UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE CIENCIAS

Desarrollo de una aplicación para la gestión del Expediente Clínico Electrónico

REPORTE DE TRABAJO PROFESIONAL

QUE PARA OBTENER EL TÍTULO DE:

ACTUARIO

P R E S E N T A :

LUIS AGUILERA ORTEGA

TUTORA:
DRA. ELISA VISO GUROVICH

2014

Hoja de datos del jurado

1. Datos del alumno.
Aguilera
Ortega
Luis
56 75 76 73
Universidad Nacional Autónoma de México
Facultad de Ciencias
Actuaría
08167794

2. Datos del tutor.
Dra
Viso
Gurovich
Elisa

3. Datos del sinodal 1
Dra
Amparo
López
Gaona

4. Datos del sinodal 2
M en C Maria Guadalupe Elena
Ibargüengoitia
González

5. Datos del sinodal 3
L en CC
Manuel Alberto
Sugawara
Muro

6. Datos del sinodal 4
M en CC Carlos
Zerón
Martínez

7. Datos del trabajo escrito
Desarrollo de una aplicación para la gestión del Expediente Clínico Electrónico
65 p
2014

Dedico este trabajo a mi papá, quien nunca quitó el dedo del renglón.

Índice

1. Introducción

De Mayo de 2011 a Abril de 2013 fuí contratado por una universidad pública[1] para participar en el desarrollo del "Expediente Clínico Electrónico" (ECE) de una institución gubernamental de seguridad social[1].

Este proyecto buscaba concretar la construcción del ECE a partir de la documentación y módulos desarrollados por dos compañías de servicios de software altamente reconocidas[1]. Esta aplicación sería presentada a la Dirección de Prestaciones Médicas para su autorización y liberación para pruebas piloto a nivel nacional.

El proyecto se dividió en tres equipos de trabajo:

1. Estabilización de ECE 1.0. Esta versión fue desarrollada por una de las compañías antes citadas y estaba formada por los módulos de Asignaciones, Agenda, Citas y Consulta Externa.
2. Desarrollo de funcionalidad para soportar unidades médicas de segundo y tercer nivel.
3. Desarrollo de funcionalidad para soportar unidades médicas de medicina familiar.

Mi primer responsabilidad fue coordinar el desarrollo de los módulos de medicina familiar, respetando los lineamientos y entorno de trabajo establecidos por la institución y cumpliendo con requerimientos establecidos en casos de uso, prototipos de pantallas y reportes y reglas de negocio.

En particular tuve como responsabilidades directas:

- La creación del modelo de datos para 4 módulos (Nutrición, Trabajo Social, Estomatología y Programas Integrados).
- Integrar el modelo de datos propuesto para cada módulo al modelo de datos de la línea base.
- La construcción de las clases java para la capa de persistencia para los módulos antes citados.
- Diseño y construcción de las clases DTO.
- Supervisión de la integración de este código con el código de las capas de servicios y presentación.
- Identificación de conflictos en catálogos y coordinación de la carga a base de datos.
- Coordinar pruebas de sistema con analistas funcionales.

Una vez concluido este desarrollo, la siguiente tarea que se me encomendó fue la de integrar el código de los tres esfuerzos de desarrollo en un solo ejecutable para hacer las pruebas de aceptación de usuario.

[1] Se omiten los nombres por razones de confidencialidad

En esta etapa mi responsabilidad era la de supervisar el trabajo de un equipo de 5 programadores que se encargaban de integrar el código. Mi trabajo consistía en verificar que las nuevas versiones no omitieran funcionalidad y coordinar la solución de conflictos entre versiones de código.

Durante esta etapa se me encomendó la generación de los archivos ejecutables para diferentes pruebas. También tomé la responsabilidad de la administración de los servidores para los ambientes de desarrollo, pruebas y pre-producción. Asimismo se me encomendó la administración del servidor de control de versiones y de las diferentes ramas del proyecto.

Al término de la integración, se inició un proceso iterativo con personal de la Dirección de Prestaciones Médicas para validar la aplicación, aplicar "casos trazadores" para verificar que los flujos de información fueran correctos y, en algunos casos, modificar la funcionalidad entregada para reflejar cambios recientes en la operación en las unidades médicas. Esto porque los casos de uso tenían más de tres años de antigüedad y no habían sido actualizados.

Finalmente la aplicación fue aprobada y presentada a la Dirección General de la institución el 4 de Julio de 2012. A partir de ese momento se liberó la aplicación para la prueba piloto.

Durante esta etapa mis responsabilidades incluían: Preparar archivos ejecutables con diferentes configuraciones, monitorear el desempeño de cada componente de la aplicación, identificar causas de problemas y coordinar con los equipos de desarrollo, administración de base de datos o de operaciones las acciones necesarias para su solución.

El objetivo de este reporte es describir las diferentes funciones que realizé durante mi participación en el proyecto. Este documento está organizado de la siguiente manera:

- La sección 2 describe el entorno en el que había que desarrollar los componentes, las decisiones de arquitectura que la institución había tomado y los lineamientos de desarrollo.
- La sección 3 detalla las tareas específicas que se realizaron durante el desarrollo del código, incluyendo una descripción de la forma como se organizaron los equipos de trabajo.
- La sección 4 muestra la manera como se integró el código de los diferentes módulos y los problemas a los que nos enfrentamos.
- La sección 5 relata los problemas que enfrentamos al presentar la aplicación a los usuarios finales y la mecánica de trabajo que establecimos para resolverlos.
- La sección 6 describe las actividades para supervisar el correcto funcionamiento de la aplicación, las herramientas que usamos para este fin y los problemas que encontramos.

- La sección 7 resume las recomendaciones que se hicieron a la institución para resolver de fondo los problemas encontrados durante la prueba piloto.
- La sección 8 plantea una propuesta para llevar a cabo las recomendaciones descritas en la sección 7.
- La seccion 9 presenta las conclusiones obtenidas.
- En los anexos se describe el uso de herramientas para analizar el comportamiento de los principales componentes de la aplicación.

2. Marco de trabajo

Para el desarrollo del ECE, la institución estableció como requerimiento que se utilizara un *framework* de desarrollo establecido por la misma institución, basado en componentes *open source* de uso generalizado con un estilo arquitectónico organizado en capas jerárquicas, que en principio facilita el desarrollo y la gestión del software (reparto de requisitos, trabajo del equipo), así como la reutilización, portabilidad y seguridad.

El propósito de esta organización por capas es diferenciar y separar funciones únicas de las aplicaciones. Este enfoque permite un desarrollo donde los componentes de software tienen un bajo nivel de acoplamiento. Por ejemplo, la complejidad del almacenamiento de datos no debería afectar la forma en que se ejecuta la lógica de negocio, la cual sólo tendría dependencia y origen en las necesidades del negocio.

Con este enfoque, un componente realiza un conjunto único de funciones encapsuladas, exponiendo únicamente sus interfaces, con el objetivo de favorecer el entendimiento, facilidad de mantenimiento y escalabilidad de la solución.

La intención de la institución al requerir el uso de este framework propio en la implementación del ECE era separar de los módulos de negocio toda la funcionalidad que fuera de uso genérico, promoviendo que el equipo de desarrollo del ECE se enfocara exclusivamente en implementar funcionalidad que soportara los requerimientos de negocio.

A continuación se enumeran los objetivos arquitectónicos planteados por la institución:

- Integrar la funcionalidad provista por cuatro aplicaciones en un sistema único y centralizado, accesible desde todas las Unidades Médicas y Hospitales del territorio nacional.
- Brindar una arquitectura orientada a servicios, los cuales serán expuestos a través del canal de datos e interoperabilidad de Sistemas, con el objetivo de ser consumidos por otros sistemas de la Institución.
- En relación al punto anterior, identificar los procesos de negocio que pueden ser expuestos como servicios, siguiendo las mejores prácticas de una arquitectura orientada a servicios (SOA).
- Proporcionar un mecanismo de autentificación de usuarios de acuerdo a su perfil, basado en software de administración de identidad implementado por la institución de modo tal que cada usuario tenga acceso únicamente a las funcionalidades a las que está autorizado.
- Utilizar el estándar internacional de mensajería para servicios de salud, HL7, para establecer comunicación entre servicios y sistemas externos al ECE.
- Proporcionar a los usuarios finales del sistema una interfaz que facilite las funciones de captura, consulta y manejo de información crítica de salud.

La figura 1 muestra las capas en las que se organiza el sistema, y las responsabilidades asociadas a cada una de éstas, así como el diagrama de paquetes, en el cual se puede ver la relación de cada capa y su correspondiente a nivel del Framework. En este sentido, el Framework es extendido por las capas y componentes del ECE.

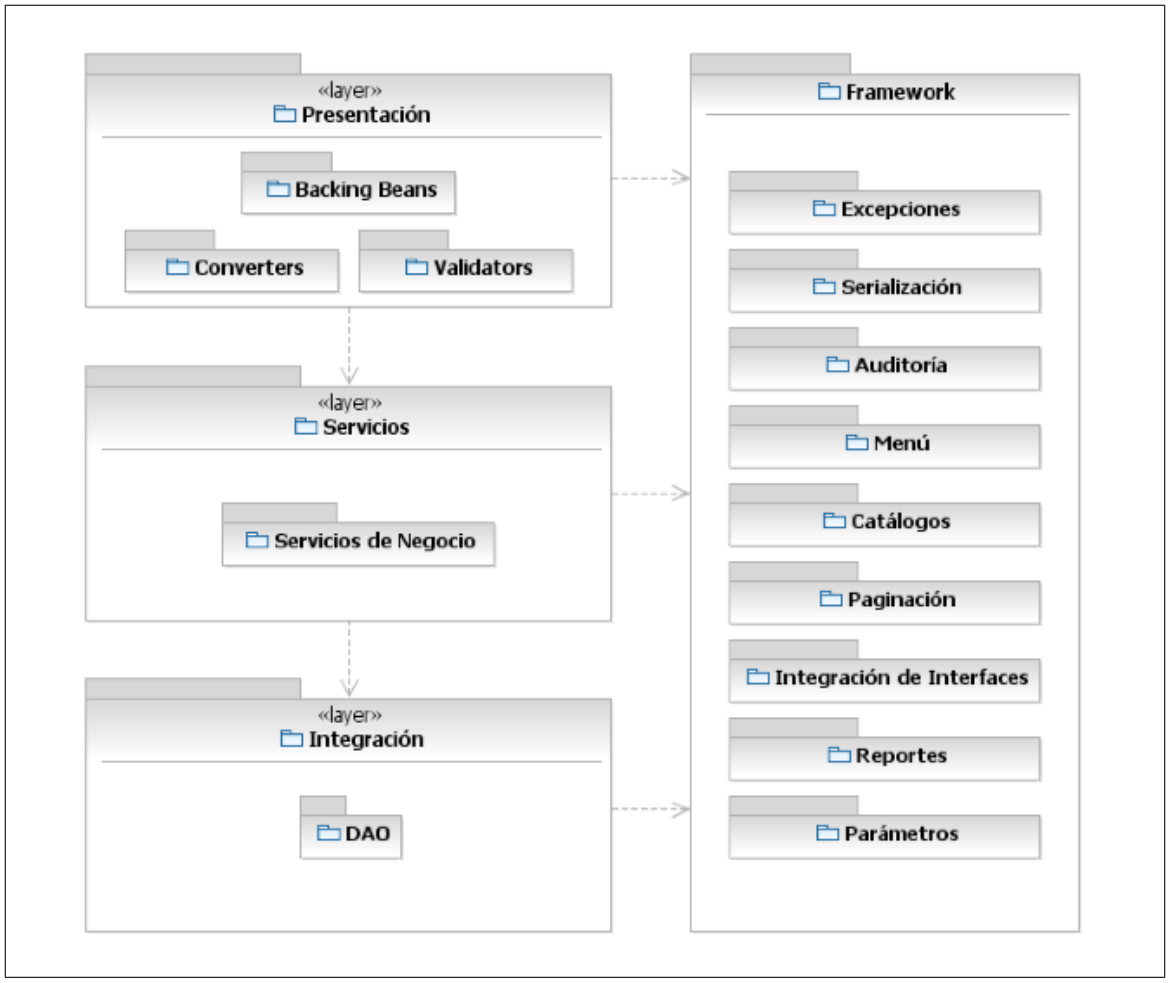

Figura 1: Marco de trabajo requerido por la institución

El paquete de la derecha corresponde al *framework* de desarrollo de la institución, muestra de manera esquemática los paquetes de funcionalidad genérica que deberían ser usados para el desarrollo de la funcionalidad específica de los módulos del expediente clínico. Mientras que los paquetes de la izquierdan muestra la arquitectura esperada para los módulos que nos comisionaron para su desarrollo. Si en el proceso de construcción de éstos módulos nos encontrábamos con componentes que pudieran ser reutilizados, estos componentes se convertirían en candidatos para extender el framework de la institución.

2.1. Lineamientos

Con el propósito de garantizar que el código desarrollado por los diferentes proveedores fuera de alta calidad y apegado a las mejores prácticas de la industria, la institución incluyó en los requerimientos una serie de lineamientos de diseño y desarrollo que deberían ser observados por todos los participantes en el proyecto.

Los lineamientos de desarrollo incluyen recomendaciones respecto a número máximo de líneas de código por clase, nombres de clases, atributos y métodos, niveles máximos de anidamiento en estructuras condicionales o de iteración.

En la solución propuesta por la institución, el ECE adoptará un diseño MVC [2] con un enfoque SOA[3] a nivel institucional, donde podrá consumir los servicios ofrecidos por las distintas aplicaciones cuando se requiera, sin necesidad de colisionar con las arquitecturas y modelos de estas otras aplicaciones.

La propuesta SOA es un estilo de arquitectura que se basa en la creación de un conjunto de servicios, de diferente granularidad, entre los procesos de negocio y las diversas aplicaciones, con el objetivo de modelar la lógica de negocio como servicios para poder expresar la capa de servicios mediante la facilidad que ofrece la orquestación de los mismos, utilizando el siguiente razonamiento:

- Hacer uso de la capa de servicios que ofrece la funcionalidad de la capa de aplicación de forma independiente a la tecnología que la soporta.
- Minimizar las dependencias entre la capa de servicios y la de presentación para desacoplar el negocio de la tecnología y, de este modo, permitir los cambios en cualquiera de ellas. El objetivo sería favorecer la agilidad para el negocio.
- Reutilizar los servicios de negocio, por medio de su publicación en el Bus de Servicios Corporativos (ESB-Enterprise Service Bus).

2.2. Persistencia de Datos

La capa de persistencia del framework integrado por la institución está basada en Hibernate 2.1, que es un proyecto open source que proporciona una biblioteca de funciones Java para mapear objetos Java a un modelo tradicional de base de datos relacional (ORM). La principal característica de Hibernate es que permite la conversión de clases Java a tablas en la base de datos así como el mapeo de tipos de dato Java a tipos de dato SQL. Adicionalmente Hibernate proporciona métodos para el guardado, consulta, actualización y borrado de información en la base de datos.

[2] Patrón de diseño Model View Controller por sus siglas en inglés.
[3] Arquitectura Orientada a Servicios por sus siglas en inglés.

2.3. Entorno de desarrollo

El *framework* provisto por la institución usa Spring 2.5, que es un entorno open source para el desarrollo de aplicaciones basado en Java, que surgió como una alternativa a la propuesta de EJB[4] contenida en la especificación JEE[5] . Este framework permite la configuración de los componentes de aplicación y la administración del ciclo de vida de los objetos Java. Spring proporciona además un contenedor de inversión de control y permite el uso de interceptores, dos características que se usaron ampliamente durante el desarrollo del ECE.

La funcionalidad correspondiente a la capa de servicios del framework de la institución debería ser construida y orquestada usando el marco de desarrollo Spring.

Adicionalmente, el framework de la institución usa el módulo *Spring Security* para implementar la seguridad basada en roles.

2.4. Presentación

Para el desarrollo de la interfaz gráfica para el usuario, el framework de la institución considera el uso de JSF[6] que es una especificación de JEE para el desarrollo de aplicaciones web. JSF usa Java Server Pages para mostrar las páginas y proporciona un biblioteca de componentes para la interfaz de usuario, incluyendo los elementos básicos de HTML para representar formularios. Estos componentes pueden ser usados como base para la definición de componentes nuevos.

JSF usa un modelo de JavaBeans para comunicar eventos en los controles de la interfaz de usuario en el cliente (browser) al servidor donde se aloja la aplicación.

Adicionalmente, el framework contempla el uso de Rich Faces para agregar funcionalidad AJAX, principalmente validación del lado del cliente, auto completado de textos y encolado de mensajes entre el cliente y el servidor de la aplicación.

2.5. Reportes

Para la generación de los diversos documentos requeridos por los usuarios el framework de la institución establece el uso de Jasper Reports.

Esta herramienta consiste de un ambiente gráfico donde se diseñan los reportes y se generan los archivos ejecutables que son interpretados por un servidor de reportes.

[4] Enterprise Java Beans
[5] Java Enterprise Edition
[6] Java Server Faces

Estos reportes pueden embeberse en una aplicación web, integrándose al esquema de seguridad de la misma. En este proyecto se usó la versión open source de esta herramienta.

2.6. Canal de comunicaciones

Como ya se mencionó, uno de los objetivos a nivel tecnológico de la institución es implementar una SOA a nivel institucional, soportada por una infraestructura orientada a servicios (SOI), en la cual el Enterprise Service Bus (ESB) será uno de los componentes que proporcionará los servicios de infraestructura a las aplicaciones.

Para lograr lo anterior, además de los servicios orquestadores del negocio y reglas de negocio que conformarán las propias transacciones del sistema ECE, es necesario contar con servicios de integración y conectividad que permitan intercambiar y compartir información de otros sistemas existentes. La integración de estos servicios mencionados se manejará a través del ESB del que dispone la institución. El framework desarrollado por la institución incluye componentes que permiten consumir o exponer servicios en este canal.

La figura 2 muestra una vista de alto nivel de la arquitectura propuesta.

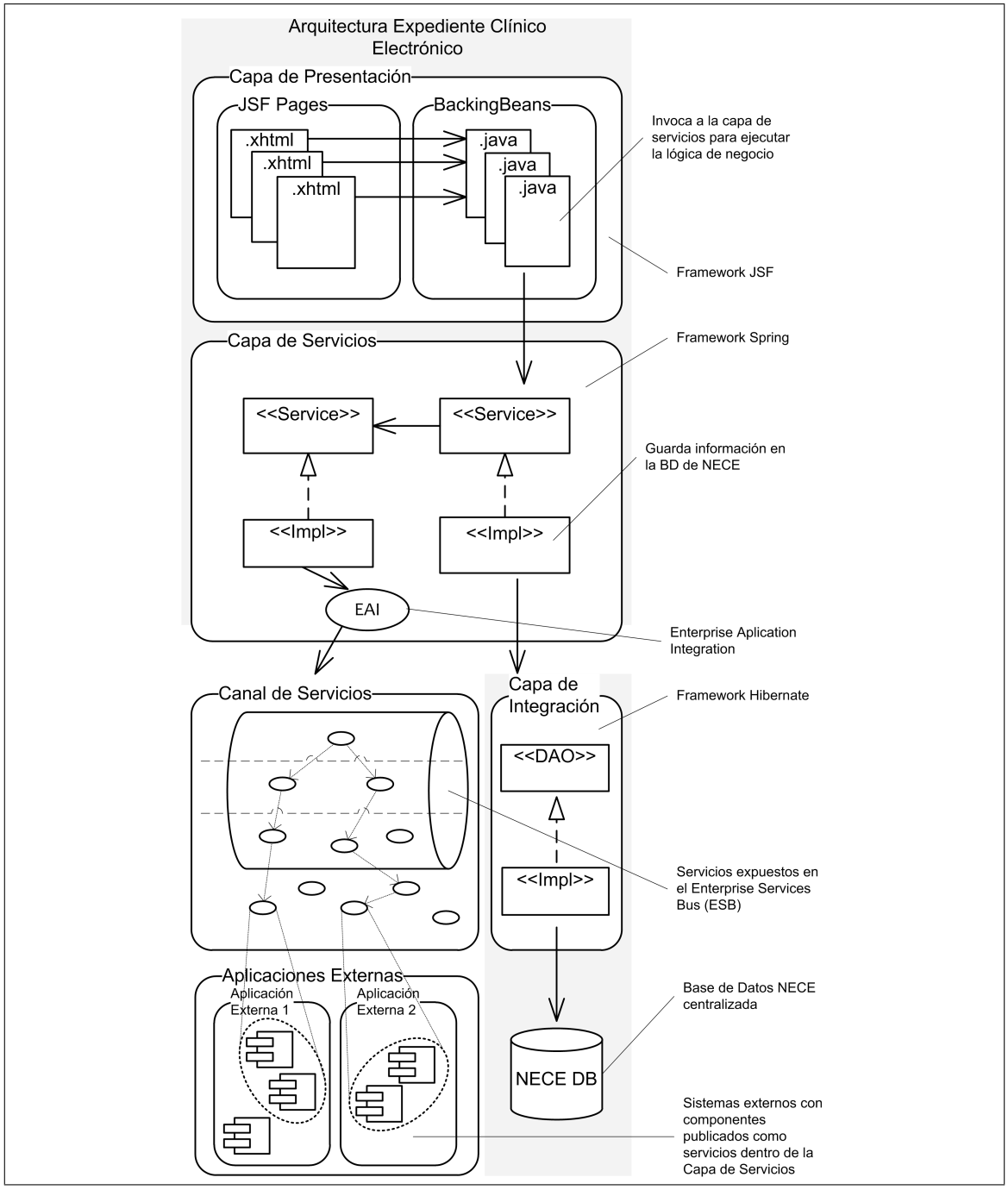

Figura 2: Arquitectura y modelo de interoperabilidad.

