CIS0830-TK02

FedExpert: Federación de Sistemas Expertos

http://pegasus.javeriana.edu.co/~CIS0830TK02/

RICARDO ANDRÉS LÓPEZ PUMAREJO

PONTIFICIA UNIVERSIDAD JAVERIANA
FACULTAD DE INGENIERIA
CARRERA DE INGENIERIA DE SISTEMAS
BOGOTÁ, D.C.
2009

CIS0830-TK02

FedExpert: Federación de Sistemas Expertos

Autor:

Ricardo Andrés López Pumarejo

MEMORIA DEL TRABAJO DE GRADO REALIZADO PARA CUMPLIR UNO DE LOS REQUISITOS PARA OPTAR AL TITULO DE INGENIERO DE SISTEMAS

Director

Enrique González Guerrero

PONTIFICIA UNIVERSIDAD JAVERIANA
FACULTAD DE INGENIERIA
CARRERA DE INGENIERIA DE SISTEMAS
BOGOTÁ, D.C.
Diciembre, 2009

PONTIFICIA UNIVERSIDAD JAVERIANA
FACULTAD DE INGENIERIA
CARRERA DE INGENIERIA DE SISTEMAS

Rector Magnífico

Joaquín Emilio Sánchez García S.J.

Decano Académico Facultad de Ingeniería

Ingeniero Francisco Javier Rebolledo Muñoz

Decano del Medio Universitario Facultad de Ingeniería

Padre Sergio Bernal Restrepo S.J.

Directora de la Carrera de Ingeniería de Sistemas

Ingeniero Luis Carlos Díaz Chaparro

Director Departamento de Ingeniería de Sistemas

Ingeniero Germán Alberto Chavarro Flórez

Artículo 23 de la Resolución No. 1 de Junio de 1946

"La Universidad no se hace responsable de los conceptos emitidos por sus alumnos en sus proyectos de grado. Sólo velará porque no se publique nada contrario al dogma y la moral católica y porque no contengan ataques o polémicas puramente personales. Antes bien, que se vean en ellos el anhelo de buscar la verdad y la Justicia"

AGRADECIMIENTOS

A mis padres Luz Helena Pumarejo de López y Ricardo López Solano, a mis hermanas Diana Patricia López y Margarita Rosa López, y a todos mis familiares por su significativo apoyo en la consecución de este objetivo. Agradezco enormemente a mi director de trabajo de grado, el Ing. Enrique González Guerrero, dado que gracias a su compromiso y dedicación pude sacar adelante mi proyecto enfocado siempre a la mejoría del mismo.

Contenido

ABSTRACT

The solution of a complex problem or a problem that requires a wide knowledge in a general area will require the knowledgement of not just one but many Expert Systems. To offer all the services of the different Expert Systems like only one service, we need to implement a coordination mechanism, for the selection of the most adequate Expert System to solve the particular problem. We propose the FedExpert platform, which allows the integration of different Expert Systems.

RESUMEN

La solución de un problema complejo o un problema que requiere de un conocimiento amplio en un área general puede requerir del conocimiento de no sólo uno sino muchos Sistemas Expertos. Para ofrecer los servicios de los diferentes Sistemas Expertos como uno sólo, se debe implementar un mecanismo de coordinación, para la selección del Sistema Experto más adecuado para la solución del problema. Se propone la plataforma FedExpert, la cuál permite la integración de diferentes Sistemas Expertos para ofrecer múltiples áreas de conocimiento como una sola.

RESUMEN EJECUTIVO

Este trabajo de grado describe el desarrollo de una plataforma para realizar la colaboración entre múltiples Sistemas Expertos, cada uno especializado en un área de conocimiento similar pero diferente. Como ejemplo podemos utilizar el sistema médico actual; el cual se integra de diferentes médicos especializados cada uno en un área medica diferente pero todos en común saben de medicina. Un Sistema Experto es un programa de software que emula el comportamiento de un experto en un dominio concreto. El conjunto de todos los Sistemas Expertos se llamara Federación de Expertos. El propósito de la plataforma es brindarle a los usuarios la posibilidad de integración de múltiples fuentes de conocimiento ofrecidas por los múltiples Sistemas Expertos como si solo fuera un Sistema Experto más grande. Para realizar la tarea mencionada anteriormente dividimos este documento en 8 capítulos. Cada capítulo ofrece un resultado de los objetivos propuestos y son totalmente dependiente unos de otros para la solución del objetivo principal del problema.

El primer capítulo del trabajo de grado introduce la oportunidad y la problemática que se presentó para la realización del proyecto. El segundo capítulo del trabajo de grado inicia la tarea de investigar sobre el estado del arte de las herramientas y metodologías que existen y están implementadas en la actualidad para la colaboración, cooperación y/o integración de múltiples Sistemas Expertos con conocimientos similares o diferentes dentro de un área general. El estado del arte permitió realizar un análisis comparativo y sacar conclusiones de las herramientas y metodologías ya existentes, lo que permitió explorar una serie de preguntas y oportunidades para la realización de la plataforma. Estas preguntas y oportunidades están descritas en el tercer capítulo. En el cuarto capítulo se toman a consideración las oportunidades que fueron expuestas para la realización del modelo de negocio. El modelo de negocio permitió darle una visión general a los usos y funcionalidades que tendría la plataforma. En el modelo de negocio se introdujeron y describieron los términos principales que se usaron a lo largo del desarrollo del trabajo de grado, los cuales permitieron definir el rumbo y los alcances de la aplicación.

Una vez que se tenía la descripción del modelo de negocio para el trabajo de grado, se dio inicio al desarrollo del diseño de la plataforma el cuál se describe en el quinto capítulo. Se realizó el diseño del modelo general, el cuál describe todas las capas y componentes que hacen parte de la plataforma, con las interacciones que existen entre ellas. De la misma manera se realizó el diseño del modelo de agentes, el cuál introduce todos los agentes que componen el sistema, con la descripción de sus metas, estados, entradas sensoriales, actuadores y comportamientos. En el sexto capítulo a partir de la descripción general y de agentes del modelo, se pudo dar inicio al desarrollo de la descripción del modelo pero a un nivel más detallado. El diseño de los componentes de la plataforma brinda una descripción más detallada de las capas y los componentes. Se presentó el diseño y la definición de todos los componentes y clases que integran cada una de las capas propuestas.

Con el diseño de los componentes se inició la tarea de realizar un prototipo experimental para probar las principales características de la plataforma. Se definió un alcance para el prototipo, priorizando algunos requerimientos y decidiendo cuáles de esos serían implementados de acuerdo a las principales necesidades de la plataforma. Las capas del prototipo que fueron implementadas son la lógica de negocio y la base de datos. Para remplazar la capa de presentación de la plataforma, el prototipo cuenta con una interfaz gráfica de usuario la cuál permite realizar de una manera más practica y cómoda la comprobación de las funcionalidades que fueron implementadas. Para el prototipo implementado se diseñaron y se realizaron una seria de pruebas, cuya funcionalidad consistía en realizar una verificación de los componentes de la aplicación. Se realizaron tres tipos de pruebas diferentes que están documentadas en el capítulo 7, las cuales son: pruebas unitarias, pruebas de circuitos típicos, pruebas concurrentes y pruebas de estrés. Las pruebas unitarias se encargaban de realizar pruebas a una sola funcionalidad específica. Las pruebas de circuitos típicos comprobaban la funcionalidad de un grupo de funcionalidades específicas que dependen unas de otras. Las pruebas concurrentes validaban el funcionamiento y comportamiento de toda la aplicación funcionando en conjunto. Las pruebas de estrés tenían como meta comprobar la escalabilidad de la plataforma, mostrando el nivel de desempeño a medida que se le iban agregando Sistemas Expertos a la federación.

Los resultados obtenidos a las pruebas realizadas al prototipo desarrollado, mostraron un menor tiempo de respuesta con la implementación de los componentes definidos. Las conclusiones surgieron a partir de las comparaciones de resultados. Se calcularon los tiempos de respuesta sin los componentes diseñados y con los componentes. Los resultados obtenidos permitieron concluir que la integración de múltiples Sistemas Expertos implementando mecanismos de selección de los expertos permiten ofrecer un campo más grande de conocimiento con un tiempo de respuesta reducido y eficiente.

En conclusión, el gran aporte de este trabajo de grado es ofrecer un modelo funcional probado en sus características más importantes por medio de un prototipo funcional. Dicho modelo ofrece una herramienta colaborativa para Sistemas Expertos cuyo objetivo es ofrecer el conocimiento de diferentes áreas de experticia como si solo fuera una.

INTRODUCCIÓN

La inspiración del presente trabajo de grado fue basada en el conocimiento de expertos en el área de la medicina. La idea inicial que fue planteada en la propuesta del trabajo de grado fue PCEM (Plataforma de Colaboración entre Sistemas Expertos Médicos). Plataforma que se encargaba de ofrecer el conocimiento de diferentes Sistemas Expertos médicos, cada uno especializado en un área diferente de la medicina como un solo sistema experto. Después de realizar el estudio al estado del arte y realizar el diseño de la plataforma, se decidió generalizar el área de conocimiento para la plataforma, pasando del área medica, a cualquier área en general. De la generalización surgió FedExpert (Federación de Expertos) la cual hace referencia a múltiples expertos con conocimientos en un área particular, formando una federación con la capacidad de ofrecer los diferentes conocimientos que ofrecen los expertos como una sola área de conocimiento en general.

Se selecciono el uso de sistemas expertos debido a que los Sistemas Expertos están diseñados y programados para emular el comportamiento de un humano especialista en un área de conocimiento en particular. Dichos Sistemas Expertos tienen como objetivo ofrecer las respuestas en un tiempo inferior y con una calidad igual o superior que la de un humano especialista. En la actualidad existen múltiples Sistemas Expertos especializados cada uno en áreas de conocimiento diferentes, ellos están encargados de solucionar problemas particulares. A diferencia de un especialista humano, un Sistema Experto está libre de ciertas condiciones externas que pueden afectar el juicio de sus respuestas, las cuales los humanos si pueden estar afectados. Como por ejemplo: cansancio, sentimientos involucrados, envejecimiento etc. Factores que no afectan a los Sistemas Expertos, lo que nos permite obtener respuestas más confiables sin importar las condiciones con las que cuenta el Sistema Experto en su ambiente de trabajo.

La posibilidad de integrar el conocimiento de múltiples Sistemas Expertos, cada uno capacitado en solucionar problemas individuales en un sólo Sistema Experto, nos permite tener una herramienta con un área de experticia más amplia que la de uno sólo. Ofrecer mecanismos de negociación para formar una federación y así poder realizar la colaboración de múltiples Sistemas Expertos permite a los usuarios del sistema la posibilidad de tener a disposición en cualquier momento el conocimiento de múltiples expertos a partir de una sola entrada. Los usuarios pueden utilizar la herramienta para realizar los diagnósticos en áreas de experticia en las cuales no tienen conocimiento o usar los diagnósticos para la toma de decisiones en temas que conocen y sólo necesitan un refuerzo para su decisión como especialista. A partir de la problemática mencionada anteriormente, surge la necesidad de ofrecer una herramienta que permita la cooperación de una federación de Sistemas Expertos.

I - DESCRIPCION GENERAL DEL TRABAJO DE GRADO

1. Oportunidad ó Problemática

• 1.1 Descripción del Contexto

El cuidado médico es un área de colaboración creciente que involucra una gama de servicios médicos proporcionados por muchos individuos y organizaciones. A parte de la prestación de servicios de cuidado médico a pacientes durante la hospitalización o en cualquier centro médico, la habilidad de asistir a personas que tienen necesidades médicas en sus hogares o en lugares remotos donde no hay acceso a centros médicos se ha convertido en un tema critico creciente [KOUF2008].

Como la inspiración de realizar el trabajo de grado fue el sistema médico actual como lo es una Empresa Prestadora de Salud, se analizó un ejemplo en particular. El caso qué se analizó fue el de Cafesalud EPS [CAFE]: esta EPS presta los siguientes servicios:

- Medicina General
- Medicina Especializada
- Laboratorio
- Rayos X
- Medicamentos
- Odontología
- Hospitalización
- Cirugía
- Tratamientos de alto costo y los contemplados en el POS
- Actividades de promoción y Prevención

Pero la EPS como tal es el conjunto de los anteriores servicios no cada uno de ellos individualmente. La EPS forma parte de muchos servicios independientes, que a la final se ofrecen como una red de servicios integrados.

Analizando el campo de los sistemas expertos, estas son herramientas computacionales diseñadas para capturar y poner en disposición los conocimientos de expertos en un área específica. Los sistemas expertos se encargan de resolver problemas que de otra manera requerirían conocimiento experto humano extensivo para hacerlo. Para hacer lo anterior, el sistema experto hace simulación del razonamiento humano aplicando conocimiento específico [GIAR2001]. Así que podemos resaltar el por qué usar sistemas expertos en problemas reales.

Ventajas de los Sistemas Expertos [GIAR2001]:

- Replicación: los sistemas expertos reproducen el conocimiento y heurística de los expertos humanos. Esto permite que se pueda copiar y distribuir la experiencia tanto como se necesite.

- Fácil modificación: el concepto de separar el conocimiento del mecanismo de inferencia facilita el proceso de modificación del conocimiento. Esto es importante porque el conocimiento cambia frecuentemente.

- Consistencia en las respuestas: los expertos humanos pueden dar soluciones diferentes al mismo problema; incluso el mismo experto humano puede dar respuestas distintas en diferentes ocasiones. Por su parte los sistemas expertos son siempre consistentes en la solución de los problemas, brindando respuestas iguales todo el tiempo.

- Disponibilidad Permanente: los sistemas expertos están siempre disponibles las 24 horas del día.

- Preservación de la experiencia: el conocimiento de los expertos humanos presente en los sistemas expertos se preserva para la posteridad.

- Explicación de la solución: una característica clave de los sistemas expertos es que son capaces de explicar cómo llegaron a sus conclusiones, qué decisiones tomaron y por qué lo hicieron. Esta explicación clarifica y justifica los resultados.

La posibilidad de integrar el conocimiento de múltiples Sistemas Expertos, cada uno capacitado en solucionar problemas individuales en un sólo Sistema Experto, nos permite tener una herramienta con un área de experticia más amplia que la de uno sólo. Ofrecer mecanismos de negociación para formar una federación y así poder realizar la colaboración de múltiples Sistemas Expertos permite a los usuarios del sistema la posibilidad de tener a disposición en cualquier momento el conocimiento de múltiples expertos a partir de una sola entrada. Los usuarios pueden utilizar la herramienta para realizar los diagnósticos en áreas de experticia en las cuales no tienen conocimiento o usar los diagnósticos para la toma de decisiones en temas que conocen y sólo necesitan un refuerzo para su decisión como especialista.

Tres maneras de colaboración entre sistemas expertos son los siguientes:

- Utilizar mecanismos de selección del Sistema Experto que esté más calificado para resolver el problema.
- Utilizar sistemas expertos separados para resolver sub problemas de un problema más grande lo cual permite que las tareas se realicen paralelamente en menos tiempo. La combinación de los sub problemas resueltos provee una solución completa al problema grande original.
- Utilizar diferentes fuentes de conocimiento para resolver un problema que requiere conocimiento multidisciplinario. De esta manera se puede ofrecer un sistema experto más grande conformado por pequeños sub sistemas con conocimientos individuales.

Sistemas Expertos en el Área Médica

Teniendo las ventajas que ofrecen los sistemas expertos se empiezan a construir pequeños sistemas para el diagnóstico médico automático. Esto empezó en la década de los 60 donde se desarrollaron un gran número de programas basados en el cálculo de probabilidades para realizar diagnósticos de un pequeño grupo de enfermedades.

En la década de los 70, tres hechos incidieron positivamente en el desarrollo de los programas de diagnóstico médico: la aparición de los computadores personales lo que generó mayores niveles de cómputo, la aplicación de técnicas interactivas que facilitaban el uso de las computadoras y el desarrollo de técnicas de sistemas expertos. Los programas se volvieron más fáciles de manejar, más interactivos, capaces de justificar sus resultados y de explicar sus procesos, y tratando de simular un comportamiento similar al de los médicos humanos.

Durante esos años se construyeron los primeros sistemas expertos en diagnóstico médico, entre ellos *MYCIN* [BUCH2008], que fueron los que dieron impulso y fama a la Inteligencia Artificial. A lo largo de los años se han creado más y más sistemas expertos médicos especializados en áreas diferentes de conocimiento. Entre ellos se encuentran:

- *ESTDD* (Expert System for Thyroid Diseases Diagnosis). [KELE2008]
- A self-learning expert system for diagnosis in traditional Chinese medicine. [WANG2003]
- An intelligent system for the detection and interpretation of sleep apneas. [CANO2003]
- A neural network based clinical decision-support system for efficient diagnosis and fuzzy-based prescription of gynecological diseases using homoeopathic medicinal system. [MANG2005]
- *CASNET* (Causal ASsociation NETwork) diagnostica y realiza el tratamiento del glaucoma. [DOWI1988]
- *PIP* (Present Illnes Program), desarrollado en el MIT en 1976. Diagnostica enfermedades del riñón. [JOHN1985]

Como se puede observar, el diagnóstico médico se ha convertido en uno de los campos donde se han construido un gran número de sistemas expertos. Este hecho se ha debido principalmente a que la experiencia es fundamental para realizar diagnósticos certeros y correctos y a que los conocimientos en medicina son muy extensos y es frecuente el uso de datos inciertos e incompletos.

• 1.2 Formulación

Teniendo en cuenta la necesidad de integrar el conocimiento de múltiples sistemas expertos para generar una colaboración entre ellos, se preguntó por una herramienta que se encargue de brindar los servicios de varios sistemas expertos con diferentes áreas de conocimiento como un sólo sistema experto. Entonces sale a relucir la pregunta:

¿Cómo proponer una plataforma que permita la integración de sistemas expertos en un área de conocimiento especifica para ofrecer servicios integrados?

2. Descripción del Proyecto

• 2.1 Visión Global

Primero se realizó una investigación del estado del arte sobre el área de la cooperación de Sistemas Expertos. Sobre el estudio al estado del arte se realizó un análisis integral comparativo de las herramientas y tecnologías utilizadas. A partir del análisis integral comparativo surgieron una serie de oportunidades. Las oportunidades fueron útiles como base para identificar los requerimientos del sistema propuesto. Una vez tenemos las oportunidades a explotar, se describió el modelo de negocio de la plataforma. El modelo de negocio describe en una visión general el uso y las funcionalidades de la plataforma.

Se realizó el diseño del modelo general y el modelo de agentes del sistema. El modelo general muestra el diseño y la descripción y la interacción de los componentes que hacen parte de la plataforma. El modelo de agentes describe el desglose de los agentes actuantes en el sistema, mostrando todas sus interacciones. A partir del modelo general y el modelo de agentes se hizo la descripción detallada de los componentes. Para el desarrollo del prototipo de la aplicación se fijó un rumbo y un alcance. Se desarrolló el prototipo y a partir de éste se realizaron las pruebas. Las pruebas arrojaron resultados que fueron registrados y mostrados al final del documento.

• 2.2 Justificación

Los Sistemas Expertos emulan el comportamiento de un experto. Un Sistema Experto es capaz de solucionar un conjunto de problemas que exigen un gran conocimiento sobre un determinado tema. La posibilidad de realizar una cooperación entre varios Sistemas Expertos con diferentes áreas de conocimiento nos permite ofrecer una amplia fuente de conocimiento como un solo Sistema Experto grande. El proyecto FedExpert se realizó aprovechando la oportunidad mencionada anteriormente.

