self.base_value = data.get("base_value")
    self.calculation = data.get("calculation")
    self.base_dif = data.get("base_dif")

def __contains__(self, value):
    if value >= self.minimum:
        if self.maximum is not None:
            if value <= self.maximum:
                return True
        else:
            return True
    return False

def calculate(self, value):
    if self.operator == u"+":
        return self.value + ((self.base_value / 100.0) * self._result(value))
    if self.operator == u"-":
        return self.value - ((self.base_value / 100.0) * self._result(value))
```

```
    def _result(self, value):
        if self.calculation == u"Valor":
            return 0.0
        if self.calculation == u"Diferencia MÃnimo":
            return value - self.minimum
        if self.calculation == u"Base Dif":
            return value - self.base_dif


class ContainerIndex(grok.View):

    grok.context(interfaces.ITableContainer)
    grok.name("index")
    grok.require("sirh.Admin")

    def update(self):
        self.tables = self.context.values()


class AddTable(grok.AddForm):
```

```
    grok.context(interfaces.ITableContainer)
    grok.name("add")
    grok.require("sirh.Admin")

    form_fields = grok.Fields(interfaces.ITable)
    template =
grok.PageTemplateFile(os.path.join(os.path.dirname(__file__),
                                       "custom_templates",
                                       "paymentreferencegroupform.pt"))

    @grok.action("Agregar")
    def add(self, **data):
        table = Table(**data)
        name = INameChooser(self.context).chooseName(u"", table)
        self.context[name] = table
        self.redirect(self.url(self.context))

    @form.action("Cancelar", validator=lambda *a: ())
    def cancel(self, action, data):
        self.redirect(self.url(self.context))
```

```
class EditTable(grok.EditForm):

    grok.context(interfaces.ITable)
    grok.name("edit")
    grok.require("sirh.Admin")

    form_fields = grok.Fields(interfaces.ITable)
    template =
grok.PageTemplateFile(os.path.join(os.path.dirname(__file__),
                                       "custom_templates",
                                       "paymentreferencegroupform.pt"))

    @grok.action("Guardar")
    def edit(self, **data):
        changes = self.applyData(self.context, **data)
        if changes:
            self.flash("Los cambios fueron guardados")
        self.redirect(self.url(self.context))

    @form.action("Cancelar", validator=lambda *a: ())
    def cancel(self, action, data):
        self.redirect(self.url(self.context))
```

```
class TableIndex(grok.View):

    grok.context(interfaces.ITable)
    grok.name("index")
    grok.require("sirh.Admin")

    def update(self):
        self.ranges = self.context.values()


class AddRange(grok.AddForm):

    grok.context(interfaces.ITable)
    grok.name("add")
    grok.require("sirh.Admin")

    form_fields = grok.Fields(interfaces.IRange)
    template =
grok.PageTemplateFile(os.path.join(os.path.dirname(__file__),
                                   "custom_templates",
```

```
                                    "paymentreferencegroupform.pt"))

    @grok.action("Agregar")
    def add(self, **data):
        myrange = Range(**data)
        name = INameChooser(self.context).chooseName(u"", myrange)
        self.context[name] = myrange
        self.redirect(self.url(self.context))

    @form.action("Cancelar", validator=lambda *a: ())
    def cancel(self, action, data):
        self.redirect(self.url(self.context))


class Index(grok.View):

    grok.context(interfaces.IRange)
    grok.require("sirh.Admin")


class Edit(grok.EditForm):
```

```
    grok.context(interfaces.IRange)
    grok.require("sirh.Admin")

    form_fields = grok.Fields(interfaces.IRange)
    template =
grok.PageTemplateFile(os.path.join(os.path.dirname(__file__),
                                       "custom_templates",
                                       "paymentreferencegroupform.pt"))

    @grok.action("Guardar")
    def edit(self, **data):
        changes = self.applyData(self.context, **data)
        if changes:
            self.flash("Los cambios fueron guardados")
        self.redirect(self.url(self.context))