3. Desarrollo de la aplicación

Al equipo de desarrollo a mi cargo se le asignó el desarrollo de los componentes para las clínicas de medicina familiar. Los módulos a desarrollar fueron los siguientes:

1. Vigilancia prenatal. Atención para que las mujeres mantengan embarazos normales por medio de acciones médicas que incluyen la identificación de condiciones de salud pre-existentes, detección temprana de complicaciones y fomento a la salud y prevención de enfermedades.
2. Climaterio y menopausia. Registro de diagnósticos y tratamiento para casos de climaterio.
3. Salud reproductiva. Registro y seguimiento de acciones orientadas a difundir información sobre planificación familiar. Diagnóstico y tratamiento de problemas de salud reproductiva.
4. Nutrición. Registro del estado nutricional de los pacientes así como planes de acción para promover una alimentación saludable.
5. Trabajo Social. Identificar los factores sociales que inciden en la salud de los pacientes. Registro y seguimiento de intervenciones de trabajo social.
6. Programas Integrados. Registro y seguimiento de las cartillas de salud y acciones asociadas para distintos tipos de pacientes (infantes, adultos mayores, climaterio, etc.) Estos programas están orientados principalmente a la ejecución de acciones preventivas.
7. Estomatología. Registro y seguimiento de diagnósticos y tratamiento a problemas bucales así como acciones orientadas a mejorar la higiene bucal.
8. Salud en el trabajo. Calificación y dictamen de riesgos y accidentes de trabajo.
9. Hojas de control. Registro y seguimiento de pacientes con enfermedades crónicas (Obesidad, Diabetes, Hipertensión, etc.)

3.1. Requerimientos del sistema

Los requerimientos funcionales para la aplicación estaban plasmados principalmente en tres tipos de documentos:

1. Casos de uso que establecían los actores, condiciones previas, resultados esperados y condiciones posteriores a la ejecución del caso de uso.
2. Prototipos de las pantallas donde se definía claramente la organización de cada pantalla y los mecanismos para comunicar al usuario el resultado del proceso de los datos capturados.

3. Reglas de negocio:
    a) Tablas de referencia para signos vitales o indicadores médicos.
    b) Instrucciones para calcular índices dependiendo del tipo de paciente y padecimiento a tratar.
    c) Rangos y valores para validar datos.
    d) Información específica o adicional dependiente de los datos proporcionados al sistema.

Adicionalmente se contaba con acceso a las aplicaciones que este proyecto debería sustituir y a los responsables de su mantenimiento y operación.

Por otro lado, la aplicación debería ser compatible con la primera versión que ya estaba en pruebas piloto con usuarios finales y con los módulos para atención de segundo y tercer nivel que estaban siendo desarrolladas simultáneamente por otro equipo.

Esto establecía una serie de requerimientos no funcionales que la aplicación debería cumplir para garantizar dicha compatibilidad. El desarrollo de estos módulos debería estar listo para pruebas de aceptación en 6 meses.

Para iniciar los trabajos en los dos módulos nuevos se estableció como línea base la versión 1.0 de la aplicación. Es decir, nuestro trabajo no iniciaría desde cero sino que deberíamos extender la versión 1.0 con nuestro código, garantizando que toda la funcionalidad existente continuaría funcionando tal como se nos entregó. Se estableció como requerimiento adicional que ningún componente de la línea base debería ser modificado.

3.2. Estrategia de trabajo

Con el fin de acelerar el desarrollo lo más posible, se crearon grupos de trabajo para cada módulo de medicina familiar, que estaban integrados de la siguiente manera:

- Un arquitecto responsable del modelo de datos, las clases de la capa de persistencia y la definición de los servicios a construir.
- Un programador *senior* responsable del diseño y construcción de los *backing beans*[7] y pantallas HTML, así como de supervisar el trabajo de los programadores a su cargo.
- Un número variable de programadores *junior* responsables de desarrollar el código Java y HTML necesario.
- Un número variable de programadores especializados en Jasper responsables de desarrollar e integrar todos los reportes y formatos para impresión necesarios.

[7] Bean administrado que es referenciado por una página JSF.

Adicionalmente un grupo de DBA[8] se encargaba de mantener actualizados y estables el modelo de datos y las tablas de catálogos del sistema. Este grupo era responsable, junto con los arquitectos de cada módulo, de garantizar que las extensiones propuestas al modelo de datos fueran congruentes entre sí y con los modelos de los otros dos esfuerzos de desarrollo.

En cada módulo se trabajaba desde los dos extremos de la aplicación simultáneamente. Por un lado los arquitectos definían el modelo de datos y las clases de persistencia. Al mismo tiempo, el líder de desarrollo trabajaba con su equipo en la construcción de las pantallas y todos los componentes necesarios para implementar las reglas de negocio.

La premisa de esta decisión era que el equipo de desarrollo no estuviera ocioso mientras se terminaba de definir el modelo de datos. Las actividades del grupo de desarrollo incluían familiarizarse con las clases que proporcionaban la seguridad de la aplicación y las que mostraban los menús de opciones a los usuarios, dependiendo del rol con el que accedieron al sistema.

Para el caso del módulo de Salud en el Trabajo, además de los requerimientos ya comentados, el sistema debería usar *Web Services* para intercambiar información con otras aplicaciones. Sin embargo, estos servicios no estaban completamente definidos ni integrados en el ESB de la institución, por lo que se acordó avanzar en el resto de la funcionalidad mientras estas definiciones se terminaban.

3.3. Modelo de datos

Después de analizar los requerimientos y compararlos con el modelo de datos entregado como parte de la línea base, nos dimos cuenta que los cambios necesarios al modelo de datos se podían clasificar en tres categorías:

1. Que las entidades y relaciones fueran completamente nuevas.
2. Que fuera necesario extender alguna entidad existente con datos adicionales. Por ejemplo, ya existía una entidad para modelar información histórica del paciente, pero esta entidad no consideraba escolaridad, último grado escolar cursado o afiliación religiosa.
3. Que las entidades existentes estuvieran en franca contradicción con lo requerido en los nuevos módulos.

El modelo de datos de la aplicación es enorme y no es el propósito de este documento el hacer un análisis detallado del mismo. Sin embargo, para fines de ejemplo de los problemas antes descrito se incluyen imágenes parciales del modelo.

[8] Administrador de base de datos por sus siglas en inglés.

El escenario 1 fué el más fácil de manejar ya que bastaba con extender el modelo con los nuevos objetos. La figura 3 muestra un caso de este escenario. En el módulo de trabajo social se requería crear un registro de la nota de trabajo social, sin necesidad de tener una nota médica asociada, pero identificando claramente a qué atención correspondía cada nota. La solución en este caso fue simplemente añadir la tabla y relaciones necesarias. La tabla NET_NOTA_TRABAJO_SOCIAL es la estructura propuesta.

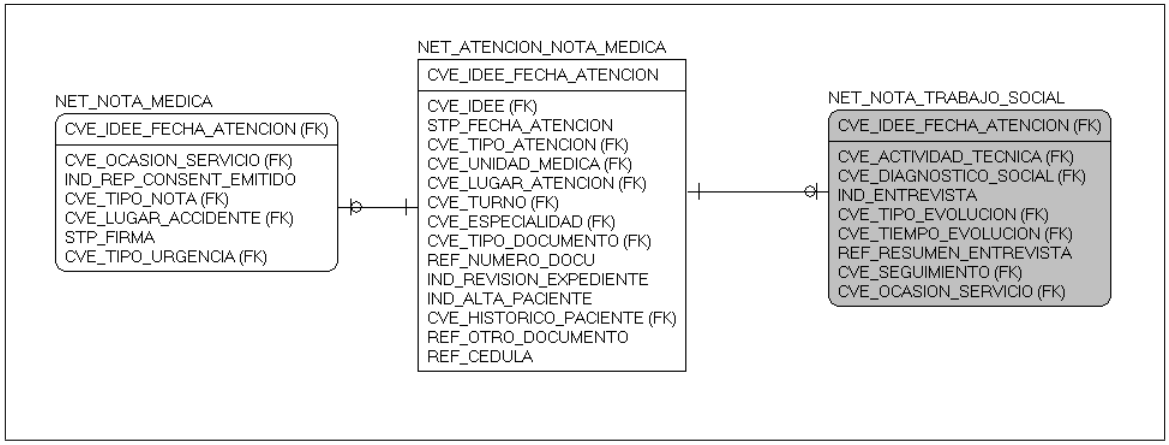

Figura 3: Agregar entidades nuevas al modelo.

Para resolver los problemas presentados por el escenario 2 se decidió crear una tabla nueva, con una relación uno a uno con la tabla existente; y agregar a esta nueva entidad la información requerida. Se decidió hacerlo de esta manera para evitar modificar el código existente o que éste tuviera errores al tener nuevos datos o restricciones en la base de datos.

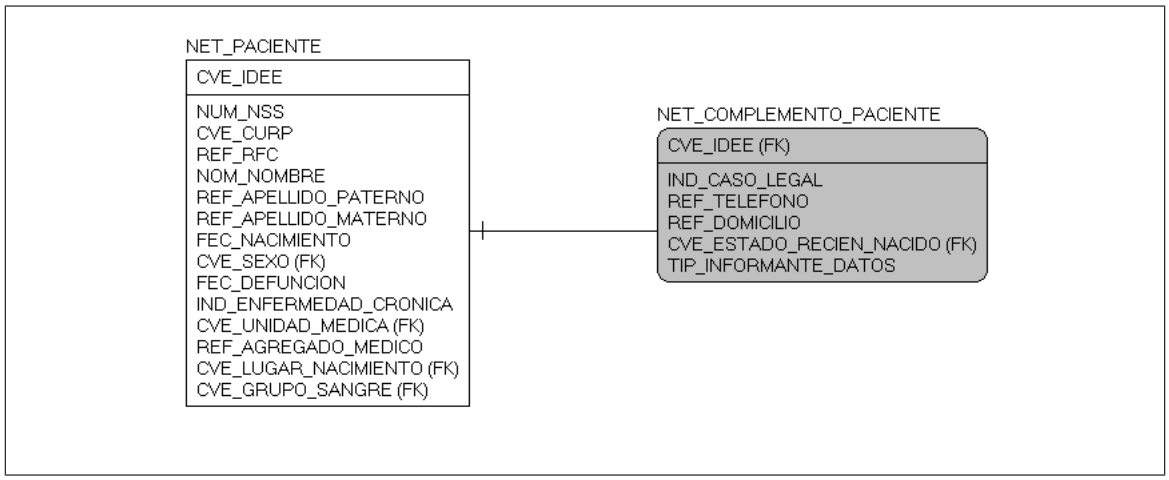

Figura 4: Tabla que extiende una entidad existente.

La figura 4 muestra uno de estos casos. El modelo de la línea base ya consideraba una tabla para modelar la información de los pacientes, aunque en los requerimientos de

medicina familiar se establecían requerimientos de información adicionales. En la tabla NET_COMPLEMENTO_PACIENTE se consideró la información adicional requerida.

Para resolver los problemas presentados por el escenario 3, se acordó revisar cada caso con los responsables de los otros dos esfuerzos de desarrollo. De tal manera que en algunos casos se modificó el modelo de la línea base mientras que en otros nos dimos cuenta que el origen del problema eran huecos en los requerimientos de usuario. Como nuestro compromiso era la entrega del sistema tal y como se nos había requerido, en estos casos se decidió crear una estructura de datos alterna e informar a los responsables dentro de la institución del hallazgo y medidas tomadas.

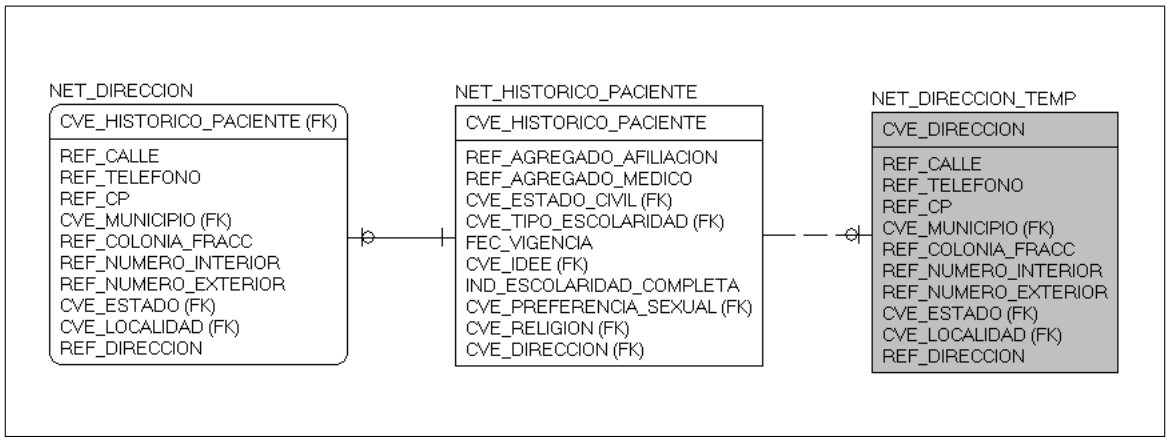

Figura 5: Propuesta para resolver conflictos en el modelo.

La figura 5 muestra un caso de este escenario. En el modelo original, la entidad que modela las direcciones de los pacientes depende de la entidad de datos históricos del paciente, mientras que en los requerimientos de medicina familiar se solicitaba guardar varias direcciones de un paciente sin guardar información en el registro histórico, como direcciones de empleos, etc.

Para resolver esta situación se propuso que la entidad dirección no dependiera de otras entidades y que se agregaran referencias a dirección en las entidades que así lo requirieran, incluyendo la tabla de datos históricos.

El acuerdo al que se llegó es que los dos módulos nuevos usarían la nueva propuesta y sólo se modificaría la tabla NET_HISTORICO_PACIENTE para agregar la nueva referencia. En una revisión posterior se adaptaría la versión 1.0 al nuevo modelo de datos. La tabla NET_DIRECCION_TEMP es la propuesta de modificación.

3.4. Capa de persistencia de datos

Una vez que se terminó el modelo de datos, la siguiente tarea para el equipo de diseño consistía en crear las clases para establecer las relaciones entre la base de datos relacional

y los objetos java. El framework de la institución contempla el uso de Hibernate para tal fin y el código de la línea base incluía muchas clases de este tipo que podían ser usadas como referencia para el desarrollo de las nuevas clases.

El listado 1 reproduce parte de una clase de la línea base. No se muestran todos los atributos de la tabla ni los setters y getters porque esa parte del código es muy repetitiva. Por razones de confidencialidad se omiten algunos comentarios.

En la primera parte de la clase pueden observarse una serie de imports de elementos del paquete javax.persistence. En JEE, el paquete javax.persistence contiene las clases e interfaces que definen los contratos entre un proveedor de persistencia y las clases manejadas y los clientes de la API de Java Persistence.

Listado 1: Ejemplo de imports en entidad hibernate

```
package ****.****.****.nece.paciente.integracion.hbm;
import java.util.Date;
import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
```

El código en el listado 2 usa anotaciones para establecer que esta clase es una entidad hibernate relacionada con la tabla NET_PACIENTE y que extiende la clase EntidadPersistente del framework de la institución.

Listado 2: Uso de anotaciones

```
@Entity
@Table(name = "NET_PACIENTE")
public class Paciente extends EntidadPersistente {
```

El listado 3 muestra código donde se establece que el identificador de esta clase es el atributo idee, mismo que está relacionado con la columna CVE_IDEE, de tipo cadena y largo 18 caracteres, que es único en la tabla, es un dato requerido que puede ser insertado y modificado

Listado 3: Definición de identificador

```
@Id
@Column(name = "CVE_IDEE", unique = true, nullable = false,
                insertable = true, updatable = true,
                length = 18)
        private String idee;
```

El código en el listado 4 establece que el atributo grupoSangre está asociado a la columna CVE_GRUPO_SANGRE, que no es único ni requerido y que además es una referencia a otra entidad, en este caso un catálogo

Listado 4: Referencia a otras entidades

```
@ManyToOne
@JoinColumn(name = "CVE_GRUPO_SANGRE", unique = false,
                    nullable = true,insertable = true,
                    updatable = true)
        private GrupoSangre grupoSangre;
```

Definir este tipo de relaciones causa que cuando se recupera información de estas entidades, Hibernate haga los *join* necesarios para recuperar información de todas las tablas asociadas. En el ejemplo, una consulta a la tabla paciente trae el código de grupo de sangre y, por medio del *join* correspondiente, la descripción del grupo de sangre.

Como puede apreciarse en la porción de código antes referida, escribir estas clases es un trabajo muy repetitivo, propenso a errores y que depende totalmente del modelo de datos, ya que las entidades Hibernate están íntimamente ligadas con las tablas relacionales que representan.

Por lo anterior se tomó la decisión de no escribir estas clases manualmente. En su lugar se construyó un generador de código basado en herramientas del mismo Hibernate [9].

Para tratar de limitar el número de *join* generados automáticamente por Hibernate se tomó la decisión de mapear solamente las relaciones entre tablas que representaran datos del negocio, sin incluir las relaciones con los catálogos.

3.5. Desarrollo de pantallas

La primer tarea del grupo de desarrollo asignado a la construcción de las pantallas del sistema fue identificar cuales de los componentes del código proporcionado como línea base deberían ser reutilizados en nuestro esfuerzo de desarrollo.

En principio, todos los elementos de apariencia visual, hojas de estilos y componentes javascript, deberían ser reutilizados y, de ser necesario, extendidos para soportar algún requerimiento adicional. De igual manera, todas las pantallas del sistema comparten el mismo encabezado y estilo de despliegue de los menús de opciones, por lo que esos elementos también deberían ser reutilizados.

Una vez identificados los elementos a reutilizar, el equipo construyó las pantallas de acceso para cada módulo. Estas pantallas contenían diferentes pestañas para implementar en cada una de ellas las pantallas requeridas.

En JSF, una pantalla de la aplicación web tiene al menos dos componentes:

[9] En la actualidad existe un plug-in de eclipse que genera este tipo de clases a partir de una conexión a la base de datos

1. El código HTML que presenta los formatos. La institución decidió que para este proyecto esto se codificara en archivos XHTML.
2. Un bean java manejado que proporciona la lógica de validación y navegación.

De acuerdo a los lineamientos de la institución, (ver figura 1) la lógica de validación de campos debería ser codificada en archivos separados, lo mismo que la lógica de conversión de datos.