• 2.3 Objetivo general

Diseñar, desarrollar y validar una plataforma que permita la integración de servicios permitiendo la colaboración de diferentes sistemas expertos especializados.

• 2.4 Objetivos específicos

1. Realizar un análisis del estado del arte de la aplicación de los sistemas expertos en el área médica y de modelos de integración y cooperación.
2. Diseñar el modelo arquitectónico de la plataforma de colaboración entre sistemas expertos médicos.
3. Desarrollar el programa implementando el diseño de la plataforma.
4. Validar en forma experimental la plataforma implementada.

Con la presentación de la problemática y la descripción del proyecto tenemos una descripción de los objetivos del trabajo de grado. Se termina el capítulo 1 lo cual nos permite continuar a realizar el estudio al estado del arte, el cual será descrito en el capítulo 2.

II – ESTADO DEL ARTE

Se realizó la investigación del estado del arte de las herramientas diseñadas e implementadas actualmente para la colaboración y/o cooperación entre múltiples sistemas expertos. Se presenta el punto de vista de los diferentes autores con las posibles herramientas, ideas y metodologías que se pueden utilizar en el desarrollo del proyecto. Las ideas más importantes que se trataron fueron: la cooperación entre Sistemas Expertos utilizando agentes, la cooperación entre Sistemas Expertos realizando la descomposición de problemas a subproblemas y la solución de problemas por medio de fuentes de conocimiento separadas.

1. Cooperación entre Sistemas Expertos basado en Agentes

Un agente, es una entidad capaz de percibir su entorno, procesar tales percepciones y responder o actuar en su entorno de una manera correcta. Para maximizar el resultado esperado en la solución de problemas se estudió la posible cooperación entre sistemas expertos utilizando de por medio los agentes inteligentes.

• 1.1 Agriculture Expert system (Diagnosis and treatment for grapes) [SHAA2004]

Desde el punto de vista arquitectónico el sistema provee un marco para coordinar el comportamiento de varios agentes específicos. La sociedad de agentes en este sistema consiste en agentes de Sistemas Expertos (agentes de diagnóstico y un agente de tratamiento) que trabajan del lado del servidor.

Los agentes de Sistemas Expertos incluyen un módulo de traducción bilingüe KQML (Knowledge Query Manipulation Language). Los mensajes transferidos por el sistema utilizan el formato KQML. Los agentes se comunican enviando mensajes al agente de coordinación como se ilustra en la figura 1. Los clientes se comunican con el servidor mediante sockets por medio de Internet. Del lado del cliente el agente de la interfaz de usuario es el encargado de enviar las peticiones de diagnóstico o tratamiento al servidor. Las respuestas que se generan del lado del servidor son enviadas de vuelta a los usuarios humanos.

El conocimiento de diagnóstico está distribuido a lo largo de los agentes de diagnóstico. Cada agente es un experto autónomo de un cierto dominio de conocimiento de distintos pero parecidos dominios.

Como se ilustra en la figura 1 los agentes con los que cuenta el sistema son:

- Agente de Diagnóstico: responsable de producir el diagnóstico a partir de las observaciones insertadas.
- Agente de Tratamiento: responsable de encontrar una recomendación a cierto desorden o síntoma presentado.

- Agente de la Interfaz de Usuario: provee acceso a los agentes de sistemas expertos. Para la portabilidad del sistema este programa es un applet de java.
- Agente de Coordinación:
    - Actúa como interfaz entre el agente de la interfaz de usuario y los demás agentes.
    - Responsable de establecer el link de comunicación con el experto deseado a partir de la petición del usuario.
    - Realiza comunicación entre los agentes.
    - Cada vez que el agente de la interfaz de usuario necesita comunicarse con los otros agentes, éste envía un mensaje al agente de coordinación.
    - El agente de coordinación crea una instancia de la clase dándole los nombres de host y los puertos de los agentes.

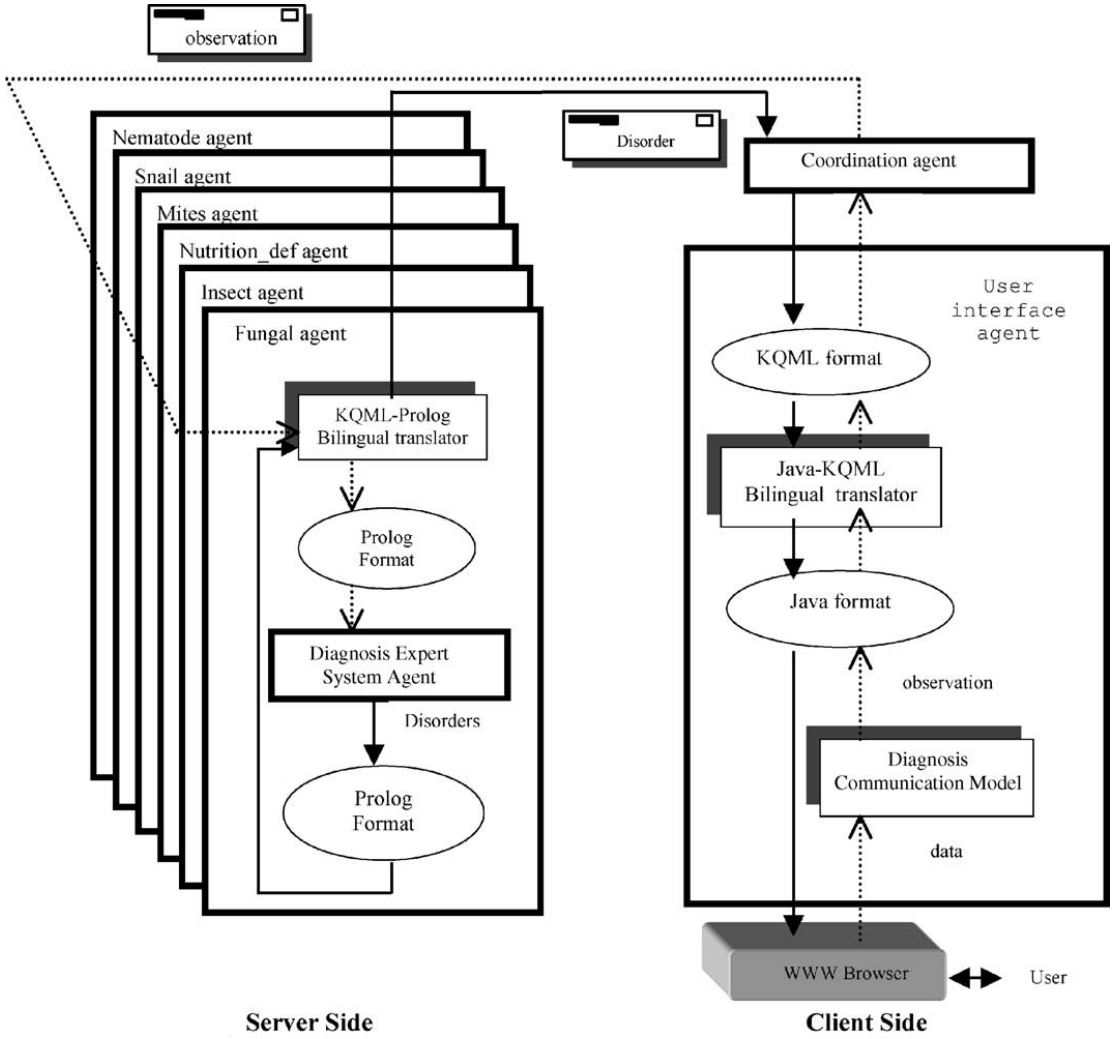

Figura 1 - Flujo de observación y desorden en una sesión de diagnóstico [SHAA2004]

El sistema tiene la capacidad de resolver problemas que requieren la experticia acumulativa de la comunidad de agentes y tiene como objetivo principal simular lo que pasa en la vida real, la forma de cómo operan los expertos en un ambiente de trabajo.

• 1.2 COOP [SHEK1989]

El ambiente Coop estructura la sociedad de agentes como un grupo de organizaciones que proveen infraestructuras y servicios necesitados por sistemas expertos individuales. Una organización es un grupo de agentes que proveen un único servicio. El ambiente Coop provee un conjunto de servicios y mecanismos para crear nuevas organizaciones.

Cuando un experto decide buscar ayuda, Él procede a identificar los agentes acordes para cooperar. Esto requiere información acerca de la experticia de otros agentes en el ambiente. La información se guarda en un agente servidor que sirve como servicio de directorio.

Una manera de reducir el esfuerzo requerido para entender y consultar la experticia existente es clasificando adecuadamente la colección de los sistemas disponibles utilizando notaciones de dominios y conceptos. Los dominios son los nombres que se le dan a una clase de problemas. Los dominios a menudo son sub divididos en sub dominios más pequeños.

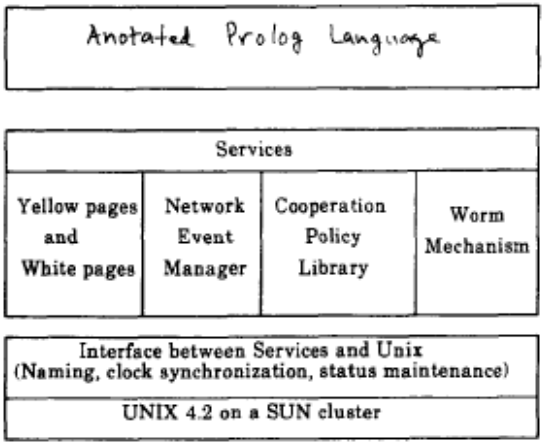

Figura 2 - El ambiente Coop par sistemas expertos cooperativos [SHEK1989]

Como se ilustra en la figura 2, el ambiente Coop provee los siguientes servicios:

- Cada regla en la base de reglas *prolog* está anotada con un par de número que cuantifican la confianza en dicha regla que da soporte de la regla. Cuando el soporte para una solución es menor que cierto umbral, el sistema experto decide buscar ayuda.
- La forma en que los agentes localizan a los especialistas a consultar es por medio, de *páginas amarillas* y para establecer la comunicación con el sistema utilizan las *páginas blancas.*

- Una *librería* de rutinas es suministrada para dar soporte a varias políticas y mecanismos de comunicación. Varios sistemas expertos pueden utilizar la librería para implementar políticas específicas de cooperación para usar en un problema dado.
- El *manejador de eventos de red* provee un servicio de mantenimiento de estado para recolectar el estado global de la red. Este servidor colecciona los estados locales de máquinas individuales en la red y las fusiona. El manejador de eventos monitorea el estado de la base de datos y detecta la ocurrencia de diferentes eventos. Cuando ocurre un evento se realiza la acción apropiada.
- El *Worm mechanism* es un servicio que ayuda en la iniciación del proceso. Controla y termina un grupo de procesos en el clúster que está conectado por Ethernet. Permite al agente ser tolerante a fallos.

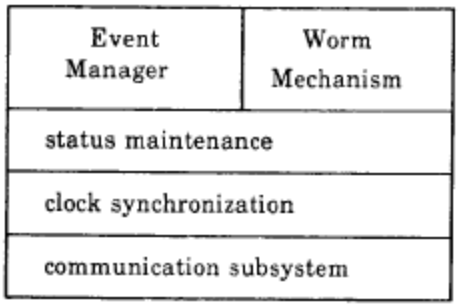

Figura 3 - Capas de la interfaz con Sistemas Operativos distribuidos [SHEK1989]

La implementación de Sistemas Expertos cooperativos necesita facilidades para iniciar monitorear y controlar los procesos en la red. Para dar soportes a estas necesidades se diseño un conjunto de capas como se ilustra en la figura 3:

- *Comunication Subsystem Layer:* establecer comunicación entre procesos.
- *Clock Synchronization:* garantiza que los relojes de todas las máquinas estén sincronizados.
- *Status Maintenance layer:* es responsable de mantener el estado de las bases y recoger los reportes.

• 1.3 Framework for distributed theorem proving [HUNG2005]

El llamado *theorem proving* (demostración de teoremas) es una técnica en un sistema de deducción la cual, trata de demostrar la exactitud de un conjunto de cláusulas que describen un problema. El *hiper-linking proof procedure* (HLPP) fue propuesto para eliminar la duplicación de cláusulas que ocurren en varios métodos de deducción y ha probado ser muy eficiente para demostrar algunos problemas lógicos de primer orden.

Esto motiva a los investigadores a explorar la computación paralela y distribuida para generar sistemas automatizados de deducción con alto rendimiento. Para implementar un sistema con dichas características se tienen que tomar las siguientes consideraciones:

- La partición de tareas: las tareas que se van a realizar de una manera distribuida deberían ser lo más independiente posible y de esta manera, cada tarea puede ser realizada por un agente independiente.
- El balanceo de cargas: las tareas que van a ser realizadas por los agentes tienen que ser distribuidas equitativamente.
- El monitoreo: elementos de procesamiento encargados de *monitorear* el progreso de las demostraciones y la confiabilidad del sistema para: reconocer cuántos hosts disponibles hay en el Internet, decidir los tipos de colaboración a realizar, ajustar las clausulas/literales que se reenviaran a otros hosts, monitorear el tráfico y predecir el tiempo de giro de cara host.

Las siguientes señales pueden ser necesarias para realizar la comunicación con todos los elementos de procesamiento:

- Informar la necesidad de ayuda para demostrar una nueva tarea.
- Informar a los elementos de procesamiento cuándo empezar o terminar la tarea.
- Informar a todos los elementos de procesamiento la finalización de toda la demostración.

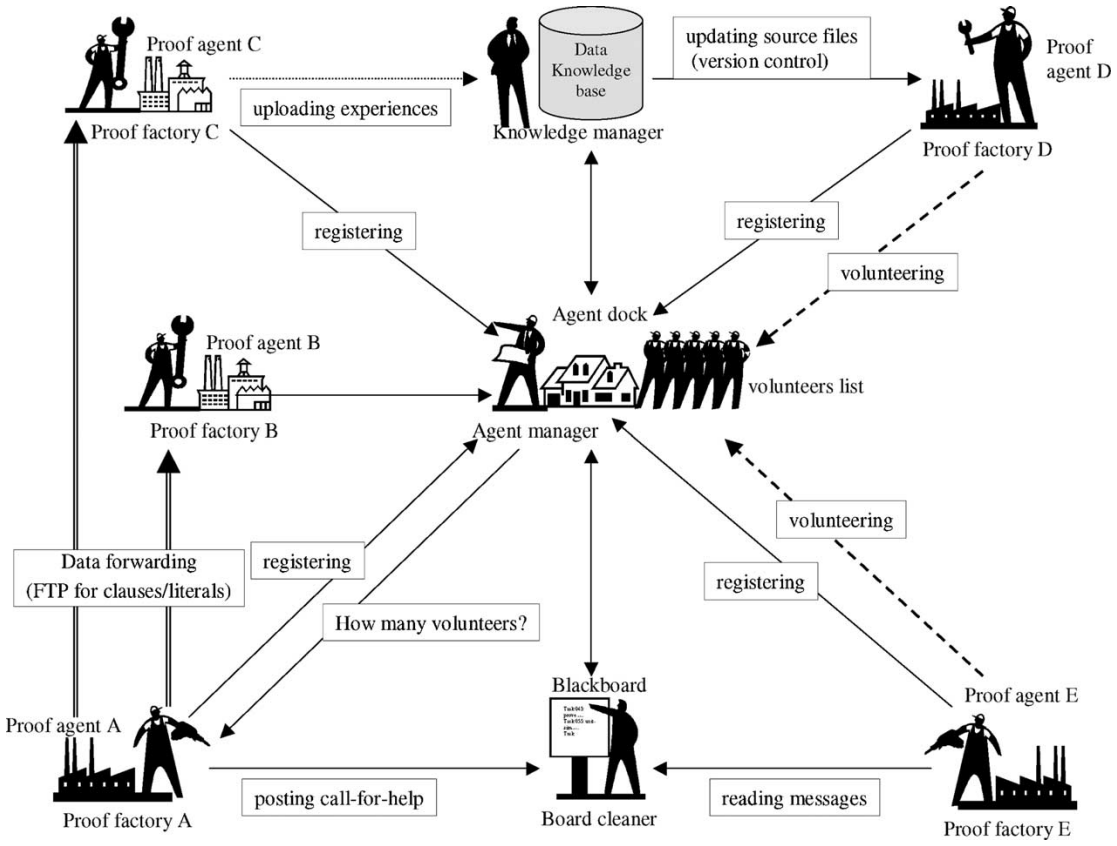

Figura 4 - Arquitectura de alto nivel del sistema [HUNG2005]

El sistema propuesto está basado en multi-agentes y se componen de los siguientes agentes, ilustrados en la figura 4:

1. Proof Agents

Es un agente que reside en un *Proof Factory* y tiene las siguientes tareas:

- Ofrecer su ayuda cuando su carga no está pesada.
- Invocar HLPP cuando un trabajo de demostración le fue asignado.
- Pedir ayuda cuando está sobrecargado.

El *Prof factory* es un elemento de procesamiento con la habilidad de invocar HLPP. Cada proof agent está asociado con una base de conocimiento que contiene respuestas esenciales y acciones predefinidas para la colaboración o comunicación.

2. System Agents

Se diseñaron para monitorear y controlar el estado del marco multia-gente presentado.

Agent Manager: reside en el agent dock (comunidad que está recolectando los proof agents disponibles), el cual nombra los agentes, monitorea la vida de los agentes registrados y lista los agentes voluntarios para una petición especifica. Antes de empezar la colaboración un proof factory tiene que registrarse en el agent dock. Cuando un proof agent se registra en el agent dock; su nombre, su dirección IP, el sistema operativo y el tiempo de registro son almacenados. Una lista de espera es asociada con el agent dock la cual es una lista de registros de los proof agents que se ofrecen a diferentes peticiones que son puestas en el Blackboard. El que llama puede acceder a la lista de espera para obtener el nombre y la dirección IP y reenviarle la información asociada a los voluntarios directamente vía FTP.

Blackboard y Board Cleaner

- En el Blackboard los agentes ponen sus peticiones de ayuda y las respuestas correspondientes a las peticiones.
- Los proof agents leen las peticiones que están en el Blackboard y si estiman que pueden atender esa petición ellos se ofrecen al agent dock.
- Una petición en el Blackboard tiene tres estados: nuevo (publicado pero no ha sido atendido), servido (en procesamiento) y fuera de fecha (no ha sido realizado por ningún agente en un periodo largo de tiempo).
- El board cleaner agent se encarga de la lista de voluntarios y mira los mensajes que son posteados en el Blackboard y rastrea sus estados.

Knowledge Manager

- Está diseñado en este marco para la administración del conocimiento y el control de versiones.
- Tiene dos tareas: actualizar e instalar la última versión del HLPP.

Comunicación basada en KQML

- Los agentes pueden comunicarse unos con otros por medio de KQML utilizando una red Peer to Peer. Esto permite que los agentes se puedan comunicar sin la intervención de servidores fijos. Cada agente funciona como servidor y como cliente.

- Cada agente es asociado con un motor de inferencia, una memoria funcionando y una base de reglas con la cual analiza los mensajes KQML pasados de otros agentes y tener las acciones correspondientes.

• 1.4 CIDVS (Collaborative Injection-mold-product Development Virtual Space) [SHAO2007]

La disminución del tiempo en el que se realizan los productos y el incremento de la complicación de estos requieren que expertos multidisciplinarios cooperen de forma dinámica y cercana en el ciclo de vida de desarrollo del producto. Se presenta un diseño de arquitectura cooperativo utilizando Internet, agentes y tecnologías Web. Con este ambiente cooperativo dinámico, todos los participantes involucrados y su conocimiento distribuido pueden integrarse como un todo para alcanzar un diseño cooperativo más eficiente y más inteligente.