    @form.action("Cancelar", validator=lambda *a: ())
    def cancel(self, action, data):
        self.redirect(self.url(self.context))
```

```
class TableVocabulary(vocabulary.BaseVocabulary):

    def titleValue(self, value):
        return "%s: %s" % (value.code, value.name)


class TableVocabularyFactory(object):

    grok.provides(schema.interfaces.IVocabularyFactory)

    def getTableContainer(self, context):
        obj = context
        while obj is not None:
            if grok.interfaces.IApplication.providedBy(obj):
                container = [value for value in obj.values()
                             if interfaces.ITableContainer.providedBy(value)][0]
```

```
            return container
        obj = obj.__parent__
    raise ValueError("No TableContainer found.")


def __call__(self, context):
    container = self.getTableContainer(context)
    values = sorted([ref for ref in container.values()],
                    key=lambda x:x.code)
    return TableVocabulary(values)
```

testing.py

```
import os.path

import sirh

from zope.app.testing.functional import ZCMLLayer


ftesting_zcml = os.path.join(

    os.path.dirname(sirh.__file__), 'ftesting.zcml')

FunctionalLayer = ZCMLLayer(ftesting_zcml, __name__,
'FunctionalLayer')
```

unit.py

```
# -*- coding: utf8 -*-

import grok
import os
from zope.formlib import form
from zope.app.container.interfaces import INameChooser
from zope.traversing.api import getName
from zope.app.form.browser.textwidgets import escape
from zope.app.generations.utility import findObjectsProviding
from zope.component import getMultiAdapter

from sirh import interfaces



# Models

class UnitContainer(grok.Container):

    grok.implements(interfaces.IUnitContainer)
```

```
class Unit(grok.Container):

    grok.implements(interfaces.IUnit)

    def __init__(self, **data):
        super(Unit, self).__init__()
        self.code = data.get("code")
        self.name = data.get("name")


# Views

class ContainerView(grok.View):

    grok.context(interfaces.IUnitContainer)
    grok.name("index")
    grok.require("sirh.Edit")

    def update(self):
        self.units = sorted([item for item in self.context.values()
```

```
                        if interfaces.IUnit.providedBy(item)],
                       key=lambda x:x.name)


class UnitView(grok.View):

    grok.context(interfaces.IUnit)
    grok.name("index")
    grok.require("sirh.Edit")

    def update(self):
        self.units = sorted([item for item in self.context.values()
                             if interfaces.IUnit.providedBy(item)],
                            key=lambda x:x.name)
        self.positions = sorted([item for item in self.context.values()
                                 if interfaces.IPosition.providedBy(item)],
                                key=lambda x:x.name)

    def breadcrumbs(self):
        parents = []
        parent = self.context.__parent__
```

```
        while interfaces.IUnit.providedBy(parent):
            parents.insert(0, {"url": self.url(parent),
                               "name": escape(parent.name)})
            parent = parent.__parent__
        parents.insert(0, {"url": self.url(parent), "name": u"Facultad"})
        result = []
        for parent in parents:
            result.append('<a href="%(url)s">%(name)s</a>' % (parent))
        result.append(escape(self.context.name))
        return " / ".join(result)

def sortinventory(x):
    if x.get("person") is not None:
        return x["person"].last_name
    return ""

class Inventory(grok.View):

    grok.context(interfaces.IUnit)
    grok.require("sirh.Edit")
```

```
def update(self):
    self.rows = []
    for position in [i for i in self.context.values()
                     if interfaces.IPosition.providedBy(i)]:
        info = {"position": position}
        if getMultiAdapter((position, self.request), name="filled")():
            info["person"] = getMultiAdapter((position, self.request),
                                             name="get_person")()
        else:
            info["person"] = None
        self.rows.append(info)
    self.rows = sorted(self.rows,
                       key=sortinventory)


def breadcrumbs(self):
    parents = []
    parent = self.context.__parent__
    while interfaces.IUnit.providedBy(parent):
        parents.insert(0, {"url": self.url(parent),
                           "name": escape(parent.name)})
        parent = parent.__parent__
```