El listado 5 muestra código muestra donde se importan los paquetes de JSF en un archivo XHTML

Listado 5: Imports de JSF

```
html xmlns="http://www.w3.org/1999/xhtml"
        xmlns:h="http://java.sun.com/jsf/html"
        xmlns:f="http://java.sun.com/jsf/core"
        xmlns:ui="http://java.sun.com/jsf/facelets"
        xmlns:a4j="http://richfaces.org/a4j"
        xmlns:rich="http://richfaces.org/rich"
```

El listado 6 muestra cómo se especifica un formato con un panel con pestañas y cómo se define el formato de una pestaña y el bean java al que está vinculado. El ejemplo corresponde al módulo de nutrición, la sección donde se registra el motivo por el que se envió al paciente al servicio.

Listado 6: Vinculación entre formato XHTML y bean java

```
<rich:tabPanel switchType="ajax"
     id="tab_panel_nutricion"
     binding="#{registrarMotivoEnvioBean.nutricionContainer}">
<rich:tab name="tab_motivo_envio"
     label="#{msgNutricion.nutricion_tab_header_motivo_envio}">
<ui:include
     src="/pages/nutricion/cunesdand01/registrarMotivoEnvio.xhtml" />
</rich:tab>
```

El bean java que soporta la funcionalidad de esta pantalla es registrarMotivoEnvioBean

En todas las pantallas del sistema se debe mostrar siempre la información del paciente al que se hace referencia. Los datos y formato de presentación son siempre los mismos, por lo que se identificó el componente que busca y presenta esta información para ser re-utilizado en lugar de generar un nuevo componente con la misma información.

El listado 7 muestra cómo se incluyen componentes preconstruidos en una pantalla JSF. En este caso, paciente.xhtml es el componente que localiza y presenta la información de cada paciente.

Listado 7: Vinculación entre formato XHTML y bean java

```
<rich:tabPanel switchType="ajax"
<ui:define name="contenido">
        <ui:include src="/WEB-INF/templates/paciente.xhtml" />
        <rich:spacer width="1" height="5"/>
```

En general, el proceso de construcción de las pantallas fue laborioso pero relativamente simple. El mayor reto lo encontramos en formatos que tenían muchas pestañas donde los usuarios querían guardar avances parciales sin que esto se considerara como un registro completo.

Este requerimiento surge de una restricción del sistema: dada la naturaleza de la información, el sistema no debería permitir cambios a los registros una vez salvados. En caso de que un usuario decidiera que la información registrada en el sistema no era correcta, debería crear un registro nuevo y marcar el anterior como actualizado, pero no borrarlo o sobre-escribirlo.

El sistema tiene algunos módulos que requieren mucha información, donde la captura se organizó en diferentes pantallas a las que se accede por medio de pestañas en un formato. En ocasiones el usuario avanza en las pestañas sólo para llegar a un punto donde no tiene la información necesaria para seguir adelante. De acuerdo a la especificación original, el usuario solo tenía dos opciones en esta situación: cancelar el registro con la consecuente pérdida de tiempo y necesidad de repetir el proceso cuando se tuviera la información completa. o bien, tratar de salvar los datos, cosa que a veces no era posible porque faltaba información requerida, o que no era conveniente, pues cuando se tuviera la información completa debería ser ingresada como una corrección al registro original.

La solución mas simple para este problema era partir el registro largo en registros más pequeños, que pudieran ser guardados individualmente, como pudieran ser los datos de cada pestaña. Pero esta solución no fue aceptada por el área médica, quienes argumentaron que esto era información incompleta y que podría ser causa de interpretaciones erróneas. Se hicieron tres propuestas de solución:

1. Crear una estructura paralela en la base de datos donde se guardaran los registros incompletos.
2. Guardar clases serializadas en el sistema de archivos de los servidores.
3. Crear una tabla nueva que permitiera guardar un archivo XML con los datos parciales.

En cualquiera de los tres escenarios era necesario extender los requerimientos iniciales para permitir el guardado parcial por cualquiera de los tres medios antes descritos y re-

cuperar la información cuando algún usuario ingresara a la misma pantalla para el mismo paciente.

Al final se optó por implementar la solución número 3, ya que la solución número 1 demandaba la creación de nuevas tablas en el sistema cada vez que se presentara esta situación. Estas tablas complicaban el darle mantenimiento a la base de datos y podían ser causa de errores. La solución 2 fue descartada porque no funcionaría correctamente cuando la aplicación se ejecutara en un cluster[10] de servidores.

Otro problema que enfrentamos en el desarrollo de esta capa se debió a que la falta de un inventario completo de componentes JSF, en particular validadores y convertidores, era la causa de que el re-uso de componentes se limitó a grandes componentes, principalmente los que presentan información de pacientes, información de médicos, el encabezado y pie de página de cada pantalla y reporte, y los componentes que muestran padecimientos y tratamientos.

Adicionalmente a lo anterior, la gran rotación de personal y la premura por mostrar avances al área de prestaciones medicas causó que se codificaran muchos métodos que tienen funcionalidad idéntica o muy parecida, incluso no se respetaron los lineamientos en el sentido de codificar los validadores y convertidores en clases separadas, construyendo todos estos métodos como parte de los *backing beans*.

3.6. Desarrollo de servicios

De acuerdo a los lineamientos de la institución, la capa de servicios debería ser responsable de dos tipos de servicios:

1. Guardar o recuperar información que tuviera sentido desde la perspectiva médica. Por ejemplo guardar o recuperar todos los datos de una nota médica. No recuperar sólo signos vitales, o algún otro dato de forma aislada.
2. Validar que la información cumple con las reglas de negocio. Es importante señalar que se permitía la validación de algunas reglas desde el momento mismo de la captura, para proporcionar una mejor experiencia de usuario.

En general se desarrollaron clases de servicios para cada módulo, con un servicio para guardar información y al menos otro para recuperar información. El listado 8 muestra parte del servicio que guarda los antecedentes de problemas nutricionales.

Listado 8: Guardado de información en la base de datos (continúa ...)

```
@Override
public Boolean registrarAntecedentesFactores(
   AntecedentesFactoresDto antecedentes) throws AbstractException {
       List<AntecedentesQuirurgicosDto> antQuirList =
```

[10] Cúmulo o agrupación de servidores

Listado 8: Guardado de información en la base de datos (continúa ...)

```
            antecedentes.getAntecedentes();
        AntecedenteNutricional antNutri = new AntecedenteNutricional();
       // Farmacos
       List<FarmacoNutrimentoDto>
          farmacos = antecedentes.getFarmacos();

       for(FarmacoNutrimentoDto farmaDto: farmacos){
       logger.debug("Farmaco:"+farmaDto.getClaveMedicamentoNutricion());
       FarmacoNutrimento farNutri = new FarmacoNutrimento();
       FarmacoNutrimentoId id = new FarmacoNutrimentoId();
     id.setClaveIdeeFechaAtencion(farmaDto.getClaveIdeeFechaAtencion());
       id.setClaveMedicamentoNutricion
            (Integer.parseInt(farmaDto.getClaveMedicamentoNutricion()));
       farNutri.setId(id);
    }
```

Este ejemplo muestra el método registrarAntecedentesFactores, que recibe un objeto AntecedentesFactoresDto como parámetro y que regresa verdadero en caso de guardar todos los antecedentes correctamente, falso si hubo algún error controlable en el proceso o una excepción si ocurrió algún problema mayor.

El objeto AntecedentesFactoresDto es un objeto complejo que está compuesto por otros objetos. En el ejemplo, el código verifica si incluye una lista de fármacos consumidos por el paciente y, de ser así, prepara los objetos para insertarlos en la base de datos.

Dependiendo de los requerimientos, en algunos casos se agregaron servicios para validar el cumplimiento de las reglas de negocio antes de intentar guardar la información. Cuando estos servicios encontraban que no se cumplían las reglas de negocio, se omitía el guardado de datos y se notificaba al usuario los errores encontrados.

En la capa de servicios se hicieron las conversiones de datos entre las estructuras necesarias para mostrar información en las pantallas y las estructuras con las que se debería guardar la información en la base de datos.

Es importante señalar que en esta capa se trató de no usar las descripciones de los catálogos, sino que sólo usábamos las claves y dejábamos la tarea de localizar las descripciones a los servicios del *framework* que estaban disponibles en la capa de presentación.

3.7. Transporte de datos

Para cumplir con el requerimiento de desacoplar la capa de servicios y la capa de presentación se diseñaron objetos de transporte (DTO por sus siglas en inglés) que serían usados para intercambiar información entre estas dos capas.

La idea es simple: se construyeron clases que contenían todos los elementos de información que necesitaba una pantalla y estas clases se pasaban a los servicios para que se validaran y se iniciaran transacciones en la base de datos que guardaran los componentes en las tablas que correspondiera. De esta manera garantizábamos la integridad de la información y no teníamos que hacer una multitud de intercambios entre las capas de presentación y de servicios.

El listado 9 muestra una estructura típica. Incluye una lista de antecedentes socioeconómicos, cada uno de estos registros incluye información del número de hogares al que ha pertenecido un paciente, una lista de registros de actividad física, donde se detalla el tipo de actividad y tiempo que le dedica y una lista de fármacos consumidos por el paciente y sus efectos metabólicos.

Listado 9: Guardado de información en la base de datos

```
@Override
public Boolean registrarAntecedentesFactores(
public class ReporteEvaluacionNutriciaDto {
private String fechaExp  ;
private ArrayList<String> anteceden_socioe ;
/*  agua potable
     drenaje
     refrigerador
     No. de familia
     adultos
     menores
     lugar
*/
private String datos_nutricionales ;
private ArrayList<String > actividadFis ;
/*  ejercicio
     tiempo
     frecuencia
*/
private ArrayList<String > interacFarmaco ;
/*  medicamento
     efecto metabÛlico
*/
private String antecedentes ;
private String antecedenQuirurgic ;
private String transtornosGastro ;
private String usoSuplementos ;
```

El uso de los objetos DTO fue controvertido por el grupo a cargo del mantenimiento de la versión 1.0 de la aplicación. Ellos argumentaban que este tipo de estructuras no se usaba en la versión inicial del sistema y que constituían un uso innecesario de recursos y de proceso al tener que poblar estas estructuras en lugar de referir directamente atributos de los backing beans de la capa de presentación u objetos Hibernate.

A favor del uso de estos objetos argumentamos que de otra forma no se lograba un verdadero desacoplamiento entre las tres capas del sistema y que al no tener estas estructuras, que incluían toda la información necesaria para registrar una transacción de negocio, se incrementaba mucho el diálogo entre las diferentes capas y se aumentaba mucho la complejidad de los servicios, al tener que agregar funcionalidad para garantizar que no se guardaban datos parciales, violando así la integridad de la base de datos.

En la medida que se iban terminando los módulos se sometían a pruebas por parte de un grupo interno, el cual aplicaba una serie de casos de prueba. Si encontraban algún problema se reportaba al equipo de desarrollo para su atención. Después de algunas iteraciones se consideró que el desarrollo era correcto y se inició la presentación a los usuarios finales.

En ese momento sólo se podían probar módulos aislados, por lo que no se podía realizar una prueba integral del sistema.

4. Integración de los sub-proyectos

Al equipo a mi cargo se le asignó la responsabilidad de integrar en un solo proyecto el esfuerzo de desarrollo. Como ya se explicó anteriormente, el proyecto estaba dividido en tres sub-proyectos:

1. Mantenimiento a la versión 1.0.
2. Desarrollo de la funcionalidad para atención hospitalaria.
3. Desarrollo de la funcionalidad para medicina familiar.

Esta integración debía realizarse sin detener los esfuerzos unitarios y garantizando que en cada iteración se tendría una aplicación estable que podría usarse para iniciar las pruebas integrales del sistema.

El esfuerzo de integración requería crear un solo modelo de datos e implementar este modelo; había que poblarlo con datos para pruebas que fueran congruentes y correctos.

Se debía unificar el código desarrollado por los tres grupos de trabajo. Esta tarea incluía clases java, archivos de configuración y demás recursos de la aplicación en un solo proyecto y preparar los ambientes: uno donde se pudieran corregir los errores detectados durante la integración, un segundo ambiente para hacer pruebas unitarias e integrales y uno más donde se pudieran hacer demostraciones a los usuarios finales.

La integración del modelo de datos fue relativamente simple. Con base en los acuerdos tomados al inicio del desarrollo, los casos problema eran muy pocos y bien conocidos, por lo que el código ya estaba listo para manejar estas situaciones.

Sin embargo, la integración del contenido de la base de datos fue una historia completamente diferente. El problema más común se presentaba con los catálogos que eran compartidos por dos o más sub-proyectos. Era muy frecuente que las llaves se hubieran duplicado y asignado a elementos totalmente diferentes.

Para ayudar a solucionar esta situación se creó una aplicación que comparaba dos versiones de un catálogo y reportaba incongruencias; este reporte se entregaba al analista que estuviera a cargo del módulo y al equipo de DBA para que resolvieran los conflictos y se actualizara la base de datos con los valores correctos. Como la versión 1.0 ya estaba en producción, en general los catálogos de esta versión se conservaban y se corregían los datos requeridos por los otros dos sub-proyectos.

El siguiente paso consistió en integrar el código fuente de la aplicación. Para esta tarea se decidió usar el sistema de gestión de versiones Mercurial, ya que esta herramienta ofrecía más facilidades para el manejo de ramas del repositorio y para la comparación y combinado de archivos.

Se crearon cuatro repositorios Mercurial, uno para la versión final de la aplicación, al que llamamos ece-integrado, y tres más para cada sub-proyecto de desarrollo, a los que lla-

mamos ece-base, ece-hospital y ece-familiar, respectivamente.

De esta forma no era necesario detener los esfuerzos de desarrollo de cada sub-proyecto mientras se hacía la integración. Cada grupo cargaba en estos repositorios el trabajo diario mientras que el equipo encargado de la integración analizaba el impacto de los cambios. Primero se inicializó el repositorio ece-integrado con un clon del repositorio ece-base, se compiló el código para garantizar que se tenía una versión estable y se desplegó usando el nuevo modelo de datos para aplicar una serie de pruebas básicas.

El siguiente paso fue combinar el repositorio ece-integrado con el repositorio ece-familiar. Esta tarea nos permitió identificar diferentes situaciones:

1. La mayoría de los archivos del repositorio ece-familiar eran completamente nuevos para el repositorio ece-integrado.
2. Algunos archivos del repositorio ece-integrado habían sido modificados en el repositorio ece-familiar. Estas modificaciones consistían únicamente en la adición de código nuevo, sin modificar al código de la línea base.
3. Algunos archivos habían sido modificados tanto en el proyecto ece-base como en ece-familiar.

Los casos 1 y 2 no presentaban problema alguno pues bastaba con agregar al repositorio ece-integrado los archivos nuevos o las versiones modificadas del repositorio ece-familiar. Por otro lado, el caso 3 requería de un análisis detallado del código modificado, incluso en algunos casos fue necesario reunir a los responsables de los cambios y buscar la mejor solución al problema. Una vez terminada la integración del sub-proyecto de medicina familiar, el código fue compilado y desplegado para hacer pruebas y corregir cualquier problema que hubiera surgido del proceso de integración.

Al terminar esta etapa, se agregó al repositorio ece-integrado el contenido del repositorio ece-hospital con un procedimiento similar al usado para integrar ece-familiar. En esta tarea encontramos un número considerable de problemas del tipo 3, normalmente causados porque en los dos sub-proyectos se había desarrollado funcionalidad similar. Cuando esto pasaba elegíamos los métodos que parecían estar más completos y se actualizaban las clases y configuraciones para considerar estos cambios.

Como ya se comentó, todo esto se realizó sin detener el avance de los sub-proyectos, que seguían generando código. Por esta razón el repositorio ece-integrado no se usaba como la fuente del código de referencia del proyecto, en su lugar siempre se refería a cualquiera de los tres repositorios de los sub-proyectos.

En Marzo de 2012 se consideró que los tres sub-proyectos habían alcanzado un nivel aceptable de madurez y se dio la instrucción de generar una versión integrada del código.

Desde la perspectiva del código la instrucción era relativamente simple de implementar. Para tal fin bastaba con pedir una ventana de tiempo durante la cual no se aceptarían

actualizaciones en los repositorios de los sub-proyectos. Durante este tiempo se integraría el código de acuerdo al proceso antes descrito.

Al final de esta ventana, se tendría una versión unificada de la aplicación a la que se le repetirían las pruebas individuales por módulo para garantizar que no se hubieran añadido errores durante el proceso de integración y, por primera vez desde el inicio del proyecto, pruebas integrales de sistema para garantizar que los diferentes módulos podían interactuar entre sí de acuerdo a lo establecido en las especificaciones del proyecto.

Al mismo tiempo, esta integración significaba cambios importantes en el proceso de gestión de desarrollo de software:

1. Los líderes de cada sub-proyecto se convertían en los responsables de subir los cambios al repositorio central y ningún otro elemento de los equipos podía realizar esta tarea.
2. Estos líderes eran responsables de que el código en este repositorio compilara correctamente y pudiera ser desplegado usando el modelo de datos unificado.
3. Todas las versiones de la aplicación que fueran presentadas a los usuarios finales a partir de este momento deberían ser construidas con base en el código de este repositorio.
4. Los scripts para modificar el modelo de datos o cargar datos a las bases de datos de los ambientes de pruebas deberían estar cargados en el repositorio ece-integrado.

La implementación de este proceso no fue recibida con simpatía por parte de los diferentes equipos de desarrollo, quienes estaban acostumbrados a no seguir un estricto control de versiones, modificar con total libertad la estructura de las bases de datos o su contenido y no se preocupaban por el impacto que pudieran tener sus cambios en otros módulos del sistema.

Para forzar la aplicación de las medidas antes descritas se tomaron las siguientes decisiones:

1. Sólo los líderes de cada módulo podían pedir que se agregaran cambios al repositorio ece-integrado.
2. Antes de solicitar la integración del código, los líderes deberían asegurarse de que la base de datos del ambiente de pruebas tenía los cambios necesarios en la estructura de la base de datos y el contenido de las tablas.
3. La persona responsable de la integración del código era responsable de verificar que el código compilara correctamente y de coordinar la instalación de la nueva versión de la aplicación con el líder del equipo de analistas funcionales.
4. En caso de detectar algún problema en la compilación del código o en la instalación de la aplicación, la persona responsable de la integración del código informaba al lí-

der de desarrollo de los problemas encontrados y revertía los cambios para regresar a una versión estable de la aplicación.

A partir de la implementación de este procedimiento se cerró lo posibilidad de subir código en cualquiera de los repositorios de los sub-proyectos.

5. Aprobación de usuarios

Una vez que se terminó la integración del código, se reanudaron las tareas de verificación de la aplicación con los usuarios finales.

Durante estas tareas nos encontramos con una situación no prevista. Los usuarios no aceptaban la funcionalidad soportada por la aplicación, aunque estuviera apegada a las especificaciones del sistema, aduciendo dos razones:

1. Las especificaciones de la aplicación no fueron aprobadas por los usuarios finales, sino únicamente por áreas internas de la Dirección responsable del desarrollo de la aplicación.
2. Cuando existían, los requerimientos de los usuarios de la aplicación no habían sido actualizados desde 2009, razón por la cual muchos requerimientos no correspondían con las necesidades actuales de los usuarios.

Para resolver esta situación se decidió abandonar el enfoque tradicional, o de cascada, del proceso de desarrollo de software e implementar un enfoque ágil. Para esto se acordó reducir significativamente los requerimientos de documentación de la aplicación. Por ejemplo, no se actualizaron los casos de uso o los documentos formales de reglas del negocio, no se crearon diagramas UML de los cambios necesarios, no se siguió la ruta normal de autorización de cambios y se permitió que el equipo de desarrollo tuviera sesiones de trabajo con los usuarios finales, cosa completamente prohibida hasta este momento. El soporte documental de los acuerdos tomados con este proceso consistía fundamentalmente de minutas aprobadas y firmadas por las partes involucradas.

Como resultado de las reuniones con los usuarios finales se elaboraron documentos donde se reportaban los problemas de la aplicación, que eran denominados hallazgos en la terminología del proyecto.

Estos documentos fueron analizados por la Dirección de Tecnología y por la Dirección de Prestaciones Médicas para establecer un mínimo aceptable de cambios para ambas partes. Estos cambios deberían estar implementados en un periodo no mayor a 60 días.