La naturaleza de la colaboración es la adquisición de servicios, lo que permite adquirir los recursos y conocimientos más apropiados que sean escalables, modificables y reutilizables. La metodología propuesta CIDVS (Collaborative Injection-mold-product Development Virtual Space) permite la composición de recursos de manera ágil, flexible y dinámica. Permite su interacción en una variedad de estilos que coincide con las necesidades actuales y cambiantes de los objetivos de diseño. Realiza la integración de agentes y cooperación por medio de Internet para poder tener recursos reutilizables en línea.

Los agentes propuestos para ello están clasificados en tres categorías e ilustrados en la figura 5:

- *Discipline-dependent application agent*: tiene la estructura más simple y provee solamente un mecanismo de comunicación por encima de la capa de encapsulamiento Web.
- *Coordinate-oriented collaboration agent:* no contiene la parte de interfaz de usuario pero tiene más lógica de negocio compleja correspondiente a la encapsulación de un servicio estático.
- *Distributed user agent*: tiene más unidades incluidas, junto con funciones dinámicas de administración de servicios adicionalmente. El agente de usuario es responsable de construir y transferir las peticiones en mensajes SOAP y después re-direccionar al servicio Web apropiado.

Estos agentes sirven para mediar y encapsular los servicios locales y distribuidos.

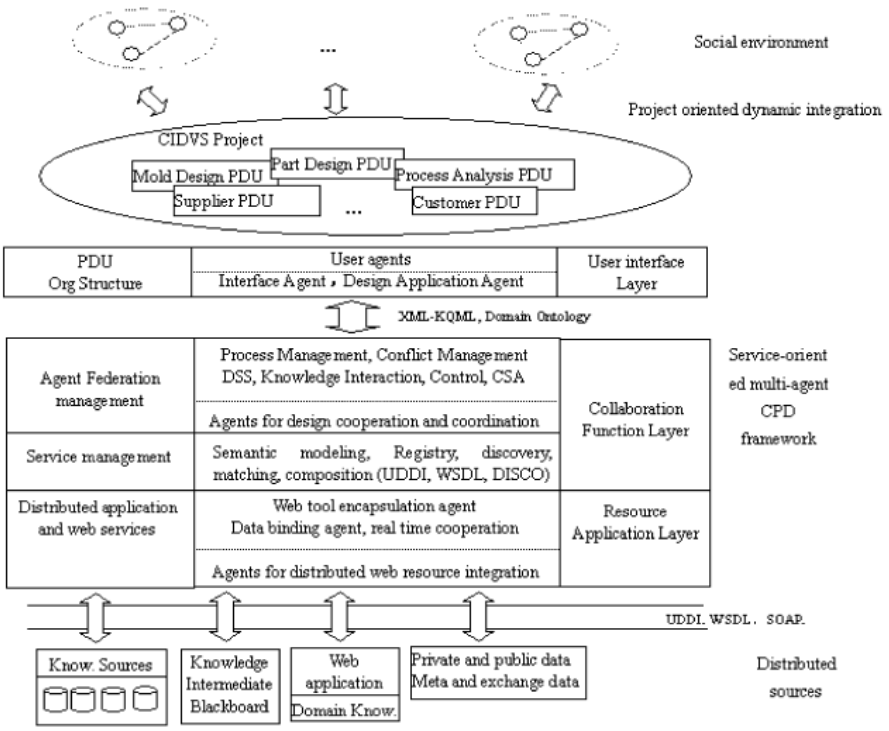

Figura 5 - Arquitectura del diseño cooperativo basado en agentes y servicio por medio de Internet [SHAO2007]

La arquitectura puede verse desde tres capas de despliegue:

1) El *product development unit* layer se compone por varios PDU, el cual contiene un usuario de desarrollo, un agente de usuario, varios agentes de interfaz y agentes de aplicación.

- Los agentes de usuario construyen un ambiente de desarrollo integrado para los PDU con diferentes roles y derechos.
- Los agentes de interfaz: proveen interfaces específicas para los usuarios.

2) El *collaborative function layer* facilita la colaboración entre los diferentes PDU. Las funciones principales están compuestas de dos categorías:
Agent federation: tiene como objetivo la coordinación y el control de los comportamientos de los siguientes grupos de agentes:

- *Comunication service agent:* construye el modelo global del agente de servicio y guarda los registros de las capacidades, nombres, direcciones, y estados de todos los agentes distribuidos.
- Process management agent: tiene procesos orientados al proyecto para la administración, incluyendo definición de procesos, análisis, optimización y alteración.
- *User management agent*: maneja toda la información de los usuarios, roles y autoridades.
- *Knowledge interaction agent:* ayuda a la toma de decisiones de las actividades para obtener el consenso óptimo.

- *Conflict coordination agent:* tiene la función de detectar y resolver conflictos entre los PDU.
- *History agent:* adquiere la información histórica cooperativa.
- *Graphic agent:* controla la presentación y operación gráfica.
- *Service Management:* provee un mecanismo eficiente para integrar los servicios de conocimiento independiente.

3) El *resource application layer* incluye todos los recursos distribuidos desplegados mundialmente que necesitan ser organizados eficientemente por las capas intermedias.

Los servicios se comunican de manera *peer to peer* y utilizan los conceptos de descubrimiento y despliegue para publicar sus capacidades o atributos sin importar su ubicación geográfica. La tecnología de agentes se utiliza para coordinar y encapsular los servicios para asistir el diseño colaborativo en una manera más eficiente. Los agentes actúan como la fuerza conductora para coordinar la utilización del servicio Web de Internet, mientras que los servicios Web existen como aplicaciones de fuentes de conocimiento.

La comunicación entre los diferentes componentes es basado en los protocolos estándares Web. La comunicación entre agentes está basado principalmente en los métodos HTTP GET y HTTP POST o por medio de un subconjunto de KQML. Los contenidos de KQML son encapsulados con el lenguaje XML.

Los servicios Web pueden ser descubiertos dinámicamente e integrarlos en el sistema. Cuando aparece una petición de descubrimiento, la petición y los anuncios registrados son comparados. Un puntaje puede ser obtenido por medio de un algoritmo de emparejamiento. El servicio con el puntaje más alto se considera que es el servicio más deseable para el caso.

En la herramienta propuesta existen dos formas para realizar la invocación de servicios:

- Invocación estática: se invoca el servicio requerido por medio de la información de la configuración inicial.
- Invocación dinámica: se invoca un servicio desconocido por medio de una búsqueda en tiempo real.

La implementación del ambiente se realizó en la plataforma .NET, el servidor Web es en IIS (Internet Information Server). Del lado del cliente para construir los componentes de los agentes se utilizó: applets de Java, controles ActiveX y Javascript.

• 1.5 CoopES [RUFA1990]

La cooperación aparece cuando múltiples procesos que se encuentran separados con diferentes propósitos interactúan para construir una solución a un problema común.

La coordinación en esta aplicación se realiza de dos formas: los agentes pueden ofrecer voluntariamente información o requerirla. Para ofrecer información útil, el agente necesita saber para quien es importante la información que Él maneja. Para requerir información y evitar transmisiones costosas, un agente necesita saber quién tiene el control sobre los recursos que

tienen influencia sobre su propio sistema. La comunicación durante la iniciación cada agente solicita dos sockets sin conexión y envía su dirección IP al servidor. Un socket es utilizado para enviar mensajes y el otro es para recibir.

La interfaz de usuario consulta la red actual para determinar qué máquinas están disponibles y para probar limitaciones en máquinas con agentes. Múltiples agentes pueden correr en todas las máquinas. Cuando un agente se sube, Él envía su dirección al servidor, el servidor después informa a todos en la red la dirección del nuevo agente.

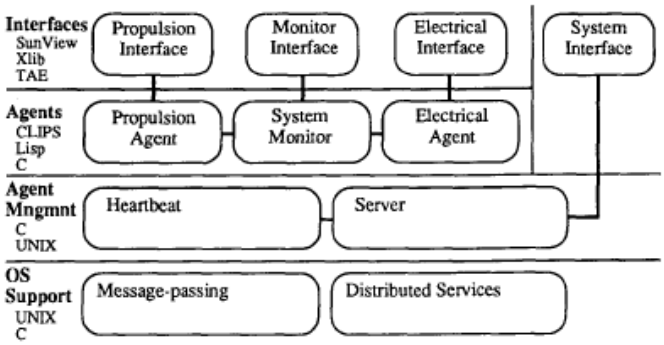

Figura 6 - Prototipo CoopES.

El *heartbeat agent* administra todos los agentes dinámicamente. Como parte de las primitivas de comunicación en cada agente está la habilidad de responder a un mensaje de latido del corazón. Un latido de corazón es enviado cada 10 segundos a cada agente. Cuando un latido de corazón no es devuelto en 30 segundos, el *heartbeat agent* asume que el agente que no responde está desactivado y empieza a buscar un nuevo host en el cual puede reiniciar dicho agente.

El prototipo *CoopES* actualmente utiliza tres agentes de aplicación: un *shuttle propulsión controller, unShuttle electrical system controller y un system monitor* el cual es responsable de coordinar los otros dos agentes. Cada agente tiene una interfaz gráfica de usuario que describe sus estados internos.

2. Solución de problemas por medio de la descomposición

La descomposición de problemas en sub-problemas o problemas más pequeños, permite utilizar diferentes sistemas especializados en diferentes áreas de conocimiento para solucionar problemas particulares. Se realizó un estudio de diferentes sistemas diseñados y/o implementados para la solución de problemas por medio de la descomposición de éstos, los cuales se presentan a continuación.

• 2.1 Frameworks for Cooperation in Distributed Problem Solving [SMIT2001]

Podemos observar el problema de la solución de problemas en un procesamiento distribuido como un grupo de expertos humanos con experiencia trabajando juntos, tratando de comple-

tar una tarea grande. Como lo realizan los expertos humanos, con los sistemas expertos se trata de dividir la carga de trabajo entre ellos y cada nodo independientemente puede resolver algunos sub-problemas del problema grande.

Como se ilustra en la figura 7, en la primera fase el problema se descompone en sub-problemas. La descomposición sigue hasta que se generan problemas que no se pueden descomponer los cuales se llaman problemas atómicos. Los problemas atómicos pueden ser resueltos por unidades individuales. En la segunda fase, se solucionan los sub-problemas. Puede necesitar comunicación y cooperación entre los nodos que intentan resolver los sub-problemas individuales. En la tercera fase, se realiza la integración de los resultados de los sub-problemas para llegar a la solución general del problema.

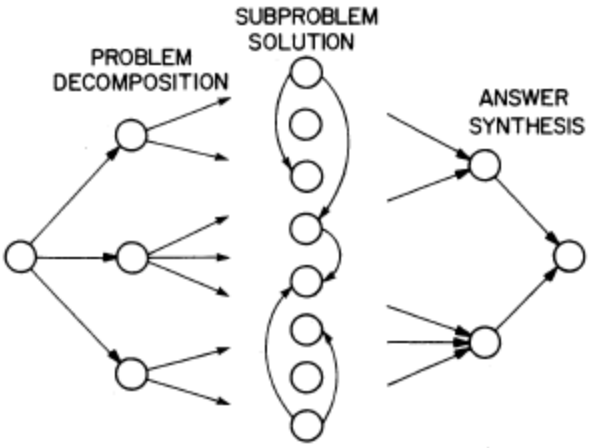

Figura 7 - Fases de la solución de problemas distribuidos [SMIT2001]

El desempeño depende en la arquitectura de solución de problemas.

Dos formas de cooperación en la solución de problemas distribuidos son considerados:

- *Task Sharing:* los expertos deciden quién hace cuál tarea. Si el experto sabe que otro experto tiene los conocimientos necesarios para resolver el problema, él le notifica directamente. Si no sabe quien en particular sabe cómo hacer el trabajo le avisa a todo el grupo sobre la tarea y espera que le responda quién lo sabe hacer. Es una forma de cooperación dirigido a metas en la cual los nodos individuales trabajan independientemente con comunicación mínima entre sí y se asisten entre ellos compartiendo la carga computacional para la ejecución de sub-tareas de un problema en general. La clave a resolverse en el *task-sharing* es como son distribuidas las tareas entre los nodos de procesamiento. El problema de conexión consiste, en encontrar el nodo disponible más apropiado para ejecutar la tarea. Para resolver el problema de conexión se considera la negociación como mecanismo que puede ser usado para estructurar las interacciones de los nodos y resolver el problema de conexión en los sistemas *task-shared.* El protocolo *Contractnet* puede servir como aproximación para resolver el problema de conexión.
- *Result sharing*: los expertos periódicamente reportan entre si los resultados parciales que han obtenido durante la ejecución de las tareas individuales con diferentes perspectivas

del problema en general. Depende de la información que está disponible. La solución es construida mediante la agregación incremental de soluciones parciales.

• 2.2 Organizations for Cooperationg Expert Systems [GROS1990]

Grupos de expertos que colectivamente resuelven los sub-problemas de un problema estructurado cooperan entre sí. Cada sub-problema debe ser escogido para que el sub-problema pueda resolverse de manera independiente.

El planeamiento es el proceso en el que un conjunto de expertos colectivamente selecciona los sub-problemas o metas a cumplir. El protocolo a seguir por los expertos cuando planean será designado para garantizar la coordinación y cooperación entre los expertos. Cada experto escoge la mejor meta posible sin forzar a otro experto a tomar la decisión errada.

La información compartida por los expertos puede ser organizada en el Blackboard con diferentes niveles de abstracción. Cada experto manda una señal cuando puede ejecutar una tarea. Los expertos realizan las tareas oportunamente explotando nueva información que aparece en el Blackboard.

Una organización de expertos para resolver problemas estructurados debe tener en consideración la planeación y ejecución de fases de la solución de problemas. La estructura de coordinación determina el patrón de interacción entre los expertos cuando determinan qué metas se van a lograr. Dicta qué fracción de toda la información disponible para la organización estará disponible para cada experto.

Los expertos pueden interactuar de dos maneras:

- *Peer to Peer*: requiere un consenso entre los expertos durante la fase de selección de meta del planeamiento.
- Jerárquicamente: permite que un experto decrete un conjunto de metas que otro agente debe cumplir.

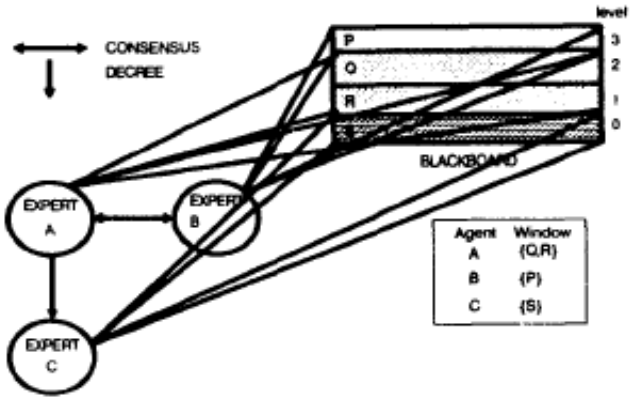

Figura 8 - Estructura de coordinación para una organización con tres agentes [GROS1990]

El protocolo de consenso asegura que los expertos participantes escojan una meta de acuerdo a clasificación conjunta. Cuando un experto no puede completar su parte del plan o se completó el plan, tiene que enviar una señal a los demás expertos. Los expertos mediante el Blackboard intercambian sus planes preferidos de acción. Los expertos examinan los planes propuestos y sus puntuaciones para determinar cuál es el mejor.

El protocolo de decreto le da a un experto la habilidad de dictar a uno o a un grupo de expertos qué metas debe buscar. La estructura organizacional determina las tareas que pueden ser realizadas por los expertos individuales y el flujo de datos entre ellos. Se propone que como parte de la estructura de coordinación una "ventana" o campo de vista debe ser específico para cada experto como se muestra en la figura 8.

La ventana del experto indica qué porción de la información disponible a la organización puede utilizar cuando va a desarrollar las tareas. La asignación de una ventana a un experto fuerza al experto a desarrollar sus tareas con información incompleta. La información disponible para los expertos cuando utilizan el protocolo de consenso incrementará a la ventana combinada de todos los expertos en el grupo.

• 2.3 KAMET [CHUN2007]

Como múltiples expertos pueden tener diferentes conocimientos de un mismo dominio, es necesario obtener e integrar ese conocimiento para construir un sistema experto más efectivo. Como ejemplo se puede tomar el caso en el que varios médicos, cada uno egresado de una universidad diferente pueden saber todos de medicina en general, pero no saben exactamente la misma información. Para hacer frente con este problema, se propone un acercamiento basado en *Delphi* para obtener conocimiento de múltiples expertos.

Es muy difícil obtener e integrar conocimiento de múltiples expertos, en especial en los dominios de aplicación donde varias escalas de tiempo de los elementos tienen que tenerse en cuenta. Para solucionar este problema se propone, KAMET (Knowledge Acquisition for Multiple Experts with Time scales), en el cual se tienen en cuenta las escalas de tiempo mientras se obtiene la experticia de los múltiples expertos.

Cada entrada KAMET es una tripleta que consiste de tres valores:

- Puntaje: grado de relevancia del elemento. El rango es de 1-5.
- Certeza: grado de certeza del puntaje dado. Puede ser S (Seguro) o N (No Seguro).
- Factor de Impacto: grado de importancia del elemento.

KAMET emplea el método *Delphi* en las etapas iníciales de negociación. *Delphi* se define como: un método para realizar solicitación sistemática y colección para juzgamientos en un particular tema por medio de un conjunto de cuestionarios entremezclados con información resumida y retroalimentación de opiniones entregadas de respuestas anteriores.

Delphi tiene las siguientes tres características: grupos anónimos de interacciones y respuestas, múltiples iteraciones o rondas de cuestionarios, presentación de estadísticas de las respuestas de grupo.

Cuando un número predeterminado de expertos están de acuerdo a participar, el investigador usa múltiples iteraciones o rondas de cuestionarios para recolectar la información:

- El cuestionario de la primera ronda utiliza un formato amplio para obtener los juicios individuales u opiniones de cada miembro del panel acerca de un asunto particular o problema bajo estudio.
- En el cuestionario de la segunda ronda, el investigador requiere que el panel de expertos considere puntuar, calificar, editar y comentar acerca de las respuestas dadas en la primera ronda.
- La meta de la tercera ronda o cualquier otra ronda de cuestionarios es de obtener un consenso o estabilidad en las respuestas del panel de miembros.
- Cuando se consigue la estabilidad o consenso, el procedimiento *Delphi* se completa.

Cuando se desarrolla un sistema experto, una de las más difíciles tareas es recolectar los dominios de conocimiento de múltiples expertos. Por eso se necesita una tecnología de integración.

KAMET consiste en 4 componentes principales:

- Módulo de obtención de conocimiento: es una herramienta de obtención de conocimiento basada en Web con la aproximación KAMET, la cual provee facilidades de recuperar, mantener, crear y almacenar el conocimiento.

- Módulo de transformación de conocimiento: puede transformar experticia en un dominio en un formato de conocimiento basado reglas de algún motor de inferencia.

- Módulo de toma de decisiones en grupo basado en *Delphi*: es implementado basado en la tecnología *Delphi*. Emplea procedimientos para asistir a múltiples expertos en determinar elementos, síntomas y el número de escalas de tiempo para cada elemento durante el proceso de adquisición del conocimiento.