```
        payroll_container = parent
        faculty_name = payroll_container.__parent__.name
        parents.insert(0, {"url": self.url(parent), "name":
escape(faculty_name)})
        parents.append({"url": self.url(self.context),
                        "name": escape(self.context.name)})
        result = []
        for parent in parents:
            result.append('<a href="%(url)s">%(name)s</a>' % (parent))
        return " / ".join(result)



class AddUnit(grok.Form):

    grok.context(interfaces.IUnitContainer)
    grok.name("add")
    grok.require("sirh.Edit")

    form_fields = grok.Fields(interfaces.IUnit)
    template =
grok.PageTemplateFile(os.path.join(os.path.dirname(__file__),
                                       "custom_templates",
```

```
                                "unitform.pt"))

    def __init__(self, *args, **kw):
        super(AddUnit, self).__init__(*args, **kw)
        if interfaces.IUnitContainer.providedBy(self.context):
            label = self.context.__parent__.name
        if interfaces.IUnit.providedBy(self.context):
            label = self.context.name
        self.label = u"Agregar SubUnidad a %s" % (label)

    @grok.action(u"Agregar")
    def add(self, **data):
        unit = Unit(**data)
        name = INameChooser(self.context).chooseName(u"", unit)
        self.context[name] = unit
        self.redirect(self.url(self.context))

    @form.action("Cancelar", validator=lambda *a: ())
    def cancel(self, action, data):
        self.redirect(self.url(self.context))
```

```
class AddUnitToUnit(AddUnit):

    grok.context(interfaces.IUnit)
    grok.require("sirh.Edit")


class EditUnit(grok.EditForm):

    grok.context(interfaces.IUnit)
    grok.name("edit")
    grok.require("sirh.Edit")

    form_fields = grok.Fields(interfaces.IUnit)
    template =
grok.PageTemplateFile(os.path.join(os.path.dirname(__file__),
                                       "custom_templates",
                                       "unitform.pt"))

    def __init__(self, *args, **kw):
        super(EditUnit, self).__init__(*args, **kw)
        self.label = u"Editar %s" % (self.context.name)
```

```
@grok.action(u"Guardar")
def save(self, **data):
    changes = self.applyData(self.context, **data)
    if changes:
        self.flash(u"Los cambios fueron guardados")
    self.redirect(self.url(self.context))

@form.action("Cancelar", validator=lambda *a: ())
def cancel(self, action, data):
    self.redirect(self.url(self.context))
```

vocabulary.py

```
# -*- coding: utf8 -*-

import grok

import os

from persistent.list import PersistentList

from zope import interface, schema, component, app, formlib


from sirhi import interfaces



# Models

class VocabularyDataContainer(grok.Container):

    grok.implements(interfaces.IVocabularyDataContainer)



class VocabularyData(grok.Model):

    grok.implements(interfaces.IVocabularyData)
```

```
    def __init__(self, **data):
        self.name = data.get("name")
        self.values = PersistentList(data.get("values"))
        self.display_name = data.get("display_name")


# Utilities

class BaseVocabulary(object):

    grok.implements(schema.interfaces.IVocabularyTokenized)

    def __init__(self, values):
        self.values = {}
        self.tokens = {}
        for i, value in enumerate(values):
            token = "%(token)02d" % {"token": i}
            term = schema.vocabulary.SimpleTerm(value,
                                                token,
                                                self.titleValue(value))
```

```
            self.values[value] = self.tokens[token] = term

    def titleValue(self, value):
        raise NotImplementedError("Subclasses should override titleValue
to "
                                  "provide a title for the value")

    def getTerm(self, value):
        try:
            return self.values[value]
        except (KeyError):
            raise LookupError(value)

    def getTermByToken(self, token):
        try:
            return self.tokens[token]
        except (KeyError):
            raise LookupError(token)

    def __iter__(self):
        return iter([self.tokens[token] for token in sorted(self.tokens)])
```