Para verificar el cumplimiento de estos acuerdos se definió un grupo de pruebas de sistema que consideraban diferentes escenarios, tipos de paciente y flujos de información entre diferentes módulos de la aplicación. Como la detección de un problema de salud en los servicios médicos de primer nivel. El seguimiento de este problema en unidades de segundo nivel, la hospitalización y cirugía en unidades de tercer nivel de atención; y finalmente la contra-referencia a la unidad de medicina familiar que detectó el problema. A estos casos de prueba se les llamó “pruebas trazadoras”.

Para poder hacer pruebas preliminares a la verificación de las “pruebas trazadoras” fue necesario crear scripts de base de datos que pudieran regresar el estado de la base de datos a puntos intermedios de cada caso.

Después de algunas iteraciones, la dirección de prestaciones médicas aceptó la funcionalidad desarrollada por la Universidad. Este evento fue de gran importancia, ya que hasta ese momento no se tenía un sistema aprobado por los usuarios finales, situación que ponía en riesgo un esfuerzo de más de 4 años y una inversión considerable de recursos.

El 4 de Julio de 2012 se presentó la aplicación a la Dirección General de la Institución con el visto bueno de la Dirección de Prestaciones Médicas. En esta presentación se consiguió la autorización para utilizar la nueva aplicación en una prueba piloto en un número limitado de Unidades Médicas en diferentes estados de la republica.

A cambio de esta aprobación, la Dirección de Prestaciones Médicas solicitó que algunos cambios y mejoras a la aplicación, no considerados parte del requerimiento inicial, fueran implementados en el corto plazo, y estos cambios deberían estar verificados y en producción antes de dar por terminada la etapa de pruebas piloto.

6. Monitoreo y estabilización

Al momento de su liberación, el ECE consistía de más de 4,000 clases Java y cerca de un millón de líneas de código. Al equipo a mi cargo se le encomendó la tarea de monitorear el desempeño de las pruebas piloto, atender cualquier contingencia que se presentara y coordinar con los líderes de cada módulo los esfuerzos para corregir dichos problemas.

Los insumos para esta tarea eran los siguientes:

- Reportes de la mesa de ayuda.
- Bitácoras de la aplicación.
- Reportes de monitoreo de la base de datos.
- Reportes de rendimiento del sistema operativo.

Durante este periodo se analizaron y corrigieron muchos problemas de la aplicación, principalmente derivados de la falta de afinación en la base de datos y algunos problemas derivados directamente del código.

Los reportes que recibía la mesa de ayuda tenían que ver en general con tres tipos de problemas:

1. Lentitud generalizada del sistema.
2. El sistema no respondía o mostraba pantallas en blanco.
3. El sistema no guardaba información sin mostrar mensajes de error.

6.1. Plataforma

Analizando las posibles razones de la lentitud del sistema encontramos que el ECE fue desplegado en dos nodos de Weblogic 10.3 con 2GB de RAM, cada uno sobre Solaris de 32bits. Esta decisión limita severamente la cantidad de recursos que se le puede asignar a Weblogic y, en consecuencia, el desempeño del ECE. Por otro lado, pese a que cada nodo tiene 8Gb RAM de memoria física, gran parte de esta memoria no es utilizada, ya que la versión Solaris de 32Bits no puede acceder a más de 4Gb de memoria RAM.

La aplicación ECE hace uso intenso de la memoria asignada al contenedor JEE. Incluso no es posible ejecutarla si no se le asigna un mínimo de 512Kb de memoria RAM a la PERMGEN al momento de iniciar Weblogic.

Por lo anterior, se recomendó actualizar la plataforma a Solaris de 64bits y al mismo tiempo actualizar la versión de Weblogic a 10.3.5. De esta manera sería posible asignar al menos 6GB de RAM a cada nodo Weblogic.

Nuestra recomendación fue recibida con muchas reservas por parte del equipo de operaciones de sistemas. Desde su perspectiva, esta propuesta no era más que un intento por

resolver con grandes recursos de hardware el uso ineficiente de los recursos asignados a la aplicación. Este argumento tiene mérito. Como se analizará en las secciones subsecuentes, la aplicación tiene grandes áreas de oportunidad. Sin embargo, aun resolviendo dichos problemas, un sistema operativo de 32bits limita severamente la cantidad de memoria RAM que se puede asignar a una aplicación y no es el entorno óptimo para una aplicación en la que se anticipa un gran número de usuarios concurrentes.

Esta recomendación se puede implementar sin tener que modificar la aplicación en ninguna forma y se considera que es un paso importante en el camino para resolver los problemas de lentitud reportados en la mesa de ayuda.

Otra área de oportunidad es que el ECE se ejecuta en una configuración estándar de Weblogic, sin afinación alguna. Para resolver esta situación se propuso hacer un ajuste fino de los parámetros de configuración de Weblogic, incluyendo número de conexiones, replicación y balanceo de servicios. En particular se recomendó la creación de *work managers* adicionales ya que toda la funcionalidad del ECE era soportada por un solo *work manager*, lo que era causa de mucha contención en el servidor de aplicaciones.

6.2. Arquitectura

La arquitectura propuesta por la institución requiere el uso de un enfoque orientado al uso de servicios y una estructura de al menos 3 capas: presentación, lógica de negocio y persistencia de datos. Pero no existe ningún mecanismo en la estructura de la aplicación que obligue a los programadores a respetar estos lineamientos. De hecho es una práctica común usar objetos hibernate en la capa de presentación, con lo que se contraviene la premisa de crear una aplicación con capas separadas y se causan problemas de desempeño que serán analizados en la sección de persistencia.

El ECE se distribuye en un solo archivo WAR que incluye todos los módulos del sistema. Este diseño monolítico impide la distribución y replicación de componentes de la aplicación en distintos servidores de acuerdo a la carga que se presente en cada módulo o componente de la aplicación.

Por ejemplo, analizando las bitácoras de acceso a la aplicación observamos que más del 85 % de los accesos al sistema se concentraban en los módulos Agenda de Citas y Nota Médica. Una posible solución para balancear la carga es separar los componentes más usados y utilizarlos en nodos especializados, cosa que no es posible con la forma como se construye el ECE.

Para corregir esta situación se propuso modificar el proyecto *maven*[11] con el que se compila la aplicación para que en lugar de que se produjera un solo archivo WAR se separara la aplicación en cuatro aplicaciones:

[11] Herramienta para la construcción de proyectos Java.

1. Configuración de la aplicación y administración de recursos.
2. Agenda de citas y consulta externa.
3. Hospitalización
4. Medicina Familiar.

Al mismo tiempo, se re-organizarían los componentes para ocultar los detalles de cada capa del sistema y forzar que la comunicación entre las capas se hiciera usando objetos de transporte de datos (DTO por las siglas en inglés) en lugar de referir directamente a los objetos hibernate. Los servicios comunes a las cuatro aplicaciones serían desplegados en la aplicación número 2.

La estructura monolítica de la aplicación obliga que a cada programador se le dé acceso a la totalidad del código, por lo que éstos pueden modificar código en otros módulos sin tener que acordar los cambios o informar al resto del equipo. Al implementar la estructura propuesta los programadores solo requieren acceso al código del módulo en el que están trabajando. Este acceso restringido al código obligaría a los programadores a enfocarse en el módulo que les fue asignado. Cuando requieran que se modifique o extienda la funcionalidad en otro módulo deberán requerir el cambio al responsable ya que no podrán realizarlo directamente.

Esta propuesta tiene un beneficio secundario que es el restringir el acceso al código, lo que disminuye significativamente el riesgo de que el código pueda ser copiado ilícitamente. También se recomendó hacer cambios al proceso de desarrollo. Debido a las presiones de tiempo se le dio total libertad a cada líder de módulo para que implementara los componentes necesarios para entregar la funcionalidad requerida.

Esto causó que se desarrollaran muchos componentes duplicados y que no se respetaran los lineamientos marcados por la Institución.

Un ejemplo muy claro de esto son los textitmanaged beans de JSF, que en general son las clases java más grandes del sistema y en algunos casos excedieron por mucho los límites establecidos por la institución.

Los lineamientos de desarrollo establecen que las clases java no deben ser de más de 1,500 líneas. Pese a lo anterior, en el proyecto 155 clases no cumplen con este requerimiento. El siguiente cuadro muestra las clases Java más grandes.

La propuesta de mejora consistía en fragmentar los objetos antes mencionados en objetos más manejables, implementando las mejores prácticas de JSF que proponen el uso de 5 tipos de Beans (Model Managed-Bean, Backing Managed-Bean, Controller Managed-Bean, Support Managed-Bean, Utility Managed-Bean) encargados de tareas específicas y reusables en lugar de un solo gran Bean como es el caso en los ejemplos anteriores.

Además de los managed beans, hay algunas clases que llaman la atención por su tamaño, como AtencionServiceImpl.java y HojaFrontalUrgencias.java con 4760 y 3841 lineas de código respectivamente. No es común que clases distintas a los managed beans tengan esta

Cuadro 1: Clases Java con el mayor número de líneas de código

Líneas de Código	Clase Java
11,275	IntervencionQuirurgicaBean.java
7,862	CitaBean.java
6,913	RegHojaFrontalUrgenciasBean.java
6,804	RegistraSignosVitalesCEBean.java
6,418	RegTratamientosYDietasBean.java
6,142	EdiHojaFrontalUrgenciasBean.java
6,013	RegistraNotaMedicaBean.java
5,936	ConsultaDiabetesMellitusBean.java
5,904	RegNotaMedUrgBean.java
5,634	RegAnteVigilanciaPrenatalBean.java

cantidad de líneas de código por lo que vale la pena analizar estas clases a detalle.

Para resolver el problema de los managed bean, se propuso hacer una prueba de concepto, eligiendo un managed bean donde, además de tener un número muy grande de líneas de código, se tuvieran reportes frecuentes de errores de la aplicación o de problemas con los tiempos de respuesta. Este componente sería re-construido apegándose a las mejores prácticas de JSF. Una vez terminado el re-diseño de este componente, lo usaríamos como referencia para la construcción de nuevos módulos y poco a poco migrar los componentes existentes a este modelo. Con este ejercicio es posible construir una biblioteca de beans que realicen tareas comunes y puedan ser re-utilizados en diferentes módulos.

En proyectos de esta magnitud es común que el código sea revisado por pares. Esta práctica aumenta la calidad del código producido ya que permite detectar errores y malas prácticas de programación, fomenta el uso de estándares, promueve el seguimiento de los lineamientos y reduce la dependencia en los autores del código.

La propuesta era que, como parte del proceso de aceptación del código, un grupo de expertos, por ejemplo personas elegidas entre arquitectos y líderes de módulo, analizara junto con los autores cada módulo, se generaran recomendaciones y que no se considerara que el módulo estaba terminado hasta haber atendido todas las recomendaciones. Este proceso permitiría corregir malas prácticas de programación e ir creando un manual para desarrollar componentes en este proyecto.

La aplicación gradual de esta recomendación permitiría que se detectara funcionalidad compartida por varios componentes, construir bibliotecas de componentes reutilizables e ir detectando y eliminando malas prácticas de programación.

Se creó el sub-proyecto ece-componentes como prueba de concepto, en el que se implementaron las propuestas de modificaciones a la arquitectura monolítica, para dividir la aplicación en cuatro aplicaciones más pequeñas. En esta prueba de concepto se aisló en una aplicación separada la funcionalidad de agenda de citas. Quedó pendiente la valida-

ción de esta propuesta por parte del equipo de Interoperabilidad de la institución.

6.3. Capa de persistencia

El análisis del comportamiento de la aplicación y de la base de datos mostraron que una causa muy importante de los problemas con los tiempos de respuesta era el número de consultas generadas por la aplicación y el tiempo que le tomaba a la base de datos ejecutar dichas consultas.

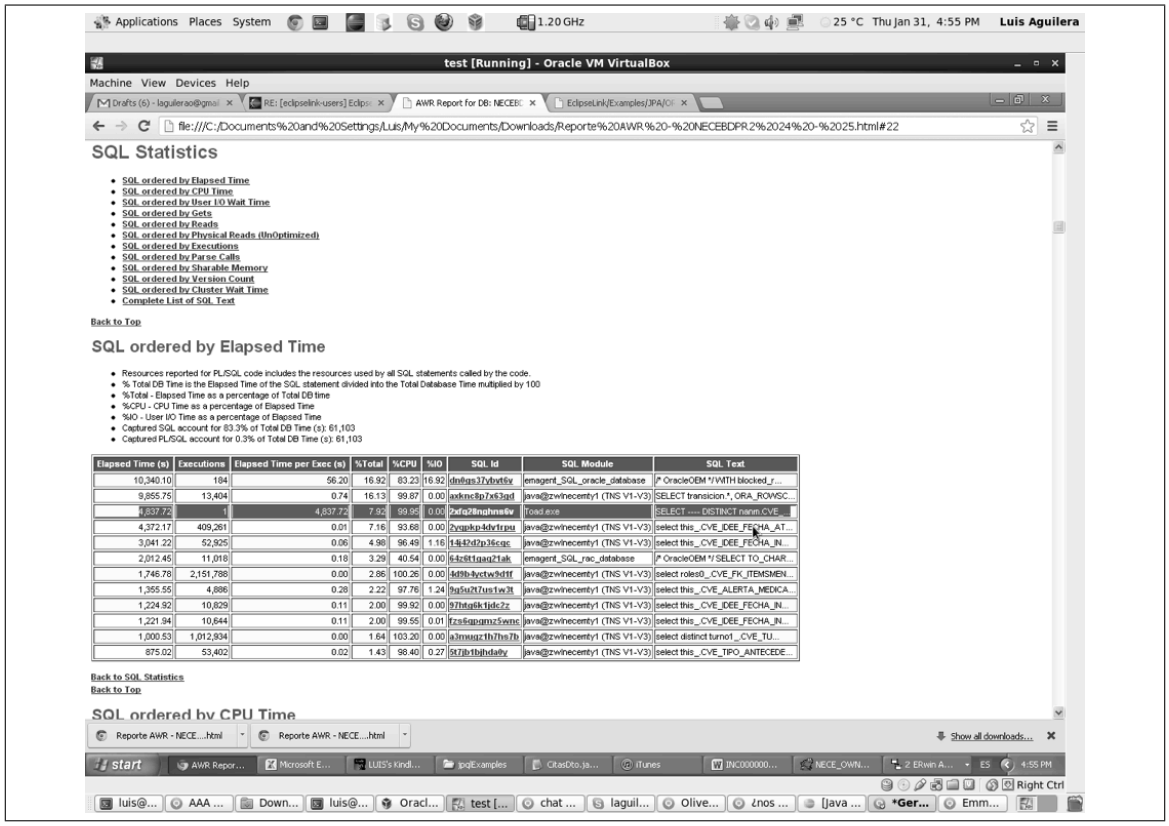

Figura 6: Estadísticas del servidor de base de datos.

El problema más simple de resolver era que muchas consultas se resolvían por medio de operaciones full-table scan debido a la falta de índices. En estos casos bastaba con identificar la consulta con este tipo de problema y generar índices que evitaran que el manejador de la base de datos usara full-table scan. Para esto se solicitaban reportes del monitoreo a la base de datos donde se identificaba cuales eran las consultas mas frecuentes o las que tomaba más tiempo resolver. La figura 6 muestra uno de estos reportes, en él podemos apreciar que algunas consultas que se repiten con exagerada frecuencia; por ejemplo tenemos una consulta que se realizó más de 2 millones de veces y otra que se realizó más de un millón de veces.

Esto es causado porque el sistema busca en la base de datos los catálogos que necesita cada vez que muestra una pantalla. Con estas consultas carga en memoria por lo menos las descripciones y claves a mostrar; en otros casos también se cargan valores para validaciones y mensajes de error. Esto en sí representa una carga innecesaria para la base de datos, ya que los registros en los catálogos son estáticos y con la excepción de los catálogos de enfermedades, el de padecimientos y el de medicamentos, el resto de los catálogos son muy pequeños en cuanto al número de elementos.

En algunos módulos esta situación es causa de lentitud en el despliegue de las pantallas, ya que se realizan múltiples accesos a la base de datos, como en el caso de la pantalla de captura de signos vitales, que realiza 42 consultas a base de datos cada vez que muestra la pantalla. La recomendación para resolver esta situación era crear un caché en memoria donde se cargarían los datos de los catálogos más comunes y desarrollar componentes que reemplacen las consultas a la base de datos con consultas a este caché.

En estos componentes se cargaría la información una sola vez al iniciar el sistema en lugar de hacerlo cada vez que se muestra la pantalla, logrando dos beneficios: 1) Menor uso de memoria ya que existiría una sola instancia del catálogo en memoria, a diferencia de la situación actual donde se carga en memoria algunos catálogos cada vez que se muestra una pantalla. 2) Reducir drásticamente el número de accesos a la base de datos, logrando con esto un flujo más rápido de la información y eliminando tráfico innecesario hacia la base de datos.

Otro problema común es el uso de la funcionalidad *criteria* de hibernate para la recuperación de información. Esta funcionalidad permite que el programador haga una consulta a la base de datos sin especificar la sentencia SQL. En su lugar, hibernate usa las clases HBM y las relaciones definidas entre ellas para crear la sentencia SQL. El listado 10 muestra código de una consulta usando *criteria* de hibernate.

Listado 10: Uso de criteria de hibernate

```
@Override
public Admision consultarUltimaAdmision(String idee) {
    Criteria criterio =
        getSession().createCriteria(Admision.class, "admision");
    if (idee != null) {
        criterio.add(Restrictions.eq("admision.paciente.idee", idee));
     }
    criterio.createAlias("admision.ingreso", "ingreso");
    criterio.addOrder(Order.desc("ingreso.fechaIngreso"));
         criterio.setFirstResult(0);
         criterio.setMaxResults(1);
     return (Admision) criterio.uniqueResult();
}
```

Esta consulta recupera todos los registros de las tablas admisión, ingreso y paciente donde la columna paciente_idee es igual a la cadena idee, los ordena descendentemente por

fecha de ingreso y escoge el primer registro.

El ejemplo muestra dos problemas de criteria:

1. El programador no tiene control sobre la sentencia SQL que será usada para recuperar la información. La clase Admisión no incluye al atributo paciente_idee y en su lugar tiene una relación con la clase Paciente. Tampoco tiene la fecha de ingreso, dato que está definido en la Clase Ingreso, con la que también tiene definida una relación. Hibernate usa estas relaciones para determinar cómo acceder a la información.
2. Criteria solo proporciona métodos para especificar restricciones, no tiene métodos para especificar qué columnas se quieren recuperar de cada tabla. En este ejemplo, se recuperan todas las columnas de las tablas admisión, ingreso y paciente, aunque sólo se necesitan dos columas: paciente_idee y fecha de ingreso.

La mayoría de los módulos usan criteria para consultar la base de datos, por lo que recuperan registros o colecciones de registros que incluyen los join de varias tablas, cuando sólo se van a utilizar algunos datos. En algunos casos extremos, las consultas generadas automáticamente por criteria recuperan más de 300 columnas haciendo joins de más de 25 tablas, lo que afecta al rendimiento del sistema al crear un tráfico innecesario hacia la base de datos y utilizar, en cada sesión, más memoria que la que realmente es necesaria para cumplir con los requerimientos.

Otro problema derivado del uso de criteria es que un programador con poca experiencia puede olvidar poner validaciones del tipo de parámetros recibidos para hacer las consultas, lo que en muchas ocasiones resultó en que se mandaron consultas a la base de datos sin restricción alguna, lo que ocasionaba que el servicio regresaba todos los registros de un join arbitario de tablas, información que era completamente inútil para la aplicación, pero que causaba una gran carga de trabajo en la base de datos, tráfico innecesario en la red y un gran uso de recursos en el servidor de aplicaciones. Este tipo de problemas no es atribuible a hibernate, aunque, en mi opinión, la opacidad derivada del uso de criteria, aunada a la falta de supervisión del código producido, contribuyen a este tipo de problemas. El listado 11 muestra un ejemplo de esta situación.