- Módulo de integración de experticia: emplea un conjunto de reglas de integración de conocimiento para integrar los puntajes, los grados de certeza y los grados de importancia correspondiente a los elementos y los síntomas.

Después de que todos los expertos proveen los elementos de una aplicación de dominio especifica, KAMET genera una unión por medio de la integración de los conjuntos de elementos obtenidos por los expertos individuales y removiendo los elementos redundantes. KAMET después emplea la tecnología *Delphi* para analizar cada opinión y comentario de cada experto. KAMET resume las respuestas dadas por los expertos en la primera ronda y muestra el análisis de los resultados a los expertos antes de empezar la segunda ronda.

3. Solución de problemas por medio de fuentes de conocimientos separadas físicamente

En muchos casos las fuentes de conocimiento que se utilizan para solucionar algunos problemas en particular, se pueden encontrar geográficamente separados. La idea de solucionar un

problema utilizando diferentes fuentes de conocimiento que están remotamente separadas puede ser muy útil a la hora de combinar diferentes áreas de conocimiento. A continuación se presenta una herramienta que soluciona problemas por medio de fuentes de conocimientos separadas físicamente.

• 3.1 CPS-ID [BASK1989]

El diseño integrado representa un acercamiento para producir diseños que tienen en cuenta requerimientos para un producto. Tiene la ventaja de ser un medio más unificado para la comunicación y cooperación.

El solucionador de problemas cooperativos provee un marco natural para el desarrollo de diseño integrado. El CPS puede ser caracterizado como la solución cooperativa de problemas por un conjunto de unidades solucionadoras de problemas que están lógicamente o físicamente separadas llamadas fuentes de conocimiento. Es diferente al procesamiento distribuido. Puede haber múltiples fuentes de conocimiento que tienen cierta experticia para resolver la mayoría de sub-problemas.

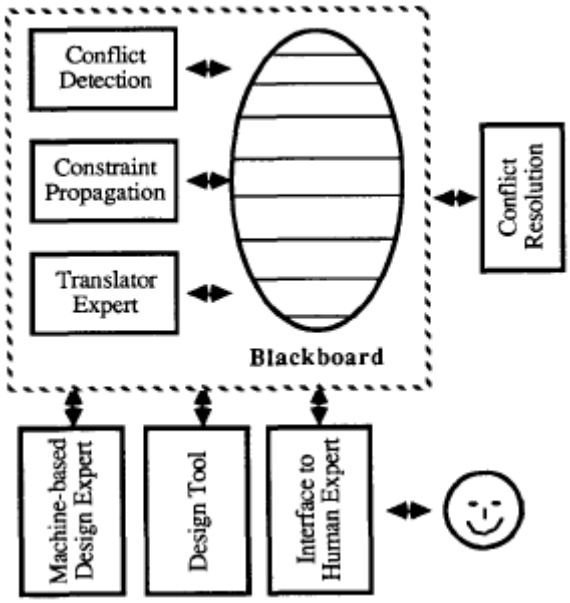

Figura 9 - Arquitectura de CPS-ID [BASK1989]

Las características del Modelo Solucionador de Problemas Cooperativo son:

- El uso de múltiples fuentes de conocimiento cooperativas con diferentes áreas de conocimiento. Cada fuente de conocimiento es independiente y puede estar instalada en diferentes máquinas, se comunican con otras fuentes de conocimiento en un lenguaje común y entendible. El uso de múltiples, separadas, identificables fuentes de conocimiento en la solución de problemas cooperativos hace el desarrollo y la validación más fácil.

- La integración se lleva a cabo de la siguiente manera: las fuentes de conocimiento interactúan modificando y evolucionando una solución guardada en el Blackboard. Cada fuente de conocimiento afirma y propaga las limitaciones de la solución. Los conflictos pueden surgir cuando hay diferentes puntos de vista entre los diferentes solucionadores de problemas.

Las características del diseño serial y el diseño paralelo son:

- Diseño serial:
    - Estaciones sucesivas.
    - Las metas son dadas por el estado anterior de los procesos.
    - Solamente un experto está activado al tiempo.
    - La calidad de integración de múltiples expertos está limitado.
- Paralelismo:
    - Múltiples fuentes de conocimiento activadas en paralelo.
    - Comunicación directa.
    - La calidad de la solución integrada es mejor que la del método secuencial.
    - Las tareas asignadas a los expertos pueden ser modificadas sin requerir mayor interacción.

El CPS-ID está basado en una comunidad de arquitectura CPS. Consiste en la colección de fuentes de conocimiento que se comunican por medio de poner afirmaciones en un *Blackboard central*. Las fuentes de conocimiento al ser implementadas son cajas negras unas con otras. La arquitectura del sistema se ilustra en la figura 9.

Después de haber expresado los puntos de vista de los autores citados anteriormente, se ha identificado una serie de oportunidades las cuales se toman a consideración en el capítulo siguiente a partir de un análisis comparativo sobre los artículos estudiados. Las oportunidades a ser detectadas serán de gran utilidad para generar los requerimientos que harán parte del sistema.

III – ANALISIS INTEGRAL COMPARATIVO

A partir de la investigación realizada al estado del arte se generó un análisis integral comparativo con las problemáticas detectadas analizando sus posibles soluciones con sus ventajas y desventajas. Mediante el análisis y las comparaciones se generan una serie de oportunidades que sirven como base para identificar los requerimientos para el sistema propuesto.

1. Problemáticas

Para la explicación de las problemáticas primero se dividieron en temas generales y dentro de cada tema se explican las diferentes problemáticas obtenidas a partir del estado del arte.

1.1 Coordinación de Tareas

Uno de los problemas más importantes que se trató fue el de la coordinación de tareas. La coordinación de tareas permite tener un mecanismo optimo para la selección del sistema o solucionador del problema más eficiente para la realización de diagnósticos.

- **Mecanismos de Negociación**

Esta problemática apareció en los artículos: [GROS1990], [SHECK1989] y [SMIT2001]. Los tres autores utilizaron el mecanismo de negociación *Contract net* para solucionar el problema.

- **Forma de Reclutar o Seleccionar el Especialista**

Esta problemática apareció en los artículos: [HUNG2005], [SHEK1989], [SMIT2001], [HUNG2005], [SHAA2004], [SHAO2007], [CHUN2007], [GROS1990], [BASK1989]. Las soluciones posibles son: [SHAA2004] propone utilizar un coordinador o médico general. La ventaja de utilizar un coordinador es que no hay necesidad de preguntarles a todos acerca del problema; el coordinador sabe a quién preguntarle.

[SHEK1989] propone utilizar páginas amarillas. La ventaja de utilizar páginas amarillas es que la información de los especialistas y sus capacidades están almacenadas ordenadamente. La correcta organización de los especialistas facilita y optimiza la búsqueda.

[CHUN2007] y [SMIT2001] les pregunta a todos los especialistas. La ventaja de preguntarle a todos es que cada experto tiene la opción de responder si es capaz o no es capaz de realizar la tarea que se le está designando. La desventaja es que una petición le puede llegar a un experto que no tiene idea de resolverla, lo que genera un trafico y consumo de recursos.

[GROS1990], [BASK1989] y [HUNG2005] utilizan Blackboard. La ventaja de utilizar Blackboard es que los expertos tienen dónde poner sus solicitudes si no saben a quién pregun-

tarle. La desventaja es que puede pasar un lapso de tiempo largo antes de que le den respuesta a su solicitud o inclusive esperar para siempre.

[SHAO2007] realiza una invocación estática o invocación dinámica de servicios Web. La ventaja es que cuando realiza una invocación dinámica no se necesita saber quién sabe hacer alguna tarea a la hora de buscar algún especialista.

1.2 Realización de Diagnósticos

El sistema que se diseñó e implementó, tiene como principal función la realización de diagnósticos. Obtener de la mejor manera un diagnóstico, permite darle un grado de confianza a futuros diagnósticos. Para eso se estudiaron diferentes factores acerca de los diagnósticos que se explican a continuación.

- **Medida de la Calidad de los Diagnósticos (Grado de Certeza)**

Esta problemática apareció en los artículos: [CHUN2007], [GROS1990]. Las posibles soluciones son: [CHUN2007] Utiliza una tripleta que consiste de tres valores: puntaje, certeza y factor de impacto. La ventaja de utilizar este sistema de puntuación es que se toman en cuenta tres factores a la hora de decidir cuál resultado escoger. La desventaja es que el experto debe tener disponible la opción de ofrecer los tres factores de puntuación a la hora de entregar el diagnóstico. Lo que significa es que si no los ofrece no puede ofrecer la puntuación correctamente.

[GROS1990] - Los expertos intercambian por medio del Blackboard los planes preferidos de acción acompañado de un puntaje de cada plan. La ventaja es que cada experto da a conocer su metodología para resolver el problema y un grado de certeza con la metodología propuesta.

- **Explicación de cómo se Obtiene la Solución**

Esta problemática apareció en los artículos: [SHAA2004], [SHEK1989] y [HUNG2005]. Los tres utilizan un agente que se encarga de generar un flujo de información con la recopilación de los pasos que se siguieron para llegar a un diagnóstico. La ventaja es que se tiene la manera de justificar los resultados que arrojó el experto. Es muy útil en casos en los que la herramienta es para asistir a la toma de decisiones.

- **Cuando Buscar Ayuda (Cooperación Entre Especialistas)**

Esta problemática apareció en los artículos: [SHEK1989] y [HUNG2005]. Las soluciones posibles son: [SHEK1989] propone que cuando el soporte para una solución es menor que cierto umbral, el sistema experto decide buscar ayuda. La ventaja es que se maneja un umbral con un mínimo puntaje con el cual se puede trabajar.

HUNG2005 propone que un *proof agent* debe pedir ayuda cuando está sobre cargado. La ventaja es que el agente tiene la capacidad de pedir ayuda cuando él lo considere pertinente.

- **Organización del Conocimiento de los Expertos (Múltiples Especialistas con Conocimiento similar)**

Esta problemática apareció en el artículo de [SHEK1989] y proponen utilizar notaciones de domino y conceptos. La ventaja es que al momento de localizar algún experto se puede utilizar notaciones, lo que facilita la búsqueda.

1.3 Fallos

Para obtener un sistema confiable y que siempre este disponible para los usuarios, es necesario tener mecanismos que permitan mantener los diferentes componentes del sistema funcionando en todo momento. A continuación se presentan diferentes factores que le permiten al sistema presentar sus diferentes funcionalidades a prueba de fallos.

- **Tolerancia a Fallos (Monitoreo)**

Esta problemática apareció en los artículos: [HUNG2005], [SHEK1989] y [SMIT2001]. Los tres autores proponen colocar un agente que se encargue de monitorear los agentes que están realizando las tareas. La ventaja de tener un agente que este monitoreando los procesos le da al sistema un grado más de fiabilidad.

- **Concepto de Excepción**

Esta problemática apareció en el artículo [SHEK198] y la solución que se le da a este problema es asignarle a cada agente un *mecanismo worm* el cual permite al agente ser tolerante a fallos. El *mecanismo worm* es un componente que hace parte del agente, el cual le permite monitorear los servicios que éste presta, de esta manera el agente puede saber cuando esta fallando un servicio.

1.4 Historial

Mantener información acerca de los sujetos que han sido diagnosticados por el sistema, permite que no se tenga que volver ingresar información que ya pudo haber sido captada con anterioridad. A continuación se presentan dos temas importantes para el concepto de historial.

- **Recuperación de los Diagnósticos**

Esta problemática apareció en el artículo [SHAO2007] y la solución que se propone para el problema es utilizar un *history agent*. La ventaja de implementar un agente de historial es que el sistema adquiere toda la historia de cooperación cuando se necesite.

- **Transferencia de la Información Ya Captada**

Esta problemática apareció en el artículo [HUNG2005] y propone que los agentes que aceptan ayudar se contacten con el que pidió la ayuda y hagan la transferencia de la información ya captada. La ventaja de transferir la información captada es que el agente nuevo no tenga que captar de nuevo la información.

1.5 Interfaz de Usuario

Permitir que los usuarios tengan una mejor interacción con el sistema, ayuda a que puedan ingresar información de manera más correcta al sistema y obtener de manera mas entendible la información de respuesta que éste le brinda. Para lo mencionado anteriormente, se presenta el concepto de interfaz de usuario.

- **Interacción del Usuario con el Sistema (Interfaz de Usuario) (Forma de responder)**

Esta problemática apareció en los artículos: [QIU2005], [SHAA2004], [HUNG2005], [CHUN2008] y [RUFA1990]. Las soluciones posibles son: [QIU2005], [SHAA2004], [CHUN2008] proponen utilizar como interfaz gráfica de usuario un explorador Web. La ventaja de utilizar un explorador Web como medio para utilizar el sistema es que permite a los clientes acceder a éste en cualquier parte del mundo y en una gran variedad de dispositivos. Este tipo de aplicaciones permite que un mayor número de personas puedan utilizarla sin importar el dispositivo o el lugar donde se encuentre.

[HUNG2005] y [RUFA1990] proponen utilizar como interfaz de usuario una aplicación *stand alone*. Las aplicaciones *stand alone* se utilizan en los casos en donde los usuarios van a ser personas específicas y no se pretende un gran numero de dispositivos diferentes para acceder-la.

1.6 Mecanismos de Comunicación

Los diferentes componentes que hacen parte del sistema necesitan mecanismos eficientes para la comunicación entre ellos, bases de datos y/o con los usuarios que los utilizarán. Dos mecanismos de comunicación que fueron estudiados se explican a continuación.

- **Necesidad de tener un Lenguaje Común de Conocimiento**

Esta problemática apareció en los artículos: [HUNG2005], [SHAA2004], [SHAO200]7. Los tres autores proponen utilizar KQML como lenguaje de comunicación común. [SHAO2007] además de utilizar el lenguaje KQML utiliza XML para encapsularlo cuando envía los mensajes a través de redes de Internet. La ventaja que se obtiene al implementar este lenguaje de comunicación común es que es ampliamente utilizado en aplicaciones de esta categoría.

- **Ontología**

Esta problemática apareció en los artículos: [SHAA2004] y [SHAO2007]. Las soluciones posibles son: en [SHAA2004] los agentes utilizan la misma ontología y entienden el mismo lenguaje de comunicación. La desventaja es que hay que instalar las ontologías en todos los agentes y al momento de realizar las actualizaciones se tendrían que actualizar todas.

En [SHAO2007] se utiliza un repositorio que guarda la información de los servicios de ontología. La ventaja de usar un repositorio es que sólo hay una fuente con los servicios de ontología y para la actualización e instalación es mejor. La desventaja es que si deja de funcionar el repositorio, todos los agentes pierden la capacidad de utilizar las ontologías.

2. Oportunidades

Después de haber realizado todo el análisis a las problemáticas encontradas en el estado del arte se generaron una serie de oportunidades que sirven como requerimientos a la hora de hacer la implementación del sistema. Las oportunidades son mostradas a continuación:

- Permitir tener un historial con dos niveles, historial actual e historial a largo plazo.
- Posibilidad de poner a disposición la información ya captada en la base de datos a los sistemas expertos.
- Permitir tener la opción de integrar varios historiales en uno sólo para poner a disposición de los consultores.
- Posibilidad de permitirle una calificación a diferentes elementos que componen el sistema para tener un grado de confianza sobre estos.
- Posibilidad de tener en cuenta el valor de confianza al momento de calificar una oferta para los agentes coordinadores.
- Posibilidad de manejar la consistencia de términos en una plataforma distribuida.
- Permitir al sistema interacción en tiempo real con los usuarios en los casos que se necesite retroalimentación de algunas preguntas realizadas por los expertos.
- Posibilidad de tener detección de fallos y mecanismos para la recuperación de componentes del sistema en el caso de que fallen.
- Permitirle al usuario coordinador seleccionar la respuesta más adecuada a partir de las enviadas por cada experto individual.
- Posibilidad de filtrar a cuál especialista se le manda la información sin necesidad de mandársela a todos.
- Posibilidad de manejar un directorio de páginas amarillas con más criterios de búsqueda.
- Poder tener la posibilidad de registrar información más detallada al momento de registrar a un especialista en las páginas amarillas. Esto permite tener más información en el momento de seleccionar los especialistas.
- Permitir que los agentes puedan decidir si son capaces de realizar una tarea o no, confrontando con un filtro inicial sin la necesidad de realizar la inferencia del Sistema Experto.
- Posibilidad de tener múltiples instancias de un mismo agente por medio de un Pool de instancias o la posibilidad de tener instancias creadas dinámicamente. Un Pool de instancias permite tener objetos ya inicializados en el sistema, lo cual permite que se puedan acceder en el momento que se necesiten.
- Posibilidad de permitir a los especialistas que puedan tomar la iniciativa de decidir cuándo transferir la información ya captada.
- Permitir la recolección de las soluciones encontradas y presentarlas de una forma ordenada y coherente.
- Posibilidad de tener dos clases de cooperación entre especialistas: una vertical y otra horizontal. La cooperación vertical consiste en la cooperación con los agentes que iniciaron el diagnostico. La cooperación horizontal permite la cooperación entre los agentes de diagnostico.
- Posibilidad que el usuario fácilmente conozca los servicios que presta el sistema por medio de servicios Web.

- Permitir que el sistema lo puedan utilizar personas con un mínimo de conocimiento técnico sobre la herramienta.
- Dividir una tarea principal en sub-tareas para que la puedan resolver independientemente los especialistas.
- Permitir que sea un humano experto interactuando con el sistema desde una interfaz como si estuviera representando a un agente de diagnóstico, esto permite ampliar las entradas de conocimiento.

3. Conclusiones Generales del Estado del Arte

A partir del análisis realizado al estado del arte y las oportunidades obtenidas se realiza unas conclusiones sobre los temas más importantes que se investigaron. Las conclusiones a partir de los temas son:

- Coordinación de Tareas: el requerimiento quizás más importante en el concepto de la cooperación, es la forma como se selecciona al especialista, debido a que la mejor forma implementada para la selección garantiza un buen y más rápido diagnóstico por parte del sistema. Una de las formas útiles para la coordinación de tareas es la división de éstas, lo que nos permite tomar un problema grande y dividirlo en sub-problemas que pueden ser solucionados por agentes independientes sin necesidad de comunicarse entre sí. Otra manera de coordinar tareas es utilizar múltiples especialistas con el mismo o similar conocimiento, lo que le permite al coordinador tener segundas opiniones sobre un mismo diagnóstico.

- Realización de Diagnósticos: la explicación de cómo se llegó a una solución sirve como referencia para justificar una respuesta de un experto. Esto es de gran utilidad para los consultores que utilizará la herramienta. La medida de calidad de los diagnósticos permite que se tenga conocimiento de la calidad de las respuestas de ciertos expertos antes de realizar la consulta. Esto le agrega un peso a la respuesta cuando se quiere hacer una selección entre muchas.

- Fallos: el tema de tolerancia de fallos juega un papel importante en el momento de implementar la aplicación, se utiliza para asegurar su correcto funcionamiento a lo largo del tiempo. Manejar un concepto de excepción nos permite tener una especie de monitoreo sobre los agentes que están corriendo en el momento, permitiendo saber sus estados actuales y para efectuar medidas. En caso de que un agente falle, las medidas serian reiniciar el agente caído y recuperar su estado en el momento antes de fallar.