```
    def __len__(self):
        return len(self.values)

    def __contains__(self, value):
        return value in self.values.keys()


class UnicodeVocabulary(BaseVocabulary):

    def titleValue(self, value):
        return value


class UnicodeVocabularyFactory(grok.GlobalUtility):

    grok.provides(schema.interfaces.IVocabularyFactory)

    def __init__(self, *arg, **kw):
        super(UnicodeVocabularyFactory, self).__init__(*arg, **kw)
        self.name = kw.get("name")
```

```
    def __call__(self, context):
        vd = component.getUtility(interfaces.IVocabularyData,
                                  name=self.name)
        return UnicodeVocabulary(sorted(vd.values))


# Views

form_template_path = os.path.join(os.path.dirname(__file__),
                                  "custom_templates",
                                  "vocabularyform.pt")

class Container(grok.View):

    grok.context(interfaces.IVocabularyDataContainer)
    grok.name("index")
    grok.require("sirh.Edit")

    def update(self):
        self.values = sorted(self.context.values(), key=lambda x:x.name)
```

```
    def in_sirhi(self):
        if interfaces.ISirhi.providedBy(self.context.__parent__):
            return True
        return False


class Add(grok.Form):

    grok.context(interfaces.IVocabularyDataContainer)
    grok.require("sirh.Edit")

    form_fields = grok.Fields(interfaces.IVocabularyData)

    template = grok.PageTemplateFile(form_template_path)

    @grok.action("Guardar")
    def add(self, **data):
        vd = VocabularyData(**data)
        chooser = app.container.interfaces.INameChooser(self.context)
        name = chooser.chooseName(vd.name, vd)
        self.context[name] = vd
```

```
        sm = grok.getSite().getSiteManager()
        sm.registerUtility(vd, interfaces.IVocabularyData, name=vd.name)
        self.flash(u"Se agregÃ³ un nuevo vocabulario")
        self.redirect(self.url(vd))

    @formlib.form.action("Cancelar", validator=lambda *a: ())
    def cancel(self, action, data):
        self.redirect(self.url(self.context))

    def in_sirhi(self):
        if interfaces.ISirhi.providedBy(self.context.__parent__):
            return True
        return False


class Index(grok.View):

    grok.context(interfaces.IVocabularyData)
    grok.require("sirh.Edit")

    def update(self):
```

```
        self.values = sorted(self.context.values)

    def in_sirhi(self):
        if interfaces.ISirhi.providedBy(self.context.__parent__.__parent__):
            return True
        return False


class Edit(grok.EditForm):

    grok.context(interfaces.IVocabularyData)
    grok.require("sirh.Edit")

    form_fields = grok.Fields(interfaces.IVocabularyData).select("values")

    template = grok.PageTemplateFile(form_template_path)

    @grok.action("Guardar")
    def edit(self, **data):
        changes = self.applyData(self.context, **data)
        if changes:
```

```
            self.flash(u"Los cambios fueron guardados")
        self.redirect(self.url(self.context))

    @formlib.form.action("Cancelar", validator=lambda *a: ())
    def cancel(self, action, data):
        self.redirect(self.url(self.context))

    def in_sirhi(self):
        if interfaces.ISirhi.providedBy(self.context.__parent__.__parent__):
            return True
        return False


class MonthVocabulary(schema.vocabulary.SimpleVocabulary):

    @classmethod
    def createTerm(cls, value):
        return schema.vocabulary.SimpleTerm(value=value["number"],
                                            token=value["number"],
                                            title=value["name"])
```

```
class MonthVocabularyFactory(object):


    grok.provides(schema.interfaces.IVocabularyFactory)


    def __call__(self, context=None):
        months = [u"Enero", u"Febrero", u"Marzo", u"Abril", u"Mayo",
                u"Junio", u"Julio", u"Agosto", u"Septiembre", u"Octubre",
                u"Noviembre", u"Diciembre"]
        values = [{"number": i+1, "name": v} for i,v in
enumerate(months)]
        return MonthVocabulary.fromValues(values)



grok.global_utility(
    MonthVocabularyFactory,
    name="months",
    )
```