Listado 11: Uso de criteria de hibernate (continúa ...)

```
@SuppressWarnings("unchecked")
public List<CitaMedica> consultar(HorarioTurno horarioTurno,
Date fechaInicio, Date fechaFin, String... codigoEstadoCita) {

DetachedCriteria cri = DetachedCriteria.forClass(CitaMedica.class);
cri.createAlias("horarioTurnoVigente", "ht");
cri.add(Restrictions.eq("ht.id", horarioTurno.getId()));
if (codigoEstadoCita != null && codigoEstadoCita.length > 0) {
cri.add(Restrictions.in("estadoCita.codigo", codigoEstadoCita));
}
```

Listado 11: Uso de criteria de hibernate (continúa ...)

```
if (fechaInicio != null) {
cri.add(Restrictions.ge("fechaInicio", fechaInicio));
}
if (fechaFin != null) {
cri.add(Restrictions.le("fechaInicio", fechaFin));
}
cri.addOrder(Order.asc("fechaInicio"));
return this.getHibernateTemplate().findByCriteria(cri);
}
```

En este ejemplo, si se llama al método consultar con todos los parámetros nulos la única restricción que se agregará a la consulta es que el atributo horarioTurno.Id sea igual a ht.id, sin importar unidad médica o fechas, información que no tiene ningún sentido.

La recomendación para solucionar este tipo de problemas es evitar, en la medida de lo posible ,el uso de *criteria* y usar en su lugar *named queries* para recuperar sólo la información necesaria.

En hibernate se le llama *named queries* a expresiones SQL que son definidas en archivos de configuración a las que se les asigna un nombre. Estas consultas pueden ser usadas después por los objetos hibernate para interactuar con la base de datos. El listado 12 muestra la definición de un *named query*.

Listado 12: Uso de criteria de hibernate

```
<sql-query name="persons">
<return alias="person" class="eg.Person"/>
     SELECT person.NAME AS {person.name},
            person.AGE AS {person.age},
            person.SEX AS {person.sex}
     FROM PERSON person
WHERE person.NAME LIKE :namePattern
</sql-query>
```

Este ejemplo define un named query llamado persons, que recupera los atributos NAME, AGE y SEX de la tabla person cuando el patrón del atributo NAME es similar a namePatern y asocia la respuesta de la consulta a la clase eg.Person.

La tarea propuesta no es menor, pues implica cambiar la forma de acceso a la base de datos en la mayoría de los módulos, crear clases DTO específicas para cada consulta y verificar el impacto en las clases que usan dichas consultas. Por ello se propone hacer una migración gradual, iniciando por las consultas que generan más contención en la base de datos.

6.4. Capa de presentación

En la sección de arquitectura ya se discutió el problema del tamaño de los *managed beans* y la falta de cumplimiento con los lineamientos de desarrollo dictados por la institución.

Esta capa de la aplicación presenta otros dos problemas importantes. JSF permite definir el alcance de la vida de cada uno de los *managed-beans*. Este atributo es muy importante ya que permite definir de una manera muy fina el ciclo de vida de los *managed beans*. El siguiente cuadro muestra las posibles opciones.

Cuadro 2: Alcance de vida de los managed beans de JSF

Tipo de Alcance	Descripción
none	Cuando se define este tipo de alcance, el bean no es instanciado a menos que otro bean lo invoque. Son creados bajo demanda. Cada instancia se mantendrá activa mientras el bean que lo creó exista.
request	Se crea una instancia por cada solicitud http. Se mantienen activos hasta que la respuesta es enviada.
view	Estos beans se mantienen activos mientras el usuario se mantenga en la misma vista. Son destruidos cuando el usuario navega hacia otra vista.
session	Estos beans se construyen cuando una nueva sesión es creada y se mantienen hasta que la sesión es invalidada, ya sea por log-out explícito o time out.
application	Este tipo de beans están disponibles para todos los usuarios mientras la aplicación está activa.

La mayoría de los *managed beans* de la aplicación están configurados con un alcance *session* lo que significa que los objetos se mantienen activos aunque ya no sean usados porque el usuario navegó hacia otras partes del sistema. Esto resulta en un gasto innecesario de memoria y en la necesidad de limpiar los objetos para evitar trabajar con información desactualizada cuando el usuario “regresa” a una pantalla visitada con anterioridad.

La recomendación para resolver este problema es cambiar la configuración de los beans, usando el alcance más limitado que pueda soportar la funcionalidad requerida, en general optando por alcances tipo *view* o incluso tipo *request*.

El otro problema ya ha sido comentado brevemente. En muchos módulos, la capa de presentación usa directamente objetos hibernate, con lo que se invalida la arquitectura de

tres capas y se hace un uso excesivo de memoria ya que, como se comentó con anterioridad, estos objetos contienen todas las columnas de todas las tablas involucradas en cada consulta. La forma de corregir este problema es evitar el uso de objetos hibernate en la capa de presentación, usando en su lugar objetos DTO definidos específicamente para cada consulta y crear los servicios correspondientes

Implementar estas dos recomendaciones implica una revisión mayor a la lógica de la aplicación, por lo que nuestra propuesta consideraba hacer este tipo de cambios a la configuración junto con la revisión de la arquitectura y la descomposición de los actuales *backing beans* en beans más especializados de acuerdo a las mejores prácticas de JSF.

6.5. Interoperabilidad

La versión en producción del ECE intercambia información con 5 sistemas externos: Verificación de derechos, Farmacia, Resultados de Laboratorio, Incapacidades y Cita Telefónica.

La implementación de la funcionalidad que soporta estas interacciones no está estandarizada:

- Verificación de derechos y Farmacia se acceden por medio del canal de intercambio de información estándar de la institución (ESB).
- Resultados de laboratorio usa un web service que opera independiente al ESB.
- Los datos para incapacidades se envían usando FTP calendarizados como tareas Quartz.
- Los datos del servicio FTP se recuperan de la tabla de parámetros de la base de datos. Para exponer los servicios de cita telefónica se implementó un EJB en una aplicación independiente.

Adicionalmente hay módulos en desarrollo y requerimientos nuevos, por lo que parece relevante establecer un mecanismo estándar para la publicación y consumo de los web services. Los servicios a los que se accede por medio del ESB presentan otro problema. El framework que desarrolló la institución se conecta al ESB de forma síncrona, lo que significa que el thread que invoca un servicio ESB queda detenido hasta que recibe la respuesta o se excede el tiempo de espera configurado (10 minutos para la aplicación en producción).

La consulta al servicio de verificación de derechos es particularmente problemática. Cuando el ESB recibe una consulta de este tipo rutea el mensaje a la unidad médica correspondiente. Si por alguna razón el sistema de verificación de derechos no está disponible, el ESB no contesta con un mensaje de error, sino que espera hasta que excede el tiempo de tolerancia.

Como el ECE tiene un comportamiento similar, cuando en una unidad médica el servicio de verificación de derechos deja de responder, las solicitudes se empiezan a encolar en el Weblogic, cada solicitud conteniendo un thread de ejecución, hasta que Weblogic se queda sin threads disponibles y el servidor de aplicaciones eventualmente se colapsa.

Se propuso atender esta problemática con tres acciones concretas:

1. Modificar la configuración de los componentes que consumen servicios del ESB para que reaccionaran más ágilmente ante la falta de respuesta. Esto implicaba que, además de las modificaciones a la configuración de Weblogic y la creación de *work managers* específicos para acceder a ESB, era necesario revisar los casos de uso con los analistas funcionales, el equipo de Interoperabilidad de la institución y, de ser necesario, los usuarios finales para definir con claridad la conducta esperada del ECE ante la falta de respuesta de servicios externos.
2. Estandarizar el acceso a los servicios externos por medio del ESB, para lo que sería necesario coordinar con interoperabilidad las modificaciones necesarias tanto al ECE como al ESB, para incorporar las consultas a los servicios de Incapacidades y Resultados de laboratorio a este canal.
3. El framework proporcionado por la institución carece de un estándar para que el ECE publique servicios para su consumo por otras aplicaciones. Nuestra recomendación fue exponer estos servicios por medio del ESB; analizar las alternativas de tecnologías para la publicación de webservices (EJB's, Axis y Spring) para definir el estándar del ECE. Una vez establecido este estándar, modificar las implementaciones existentes para apegarse al nuevo estándar.

Para demostrar los conceptos de esta recomendación se creó el sub-proyecto ece-ws, donde se cambió la consulta síncrona al ESB por consultas asíncronas y se configuraron *work manager* específicos para manejar los accesos a servicios externos. Quedó pendiente hacer las pruebas de sistema y validar con el equipo de Interoperabilidad los cambios sugeridos a la configuración de Weblogic para proceder a su liberación.

6.6. Escalabilidad

El ECE es un sistema que está pensado para soportar la actividad en línea de más de 1,200 Unidades médicas, meta que parece difícil de alcanzar teniendo en cuenta los problemas analizados en las secciones anteriores. La corrección de estos problemas reducirá de forma notable los requerimientos de memoria y tiempos de respuesta de la aplicación, lo que sin duda resultará en una mejor experiencia de usuario y permitirá soportar un mayor número de usuarios concurrentes en cada nodo.

Sin embargo, esas medidas no resuelven completamente los problemas de escalabilidad. Un recurso muy usado en aplicaciones JEE para atender a grandes números de usuarios

es distribuir la carga de trabajo entre diferentes computadoras agrupadas en *clusters*. La idea es que estas computadoras compartan copias de las sesiones que están siendo ejecutadas y de los objetos que le dan servicio a las sesiones. De esta manera, cuando una nueva petición de servicio es recibida, el cluster asigna la nueva petición a la computadora con menos trabajo. Estas copias también se usan para implementar la alta disponibilidad. Si un nodo del *cluster* deja de responder, existen otras computadoras dentro del *cluster* que tienen copias de las sesiones que estaban activas en el nodo que falló y pueden reestablecer el servicio a dichas sesiones.

Para que estas dos características funcionen óptimamente se requiere que las sesiones sean lo más pequeño posible en términos de la memoria usada y que todos los datos contenidos en la sesión sean *serializables*. Estos dos requisitos son indispensables para mover entre computadoras los datos de las sesiones con facilidad.

Algunos módulos de la aplicación no cumplen con estos requisitos. Por ejemplo en el módulo de agenda se cargan en sesión 43 Beans JSF y 145 Objetos hibernate, lo que ocupa aproximadamente 55Mb de memoria por cada usuario. Este uso excesivo de recursos limita severamente el número de usuarios concurrentes e impide que se puedan implementar esquemas de alta disponibilidad.

La propuesta para resolver esta situación es modificar radicalmente el guardado de objetos en la sesión y limitar los objetos guardados a los que sean realmente indispensables. También se recomendó el uso de objetos proxy en lugar de instancias de objetos para reducir el acoplamiento entre componentes y reducir lo más posible el consumo de memoria. Estos cambios se pueden hacer al mismo tiempo que se re-diseñan los *managed beans* de JSF y se elimina el uso de objetos hibernate en las capas de servicios y de presentación.

6.7. Mantenimiento de la infraestructura.

Además del mantenimiento a la aplicación derivado de nuevos requerimientos o la detección de errores, el equipo de desarrollo debía considerar que las versiones de las herramientas Java se actualizan con mucha rapidez. Algunas de estas actualizaciones no son compatibles con las versiones anteriores, por lo que es necesario revisar constantemente el impacto que estas actualizaciones puede causar en la aplicación y el esfuerzo necesario para modificar la aplicación para adaptarse a los cambios.

Los componentes y entornos de trabajo que se eligieron para el desarrollo de la aplicación fueron especificados en 2009. Desde entonces se han liberado nuevas versiones de estos componentes con correcciones de errores y funcionalidad nueva o modificada. Esta situación hace que sea necesario actualizar los *frameworks* y modificar el código de la aplicación en consecuencia.

Un ejemplo de esto es que los requerimientos de la aplicación especifican que el códi-

go de las pantallas debe ser 100 % compatible con Internet Explorer 7 (IE7). A finales de 2012 se empezaron a recibiir reportes de errores porque la aplicación no se desplegaba correctamente al ser accedida usando un browser distinto al Internet Explorer 7.

Si bien los reportes de error no se aceptan porque el estándar definido en la especificación técnica del ECE es IE7, es necesario analizar esta situación, ya que cada vez será más frecuente que la versión 7 del Internet Explorer sea obsoleta y no está incluida en las configuraciones de sistema operativo de equipos nuevos.

Esta situación se presenta porque las diferentes versiones de Internet Explorer no son compatibles entre si, en, particular IE7 es conocido por tener un gran número de bugs en el manejo de CSS. Otros fabricantes de browsers tienen problemas similares, aunque el número de problemas e impacto en la funcionalidad de las páginas desplegadas en ellos es mucho menor.

La figura num 7 muestra los diferentes tipos de contenidos CSS y la compatibilidad entre diferentes browsers y las versiones de éstos.

Figura 7: Comparativo del cumplimiento de estándares de W3C.

Esta tabla muestra con mucha claridad que IE7 tiene un nivel muy bajo de cumplimiento de los estándares de W3C y no es compatible con ninguno de los browsers dominantes en el mercado en las versiones actuales.

Las posibles soluciones para resolver esta situación son:

1. Crear versiones de contenido CSS para cada browser que se desea soportar. Esta solución es muy común. Requiere el uso de etiquetas condicionales para activar contenidos dependiendo del browser que está desplegando las páginas. La principal

desventaja de esta alternativa es que aumenta de forma considerable el trabajo de desarrollo para la capa de presentación y el número de casos de prueba que se deben ejecutar para aprobar versiones.

2. Usar los browsers en modo de compatibilidad. Esta opción reduce notoriamente los problemas de despliegue de contenidos, pero no los elimina completamente. Tiene la ventaja que no requiere modificaciones mayores al código.
3. Usar un estándar diferente al especificado. Si bien no hay browsers que cumplan al 100 % con las especificaciones del W3C, Firefox (FF) es más cercano al estándar y tiene mayor compatibilidad entre diferentes versiones.
4. Actualizar las pantallas para que se muestren correctamente en IE9, que es el browser que traen instalado de fábrica los equipos actuales,o bien, actualizar a IE10, que es el browser que traen instalado los equipos nuevos.

Para tener un buen dimensionamiento del problema la recomendación es:

1. Revisar el ECE en los browsers propuestos, IE9, IE10 y FF. Los problemas que se anticipan son problemas al mostrar algunos contenidos y fallas en la ejecución de javascript. Por lo que se tendrían que probar todos los casos de uso.
2. En función del número de errores detectados escoger un nuevo estándar y elaborar un plan de trabajo para la migración.

Otra forma de abordar el problema es asumir que Firefox va a seguir siendo un browser más compatible con W3C y con futuras versiones y, en consecuencia, adoptarlo como estándar. Esta opción requiere mucho menos trabajo de los testers; pero de todos modos requiere que se verifique que las adecuaciones son correctas. Una revisión preliminar y los comentarios del equipo sugieren que esta opción es la que se podría implementar más rápidamente. Los cambios propuestos afectan exclusivamente a la capa de presentación ya que el resto de la aplicación no será modificado.

Se anticipa que se presentarán situaciones similares al tener que actualizar las versiones de Weblogic, ya que el fin de soporte por parte de Oracle está anunciado para 2014. Lo más probable es que para ejecutar la aplicación en un contenedor JEE más moderno se tengan que actualizar las versiones de Hibernate, Spring, JSF y Richfaces que están especificadas en el *framework* de la institución.

7. Recomendaciones

Algunas de las decisiones que se tomaron durante el desarrollo del proyecto no se apegan a las prácticas comunes en la institución. Los buenos resultados que se obtuvieron con estas prácticas alternativas hacen pensar que sería muy conveniente considerar integrarlas a los lineamientos vigentes.

1. Uso de metodologías ágiles para la definición de requerimientos y desarrollo de componentes. Durante las pruebas de aceptación de usuario se cambiaron algunas prácticas de gestión de requerimientos que agilizaron mucho el ciclo de desarrollo y permitieron que los analistas y programadores tuvieran una mejor comprensión de las necesidades de los usuarios. Al mismo tiempo, los usuarios entendieron con mucha claridad que no era posible satisfacer algunos de sus requerimientos en los tiempos establecidos. Con esto se lograron acuerdos que permitieron liberar versiones de algunos módulos y crear planes de trabajo muy detallados para la liberación de versiones subsecuentes.
2. Tener un equipo separado de los equipos de desarrollo que sea responsable de la integración de las aplicaciones, generación de programas ejecutables y monitoreo integral de las pruebas de sistema. El uso de Mercurial como herramienta de control de versiones y el uso de ramas y sub-proyectos en la configuración del repositorio de código fuente del proyecto ayudó a mejorar significativamente la calidad de los entregables. El estricto control de versiones que se implementó permitió identificar y corregir errores en código en producción mientras que se trabajaba en nuevas versiones de la aplicación.
3. Integrar a los lineamientos de la institución las recomendaciones técnicas de este documento para garantizar que los problemas aquí descritos no se vuelvan a presentar.
4. Formar un grupo responsable de la arquitectura de la aplicación. Este grupo sería responsable del diseño de la estructura de componentes; de la creación y mantenimiento de bibliotecas de componentes y de verificar que el código cumpla con los lineamientos establecidos.
5. El uso de herramientas para monitorear el comportamiento de los servidores, aplicaciones, bitácoras y tráfico en la red que permita la identificación y corrección temprana de errores en el código, falta de afinación de la base de datos o de los parámetros del servidor de aplicaciones.
6. Incluir pruebas unitarias automáticas en los scripts para construir los ejecutables de la aplicación. También es importante hacer pruebas de carga rutinariamente.

El problema de seguir los estándares para el desarrollo de aplicaciones definidos por la institución es que están muy orientados a un proceso de desarrollo de software en cascada, donde el ciclo de vida del proyecto está dividido en etapas y normalmente transcurre

mucho tiempo desde que se recaban los requerimientos hasta que se presentan a los usuarios las primeras versiones de la aplicación. Es común que en este lapso de tiempo las prioridades, o los requerimientos, hayan cambiado y el producto entregado no tenga relevancia.

Desde la perspectiva de la arquitectura hay una desconexión entre el área que define los estándares, la que desarrolla las aplicaciones y la que despliega la aplicación en producción. Esta falta de comunicación fluida causa que no se tenga retroalimentación oportuna de las diferentes áreas y, en consecuencia, los estándares no evolucionan. De esta forma se pierde la oportunidad de enriquecer los estándares con la experiencia acumulada por los grupos de desarrollo o despliegue.

Por último, las actividades de monitoreo de la aplicación durante la prueba piloto nos permitieron recopilar muchos datos sobre el comportamiento de la aplicación en condiciones de uso normales. Trazando algunos errores a partir de los reportes de usuario, las caídas de sistema o los mensajes en las bitácoras de la aplicación nos proporcionó información suficiente para validar o, en su caso, cambiar algunas de las decisiones de diseño tomadas durante el desarrollo del proyecto. Durante estas intervenciones desarrollamos metodologías y herramientas que nos permitían diagnosticar con rapidez los problemas y encontrar alternativas de solución. Nuestra recomendación es que se formalice la existencia de este grupo, para que se tenga un monitoreo continuo de la aplicación, para garantizar que opera dentro de los parámetros establecidos por un lado y, por otro, tener una detección temprana de errores y evitar, dentro de lo posible, fallas del sistema.

8. Propuesta de implementación

Los cambios propuestos en este documento no son menores. De implementarse todas las recomendaciones, al terminar se tendrían cuatro aplicaciones independientes que serían muy diferentes a la aplicación actual. Los beneficios de implementar las recomendaciones serían principalmente:

- Aplicaciones serían más eficientes, derivado del uso más racional de los recursos
- más estables, ya que se eliminarían todas las condiciones de error que están reportadas en las bitácoras de la aplicación y que hoy es prácticamente imposible resolver
- se podría tomar ventaja de la funcionalidad de alta disponibilidad con que cuenta el contenedor JEE usado en el proyecto.
- más fáciles de mantener, ya que se tendría una biblioteca de componentes bien documentada y el código de cada módulo estaría enfocado exclusivamente en los requerimientos de negocio a soportar.

Esta propuesta no se trata de re-escribir la aplicación desde cero. Como ya se comentó, lograr la aprobación de los usuarios fue un logro mayor que requirió buscar alternativas para llenar huecos o actualizar la documentación formal del proyecto; así como para lograr que la información fluyera eficientemente entre los diferentes grupos de trabajo. Por esta razón mucha de la funcionalidad que se desarrolló no está documentada y se fue definiendo y detallando sobre la marcha. Tratar de escribir la aplicación de nuevo implicaría volver a tener que repetir ese proceso.