- Historial: la recuperación de los diagnósticos realizados anteriormente es un punto crítico en la aplicación a desarrollar, debido a que por medio de éste se logra evitar nuevos diagnósticos innecesarios. Transferir la información ya captada permite a los expertos evitar realizar el mismo diagnóstico y tener información base para dar respuestas.

- Interfaz de Usuario: la interfaz de usuario le permite al usuario poder interactuar de una manera más fácil, práctica y eficiente utilizando todo el potencial y las características de la herramienta. La forma de responder permite dar una cantidad de tiempo para la respuesta que le da el sistema al usuario que la requiere. La interacción del usuario con el sistema es importante debido a que el sistema puede realizar preguntas al usuario en tiempo real para completar su diagnóstico.

- Mecanismos de Comunicación: la comunicación entre los agentes que hacen parte del sistema es un punto vital en el desarrollo de la aplicación. Tener un lenguaje de comunicación común permite a los agentes comunicarse en el mismo lenguaje de una manera transparente. Las ontologías permiten a los expertos manejar los mismos términos sin importar las palabras que se utilizan facilitando el entendimiento entre estos. En el caso en el que los agentes no hablen en el mismo lenguaje, es necesario el componente de traductor. El componente de traductor facilita la comunicación entre ellos, esto permite que los expertos puedan tener lenguajes de comunicación diferente, permitiendo que no hayan restricciones de entrada para los diferentes expertos.

- Mecanismos de Negociación: los expertos pueden tener la necesidad de buscar ayuda, para eso se debe brindar un mecanismo para que éste pueda localizar la ayuda pertinente en su caso. Los mecanismos de negociación son útiles también en el momento de que los agentes requieren hacer distribución de tareas. Como en el caso de *Contract-Net*[SMIT2001].

- Especialista Humano: la posibilidad de tener una interfaz humana le permite al sistema que un especialista humano pueda trabajar como si fuera un sistema experto realizando diagnósticos. De esta manera el especialista humano aporta su experticia a la plataforma incrementando el área de conocimiento de la plataforma en general.

- Escalabilidad del Sistema: la necesidad de agregar dinámicamente agentes o especialistas a la federación permite al sistema ser más escalable con la posibilidad de manejar nuevos dominios de conocimiento y nuevos componentes de computo.

- Atención de Múltiples Usuarios: la necesidad de atender múltiples usuarios al tiempo nos permite utilizar los conceptos de paralelismo y así poder atender las peticiones de los usuarios más rápido y aprovechar los recursos con que se cuenta.

A partir del análisis del estado del arte se generaron las oportunidades y conclusiones que pueden ayudarnos en el camino hacia donde queremos dirigir nuestra herramienta. Dichas oportunidades nos brindan la base para la obtención de los requerimientos más importantes de la aplicación que queremos implementar. Con toda la información recopilada anteriormente se le da paso a la creación del modelo de negocio. Lo que nos permite definir a nivel general lo que se pretende con el sistema.

IV – MODELO DE NEGOCIO FEDEXPERT

El análisis y el estudio al estado del arte arrojaron una serie de preguntas y oportunidades útiles a la hora de buscarle un rumbo al desarrollo de la aplicación. Las conclusiones al estado del arte nos permitieron desarrollar las principales características que hacen parte del modelo de negocio de la aplicación.

1. Introducción

El sistema va a realizar consultas tipo diagnóstico. El diagnóstico es realizado por un usuario consultor que a través de una interfaz gráfica puede ingresarle los síntomas de un sujeto a diagnosticar al sistema. Un síntoma es una característica de alguna cosa que tiene el sujeto a diagnosticar. El sistema le va a contestar ya sea con preguntas o con el resultado de diagnóstico. El sistema se compone de la integración de múltiples especialistas con diferentes áreas de conocimiento que conforman una federación que es la que va a ofrecer los servicios. Un especialista puede ser un especialista humano o un especialista artificial, lo cual permite ofrecer un campo más amplio de conocimiento para la realización de los diagnósticos. La principal finalidad del sistema es la de facilitar a los usuarios que la utilizaran, la realización de diagnósticos, utilizando de por medio diferentes dispositivos de cómputo.

1.1 Usuarios

Los usuarios consultores del sistema propuesto pueden ser: cualquier tipo de persona que posea el conocimiento para obtener la información de los usuarios de la mejor manera en el área de conocimiento que se esté trabajando y que posean el conocimiento técnico para insertar la información obtenida al sistema. Lo importante es que los usuarios consultores que van a utilizar el sistema sean capaces de obtener la información de una manera correcta de los sujetos a diagnosticar y que tengan todo el conocimiento técnico adecuado para insertar de mejor manera dichos síntomas al sistema y así realizar el mejor diagnóstico. De esta manera se logra con el sistema ampliar el número de personas que se puedan beneficiar para realizar los diagnósticos a los interesados.

El sistema va a permitir a los usuarios consultores que estén geográficamente distribuidos acceder a los servicios de diagnóstico, sin importar el medio que utilicen los usuarios para acceder al sistema los resultados que se arrojarán serán los mismos, si se ingresan de manera correcta los síntomas obtenidos de los sujetos a diagnosticar. De esta forma se logra conectar a los usuarios con las diferentes fuentes de conocimiento. El objetivo es generar un diagnóstico más eficiente y más cooperativo. El sistema le brindaría a los usuarios que estén capacitados para utilizarlo un diagnóstico, una ayuda sobre un problema de un área que no conocen, un apoyo para la toma de una decisión o tener una segunda opinión en el caso que no confíen en su propio criterio. En vez de llamar a otra persona, se utiliza el sistema el cual le ayudaría a darle más peso a una decisión.

1.2 Especialistas

Se tienen varios agentes de diagnóstico, cada uno de ellos está especializado en un área de conocimiento similar pero diferente; estos son los que brindan las fuentes de conocimiento al sistema. Existen dos tipos de agentes de diagnóstico, pueden incluir sistemas expertos artificiales basados en reglas o pueden incluir un apoyo de un experto humano que es una persona que puede ser un experto del sistema pero simulado por un humano.

Todas las funciones que hace un agente del sistema las tiene que ser capaz de hacer los expertos humanos, es decir, que tiene que ser capaz de responder a los protocolos establecidos para los agentes. La idea es que ambos especialistas sean iguales, las preguntas que se le hagan a uno, el otro tiene que ser capaz de responderlas también. Así como el agente artificial tiene acceso al historial de los sujetos a diagnosticar el especialista humano también tiene acceso y puede consultar la información de una manera fácil y eficiente.

1.3 Modelo de Inspiración

La inspiración del sistema surge a partir del sistema médico que se utiliza actualmente. El caso en el que un paciente primero acude a un Médico General el cual le realiza un diagnóstico general. A partir del diagnóstico que realizó el Médico General, decide si puede resolver o no el problema; si puede, el Médico General resuelve el problema del paciente, si no, el remite al paciente a un médico especialista que tiene más conocimiento para resolver el problema en particular. En el caso del sistema propuesto, el sujeto a diagnosticar acude a un usuario consultor el cual hace el rol de médico general, éste realiza la obtención de los síntomas y los ingresa al sistema, el cual se encargará de remitirlo a un especialista apropiado para generar la solución al problema.

2. Sesión

Una sesión es un periodo de tiempo de actividad que un usuario pasa en el sistema cuando está conectado. Existe un concepto genérico de sesión el cual es un usuario normal y tiene unas funciones básicas que comparten todos los usuarios del sistema. Un usuario puede tener sólo una sesión registrada y sólo puede tener una sesión activa en el momento, la de su último inicio de sesión. La interfaz de inicio de sesión es igual para todos los usuarios del sistema.

En la pantalla de inicio de sesión aparecen dos campos en blanco uno para ingresar el nombre de usuario y uno para ingresar la contraseña asociada a ese nombre de usuario. Una vez ingresa los datos requeridos, éste debe hacer clic en el botón de inicio de sesión. Los dos datos que fueron ingresados por el usuario serán comparados con la información que está guardada en la base de datos; una vez que los datos son validados se autoriza el inicio de sesión del usuario. Cuando un usuario inicia sesión con el sistema tiene unos derechos con unos servicios describiendo lo que puede hacer en el sistema mientras mantiene la sesión de usuario activa.

Los usuarios pueden terminar las sesiones en el momento que lo deseen. También existe un tiempo de ausencia en el cual el sistema detecta un intervalo de tiempo establecido, lo que

genera la finalización de la sesión de usuario. Se definen tres roles que son los diferentes usuarios del sistema los cuales son: usuario consultor, usuario especialista humano y administrador del sistema. Según el rol con que se inicia la sesión se les asignan unos privilegios, servicios y obligaciones a los usuarios.

Los usuarios del sistema también pueden dormir una sesión. Lo que significa que en el caso de que no hayan terminado todas sus consultas pueden congelar el estado actual de sus actividades para cuando inicien su sesión nuevamente.

2.1 Usuario Consultor

Cuando un usuario consultor se conecta e inicia sesión puede iniciar una o varias consultas a los sujetos a diagnosticar que tengan encargados. El usuario consultor también puede revisar las consultas activas que dejó pendientes cuando realizó su último inicio de sesión y decidir cuál continuar. Cuando un usuario consultor termina sus consultas o simplemente quiera terminar la sesión lo puede hacer para que otras personas mal intencionadas utilicen sus cosas. Las personas mal intencionadas pueden utilizar la información de los pacientes, información de consultas, entre otras cosas para su beneficio propio. Si un usuario consultor tiene abierta una sesión en una máquina y después hace inicio de sesión en otro dispositivo, la sesión en la máquina anterior se cierra permitiendo abrir la misma sesión en el nuevo dispositivo guardando los cambios sin perder el estado de las consultas que tenía anteriormente. Esto permite que un usuario consultor pueda acceder a su sesión desde cualquier dispositivo en cualquier momento.

2.2 Usuario Especialista Humano

Cuando un especialista humano inicia sesión, éste tiene la capacidad de funcionar como un experto artificial especializado en un área en particular simulando de esta manera las funciones que realiza un agente artificial basado en reglas. Por medio de la interfaz gráfica de usuario, el sistema le hace preguntas y este especialista humano se encarga de responder. En la interfaz para esté usuario se le ofrecen facilidades para que el usuario pueda interactuar con el sujeto a diagnosticar como si fuera un agente más del sistema.

Un especialista humano si tiene los privilegios necesarios o los conocimientos básicos puede hacer inicio de sesión como usuario consultor o como agente especialista humano. En el momento de iniciar la sesión se le presenta la opción para escoger con que rol va a ingresar al sistema. Cuando se presenta este caso sólo puede escoger un rol al mismo tiempo; si desea cambiar de rol tiene que volver a iniciar sesión y escoger el otro rol.

2.3 Usuario Administrador del Sistema

Cuando se inicia la sesión de usuario con el rol de administrador del sistema, este usuario tiene las siguientes obligaciones: crear y administrar las cuentas de los usuarios consultores y los usuarios especialista humano, mantener siempre el sistema en línea, administrar y configurar los servicios del sistema que están habilitados, agregar y administrar las áreas de conocimiento, agregar y administrar los agentes que hacen parte de la federación.

El administrador del sistema cuenta con una herramienta para administrar los servicios del sistema. Los servicios del sistema son los diferentes programas con los que cuenta el sistema los cuáles van a ser usados por los diferentes usuarios. El administrador es el encargado de seleccionar cuales de estos deben estar funcionando, que estén funcionando y mantenerlos siempre actualizados.

El administrador del sistema también cuenta con acceso a la base de datos de los usuarios del sistema, dicha base de datos contiene la información de las sesiones de los usuarios consultores y los usuarios especialistas humanos. La información que se guarda en esta base de datos tiene un número de identificación, nombre de usuario, contraseña, privilegios que tiene e la información acerca de sus últimos movimientos en sesiones anteriores.

3. Consulta

Una consulta es el efecto de solicitar un diagnóstico al usuario consultor. La consulta se inicia en el momento en el que el sujeto a diagnosticar llega a donde el usuario consultor, el usuario consultor ingresa la información obtenida del sujeto a diagnosticar al sistema, el sistema le entrega al usuario consultor el resultado de los diagnósticos y la consulta termina cuando el usuario consultor le entrega el resultado del diagnóstico al sujeto. En otros casos también puede ocurrir que exista una frecuente interacción entre el sistema y el sujeto a diagnosticar cuando se requiere retroalimentación del sujeto de alguna información.

3.1 Historia

Cuando un usuario consultor inicia una consulta lo primero que se realiza es verificar si el sujeto a diagnosticar que llegó tiene una historia. Una historia es la información que se tiene históricamente de un sujeto a diagnosticar, a lo largo de todas las consultas que se le ha hecho. El sistema se encarga de revisar en la base de datos y si no encuentra, le avisa al usuario consultor. Si el sujeto a diagnosticar no tiene historia, el usuario consultor tiene la obligación de llenar por primera vez la historia de este sujeto al menos con sus datos generales. Los datos generales son aquellos que ayudan a la descripción e información básica de un sujeto a diagnosticar. Como ejemplo podemos utilizar en el caso de un carro: la información que se encuentra en la tarjeta de propiedad y en el caso de una persona: los datos que se encuentran en el registro civil. Cuando llena los datos el sistema se encarga de ingresar la historia a la base de datos para utilizarla en futuras consultas.

Si el sujeto a diagnosticar ya contaba con una historia, el sistema le muestra al usuario consultor la información del sujeto a diagnosticar, lo cual es una información que el usuario consultor puede usar para tomar decisiones sobre el resultado. Cada vez que se realiza una consulta a un sujeto a diagnosticar, el usuario consultor se encarga de preguntarle al sujeto por una serie de información que siempre está cambiando a lo largo del tiempo y que puede afectar un resultado. Esta información también se almacena en la historia del sujeto y se tiene como referencia también entre una consulta y otra. Otra información que se utiliza es la que es muy específica de la consulta debido a que sólo aparece en ésta. Una historia es un documento o en este caso, una serie de información que contiene los datos, valoraciones e infor-

maciones de cualquier índole sobre la situación y la evolución de un sujeto a diagnosticar a lo largo de un proceso.

3.2 Consultas Activas

Un usuario consultor puede tener varias consultas abiertas las cuales llamaremos consultas activas. Cada consulta tiene su propio registro el cual lleva la identificación del usuario consultor que la realizó, la información del sujeto a diagnosticar al cual se le hizo la consulta, la información del especialista al cual fue remitido, información de fecha y hora, y los resultados que se obtuvieron de la consulta.

Un usuario consultor puede cambiar de consulta en el momento que lo desee. Puede ser si surge una emergencia y tiene que realizar una consulta con mayor prioridad y puede regresar a la otra consulta cuando termine o cuando lo vea necesario. Dos usuarios consultores pueden realizar una consulta en paralelo que significa que dos usuarios consultores con sesiones diferentes abiertas al mismo tiempo pueden realizar una consulta sobre el mismo sujeto a diagnosticar, cuando dos usuarios consultores realizan la consulta sobre el mismo sujeto se generan registros diferentes generados por sesiones de usuarios consultores separadas.

El sistema permite tener las consultas con un estado estable, debido a que siempre se estará reflejando los datos que se están ingresando al sistema en la base de datos, esto permite que se pueda recuperar la información en los casos de que se cambie la sesión de dispositivo, cuando un usuario consultor cierra la sesión o cuando ocurran fallas en el sistema lo cual requiera reiniciar el sistema o la sesión de usuario.

Con la presentación del modelo de negocio obtenemos una perspectiva general sobre las características y requerimientos principales de la plataforma. Se termina el capítulo 4 lo cual nos permite continuar a realizar el modelo general y el modelo de agentes, los cuales serán descritos en el siguiente capítulo.

V – MODELO FEDEXPERT

A partir del modelo de negocio descrito en el capítulo anterior, se definen las principales características y funcionalidades del sistema. El modelo general describe todas las capas y componentes que hacen partes de la plataforma y las interacciones que hay entre ellas. El modelo de agentes introduce todos los agentes que componen el sistema y describe las metas, el estado, las entradas sensoriales, los actuadores y los comportamientos de los agentes.

1. Modelo General FedExpert

Para el modelo se describieron tres capas, las cuales organizan paquetes de servicios y subsistemas. En esta descripción se pueden observar los diferentes componentes principales y sus relaciones, además se tienen en cuenta detalles técnicos y un principio para la implementación de la plataforma basándose en la lógica del negocio. En la figura 10 podemos observar una descripción general y las interacciones principales del modelo para el sistema FedExpert; adicionalmente, se describen los paquetes relacionados con cada módulo de diseño. El modelo FedExpert se compone de tres capas: presentación, lógica y datos.

1.1 Descripción general

Los componentes que conforman el sistema están organizados en capas. Estas capas se encargan de proporcionarle servicios a sus capas subyacentes una vez lo requieran para llevar a cabo tareas dentro del sistema. La estructuración en capas se puede afrontar por la responsabilidad de cada una de éstas; para este caso se manejan capas por responsabilidades las cuales buscan operar módulos del sistema cuyo trabajo está enfocado a tareas específicas que las diferencian de las demás capas.

En la figura 10 que se presenta el modelo general de capas sobre el cual será diseñado FedExpert y en la figura 11 se presenta la nomenclatura de los objetos utilizados en el modelo general.

1.2 Capas que conforman el Sistema

El modelo basado en capas tiene como objetivo principal la separación de la lógica de negocio de la aplicación con la capa de la presentación al usuario. El desarrollo basado en capas permite distribuir el trabajo en diferentes niveles, permitiendo el trabajo en paralelo. A continuación se presentan las tres capas que hacen parte del modelo.

- **Capa Presentación**

Esta capa se encarga de mostrar todas las funcionalidades, opciones del sistema e interacción con los usuarios. También está encargada de manejar las diferentes sesiones para los usuarios del sistema. La capa de presentación se conforma de los siguientes componentes:

- Modelo: ésta es la representación específica de la información con la cual el sistema opera. La lógica de datos asegura la integridad de estos y permite derivar nuevos datos. Se encarga de establecer la comunicación con la capa de Lógica.
- Vista: este componente presenta el modelo en un formato adecuado para interactuar con el usuario por medio de un navegador Web. Le permite a los usuarios de la plataforma FedExpert tener control a los diferentes servicios que se ofrecen en la capa de la lógica.
- Controlador: este componente es el que está encargado de responder a los eventos, usualmente acciones del usuario que invoca cambios en el modelo y en la vista.
- Sesión: este componente está encargado de manejar las sesiones de los diferentes usuarios. Está encargado también de asignar los privilegios con los que cuentan los usuarios.

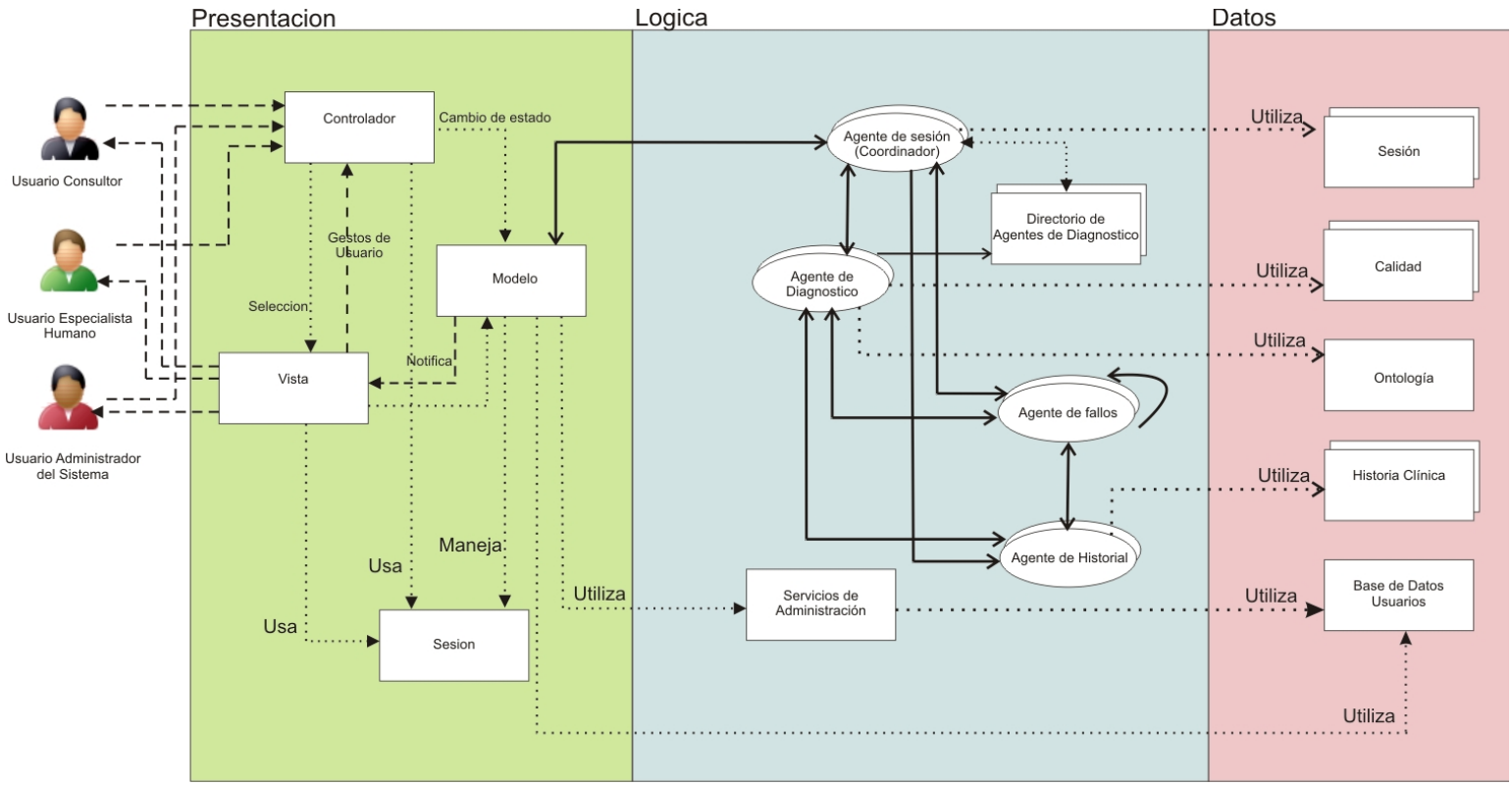

Figura 10 - Descripción general del Modelo

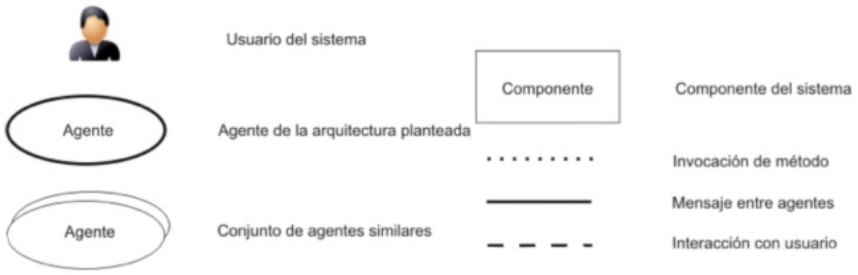

Figura 11 - Nomenclatura de los diagramas.

- ## Capa Lógica

Esta capa es la encargada de toda la lógica de negocio del sistema. La lógica de negocio es la parte del sistema que se encarga de las tareas relacionadas con los procesos del negocio con el que se esté. Es la parte del sistema que involucra a los agentes y realiza todo el procesamiento detrás de la aplicación visible para el usuario. La capa de lógica se conforma de los siguientes componentes:

- Directorio de Agentes de Diagnóstico: contiene la lista de todos los agentes de diagnóstico que están activos y disponibles en el sistema. Cada agente estará acompañado por el área de experticia que Él conoce. El agente de sesión puede utilizar en cualquier momento los agentes que estén en el directorio.
- Servicios de Administración: este componente está encargado de prestar los servicios de administración a los usuarios administradores del sistema. Los servicios con los que cuentan estos usuarios son los siguientes: servicios disponibles, cuentas de usuarios y agentes disponibles.

- ## Capa Datos

Esta capa es la encargada de realizar la persistencia de los datos más importantes del sistema. Debe realizar la comunicación con los diferentes componentes y agentes que deben persistir datos a lo largo de la utilización de los diferentes servicios que se van a utilizar. La capa de datos se conforma de los siguientes componentes:

- Sesión: base de datos que guarda los estados de las diferentes sesiones de usuario que están corriendo en el sistema. Permite realizar recuperación de sesiones en el caso de que ocurra algún fallo inesperado o algún cierre de sesión.
- Calidad: base de datos que permite llevar un registro de las puntuaciones de los diferentes diagnósticos que han realizado los agentes, para poder realizar una mejor selección de los agentes en un diagnóstico futuro.
- Ontología: base de datos que cuenta con las ontologías que el sistema va a utilizar. Permite facilitar la comunicación y permite compartir información entre diferentes sistemas y entidades.

- Historial: base de datos que cuenta con los registros médicos de todos los pacientes que han sido tratados por el sistema.
- Usuarios: base de datos que contiene la información de todos los usuarios del sistema con sus privilegios y los servicios que puede utilizar. La base de datos es útil a la hora de realizar la autenticación de los usuarios en el momento de iniciar una sesión.

2. Modelo de Agentes FedExpert

El modelo de agentes de FedExpert hace la introducción de todos los agentes que hacen parte de la plataforma. A continuación se identifican los agentes y se describen sus principales funciones: metas, estados, entradas sensoriales, actuadores y sus comportamientos.

2.1 Descripción General del Modelo de Agentes

Los diagramas del modelo de agentes están compuestos por los agentes que participan en la arquitectura y el ambiente donde se están manejando. Cada agente tiene su nombre el cual será usado como identificación y tendrá tres aspectos importantes que identifican sus características y comportamientos.

A continuación se muestra el desglose de los agentes actuantes en el sistema mostrando todas sus interacciones y manejando este esquema:

Esquema:

Ai. Nombre del Agente i

a. Meta:

M_{ij}. Meta j del agente i.

b. Estado:

Elemento 1.

c. Entradas Sensoriales:

S_{ix}.Entrada sensorial x del agente i.

d. Actuadores:

A_{iz}. Actuador z del agente i.

e. Comportamientos:

B_{ik}. Comportamiento k del agente i: Cumple con algunas metas.

Mensaje / Evento: nombre del mensaje, descripción del mensaje recibido, enviado por el agente o sensor w.

Acción: descripción de la acción.

Respuesta: mensaje de respuesta a enviar por el agente i, y la descripción del mensaje.

2.2 Agente de Sesión – AgentSesion

• Metas

M_{11}. Registrar y remover los agentes de diagnóstico en el directorio de agentes.

M_{12}. Decidir qué agente de diagnóstico va a realizar el diagnóstico.

M_{13}. Reiniciar un agente de diagnóstico cuando uno falla.

M_{14}. Retornar el resultado de diagnóstico al agente User Interface.

M_{15}. Crear el agente historial para cada paciente.

M_{16}. Mantener persistente la sesión en la base de datos.

M_{17}. Agregar y remover áreas de diagnóstico.

M_{18}. Ordenar el diagnóstico.

M_{19}. Retornar las áreas de diagnóstico disponibles.

• Estado

1. Lista de agentes de diagnóstico inscritos.

2. Información de la sesión de usuario.

3. Lista de áreas de diagnóstico.

• Entradas Sensoriales

S_{11}. Consulta de agentes de diagnóstico en la base de datos.

• Actuadores

A_{11}. Crear un agente de Historial.

A_{12}. Guardar el estado de la sesión en la base de datos.

A_{13}. Liberar el agente de historial.

- **Comportamientos**

B_{11}. Registrar agente de diagnóstico: cumple con meta M_{11}.

Mensaje / Evento: *InsertAgentDiagnosis,* el *AgentDiagnosis* envía al *AgentSesion* al momento de crearse el agente una solicitud para registrarse en la lista de agentes inscritos.

Acción: registrar en el directorio de agentes el Agente de diagnóstico *AgentDiagnosis*.

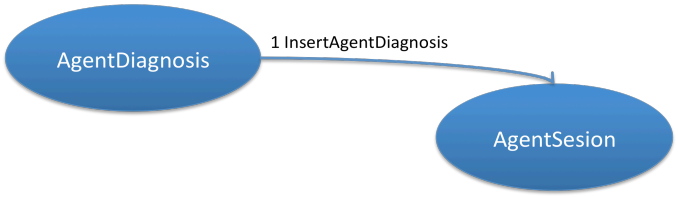

Figura 12 – Registrar agente de diagnóstico

Parámetros: *AgentDiagnosis*.

B_{12}. Crear historial de un sujeto a diagnosticar: cumple con meta M_{15}.

Mensaje / Evento: *RequestCreateAgentHistory*, envía solicitud para buscar historia clínica o crearla a partir del ID del paciente. *RequestLoad*, realiza la conexión con la base de datos y realiza la búsqueda de la información del sujeto a diagnosticar. *RequestPatientInformation*, Envía una petición al *AgentUserInterface* para llenar la información básica del paciente. *RequestCreate*, solicita al *AgentHistory* crear un nuevo historial con la información llenada anteriormente.

Acción: crear agente historial con el ID del paciente, primero realiza búsqueda del ID del paciente si lo encuentra carga los datos en el agente historial. Si no se encuentra el historial del paciente se envía la solicitud para crear uno nuevo.

Respuesta: *ResponseCreate* respuesta con la confirmación de la creación del historial.

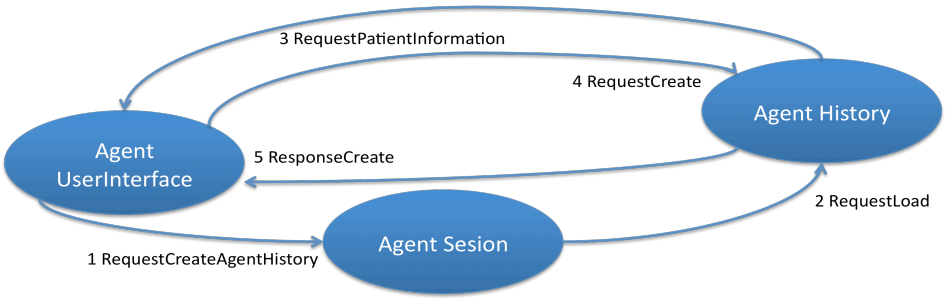

Figura 13 - Crear historial de un sujeto a diagnosticar

Parámetros: ID del sujeto a diagnosticar.

B_{13}. Cargar historial de un sujeto a diagnosticar: cumple con meta M_{15}.

Mensaje / Evento: *RequestCreateAgentHistory*, envía solicitud para buscar historia clínica o crearla a partir del ID del paciente. *RequestLoad*, realiza la conexión con la base de datos y realiza la búsqueda de la información del sujeto a diagnosticar.

Acción: realiza la búsqueda en la base de datos del usuario con el respectivo ID.

Respuesta: *ResponsePatientInfo*, envía la información al *AgentUserInterface* del sujeto a diagnosticar.

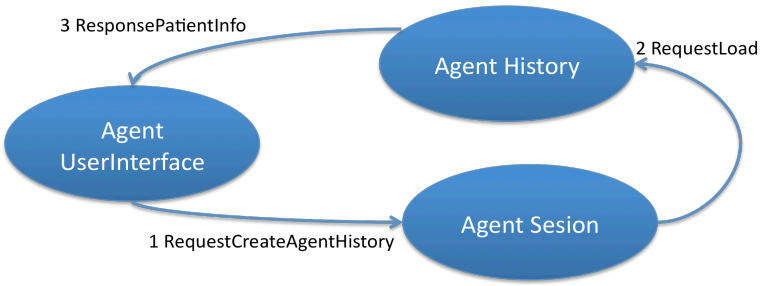

Figura 14 - Cargar historial de un sujeto a diagnosticar

Parámetros: ID del sujeto a diagnosticar.

B_{14}. Reiniciar agente caído: cumple con meta M_{13}.

Mensaje / Evento: *RequestAgentFail*, *AgentFault* le envía una solicitud al *AgentDiagnosis* indicando qué un agente ha fallado.

Acción: a partir del ID se puede saber qué tipo de agente era, se crea un nuevo agente con las mismas características y éste se encarga de enviar un mensaje de *alive* al agente de fallos.

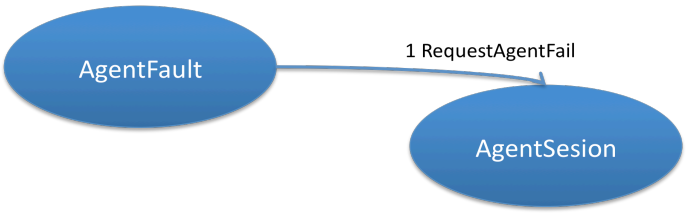

Figura 15 - Reiniciar agente caído

Parámetros: ID del agente caído.

B_{15}. Decidir qué agente de diagnóstico escoger: cumple con metas $M_{12,}$ $M_{14,}$ M_{18}.

Mensaje / Evento: *RequestSelectExpert*, el agente *AgentUserInterface* envía una solicitud al agente *AgentSesion* para realizar un diagnóstico.

Acción: a partir del filtro enviado por el *AgentUserInterface* el *AgentSesion* selecciona el *AgentDiagnosis* adecuado para realizar el diagnóstico.

Respuesta: *SetAgentInCharge*, le indica al *AgentUserInterface* cuál es el agente encargado para realizar el diagnóstico.

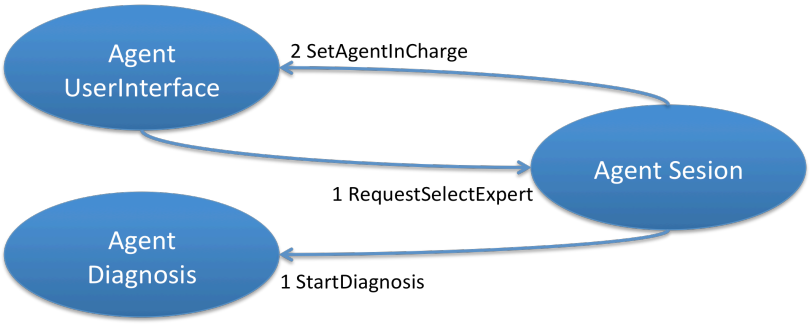

Figura 16 - Decidir qué agente de diagnóstico escoger

Parámetros: filtro.

B_{16}. Agregar área de conocimiento: Cumple con meta M_{17}.

Mensaje / Evento: *RequestInsertArea*, el *AgentUserInterface* envía al *AgentSesion* una nueva área de conocimiento para agregar a la lista.

Acción: registra en la lista de áreas de conocimiento una nueva área.

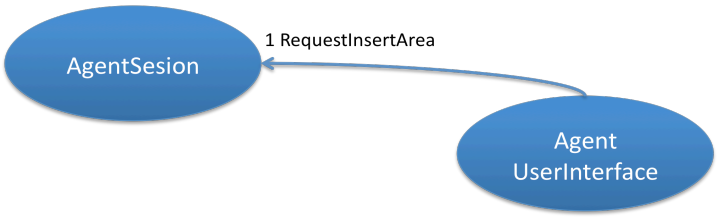

Figura 17 - Agregar área de conocimiento

Parámetros: área de Conocimiento.

B_{17}. Eliminar área de conocimiento: Cumple con meta M_{17}.

Mensaje / Evento: *RequestRemoveArea*, el *AgentUserInterface* envía al *AgentSesion* una solicitud para eliminar un área de conocimiento.

Acción: elimina de la lista de áreas de conocimiento un área seleccionada.

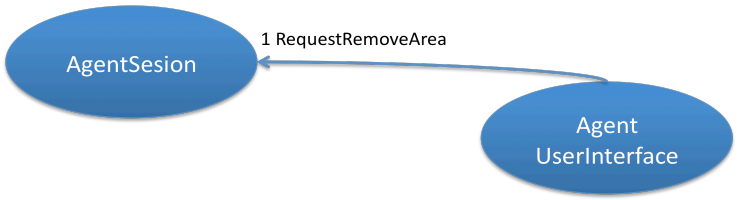

Figura 18 - Eliminar área de conocimiento

Parámetros: área de Conocimiento a eliminar.

B_{18}. Petición de la lista de áreas de conocimiento disponibles: cumple con metas M_{19}.

Mensaje / Evento: *RequestAreas*, el *AgentUserInterface* envía la solicitud para qué el *Agent-Sesion* le retorne la lista de áreas de conocimientos disponibles.

Acción: el agente de sesión le retorna la lista de las áreas disponibles qué mantiene en su estado.

Respuesta: *ResponseAreas*, le indica al *AgentUserInterface* cuál es la lista de áreas disponibles.

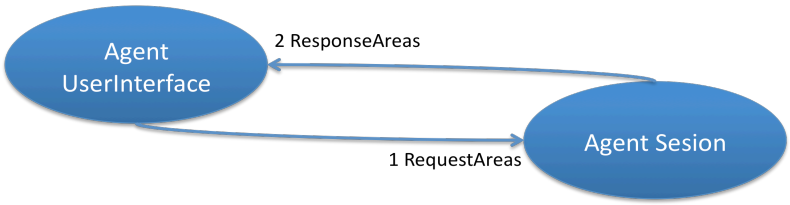

Figura 19 - Petición de la lista de áreas de conocimiento disponibles

B_{19}. Decidir qué especialista humano escoger: cumple con metas $M_{12,}$ $M_{14,}$ M_{18}.

Mensaje / Evento: *RequestSelectExpert*, el AgentUserInterface envía una solicitud para realizar un diagnóstico.

Acción: a partir del filtro enviado por el *AgentUserInterface* el *AgentSesion* selecciona al *AgentDiagnosis* adecuado para realizar el diagnóstico.

Respuesta: *SetAgentInCharge*, le indica al *AgentUserInterface* cuál es el agente encargado para realizar el diagnóstico.

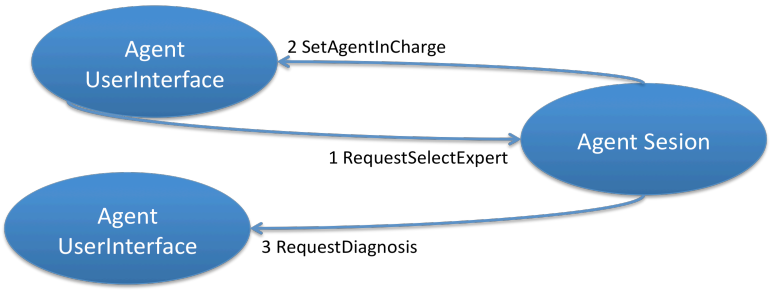

Figura 20 - Decidir qué especialista humano escoger

Parámetros: filtro.

2.3 Agente de Diagnóstico - AgentDiagnosis

- **Metas**

M_{21}. Iniciar el diagnóstico.

M_{22}. Invocar agente de Historial.

M_{23}. Responder al agente de Sesión.