La propuesta busca enfocarse en los problemas técnicos, no de negocio, que sean más costosos en términos de uso de recursos y causen mayores problemas; con el propósito de resolverlos de tal manera que no se afecte la funcionalidad, sólo la forma como se usan las tecnologías que soportan la aplicación. De hecho, esta propuesta no considera ningún cambio al modelo de datos o a las pantallas.

Para minimizar los riesgos inherentes a cambios de tal magnitud se propone un plan por etapas. Como se mencionó, ya existen avances en la prueba de concepto para separar la funcionalidad de agenda de citas de la aplicación principal. En la prueba piloto se observó que esta parte de la aplicación es por mucho la más utilizada por los usuarios, por lo que un proyecto exitoso resultaría en una carga de trabajo mucho menor para la aplicación actual.

La propuesta de trabajo para la primer etapa es:

1. Aislar la funcionalidad de agenda de citas en una aplicación independiente.
2. Crear servicios básicos para manejar el ciclo de vida de la agenda de citas.
3. Identificar cuáles componentes del resto de la aplicación usan servicios de agenda de citas.

4. Modificar esos componentes para consumir los servicios de la nueva aplicación en lugar de los servicios locales.
5. Diseñar casos de prueba para verificar que la nueva aplicación funciona tal y como lo hacía la aplicación original.
6. Ejecutar una prueba piloto.
7. Agregar la funcionalidad de asignaciones a la nueva aplicación y repetir el proceso.

En la aplicación independiente para agenda de citas se propone aplicar todas las recomendaciones descritas en este documento, de tal manera que al final del ejercicio podamos confirmar que se obtienen los beneficios esperados y se cuente con una aplicación que pudiera servir como marco de referencia para los nuevos desarrollos. Al mismo tiempo se estarían construyendo las bibliotecas de componentes y adecuando los lineamientos de la institución con las mejores prácticas identificadas durante el proceso.

9. Conclusiones

En su punto más alto, el equipo de trabajo llegó a contar con más de 150 personas, entre diseñadores, programadores, administradores de base de datos y analistas funcionales. Coordinar el trabajo de un equipo de este tamaño es, por sí mismo, una tarea compleja donde, en mi opinión, los dos retos más grandes son lograr una comunicación efectiva entre los integrantes del equipo y la verificación constante de la calidad de los entregables. En mi opinión, los factores que permitieron que nuestra participación concluyera exitosamente fueron los siguientes:

La cercanía del equipo de desarrollo con los usuarios finales fue fundamental. Sin importar lo bien preparados que estén los documentos de requerimientos, es muy probable que existan diferencias entre lo descrito en los documentos, lo que entienden los programadores y lo que espera el usuario. Esto debe hacerse con mucho cuidado y con un manejo claro de las espectativas, ya que es posible que esta cercanía propicie que los usuarios continuamente soliciten modificaciones o adiciones a los requerimientos originales.

Es muy importante definir un proceso de desarrollo de software balanceado. Por balance me refiero a que las actividades del proceso deben generar suficiente información para establecer el alcance y objetivos del proyecto con claridad, pero no debe convertirse en una distracción del objetivo final. Los artefactos derivados de este proceso deben ser herramientas útiles para la comunicación del equipo y no solamente documentos protocolarios. Los ciclos de desarrollo deben ser lo más corto posible para obtener retroalimentación temprana para reaccionar oportunamente.

Las decisiones de diseño, tanto en infraestructura como en arquitectura, deben ser comprobadas conforme el proyecto avanza. Por un lado, este seguimiento permite detectar oportunamente áreas de oportunidad en el diseño y tomar las acciones necesarias para corregirlo. Por otro lado, permite verificar que el código se apegue al diseño propuesto. La revisión sistemática del código debe ser una tarea prioritaria para el equipo de arquitectura, ya que de esta manera se pueden detectar oportunamente malas prácticas de programación o incumplimiento de los lineamientos establecidos. Adicionalmente, se facilita el mantenimiento del sistema con la participación de un número grande de programadores en las revisiones ya que el código es conocido por un mayor número de personas.

A diferencia de las experiencias con los contratistas anteriores, nuestro equipo de arquitectura tenía la capacidad técnica para detectar malas prácticas de programación o de uso de los entornos de trabajo y la autoridad para acordar con los líderes de desarrollo acciones para corregirlas.

Finalmente, para garantizar el funcionamiento correcto de la aplicación, es necesario vigilar continuamente el comportamiento de todos sus componentes. Existen diferentes herramientas, tanto comerciales como de software libre, que ayudan a esta vigilancia proporcionando información sobre diferentes aspectos: sistema operativo, redes de comu-

nicaciones, bases de datos, servidores de aplicaciones, etc.

Con estas herramientas es posible identificar métricas para determinar objetivamente si el rendimiento del sistema está dentro de valores aceptables o no. Adicionalmente, esta información es muy útil para hacer una adecuada planeación de la capacidad y permite detectar oportunamente problemas en los diferentes componentes de la solución.

10. Anexos

10.1. Anexo 1. Monitoreo de errores de la aplicación.

Como se comentó en el cuerpo del reporte, una de las principales funciones del grupo a mi cargo fue el monitoreo de la aplicación durante las etapas de pruebas de aceptación de usuario y pruebas piloto. En caso de que ocurrieran errores en la aplicación, tratar de identificar las causas de éstos y coordinar con el equipo de desarrollo las acciones necesarias para su corrección.

Uno de los elementos que con los que se cuenta para este fin son las bitácoras del sistema. Al tratarse de una aplicación web se cuenta con las bitácoras estándar de los servidores http: los archivos access.log y error.log. En estos archivos se registra cada solicitud que recibe el servidor, la respuesta a la solicitud y, en caso de que ocurran, los errores encontrados. Estos errores se refieren exclusivamente al contenido estático; para el caso del contenido dinámico, Weblogic produce bitácoras de eventos de sistema en las que se puede configurar diferentes niveles de detalle. La siguiente tabla muestra los niveles en orden descendente de verbosidad:

Cuadro 3: Alcance de vida de los managed beans de JSF

Tipo de Alcance	Descripción
ALL	Máximo nivel de detalle, muestra todos los mensajes.
TRACE	Muestra mensajes muy detallados de los eventos ocurridos durante la ejecución de la aplicación.
DEBUG	Similar al anterior, muestra un menor nivel de detalle.
INFO	Similar al anterior, con muy poco nivel de detalle; sólo produce mensajes que muestran a grandes rasgos el avance de la ejecución.
WARN	Muestra mensajes de situaciones que pueden causar problemas.
ERROR	Muestra todos los mensajes de error.
FATAL	Sólo muestra mensajes de errores muy severos que posiblemente causen que la aplicación se detenga.
OFF	No muestra ningún mensaje.

El primer problema que encontramos fue el nivel de reporte con el que estaba configurada la aplicación en producción. El equipo de desarrollo configuraba niveles TRACE o DEBUG para ayudarse en las pruebas unitarias o incluso usaban instrucciones println para

escribir información directamente en la consola de la aplicación. Los ejecutables que se preparaban para hacer pruebas de aceptación de usuario tenían esta configuración.

Lo anterior causaba que la escritura de las bitácoras se convirtiera en un cuello de botella y que los servidores estuvieran detenidos mucho tiempo, esperando que se terminara de escribir en las bitácoras. Con esto los tiempos de respuesta eran muy malos y a menudo se tenía que reiniciar los servidores para obtener tiempos de respuesta aceptables.

Para corregir esta situación se cambió el proceso para generar los programas ejecutables. Esta responsabilidad fue transferida del equipo de desarrollo al equipo de arquitectura, donde procedimos a modificar la configuración de maven para automatizar la creación de archivos ejecutables con diferentes configuraciones, dependiendo del uso que se les fuera a dar.

En particular, para los programas para producción y pruebas de aceptación se cambió el nivel de las bitácoras en los servidores de producción, optando por el nivel ERROR para limitar el volumen de información a escribir en las bitácoras, pero conservando la información mínima que permitiera investigar causas de error.

Al mismo tiempo se localizaron todas las instrucciones println y se pidió a los programadores que las eliminaran o las cambiaran por registros en la bitácora de errores. Estas medidas mejoraron notablemente los tiempos de respuesta y permitieron avanzar en las pruebas. El siguiente tema a resolver fue la aparición de las “pantallas en blanco” que el sistema mostraba cuando ocurría algún error interno. Estos errores ocurrían cuando se presentaba algún error en la aplicación java que interrumpía la ejecución de la aplicación y dejaba la información en la base de datos en un estado inconsistente.

Pese a las medidas tomadas, los archivos de las bitácoras seguían siendo muy grandes, con frecuencia con más de un millón de líneas cada uno, lo que complicaba y hacía muy tedioso en análisis de los mismos.

El primer esfuerzo consistió en usar comandos de línea para buscar la ocurrencia de registros de error, que se reconocían porque el segundo campo de cada registro era igual a la cadena Error. El listado 13 muestra un registro de error, en este caso causado por un NullPointerException en la clase CitaBean, en el método buscarPacientes: La clase fue invocada por el archivo HTML seleccionarPacientePorAgenda.html.

Listado 13: Ejemplo de registro de error en la bitácora de la aplicación (continúa ...)

```
<Jan 15, 2013 6:21:41 AM CST> <Error> <HTTP> <BEA-101017>
<[weblogic.servlet.internal.WebAppServletContext@42df79 -
    appName: 'nece-1',
    name: 'nece-1', context-path: '/ece', spec-version: '2.5']
    Root cause of ServletException.
    javax.faces.FacesException: Error calling action method of component with id
    v_busqueda:frm_sel_pac_agenda:btn_buscar
at
   org.apache.myfaces.application.ActionListenerImpl.processAction(ActionListenerImpl.java:72)
at javax.faces.component.UICommand.broadcast(UICommand.java:141)
at org.ajax4jsf.component.AjaxActionComponent.broadcast(AjaxActionComponent.java:55)
at org.ajax4jsf.component.AjaxViewRoot.processEvents(AjaxViewRoot.java:324)
at org.ajax4jsf.component.AjaxViewRoot.broadcastEvents(AjaxViewRoot.java:299)
```

Listado 13: Ejemplo de registro de error en la bitácora de la aplicación (continúa ...)

```
Truncated. see log file for complete stacktrace
  javax.faces.el.EvaluationException: javax.el.ELException: //presmed/users/wlnece/domains/applications/nece-1.1.18/
     pages/agenda/cunehichc13/seleccionarPacientePorAgenda.xhtml @53,254 action="#{citaBean.buscarPacientes}":
java.lang.NullPointerException
at javax.faces.component._MethodExpressionToMethodBinding.invoke
   (_MethodExpressionToMethodBinding.java:82)
at org.apache.myfaces.application.ActionListenerImpl.processAction(ActionListenerImpl.java:57)
at javax.faces.component.UICommand.broadcast(UICommand.java:141)
at org.ajax4jsf.component.AjaxActionComponent.broadcast(AjaxActionComponent.java:55)
at org.ajax4jsf.component.AjaxViewRoot.processEvents(AjaxViewRoot.java:324)

Truncated. see log file for complete stacktrace
javax.el.ELException:
//presmed/users/wlnece/domains/applications/nece-1.1.18/
    pages/agenda/cunehichc13/seleccionarPacientePorAgenda.xhtml @53,254 action="#{citaBean.buscarPacientes}":
 java.lang.NullPointerException
at com.sun.facelets.el.TagMethodExpression.invoke(TagMethodExpression.java:74)
at javax.faces.component._MethodExpressionToMethodBinding.invoke
    (_MethodExpressionToMethodBinding.java:78)
at org.apache.myfaces.application.ActionListenerImpl.processAction(ActionListenerImpl.java:57)
at javax.faces.component.UICommand.broadcast(UICommand.java:141)
at org.ajax4jsf.component.AjaxActionComponent.broadcast(AjaxActionComponent.java:55)

Truncated. see log file for complete stacktrace
java.lang.NullPointerException
at org.hibernate.type.LongType.next(LongType.java:79)
at org.hibernate.engine.Versioning.increment(Versioning.java:131)
at org.hibernate.event.def.DefaultFlushEntityEventListener.getNextVersion
   (DefaultFlushEntityEventListener.java:387)
at org.hibernate.event.def.DefaultFlushEntityEventListener.scheduleUpdate
   (DefaultFlushEntityEventListener.java:279)
at org.hibernate.event.def.DefaultFlushEntityEventListener.onFlushEntity
   (DefaultFlushEntityEventListener.java:151)
Truncated. see log file for complete stacktrace
>
```

Una vez que se localizaba el error se le pasaba la información al equipo de desarrollo para que se realizara la corrección correspondiente.

Con el fin de automatizar esta tarea y generar estadísticas sobre el tipo de errores que se registraban en la bitácora se decidió construir un parser que analizara los archivos de la bitácora, buscara en cada traza del stack de llamadas Java las clases escritas por el equipo y generara un histograma con esta información.

Esta herramienta nos permitió ser proactivos y atender errores antes de que los reportaran el equipo de pruebas o los usuarios finales. Fue toda una sorpresa encontrar errores que no habían sido reportados y que ocurrían con mucha frecuencia.

El cuadro 4 muestra los errores más frecuentes encontrados al analizar las bitácoras del sistema generadas en el ambiente de prueba piloto durante un lapso de 4 horas.

También vimos que errores como “Not saved view state could be found for the view identifier” no habían sido reportados. Analizando a más detalle la bitácora, encontramos que esta situación en particular se presentaba cuando se intentaba ingresar al sistema después de haber dejado la pantalla de acceso al sistema inactiva por un periodo largo.

Listado 14: Error en página de acceso a la aplicación

```
<Jan 15, 2013 7:22:43 AM CST> <Error> <HTTP> <BEA-101017> <[weblogic.servlet.internal.WebAppServletContext@6bc2f -
      appName: 'nece-1', name: 'nece-1', context-path: '/ece', spec-version: '2.5']
     Root cause of ServletException.
javax.faces.application.ViewExpiredException: /pages/acceso/login.jsf
No saved view state could be found for the view identifier: /pages/acceso/login.jsf
 at org.apache.myfaces.lifecycle.RestoreViewExecutor.execute(RestoreViewExecutor.java:88)
 at org.apache.myfaces.lifecycle.LifecycleImpl.executePhase(LifecycleImpl.java:103)
Truncated. see log file for complete stacktrace
```

Cuadro 4: Reporte de errores más frecuentes

Error	Frecuencia
No saved view state could be found for the view identifier	210
Resource not registered	208
effects.js	163
capturaOtrosDatos.xhtml	75
registrarNotaMedicaBean.registrarNotaMedica	74
citaListaConsulta.jsf	71
registrarIngresoUrgencias:id_**********_239	71
seleccionarPacientePorAgenda.xhtml	62
citaBean.buscarPacientes	58
regIngresoUrgenciasBean.registrar	57
PrescripcionServiceImpl	49
registrarNotaMedicaBean.agregarDiagnostico	44
CitaConsultaFormatoBean	33
CitaBean.agregarMensajeAComponente	29
CitaBean.getCitasDto	22
duplicate Id for a component j_id127	22
citaListaConsulta.xhtml	21

Aunque esto en sí no es un error del sistema, se trata de un mensaje que nos dice que algo no está funcionando correctamente en la aplicación. Investigando a más profundidad, encontramos que es un error conocido en la versión de RichFaces que se estaba usando y que fue corregida en versiones posteriores.

10.2. Anexo 2. Análisis de la contención en threads.

Otra actividad muy importante es el monitoreo del uso de los threads en la JVM que ejecuta la aplicación. Este análisis permite identificar porciones del código que se ejecutan con lentitud, que originan locks y que pueden causar que Weblogic esté inestable o se detenga completamente.

Para hacer este análisis se requiere hacer varios thread dump cuando la aplicación está mostrando lentitud o inestabilidad. Lo común era pedir al equipo de operaciones 5 thread dumps con un intervalo de 40 segundos entre cada dump.

Un thread dump es una instantánea en formato de texto de todos los threads que se están ejecutando en un momento dado en una JVM. Esto es equivalente a un dump de procesos a nivel de sistema operativo. La información que incluye cada registro en el dump incluye:

- Nombre del thread
- Prioridad
- Grupo al que pertencece
- Estado (Running, Blocked, Waiting, Stuck, Parking)
- Trazo del thread.

A continuación un ejemplo de un registro en el thread dump con estado WAITING:

Listado 15: Ejemplo de un registro del thread dump

```
"[STANDBY] ExecuteThread: '26' for queue: 'weblogic.kernel.Default (self-tuning)'"
     waiting for lock weblogic.work.ExecuteThread@1d4bb19 WAITING
          java.lang.Object.wait(Native Method)
          java.lang.Object.wait(Object.java:485)
          weblogic.work.ExecuteThread.waitForRequest(ExecuteThread.java:157)
          weblogic.work.ExecuteThread.run(ExecuteThread.java:178)
```

El propósito de pedir 5 dumps con un intervalo de 40 segundos entre ellos es tratar de identificar threads que tardan más de 40 segundos en ejecutar su trabajo, condición que puede causar contención de recursos y, eventualmente, la caída e inestabilidad del sistema.

En Weblogic se puede configurar un tiempo máximo de espera, de tal manera que los threads que tarden en su ejecución más de este valor se etiqueten como “Stuck” y sean eliminadas.

El listado 16 muestra un thread marcado como stuck después de pasar el tiempo de espera configurado, en este caso 500 segundos. Es importante cuidar que esto no ocurra con frecuencia, ya que cuando muchos threads son marcados como stuck el sistema se vuelve muy inestable.