- **Estado**

1. ID del agente de la Interfaz de Usuario.

2. ID del Agente Historial.

3. Sistema Experto.

4. Filtro con las áreas de diagnóstico.

5. Nombre del Sistema Experto.

- **Entradas Sensoriales**

S_{21}. Agregar hechos al Sistema Experto.

S_{22}. Recibir preguntas del Sistema Experto.

- **Actuadores**

A_{21}. Solicitar historia clínica al agente de historial asignado.

A_{22}. Utilizar el sistema experto asociado al agente para realizar el diagnóstico.

- **Comportamientos**

B_{21}. Iniciar el diagnóstico: cumple con meta $M_{21,}$ M_{22}.

Mensaje / Evento: *StartDiagnosis*, envío de mensaje por parte del *AgentSesion* para realizar un diagnóstico.

Acción: le solicita al *AgentSesion* realizar el diagnóstico al paciente utilizando el agente historial asignado al paciente.

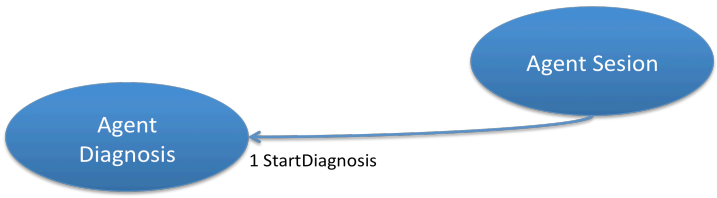

Figura 21 - Iniciar el diagnóstico

2.4 Agente de Fallos - AgentFault

• Metas

M_{31}. Registrar a los agentes en la lista de agentes a supervisar.

$M_{32.}$ Supervisar el funcionamiento de los agentes inscritos.

$M_{33.}$ Notificar al agente coordinador de los agentes que no funcionan.

$M_{34.}$ Actualizar los tiempos de los agentes qué se están supervisando.

$M_{35.}$ Remover los agentes de la lista cuando son desactivados.

• Estado

1. Lista de agentes inscritos.

2. Información del agente de sesión.

3. Intervalos de tiempo de los agentes.

• Actuadores

A_{31}. Agregar/Eliminar/Actualizar agentes en la lista de agentes inscritos.

A_{32}. Notificar al agente de sesión sobre los agentes caídos o los agentes qué fallan en su funcionamiento.

• Comportamientos

B_{31}. Notificar qué un agente está vivo: cumple con las metas $M_{31,}$ M_{34}.

Mensaje / Evento: *ResponseAreYouAlive*, se recibe un mensaje de cualquier agente notificando qué está vivo.

Acción: a partir del ID del agente qué se recibió se revisa en la lista de agentes inscritos, si está en la lista sólo se actualiza el tiempo; si no está, se agrega a la lista.

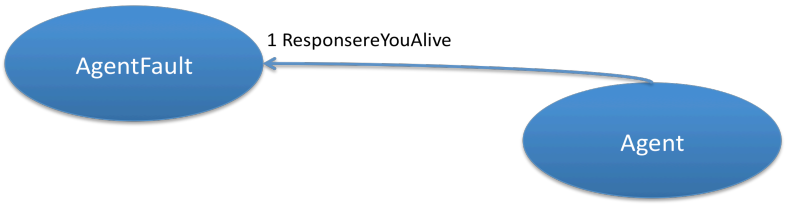

Figura 22 - Notificar qué un agente está vivo

Parámetros: ID del agente.

B_{32}. Confirmar si el agente está vivo: cumple con las metas $M_{31,}$ M_{34}.

Mensaje / Evento: *Guard(Agent Type)RequestAreYouAlive*, recibe el mensaje del agente de fallos preguntando si está vivo.

Acción: si el agente recibe el mensaje simplemente le responde al *AgentFault* con su ID.

Respuesta: *ResponseAreYouAlive*, le responde al agente de fallos si está vivo.

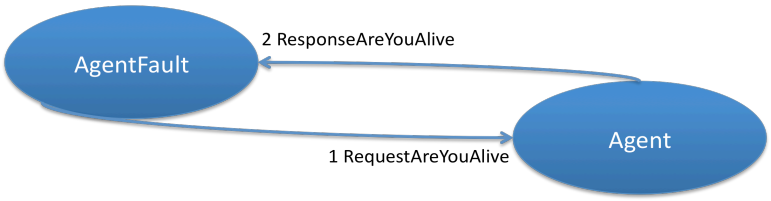

Figura 23 - Confirmar si el agente está vivo

B_{33}. Remover agente de la lista de fallos: cumple con las metas M_{35}.

Mensaje / Evento: *RequestRemoveAgent*, recibe el mensaje del *AgentSesion* notificando el agente que tiene que remover de la lista.

Acción: a partir del ID del agente qué recibió, realiza la búsqueda en la lista y lo remueve.

Respuesta: *ResponseAreYouAlive*, le responde al agente de fallos si está vivo.

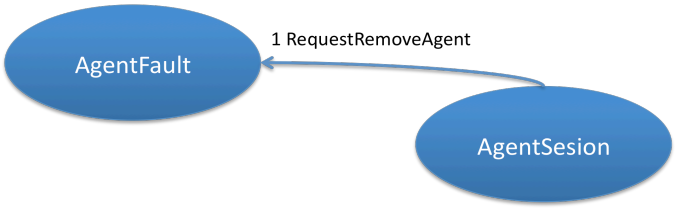

Figura 24 - Remover agente de la lista de fallos

Parámetros: ID del agente qué se quiere remover.

4.5 Agente de Historial - AgentHistory

• Metas

M_{41}. Administrar las historias clínicas.

M_{42}. Persistir las historias clínicas.

M_{43}. Verificar si los hechos están en la historia clínica del sujeto a diagnosticar.

• Estado

1. Historial del paciente en cuestión.

• Entradas Sensoriales

S_{41}. Notificación Historia de la base de datos.

• Actuadores

A_{41}. Buscar/Crear/Eliminar/actualizar los historiales.

• Comportamientos

B_{41}. Guarda los cambios hechos a un historial: cumple con M_{41}.

Mensaje / Evento: *ResponseSave*, solicitud del *AgentUserInterface* para guardar los cambios de un historial.

Acción: realizar la conexión con la base de datos y guardar los cambios qué fueron hechos al historial del sujeto a diagnosticar identificado por un ID.

Respuesta: *ResponseSave*, notificación al *AgentUserInterface* qué el historial ha sido guardado correctamente.

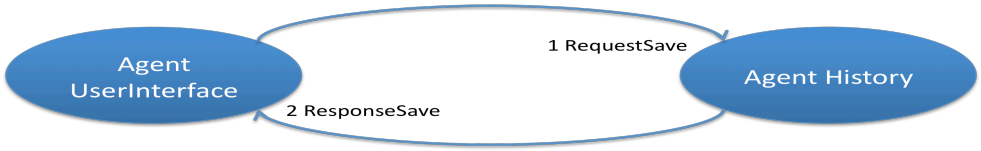

Figura 25 - Guarda los cambios hechos a un historial

Parámetros: ID del *AgentHistory*.

B_{42}. Buscar respuesta en el agente historial: cumple con M_{41}.

Mensaje / Evento: *RequestOnHistory*, el *AgentDiagnosis* le pregunta al *AgentHistory* si la respuesta la tiene.

Acción: el *AgentHistory* busca en su memoria si tiene la respuesta que el *AgentDiagnosis* necesita.

Respuesta: *ResponseAnswer*, le responde al *AgentDiagnosis* si tiene la respuesta.

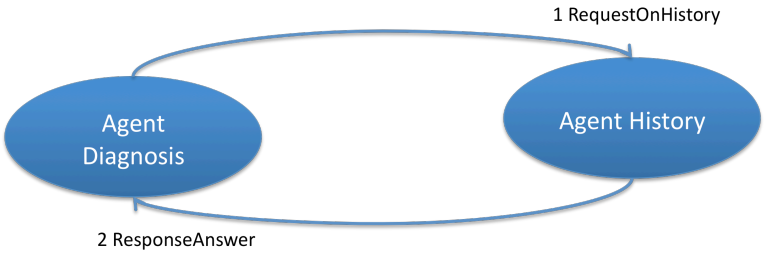

Figura 26 - Buscar respuesta en el agente historial

Parámetros: pregunta qué realizó el Sistema Experto.

B_{43}. Eliminar historial: cumple con M_{41}.

Mensaje / Evento: *RequestDelete*, el *AgentUserInterface* le envía el mensaje para eliminar un historial de la base de datos.

Acción: el *AgentHistory* realiza la búsqueda del historial en la base de datos y lo elimina.

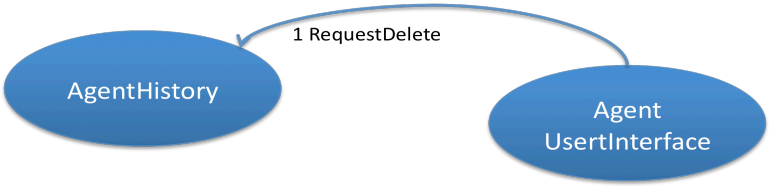

Figura 27 - Eliminar historial

4.6 Agente de la Interfaz de Usuario - AgentUserInterface

- **Metas**

M_{51}. Funcionar de intermediario entre la capa de presentación y la capa de negocio.

- **Estado**

1. Interfaz gráfica de usuario.

2. Información de la sesión de usuario.

- **Entradas Sensoriales**

S_{51}. Eventos de la interfaz de usuario.

- **Actuadores**

A_{51}. Solicitar los diferentes servicios que ofrece la lógica de la aplicación.

- **Comportamientos**

B_{51}. Interacción del usuario consultor: cumple con M_{41}.

Mensaje / Evento: *RequestQuestion*, el *AgentHistory* le envía una pregunta que no encontró en su memoria al *AgentUserInterface* para que le pregunte al Usuario Consultor.

Acción: el *AgentUserInterface* actualiza la interfaz gráfica del Usuario para que le desplegué al usuario consultor la pregunta.

Respuesta: *ResponseAnswer*, le responde al *AgentDiagnosis* la respuesta ingresada por el Usuario Consultor.

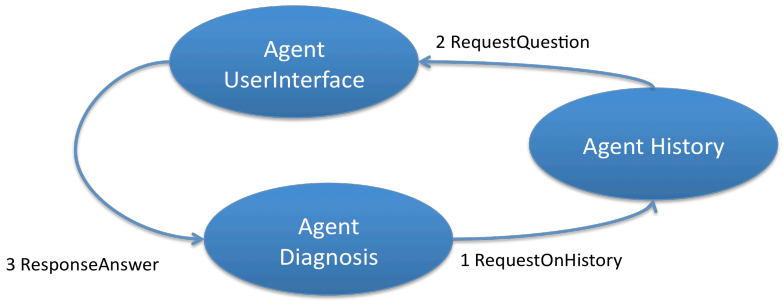

Figura 28 - Interacción del usuario consultor

Parámetros: pregunta que realizó el Sistema Experto.

Con la presentación del modelo general y el modelo de agentes obtenemos una descripción general de las capas, componentes y agentes que conforman la plataforma. Se termina el capítulo 5 lo cual nos permite continuar a realizar el diseño de componentes del sistema el cual será planteado y descrito en el siguiente capítulo.

VI –COMPONENTES FEDEXPERT

A partir de la descripción del modelo general y el modelo de agentes del sistema, se procede a realizar una descripción más detallada de las capas y los componentes que hacen parte de la plataforma FedExpert. Para el desarrollo de la implementación se decidió trabajar con la plataforma de programación JEE (Java Enterprise Edition). La plataforma de programación JEE permite el desarrollo de aplicaciones en lenguaje Java con basándose en arquitecturas de componentes distribuidos. La selección de dicha plataforma se debe a que los otros componentes que hacen parte del sistema utilizan el lenguaje de programación Java y esto permite simplificar el proceso de desarrollo. Se presenta el diseño y la definición de todos los componentes y todas las clases que integran cada una de las capas propuestas en el capítulo 5, los cuáles serán descritos y detallados con sus respectivos atributos y métodos.

1. Capa Presentación

El componente de la presentación de FedExpert se encarga de visualizar para el usuario todas las funcionalidades que presta el sistema. La principal característica de este componente es que se conforma por el patrón MVC (Modelo, Vista, Controlador), cuya función nos permite separar la lógica de negocio de nuestra aplicación de nuestra interfaz gráfica de usuario. Presentaremos cada uno de los componentes que conforman la capa de presentación con sus respectivas clases y sus descripciones.

• 1.1 Componente Modelo

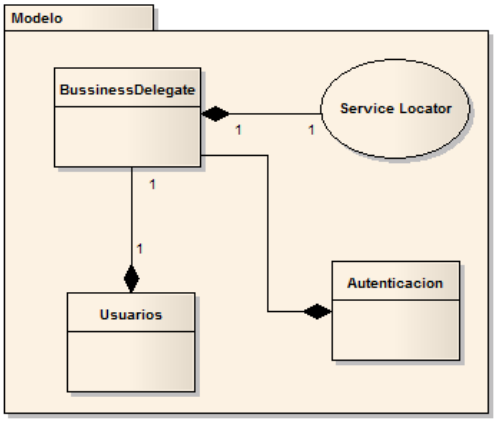

Figura 29 - Componente Modelo

BussinessDelegate: se utiliza para hacer las invocaciones de los métodos remotamente que se ofrecen por parte de la lógica de la plataforma FedExpert y se utiliza para recibir datos a través de diferentes capas. En este caso a través de la capa de presentación y la capa de la lógica.

Service Locator: se utiliza para encapsular el procedimiento de obtención de los servicios ofrecidos por la capa de la lógica de la plataforma FedExpert con una capa fuerte de abstracción. El *Service Locator* retorna datos cuando se pide alguna información necesaria para realizar una tarea.

Autenticación: encargado de hacer la autenticación de los usuarios de la plataforma FedExpert a la hora de iniciar la sesión de usuario y en el momento de realizar las diferentes operaciones de los clientes mediante el *BusinessDelegate.* Esto se realiza para que sólo puedan utilizar los servicios los usuarios que tengan los privilegios necesarios. En este caso los usuarios consultores, usuarios especialistas y los administradores del sistema.

Usuarios: cada *BusinessDelegate* tiene asociado un usuario, el cual es el que está utilizando la sesión en ese momento y también lo vincula con la autenticación para poder realizar todas las operaciones.

• 1.2 Componente Vista

Uno de las principales funciones de la plataforma de programación JEE es permitir la elaboración de componentes Web, los cuales pueden ser visualizados desde cualquier dispositivo que cuente con un navegador Web. Las páginas que son visualizadas por los navegadores son JPS (Java Server Pages), las cuales permiten la comunicación entre el sistema y el usuario. Las otras herramientas que hacen parte del sistema son descritas en el capitulo VII. La vista está conformada por diferentes páginas JSP las cuáles, son las encargadas de mostrar toda la información a los usuarios del sistema por medio de los navegadores Web.

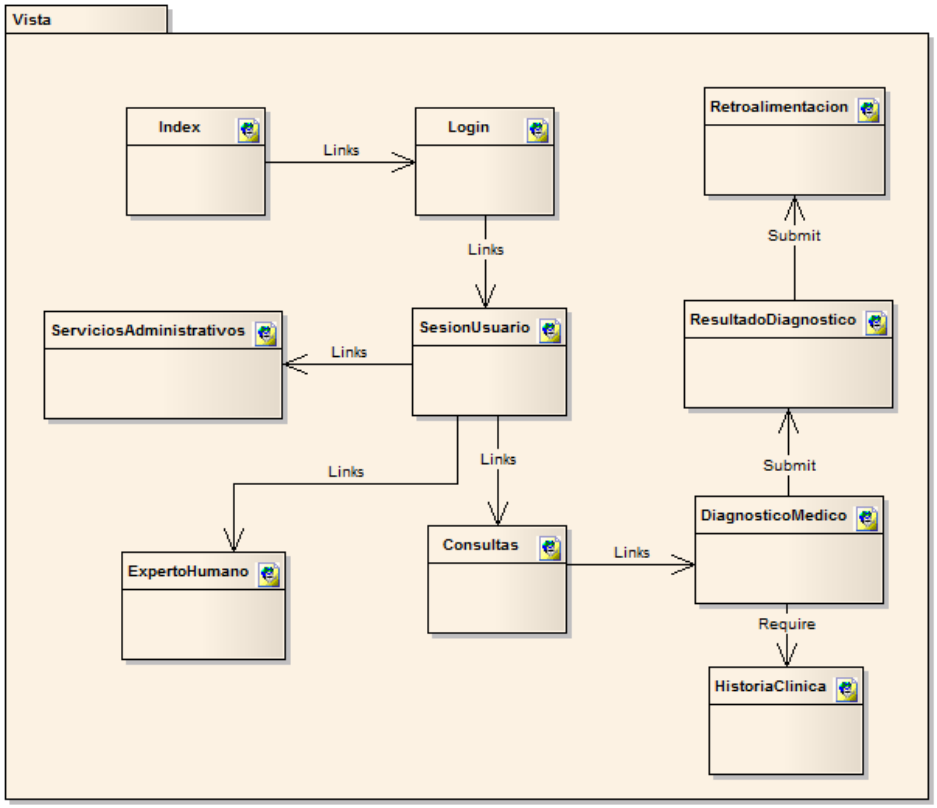

Figura 30 - Componente Vista

La vista está conformada por las siguientes páginas JSP que mostrarán los diferentes servicios de la plataforma al usuario:

Index (Página de inicio): página principal del sistema que mostrará al usuario las diferentes secciones y opciones de ésta. Entre ellas se encuentra el inicio de Sesión.

Inicio de Sesión: página en donde el usuario puede iniciar su sesión de usuario ingresando su nombre de usuario y contraseña. Una vez el usuario inicie su sesión podrá disponer de todos los servicios que su rol de usuario le permiten.

Servicios administrativos: una vez que el usuario administrador inició su sesión puede disponer de las diferentes opciones que le permite configurar los servicios administrativos de la aplicación.

Diagnóstico: página que le permite al usuario consultor iniciar un diagnóstico a un paciente.

Resultado del Diagnóstico: página que le muestra al usuario consultor el resultado del diagnóstico que había iniciado anteriormente.

Página de consultas: página que muestra las consultas que tiene activas el usuario consultor y en donde puede iniciar una nueva.

Página de sesión de usuario: página que se muestra a los usuarios una vez que ya inició sesión en el sistema.

Historial: página que permite a los usuarios consultores llenar por primera vez la historia clínica de los pacientes.

Retroalimentación: página que les permite a los usuarios consultores ingresar información cuando un experto lo solicita.

Experto Humano: página que le permite al usuario especialista humano ingresar la información cuando se le solicita un diagnóstico.

• 1.3 Componente Controlador

El controlador está conformado por servlets que son clases java que procesan peticiones y respuestas con la vista y el modelo.

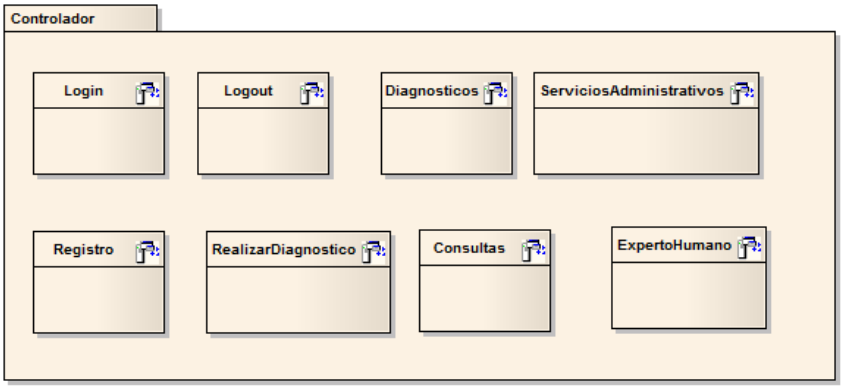

Figura 31 - Componente controlador

Login: recibe un *submit* con la información de inicio de sesión por parte del usuario de la página index y esta clase se encarga de realizar la autenticación por medio de la clase autenticación del modelo. Después de realizar la autenticación re-direcciona a la página de sesión de usuario.