Listado 16: Thread marcado como "STUCK"

```
<Nov 21, 2012 11:16:52 AM CST> <Error> <WebLogicServer> <BEA-000337> <[STUCK] ExecuteThread: '20'
for queue: 'weblogic.kernel.Default (self-tuning)' has been busy for
"502" seconds working on the request
 "weblogic.servlet.internal.ServletRequestImpl@4a38dc[
POST /ece/pages/paciente/seleccionpacientecatalogo/selecPacPorCatalogo.jsf HTTP/1.1
Accept: image/jpeg, application/x-ms-application, image/gif, application/xaml+xml,
image/pjpeg, application/x-ms-xbap, application/vnd.ms-excel,
application/vnd.ms-powe rpoint, application/msword, */*
Referer:      http://ece.****.****/ece/pages/paciente/seleccionpacientecatalogo/
selecPacPorCatalogo.jsf
Accept-Language: es-ES
User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; Trident/4.0; SLCC2; .NET
CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0;
.NET 4.0C; .NET4.0E)
Content-Type: application/x-www-form-urlencoded
Accept-Encoding: gzip, deflate
Content-Length: 384
Connection: Keep-Alive
Cache-Control: no-cache
Cookie:
JSESSIONID=MRv4QtJhLzh1GSytybCwy2HJQJXKhNpC304yJkbJGkmqmkWH8XV6!-1580287373;
BIGipServerPool_necewl=1142037164.63822.0000
X-Forwarded-For: 11.45.22.145
X-Forwarded-For: 11.45.22.145%1
]", which is more than the configured time (StuckThreadMaxTime) of "500" seconds. Stack trace:
        org.hibernate.loader.BasicLoader.getEntityAliases(BasicLoader.java:54)
        org.hibernate.loader.Loader.getKeyFromResultSet(Loader.java:1121)
        org.hibernate.loader.Loader.getRowFromResultSet(Loader.java:588)
        org.hibernate.loader.Loader.doQuery(Loader.java:724)
        org.hibernate.loader.Loader.doQueryAndInitializeNonLazyCollections(Loader.java:259)
        org.hibernate.loader.Loader.doList(Loader.java:2232)
        org.hibernate.loader.Loader.listIgnoreQueryCache(Loader.java:2129)
        org.hibernate.loader.Loader.list(Loader.java:2124)
        org.hibernate.loader.criteria.CriteriaLoader.list(CriteriaLoader.java:118)
        org.hibernate.impl.SessionImpl.list(SessionImpl.java:1597)
        org.hibernate.impl.CriteriaImpl.list(CriteriaImpl.java:306)
        org.springframework.orm.hibernate3.HibernateTemplate$36.doInHibernate
(HibernateTemplate.java:1065)
        org.springframework.orm.hibernate3.HibernateTemplate.doExecute
(HibernateTemplate.java:419)
        org.springframework.orm.hibernate3.HibernateTemplate.executeWithNativeSession
(HibernateTemplate.java:374)
        org.springframework.orm.hibernate3.HibernateTemplate.findByCriteria
(HibernateTemplate.java:1055)
        org.springframework.orm.hibernate3.HibernateTemplate.findByCriteria
(HibernateTemplate.java:1048)
        ****.****.****.nece.internamiento.integracion.dao.impl.
AdmisionDaoImpl.consultarAdmisionPaciente(AdmisionDaoImpl.java:93)
        ****.****.****.nece.internamiento.servicios.impl.AdmisionServiceImpl.
consultarAdmisionPacientePorEstatus(AdmisionServiceImpl.java:154)
        sun.reflect.GeneratedMethodAccessor726.invoke(Unknown Source)
Caused by: org.springframework.transaction.TransactionSystemException:
Could not roll back Hibernate transaction; nested exception is org.hibernate.TransactionException: JDBC rollback failed
        at org.springframework.orm.hibernate3.HibernateTransactionManager.doRollback
(HibernateTransactionManager.java:677)
        at org.springframework.transaction.support.AbstractPlatformTransactionManager.processRollback
(AbstractPlatformTransactionManager.java:823)
        at org.springframework.transaction.support.AbstractPlatformTransactionManager.rollback
(AbstractPlatformTransactionManager.java:800)
        at org.springframework.transaction.interceptor.TransactionAspectSupport.
completeTransactionAfterThrowing(TransactionAspectSupport.java:339)
        at org.springframework.transaction.interceptor.TransactionInterceptor.
invoke(TransactionInterceptor.java:110)
        at org.springframework.aop.framework.ReflectiveMethodInvocation.
proceed(ReflectiveMethodInvocation.java:171)
        at
org.springframework.aop.framework.ReflectiveMethodInvocation.
proceed(ReflectiveMethodInvocation.java:171)
        at org.springframework.aop.framework.JdkDynamicAopProxy.
invoke(JdkDynamicAopProxy.java:204)
        at $Proxy157.consultarAdmisionPacientePorEstatus(Unknown Source)
        at ****.****.****.nece.paciente.presentacion.SelecPacPorCatalogoBean.
getTieneIngresoSinEgreso(SelecPacPorCatalogoBean.java:1243)
        at sun.reflect.GeneratedMethodAccessor728.invoke(Unknown Source)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:25)
        at java.lang.reflect.Method.invoke(Method.java:597)
        at javax.el.BeanELResolver.getValue(BeanELResolver.java:261)
        at javax.el.CompositeELResolver.getValue(CompositeELResolver.java:143)
        at org.apache.myfaces.el.unified.resolver.FacesCompositeELResolver.
getValue(FacesCompositeELResolver.java:140)
        at com.sun.el.parser.AstValue.getValue(AstValue.java:118)
        at com.sun.el.ValueExpressionImpl.getValue(ValueExpressionImpl.java:192)
        at com.sun.facelets.el.TagValueExpression.getValue(TagValueExpression.java:71)
```

El análisis de los thread dumps nos permitió identificar diferentes causas de errores:

1. La llamada síncrona a servicios externos. Si por alguna razón el sistema externo no contestaba, muchos threads de Weblogic empezaban a ser marcados como stuck, hasta que eventualmente Weblogic se detenía por falta de recursos.
2. Malas prácticas de programación, como el uso deliberado de la sentencia sleep.
3. Consultas a base de datos muy costosas.
4. Iteraciones sin fin.

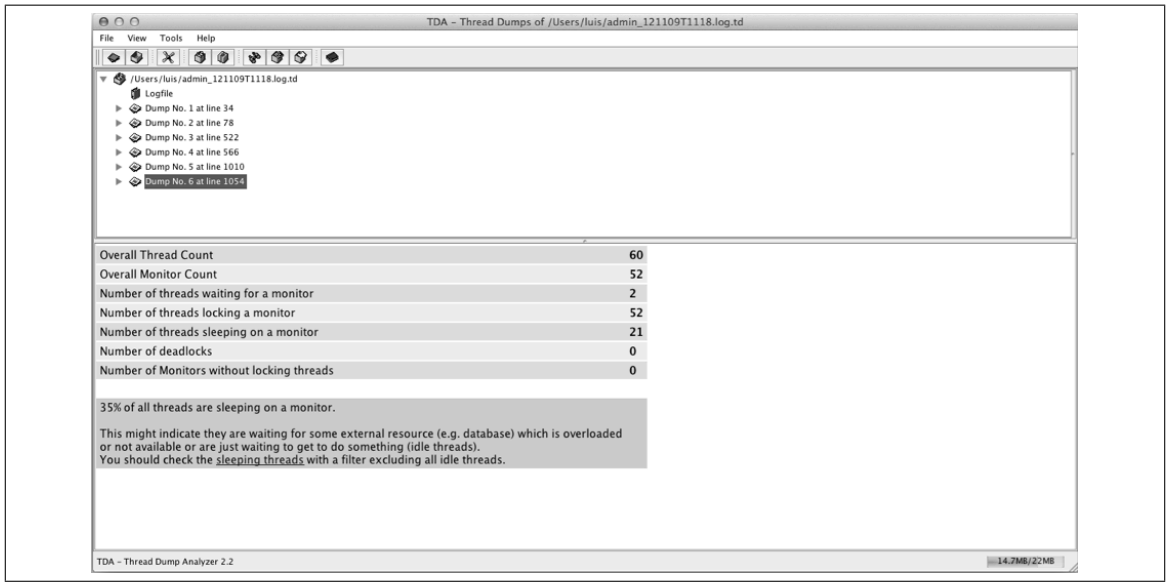

Figura 8: Pantalla de la herramienta TDA usada oara analizar el comportamiento de threads.

Existen varias herramientas que facilitan en análisis de los *thread dumps*. En este proyecto usamos TDA[12] que está disponible en https://java.net/projects/tda/downloads.

La figura 8 muestra el análisis que hace TDA de los thread dumps que están contenidos en las bitácoras del sistema. Un primer indicio de que la aplicación tiene problemas es que 21 threads de 60 se encuentran esperando turno para ser atendidos.

La figura 9 muestra una pantalla donde se aprecian threads que están en espera.

[12]Thread Dump Analizer por sus siglas en inglés

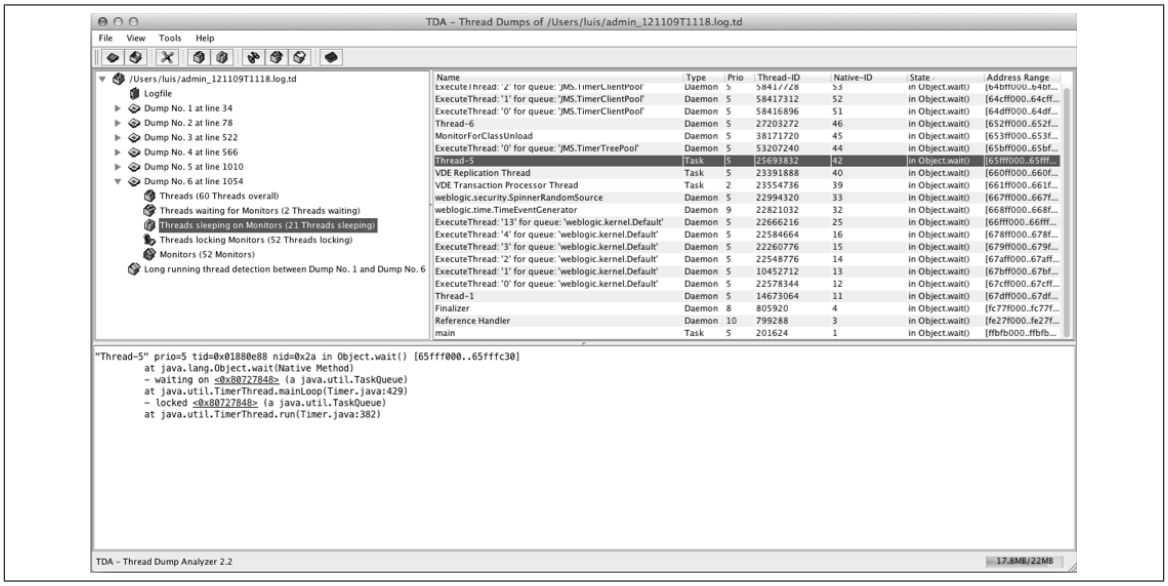

Figura 9: Threads en espera.

10.3. Anexo 3. Análisis de las consultas a base de datos.

Una de las causas más comunes de contención en el servidor de aplicaciones es la ejecución de consultas muy costosas a la base de datos.

Una vez que se identificó en el análisis de thread dumps que cierta consulta estaba causando que un thread estuviera detenido durante mucho tiempo, se procedía a analizar la consulta. Hay que recordar que se estableció como lineamiento usar la opción de criteria de hibernate para generar las consultas, lo que limitaba de manera importante el tipo de acciones que podíamos tomar para remediar la situación. A continuación se muestra a fines de ejemplo uno de los queries mas complejos generados automáticamente.

Listado 17: Ejemplo de una consulta generada por criteria de hibernate (continúa ...)

```
SELECT    this_.cve_idee_fecha_admision AS cve1_312_39_,
          this_.ORA_ROWSCN AS ora2_312_39_,
          this_.cve_especialidad AS cve3_312_39_,
          this_.cve_estado_proceso AS cve4_312_39_,
          this_.cve_proceso AS cve5_312_39_, this_.cve_idee AS cve6_312_39_,
          atencion1_.cve_idee_fecha_atencion AS cve1_38_0_,
          atencion1_.ORA_ROWSCN AS ora2_38_0_,
          atencion1_.ind_alta_paciente AS ind3_38_0_,
          atencion1_.ref_cedula AS ref4_38_0_,
          atencion1_.ref_numero_docu AS ref5_38_0_,
          atencion1_.ref_otro_documento AS ref6_38_0_,
          atencion1_.ind_revision_expediente AS ind7_38_0_,
          atencion1_.cve_tipo_documento AS cve12_38_0_,
          atencion1_.cve_especialidad AS cve13_38_0_,
          atencion1_.stp_finalizada AS stp8_38_0_,
          atencion1_.stp_fecha_atencion AS stp9_38_0_,
          atencion1_.stp_fecha_registro AS stp10_38_0_,
          atencion1_.cve_historico_paciente AS cve14_38_0_,
          atencion1_.cve_unidad_medica AS cve15_38_0_,
          atencion1_.cve_lugar_atencion AS cve16_38_0_,
          atencion1_.cve_idee AS cve17_38_0_,
          atencion1_.cve_tipo_atencion AS cve18_38_0_,
          atencion1_.cve_turno AS cve19_38_0_,
          tipodocume4_.cve_tipo_documento AS cve1_62_1_,
          tipodocume4_.fec_baja AS fec2_62_1_,
          tipodocume4_.des_tipo_documento AS des3_62_1_,
          especialid5_.cve_especialidad AS cve1_133_2_,
          especialid5_.fec_baja AS fec2_133_2_,
          especialid5_.ind_css AS ind3_133_2_,
          especialid5_.ind_ce AS ind4_133_2_,
          especialid5_.des_especialidad AS des5_133_2_,
          especialid5_.ind_hosp AS ind6_133_2_,
          especialid5_.ind_iq AS ind7_133_2_,
          historicop6_.cve_historico_paciente AS cve1_186_3_,
          historicop6_.ORA_ROWSCN AS ora2_186_3_,
          historicop6_.ref_agregado_afiliacion AS ref3_186_3_,
          historicop6_.ref_agregado_medico AS ref4_186_3_,
          historicop6_.cve_direccion AS cve7_186_3_,
          historicop6_.cve_estado_civil AS cve8_186_3_,
          historicop6_.fec_vigencia AS fec5_186_3_,
          historicop6_.ind_escolaridad_completa AS ind6_186_3_,
          historicop6_.cve_idee AS cve9_186_3_,
          historicop6_.cve_preferencia_sexual AS cve10_186_3_,
          historicop6_.cve_religion AS cve11_186_3_,
          historicop6_.cve_tipo_escolaridad AS cve12_186_3_,
          direccion7_.cve_direccion AS cve1_184_4_,
          direccion7_.ORA_ROWSCN AS ora2_184_4_,
          direccion7_.ref_calle AS ref3_184_4_,
          direccion7_.ref_cp AS ref4_184_4_,
          direccion7_.ref_colonia_fracc AS ref5_184_4_,
          direccion7_.ref_direccion AS ref6_184_4_,
          direccion7_.cve_estado AS cve10_184_4_,
          direccion7_.cve_localidad AS cve11_184_4_,
          direccion7_.cve_municipio AS cve12_184_4_,
          direccion7_.ref_numero_exterior AS ref7_184_4_,
          direccion7_.ref_numero_interior AS ref8_184_4_,
          direccion7_.ref_telefono AS ref9_184_4_,
          estadocivi8_.cve_estado_civil AS cve1_185_5_,
          estadocivi8_.fec_baja AS fec2_185_5_,
          estadocivi8_.des_estado_civil AS des3_185_5_,
```

Listado 17: Ejemplo de una consulta generada por criteria de hibernate (continúa ...)

```
        paciente9_.cve_idee AS cve1_187_6_,
        paciente9_.ORA_ROWSCN AS ora2_187_6_,
        paciente9_.ref_agregado_medico AS ref3_187_6_,
        paciente9_.ref_apellido_materno AS ref4_187_6_,
        paciente9_.ref_apellido_paterno AS ref5_187_6_,
        paciente9_.cve_curp AS cve6_187_6_,
        paciente9_.fec_defuncion AS fec7_187_6_,
        paciente9_.ind_enfermedad_cronica AS ind8_187_6_,
        paciente9_.fec_nacimiento AS fec9_187_6_,
        paciente9_.cve_grupo_sangre AS cve13_187_6_,
        paciente9_.cve_lugar_nacimiento AS cve14_187_6_,
        paciente9_.nom_nombre AS nom10_187_6_,
        paciente9_.num_nss AS num11_187_6_,
        paciente9_.ref_rfc AS ref12_187_6_,
        paciente9_.cve_sexo AS cve15_187_6_,
        paciente9_.cve_unidad_medica AS cve16_187_6_,
        preferenci10_.cve_preferencia_sexual AS cve1_112_7_,
        preferenci10_.fec_baja AS fec2_112_7_,
        preferenci10_.des_preferencia_sexual AS des3_112_7_,
        religion11_.cve_religion AS cve1_113_8_,
        religion11_.fec_baja AS fec2_113_8_,
        religion11_.des_religion AS des3_113_8_,
        tipoescola12_.cve_tipo_escolaridad AS cve1_190_9_,
        tipoescola12_.fec_baja AS fec2_190_9_,
        tipoescola12_.des_tipo_escolaridad AS des3_190_9_,
        lugaratenc13_.cve_unidad_medica AS cve1_205_10_,
        lugaratenc13_.cve_lugar_atencion AS cve2_205_10_,
        lugaratenc13_.ORA_ROWSCN AS ora3_205_10_,
        lugaratenc13_.ind_estatus AS ind4_205_10_,
        lugaratenc13_.ref_grupo_descripcion AS ref5_205_10_,
        lugaratenc13_.can_max_personas AS can6_205_10_,
        lugaratenc13_.num_sesiones AS num7_205_10_,
        lugaratenc13_.cve_tipo_lugar_atencion AS cve8_205_10_,
        tipolugara14_.cve_tipo_lugar_atencion AS cve1_36_11_,
        tipolugara14_.fec_baja AS fec2_36_11_,
        tipolugara14_.des_tipo_lugar_atencion AS des3_36_11_,
        tipolugara14_.ind_interno_externo AS ind4_36_11_,
        unidadmedi15_.cve_presupuestal AS cve1_145_12_,
        unidadmedi15_.ORA_ROWSCN AS ora2_145_12_,
        unidadmedi15_.ref_presupuestal_farmacia AS ref3_145_12_,
        unidadmedi15_.cve_clues AS cve4_145_12_,
        unidadmedi15_.cve_localidad AS cve5_145_12_,
        unidadmedi15_.cve_municipio AS cve6_145_12_,
        unidadmedi15_.cve_localizacion AS cve7_145_12_,
        unidadmedi15_.cve_servicio AS cve8_145_12_,
        unidadmedi15_.cve_tipo_inmueble AS cve9_145_12_,
        unidadmedi15_.cve_unidad_presup AS cve10_145_12_,
        unidadmedi15_.cve_delegacion AS cve23_145_12_,
        unidadmedi15_.dom_calle AS dom11_145_12_,
        unidadmedi15_.ref_cp AS ref12_145_12_,
        unidadmedi15_.ref_colonia_fracc AS ref13_145_12_,
        unidadmedi15_.ref_telefono AS ref14_145_12_,
        unidadmedi15_.cve_estado AS cve24_145_12_,
        unidadmedi15_.fec_baja AS fec15_145_12_,
        unidadmedi15_.ref_franja_horaria AS ref16_145_12_,
        unidadmedi15_.ind_horario_verano AS ind17_145_12_,
        unidadmedi15_.ref_unidad_medica AS ref18_145_12_,
        unidadmedi15_.cve_numero_economico AS cve19_145_12_,
        unidadmedi15_.cve_prei AS cve20_145_12_,
        unidadmedi15_.ind_mf AS ind21_145_12_,
        unidadmedi15_.cve_nivel_atencion AS cve25_145_12_,
        unidadmedi15_.ind_triage AS ind22_145_12_,
        paciente16_.cve_idee AS cve1_187_13_,
        paciente16_.ORA_ROWSCN AS ora2_187_13_,
        paciente16_.ref_agregado_medico AS ref3_187_13_,
        paciente16_.ref_apellido_materno AS ref4_187_13_,
        paciente16_.ref_apellido_paterno AS ref5_187_13_,
        paciente16_.cve_curp AS cve6_187_13_,
        paciente16_.fec_defuncion AS fec7_187_13_,
        paciente16_.ind_enfermedad_cronica AS ind8_187_13_,
        paciente16_.fec_nacimiento AS fec9_187_13_,
        paciente16_.cve_grupo_sangre AS cve13_187_13_,
        paciente16_.cve_lugar_nacimiento AS cve14_187_13_,
        paciente16_.nom_nombre AS nom10_187_13_,
        paciente16_.num_nss AS num11_187_13_,
        paciente16_.ref_rfc AS ref12_187_13_,
        paciente16_.cve_sexo AS cve15_187_13_,
        paciente16_.cve_unidad_medica AS cve16_187_13_,
        gruposangr17_.cve_grupo_sangre AS cve1_153_14_,
        gruposangr17_.fec_baja AS fec2_153_14_,
        gruposangr17_.des_grupo_sangre AS des3_153_14_,
        estado18_.cve_estado AS cve1_134_15_,
        estado18_.fec_baja AS fec2_134_15_,
        estado18_.nom_estado AS nom3_134_15_,
        estado18_.cve_pais AS cve4_134_15_, sexo19_.cve_sexo AS cve1_188_16_,
```

Listado 17: Ejemplo de una consulta generada por criteria de hibernate (continúa ...)

```
        sexo19_.fec_baja AS fec2_188_16_, sexo19_.des_sexo AS des3_188_16_,
        unidadmedi20_.cve_presupuestal AS cve1_145_17_,
        unidadmedi20_.ORA_ROWSCN AS ora2_145_17_,
        unidadmedi20_.ref_presupuestal_farmacia AS ref3_145_17_,
        unidadmedi20_.cve_clues AS cve4_145_17_,
        unidadmedi20_.cve_localidad AS cve5_145_17_,
        unidadmedi20_.cve_municipio AS cve6_145_17_,
        unidadmedi20_.cve_localizacion AS cve7_145_17_,
        unidadmedi20_.cve_servicio AS cve8_145_17_,
        unidadmedi20_.cve_tipo_inmueble AS cve9_145_17_,
        unidadmedi20_.cve_unidad_presup AS cve10_145_17_,
        unidadmedi20_.cve_delegacion AS cve23_145_17_,
        unidadmedi20_.dom_calle AS dom11_145_17_,
        unidadmedi20_.ref_cp AS ref12_145_17_,
        unidadmedi20_.ref_colonia_fracc AS ref13_145_17_,
        unidadmedi20_.ref_telefono AS ref14_145_17_,
        unidadmedi20_.cve_estado AS cve24_145_17_,
        unidadmedi20_.fec_baja AS fec15_145_17_,
        unidadmedi20_.ref_franja_horaria AS ref16_145_17_,
        unidadmedi20_.ind_horario_verano AS ind17_145_17_,
        unidadmedi20_.ref_unidad_medica AS ref18_145_17_,
        unidadmedi20_.cve_numero_economico AS cve19_145_17_,
        unidadmedi20_.cve_prei AS cve20_145_17_,
        unidadmedi20_.ind_mf AS ind21_145_17_,
        unidadmedi20_.cve_nivel_atencion AS cve25_145_17_,
        unidadmedi20_.ind_triage AS ind22_145_17_,
        afiliacion21_.cve_idee AS cve1_181_18_,
        afiliacion21_.ORA_ROWSCN AS ora2_181_18_,
        afiliacion21_.ref_consultorio AS ref3_181_18_,
        afiliacion21_.ind_derecho_incapacidad AS ind4_181_18_,
        afiliacion21_.ind_derecho_serv_medico AS ind5_181_18_,
        afiliacion21_.ref_domicilio AS ref6_181_18_,
        afiliacion21_.fec_fin_vigencia AS fec7_181_18_,
        afiliacion21_.fec_inicio_labores AS fec8_181_18_,
        afiliacion21_.fec_termino_labores AS fec9_181_18_,
        afiliacion21_.fec_ultima_consulta_acceder AS fec10_181_18_,
        afiliacion21_.ref_razon_social AS ref11_181_18_,
        afiliacion21_.ref_registro_patronal AS ref12_181_18_,
        afiliacion21_.ref_telefono AS ref13_181_18_,
        afiliacion21_.ref_tipo_pension AS ref14_181_18_,
        afiliacion21_.cve_turno AS cve15_181_18_,
        tipoatenci22_.cve_tipo_atencion AS cve1_59_19_,
        tipoatenci22_.fec_baja AS fec2_59_19_,
        tipoatenci22_.des_tipo_atencion AS des3_59_19_,
        turno23_.cve_turno AS cve1_209_20_,
        turno23_.fec_baja AS fec2_209_20_,
        turno23_.des_turno AS des3_209_20_,
        complement24_.cve_idee_fecha_atencion AS cve1_41_21_,
        complement24_.stp_fin_atencion_nota_medica AS stp2_41_21_,
        complement24_.ORA_ROWSCN AS ora3_41_21_,
        complement24_.ind_alta AS ind4_41_21_,
        complement24_.can_metodo AS can5_41_21_,
        complement24_.ind_metodo_riesgo AS ind6_41_21_,
        complement24_.cve_destino_envio AS cve11_41_21_,
        complement24_.cve_proceso AS cve12_41_21_,
        complement24_.num_dias_incapacidad AS num7_41_21_,
        complement24_.cve_invalidez AS cve13_41_21_,
        complement24_.ind_pase_otra_unidad AS ind8_41_21_,
        complement24_.cve_especificacion_metodo_pf AS cve14_41_21_,
        complement24_.num_recetas AS num9_41_21_,
        complement24_.num_semanas_gestacion AS num10_41_21_,
        complement24_.cve_materno_infantil AS cve15_41_21_,
        complement24_.cve_tipo_riesgo AS cve16_41_21_,
        embarazo25_.cve_idee_fecha_atencion AS cve1_248_22_,
        embarazo25_.ORA_ROWSCN AS ora2_248_22_,
        embarazo25_.cve_condicion_embarazo AS cve8_248_22_,
        embarazo25_.fec_probable_de_parto AS fec3_248_22_,
        embarazo25_.fec_termino_embarazo AS fec4_248_22_,
        embarazo25_.num_intervalo_intergenesico AS num5_248_22_,
        embarazo25_.num_meses_gestacion AS num6_248_22_,
        embarazo25_.ind_muerte_perinatal AS ind7_248_22_,
        embarazo25_.cve_termino_embarazo AS cve9_248_22_,
        embarazo25_.cve_tipo_parto AS cve10_248_22_,
        tipoparto26_.cve_tipo_parto AS cve1_105_23_,
        tipoparto26_.fec_baja AS fec2_105_23_,
        tipoparto26_.des_tipo_parto AS des3_105_23_,
        tipoparto26_.cve_parto_especifico AS cve4_105_23_,
        terminoemb27_.cve_termino_embarazo AS cve1_406_24_,
        terminoemb27_.fec_baja AS fec2_406_24_,
        terminoemb27_.des_termino_embarazo AS des3_406_24_,
        tipoparto28_.cve_tipo_parto AS cve1_105_25_,
        tipoparto28_.fec_baja AS fec2_105_25_,
        tipoparto28_.des_tipo_parto AS des3_105_25_,
        tipoparto28_.cve_parto_especifico AS cve4_105_25_,
```

Listado 17: Ejemplo de una consulta generada por criteria de hibernate (continúa ...)

```
        notamedica29_.cve_idee_fecha_atencion AS cve1_50_26_,
        notamedica29_.ORA_ROWSCN AS ora2_50_26_,
        notamedica29_.cve_area_urgencias AS cve5_50_26_,
        notamedica29_.stp_firma AS stp3_50_26_,
        notamedica29_.cve_lugar_accidente AS cve6_50_26_,
        notamedica29_.cve_ocasion_servicio AS cve7_50_26_,
        notamedica29_.ind_rep_consent_emitido AS ind4_50_26_,
        notamedica29_.cve_tipo_sub_nota AS cve8_50_26_,
        notamedica29_.cve_tipo_nota AS cve9_50_26_,
        notamedica29_.cve_tipo_urgencia AS cve10_50_26_,
        areaurgenc30_.cve_area_urgencias AS cve1_376_27_,
        areaurgenc30_.fec_baja AS fec2_376_27_,
        areaurgenc30_.des_area_urgencias AS des3_376_27_,
        subtiponot31_.cve_tipo_sub_nota AS cve1_66_28_,
        subtiponot31_.fec_baja AS fec2_66_28_,
        subtiponot31_.des_tipo_sub_nota AS des3_66_28_,
        tiponotame32_.cve_tipo_nota AS cve1_65_29_,
        tiponotame32_.fec_baja AS fec2_65_29_,
        tiponotame32_.des_tipo_nota AS des3_65_29_,
        tipourgenc33_.cve_tipo_urgencia AS cve1_69_30_,
        tipourgenc33_.fec_baja AS fec2_69_30_,
        tipourgenc33_.des_tipo_urgencia AS des3_69_30_,
        hojaalta34_.cve_idee_fecha_atencion AS cve1_337_31_,
        hojaalta34_.ORA_ROWSCN AS ora2_337_31_,
        hojaalta34_.ind_autopsia AS ind3_337_31_,
        hojaalta34_.cve_destino_envio AS cve5_337_31_,
        hojaalta34_.cve_proceso AS cve6_337_31_,
        hojaalta34_.stp_egreso_programado AS stp4_337_31_,
        hojaalta34_.cve_motivo AS cve7_337_31_,
        hojaalta34_.cve_tipo_motivo AS cve8_337_31_,
        hojaalta34_.cve_programa_atencion AS cve9_337_31_,
        hojaalta34_.cve_tipo_riesgo AS cve10_337_31_,
        riesgotrab35_.cve_idee_fecha_atencion AS cve1_320_32_,
        riesgotrab35_.ORA_ROWSCN AS ora2_320_32_,
        riesgotrab35_.fec_dia_descanso AS fec3_320_32_,
        riesgotrab35_.cve_lugar_accidente AS cve7_320_32_,
        riesgotrab35_.stp_accidente AS stp4_320_32_,
        riesgotrab35_.tim_fin_labores AS tim5_320_32_,
        riesgotrab35_.tim_inicio_labores AS tim6_320_32_,
        especialid36_.cve_especialidad AS cve1_133_33_,
        especialid36_.fec_baja AS fec2_133_33_,
        especialid36_.ind_css AS ind3_133_33_,
        especialid36_.ind_ce AS ind4_133_33_,
        especialid36_.des_especialidad AS des5_133_33_,
        especialid36_.ind_hosp AS ind6_133_33_,
        especialid36_.ind_iq AS ind7_133_33_,
        estatus37_.cve_estado_proceso AS cve1_321_34_,
        estatus37_.cve_proceso AS cve2_321_34_,
        estatus37_.fec_baja AS fec3_321_34_,
        estatus37_.des_estado_proceso AS des4_321_34_,
        egreso38_.cve_idee_fecha_admision AS cve1_151_35_,
        egreso38_.ORA_ROWSCN AS ora2_151_35_,
        egreso38_.stp_egreso AS stp3_151_35_,
        egreso38_.stp_salida AS stp4_151_35_,
        egreso38_.ref_recibi_especificar AS ref5_151_35_,
        ingreso39_.cve_idee_fecha_admision AS cve1_315_36_,
        ingreso39_.ORA_ROWSCN AS ora2_315_36_,
        ingreso39_.stp_ingreso AS stp3_315_36_,
        ingreso39_.stp_programacion AS stp4_315_36_,
        ingreso39_.cve_procedencia AS cve5_315_36_,
        ingreso39_.cve_tipo_ingreso AS cve6_315_36_,
        procedenci40_.cve_procedencia AS cve1_317_37_,
        procedenci40_.fec_baja AS fec2_317_37_,
        procedenci40_.des_procedencia AS des3_317_37_,
        tipoingres41_.cve_tipo_ingreso AS cve1_316_38_,
        tipoingres41_.fec_baja AS fec2_316_38_,
        tipoingres41_.des_tipo_ingreso AS des3_316_38_
    FROM net_admision this_ INNER JOIN net_atencion_nota_medica atencion1_
        ON this_.cve_idee_fecha_admision = atencion1_.cve_idee_fecha_atencion
        LEFT OUTER JOIN nec_tipo_documento tipodocume4_
        ON atencion1_.cve_tipo_documento = tipodocume4_.cve_tipo_documento
        LEFT OUTER JOIN nec_especialidad especialid5_
        ON atencion1_.cve_especialidad = especialid5_.cve_especialidad
        LEFT OUTER JOIN net_historico_paciente historicop6_
        ON atencion1_.cve_historico_paciente =
                                        historicop6_.cve_historico_paciente
        LEFT OUTER JOIN net_direccion direccion7_
        ON historicop6_.cve_direccion = direccion7_.cve_direccion
        LEFT OUTER JOIN nec_estado_civil estadocivi8_
        ON historicop6_.cve_estado_civil = estadocivi8_.cve_estado_civil
        LEFT OUTER JOIN net_paciente paciente9_
        ON historicop6_.cve_idee = paciente9_.cve_idee
        LEFT OUTER JOIN nec_preferencia_sexual preferenci10_
        ON historicop6_.cve_preferencia_sexual = preferenci10_.cve_preferencia_sexual
```

Listado 17: Ejemplo de una consulta generada por criteria de hibernate (continúa ...)

```
        LEFT OUTER JOIN nec_religion religion11_
        ON historicop6_.cve_religion = religion11_.cve_religion
        LEFT OUTER JOIN nec_tipo_escolaridad tipoescola12_
        ON historicop6_.cve_tipo_escolaridad =
                                        tipoescola12_.cve_tipo_escolaridad
        LEFT OUTER JOIN net_lugar_atencion lugaratenc13_
        ON atencion1_.cve_unidad_medica = lugaratenc13_.cve_unidad_medica
      AND atencion1_.cve_lugar_atencion = lugaratenc13_.cve_lugar_atencion
        LEFT OUTER JOIN nec_tipo_lugar_atencion tipolugara14_
        ON lugaratenc13_.cve_tipo_lugar_atencion =
                                      tipolugara14_.cve_tipo_lugar_atencion
        LEFT OUTER JOIN net_unidad_medica unidadmedi15_
        ON lugaratenc13_.cve_unidad_medica = unidadmedi15_.cve_presupuestal
        LEFT OUTER JOIN net_paciente paciente16_
        ON atencion1_.cve_idee = paciente16_.cve_idee
        LEFT OUTER JOIN nec_grupo_sangre gruposangr17_
        ON paciente16_.cve_grupo_sangre = gruposangr17_.cve_grupo_sangre
        LEFT OUTER JOIN nec_estado estado18_
        ON paciente16_.cve_lugar_nacimiento = estado18_.cve_estado
        LEFT OUTER JOIN nec_sexo sexo19_
        ON paciente16_.cve_sexo = sexo19_.cve_sexo
        LEFT OUTER JOIN net_unidad_medica unidadmedi20_
        ON paciente16_.cve_unidad_medica = unidadmedi20_.cve_presupuestal
        LEFT OUTER JOIN net_afiliacion afiliacion21_
        ON paciente16_.cve_idee = afiliacion21_.cve_idee
        LEFT OUTER JOIN nec_tipo_atencion tipoatenci22_
        ON atencion1_.cve_tipo_atencion = tipoatenci22_.cve_tipo_atencion
        LEFT OUTER JOIN nec_turno turno23_
        ON atencion1_.cve_turno = turno23_.cve_turno
        LEFT OUTER JOIN net_complemento_4306 complement24_
        ON atencion1_.cve_idee_fecha_atencion =
                                      complement24_.cve_idee_fecha_atencion
        LEFT OUTER JOIN net_embarazo embarazo25_
        ON atencion1_.cve_idee_fecha_atencion =
                                        embarazo25_.cve_idee_fecha_atencion
        LEFT OUTER JOIN nec_tipo_parto tipoparto26_
        ON embarazo25_.cve_condicion_embarazo = tipoparto26_.cve_tipo_parto
        LEFT OUTER JOIN nec_termino_embarazo terminoemb27_
        ON embarazo25_.cve_termino_embarazo =
                                         terminoemb27_.cve_termino_embarazo
        LEFT OUTER JOIN nec_tipo_parto tipoparto28_
        ON embarazo25_.cve_tipo_parto = tipoparto28_.cve_tipo_parto
        LEFT OUTER JOIN net_nota_medica notamedica29_
        ON atencion1_.cve_idee_fecha_atencion =
                                      notamedica29_.cve_idee_fecha_atencion
        LEFT OUTER JOIN nec_area_urgencias areaurgenc30_
        ON notamedica29_.cve_area_urgencias =
                                        areaurgenc30_.cve_area_urgencias
        LEFT OUTER JOIN nec_tipo_sub_nota subtiponot31_
        ON notamedica29_.cve_tipo_sub_nota = subtiponot31_.cve_tipo_sub_nota
        LEFT OUTER JOIN nec_tipo_nota tiponotame32_
        ON notamedica29_.cve_tipo_nota = tiponotame32_.cve_tipo_nota
        LEFT OUTER JOIN nec_tipo_urgencia tipourgenc33_
        ON notamedica29_.cve_tipo_urgencia = tipourgenc33_.cve_tipo_urgencia
        LEFT OUTER JOIN net_prealta hojaalta34_
        ON notamedica29_.cve_idee_fecha_atencion =
                                        hojaalta34_.cve_idee_fecha_atencion
        LEFT OUTER JOIN net_riesgo_trabajo riesgotrab35_
        ON atencion1_.cve_idee_fecha_atencion =
                                      riesgotrab35_.cve_idee_fecha_atencion
        LEFT OUTER JOIN nec_especialidad especialid36_
        ON this_.cve_especialidad = especialid36_.cve_especialidad
        LEFT OUTER JOIN nec_estado_proceso estatus37_
        ON this_.cve_estado_proceso = estatus37_.cve_estado_proceso
      AND this_.cve_proceso = estatus37_.cve_proceso
        LEFT OUTER JOIN net_egreso egreso38_
        ON this_.cve_idee_fecha_admision = egreso38_.cve_idee_fecha_admision
        LEFT OUTER JOIN net_ingreso ingreso39_
        ON this_.cve_idee_fecha_admision = ingreso39_.cve_idee_fecha_admision
        LEFT OUTER JOIN nec_procedencia procedenci40_
        ON ingreso39_.cve_procedencia = procedenci40_.cve_procedencia
        LEFT OUTER JOIN nec_tipo_ingreso tipoingres41_
        ON ingreso39_.cve_tipo_ingreso = tipoingres41_.cve_tipo_ingreso
ORDER BY atencion1_.stp_fecha_atencion DESC
```

En el ambiente de pruebas, esta consulta tardaba 300 segundos en ejecutarse en promedio y entregaba más de 7,000 registros. Los problemas más graves de este consulta son:

1. La consulta recupera un número muy elevado de columnas (894) la mayoría de las

cuales no se utiliza. Esto es causado por el uso de criteria de Hibernate. Para corregir esta situación es necesario eliminar el uso de criteria y usar queries nombrados en su lugar.

2. La consulta entrega un número muy grande de registros, la mayoría de los cuales son filtrados y no se presentan al usuario. Este problema y el anterior causan que se mueva mucha información que no se usa entre el servidor de aplicaciones y el de base de datos, resultando en un uso innecesario de recursos.
3. Algunos de las sub-consultas necesarias para resolver la consulta usaban “full table scan” para acceder a la información.

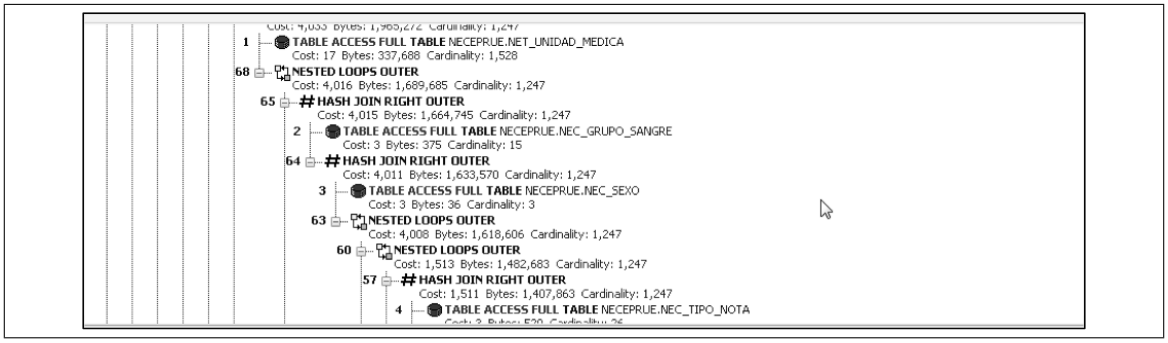

Figura 10: Vista parcial del explain plan de la consulta.

La figura 10 muestra una porción del resultado de ejecutar la sentencia explain plan para la consulta antes descrita. Aquí se puede notar que en las tablas net_unidad_medica, nec_grupo_sangre , nec_sexo y nec_tipo_nota se usa un full table scan para acceder a la información. Analizando los índices de estas tablas para modificar los índices existentes o agregar índices nuevos se redujo el tiempo de ejecución de la consulta a solo 600 milisegundos en promedio.

Con esta técnica se redujo de forma considerable el tiempo de respuesta en consultas similares, pero no se eliminó el problema del uso ineficiente de recursos. Para solucionar completamente los problemas de consultas como ésta, es necesario cambiar el uso de criteria de hibernate por consultas nombradas como se sugiere en la sección 7.

Referencias

[1] DAVE MINTER; JEFF LINWOOD , *Beginning Hibernate*, Editorial Apress, 2006; ISBN: 9781590596937

[2] ROB HARROP; JAN MACHACEK, *Pro Spring*, Editorial Apress, 2005; ISBN: 9781590594614

[3] NICOLAI M. JOSUTTIS, *SOA in Practice*, Editorial O'Really, 2007; ISBN: 9780596529550

[4] SAM ALAPATI, *Oracle Weblogic Server 11g Administration Handbook*, Editorial McGraw-Hill Osborne Media, 2011; ISBN: 9780071774253

[5] CARLO GHEZZI, *Fundamentals of Software Engineering (2nd Edition)*, Editorial Prentice Hall, 2002; ISBN: 9780133056990

[6] ED BURNS; CHRIS SCHALK *JavaServer Faces 2.0: The Complete Reference* Editorial McGraw-Hill, 2009 ISBN: 9780071625098

[7] Hibernate: http://www.hibernate.org/

[8] Java Server Faces: http://www.oracle.com/technetwork/java/javaee/javaserverfaces-139869.html

[9] Jasper Reports: http://community.jaspersoft.com/project/jasperreports-library

[10] Maven: http://maven.apache.org/

[11] RichFaces: http://www.jboss.org/richfaces

[12] Spring Framework: http://projects.spring.io/spring-framework/

[13] Spring Security: http://projects.spring.io/spring-security/