Logout: recibe un *submit* de cualquier página JSP de la que se conforma el sistema con la petición de cerrar la sesión de usuario. Una vez se cierra la sesión se reenvía al usuario a la página principal.

Realizar Diagnóstico: recibe un *submit* de la página de diagnóstico. Se encarga de evaluar la solicitud y generar una respuesta re-direccionando a la página del resultado de diagnóstico con la información generada.

Diagnósticos: recibe un *submit* de la página de sesión de usuario pidiendo los diagnósticos que ha realizado el usuario consultor. Cuando termina la operación, reenvía los resultados al usuario por medio de la página Consultas.

Consultas: recibe un *submit* de la página de sesión de usuario pidiendo las consultas que tiene activas el usuario consultor. Cuando termina la operación, reenvía los resultados al usuario por medio de la página Consultas.

Servicios Administrativos: recibe un *submit* de la página de sesión de usuario pidiendo los servicios que puede administrar el usuario administrador. Cuando termina la operación re-envía los resultados al usuario por medio de la página Servicios administrativos.

2. Capa Lógica

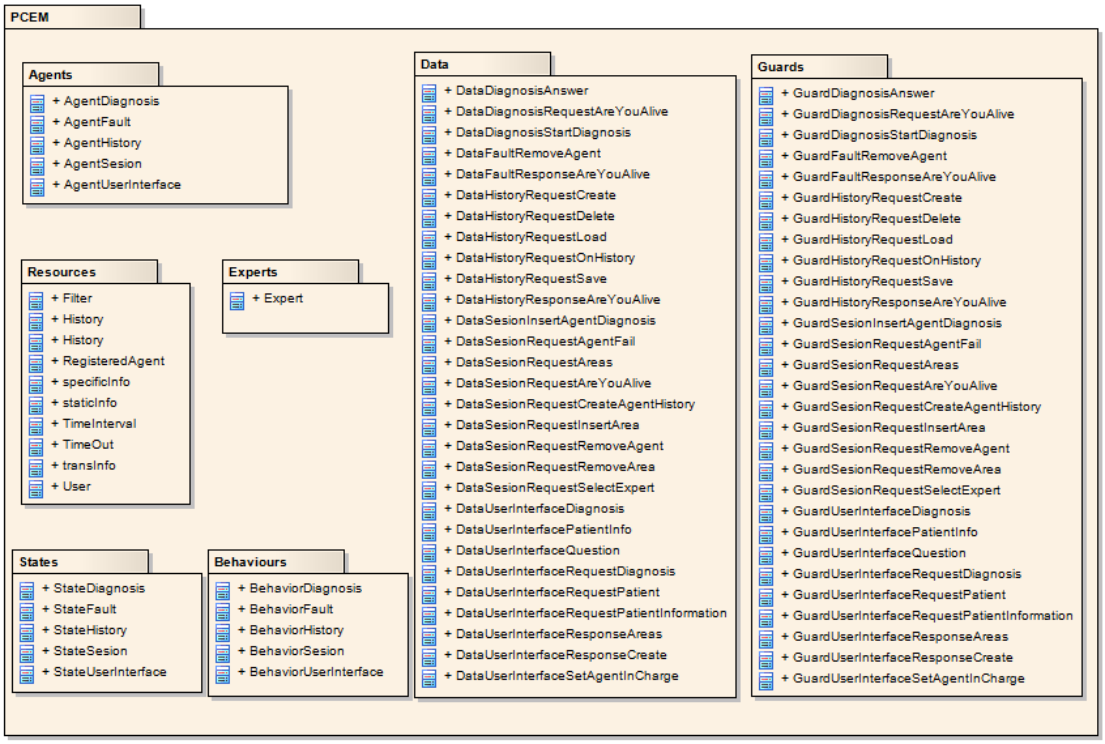

Figura 32 – Capa de la lógica

La lógica de la plataforma FedExpert está conformado por 7 componentes principales. La mayoría de los componentes son orientados a la programación basada en agentes. Los componentes orientados a la programación orientado a agentes que hacen parte del sistema son: *Agents* (Agentes) , *Behaviours* (Comportamientos), *Data* (Datos), *Guards* (Guardas) y *States*. (Estados). Los componentes que no están orientados a la programación orientada a agentes son: *Experts* (Sistemas Expertos) y , *Resources* (Recursos).

A continuación, se presentan los diagramas de clases de cada componente. En el caso de las Guardas y los Datos sólo se presentan unos ejemplos, debido a la gran extensión de clases que estos presentan y a su similitud en la mayoría de los casos. El diagrama completo de clases del sistema se encuentra como anexo en el porta Web asignado al trabajo de grado.

• Componente Agents

El componente de los agentes almacena todos los agentes que hacen parte del sistema. Cada clase se compone de los métodos: *createStructSesion*, *setupAgent* y *shutdownAgent*. Los cuáles permiten crear los agentes en el contenedor. Como se ilustra en la figura 33, el prefijo *Agent* indica que la clase es un comportamiento del agente y el sufijo indica cuál es el nombre asignado al agente.

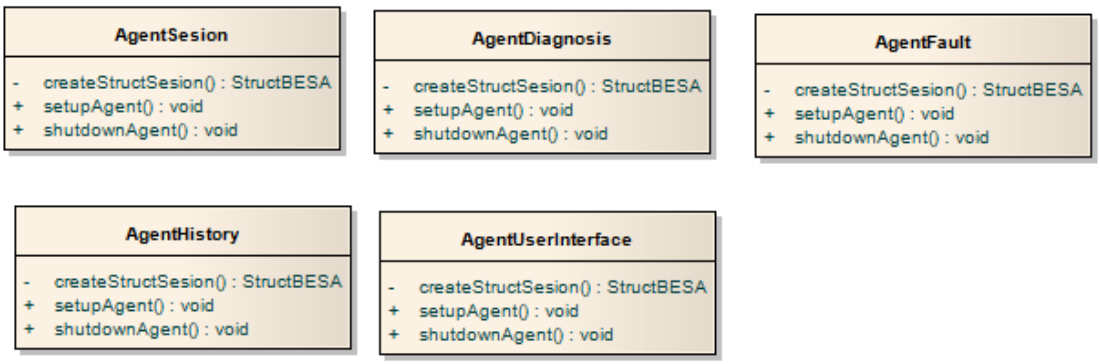

Figura 33 - Componente de los agentes

• Componente Behaviours

El componente de los comportamientos almacena todos los comportamientos de los agentes. Un comportamiento permite reaccionar a un conjunto bien definido de tipos de eventos; para demostrar su interés en un tipo específico de evento, el comportamiento es registrado ante la guarda asociada al evento; un evento le será enviado cuando la guarda sea disparada. Como se ilustra en la figura 34 el prefijo *Behaviour* indica que la clase es un comportamiento del agente y el sufijo indica a cuál agente pertenece.

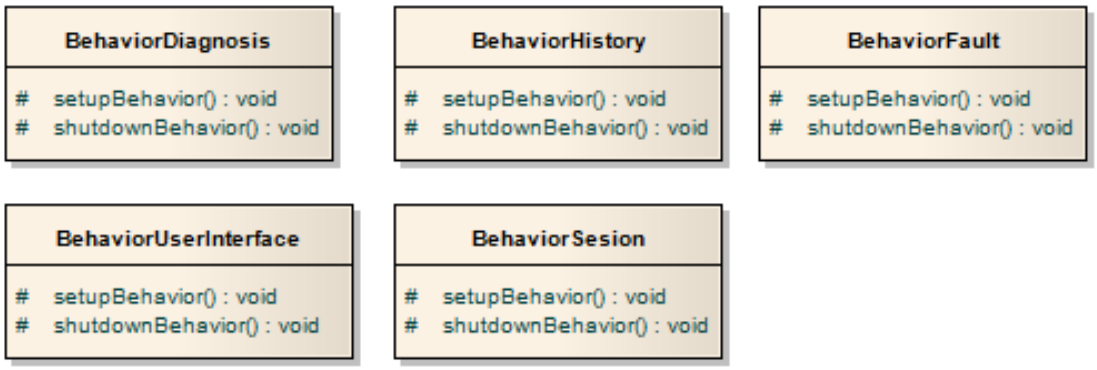

Figura 34 - Componente de los comportamientos

- ## Componente Data

El componente de datos se encarga de guardar las clases que son estructuras complejas de datos y se enviarán a través de la creación de los eventos entre los agentes que conforman el sistema. Como se ilustra en la figura 35 se puede observar que el prefijo *Data* indica que la clase es una estructura de datos, la siguiente palabra indica a cuál agente está siendo asociado y el sufijo indica el nombre de la estructura que la identifica. En ella se guarda la información que se quiere trasladar de un agente a otro para iniciar una acción de una guarda.

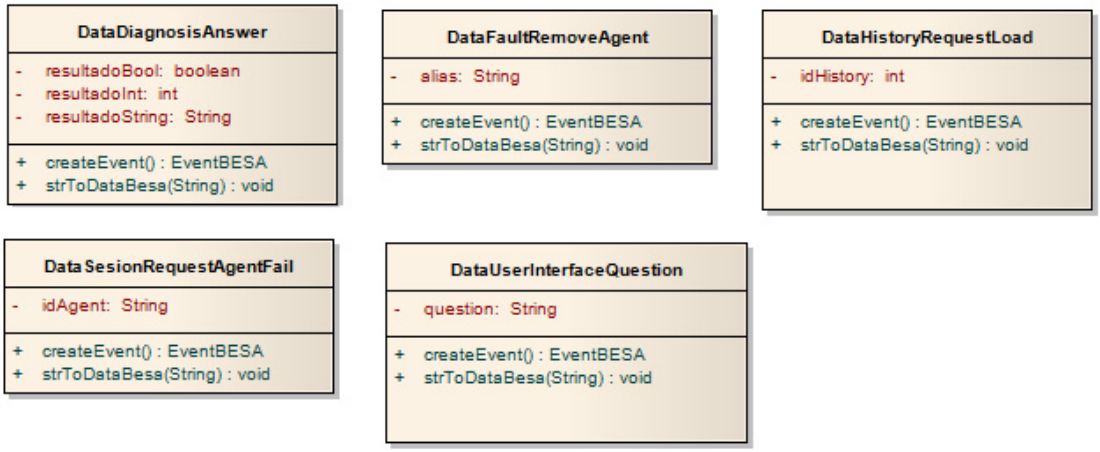

Figura 35 - Componente de los datos

- ## 2.4 Componente Experts

En el componente de los expertos se encuentra la clase *Expert*. La clase se encarga de cargar el archivo CLP el cuál es la extensión de los archivos de Clips, el Sistema Experto que se utilizó en la implementación. Como se puede observar en los métodos que se lustran en la

figura 36 la clase *Expert* esta encargada de agregar los hechos al Sistema experto y de enviar las preguntas o el resultado a la interfaz gráfica de usuario.

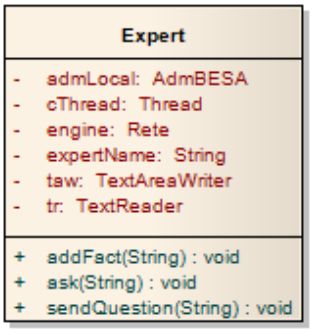

Figura 36 - Componente de los expertos

• 2.5 Componente Guards

El componente de las guardas mantiene todas las guardas asociadas a los comportamientos de los agentes que conforman el sistema. Las guardas son las encargadas de disparar las diferentes acciones de los agentes cuando se recibe un evento asociado a la guarda. Todas las guardas tienen que estar asociadas a un comportamiento. Como se ilustra en la figura 37 se puede observar que el prefijo *Guard* indica que la clase es una guarda, la siguiente palabra indica a cuál agente esta siendo asociado y el sufijo indica el nombre de la guarda en particular.

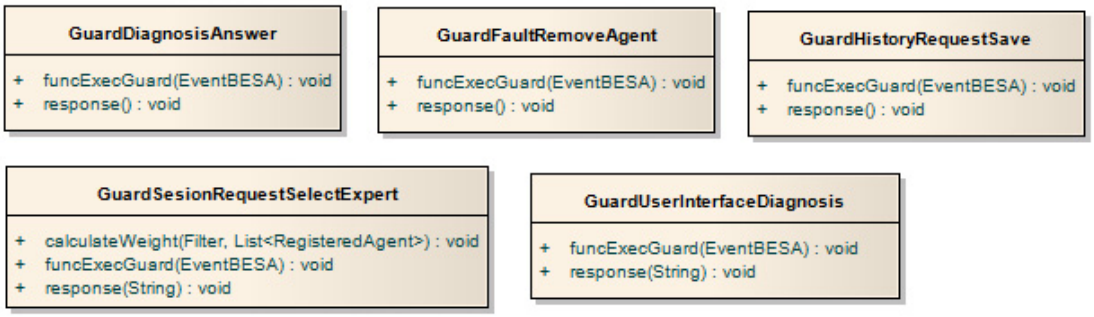

Figura 37 - Componente de las guardas

• 2.6 Componente Resources

El componente de los recursos se encarga de almacenar las clases del sistema que no pertenecen al modelo de los agentes pero, si son utilizados por los agentes para realizar las diversas funciones que tienen encargadas. Como se ilustra en la figura 38, el componente *resources* tiene clases: *Filter, TimeInterval, History, RegisteredAgent, TimeOut, User, StaticInfo, TransInfo y SpecificInfo.*

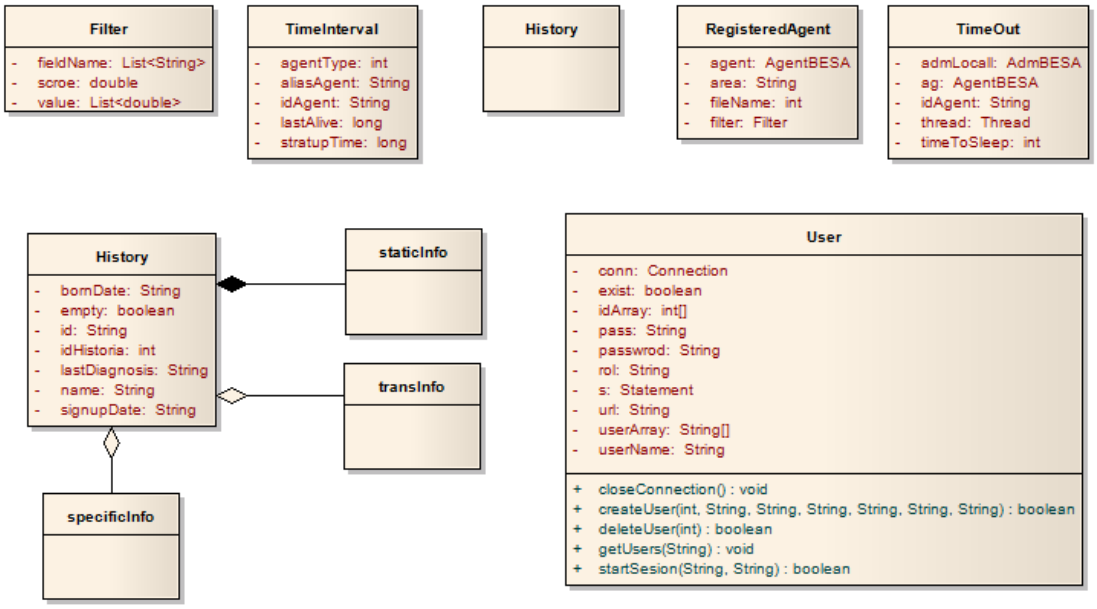

Figura 38 - Componente de los recursos

La clase *Filter* es utilizada por los *AgentDiagnosis* para indicar al agente de sesión cuáles son las áreas de experticia del Sistema Experto que tiene cargado en su estado. La clase *TimeInterval* es utilizada por el *AgentFault* para establecer los intervalo de tiempo cargados por defecto para el envío de los mensajes de vida hacia los demás agentes. La clase *RegisteredAgent* es utilizada por el *AgentSesion* para mantener una lista de los *AgentDiagnosis* disponibles para realizar un diagnóstico a un sujeto. La clase *TimeOut* utiliza un hilo para saber cada cuanto enviar los mensajes de supervivencia a los agentes que están registrados en el sistema. La clase *History* es la encargada de mantener la información que se obtiene de la base de datos y también en el caso de crear o actualizar una historia. Se compone por la información estática, transversal y especifica del sujeto a diagnosticar. La clase *User* se encarga de mantener la información del usuario del sistema. Se utiliza para realizar la obtención de la información de la base de datos y para almacenarla.

• 2.7 Componente States

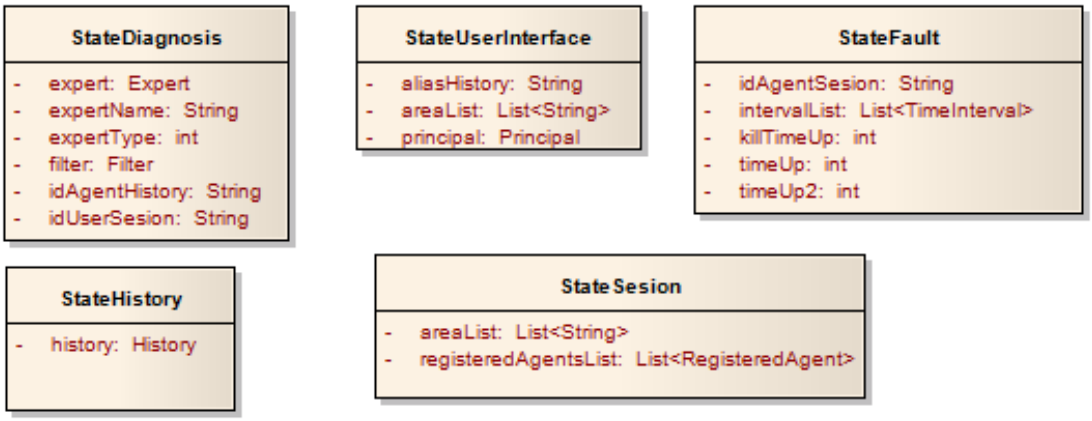

Figura 39 - Componente de los estados

El componente de estados aloja las clases que forman parte de los estados de los agentes del sistema. Un estado es una memoria compartida que puede ser utilizada por los comportamientos del agente usando sincronización de exclusión mutua para evitar problemas de concurrencia. Como se ilustra en la figura 39 el prefijo *State* indica que la clase es un estado del agente y el sufijo indica a cuál agente pertenece el estado.

3. Capa Datos

Para el desarrollo de la plataforma se utilizaron dos bases de datos, las cuales están separadas. Las bases de datos son: Usuarios y Lógica. Los motivos para realizar la plataforma con dos bases de datos separadas radica, en que es preferible mantener separada la información referente a la administración del sistema con la información para la funcionalidad de él.

• 3.1 Base de Datos Usuarios

La base de datos de los usuarios se utiliza para guardar la información de acceso de todos las personas que van a utilizar el sistema. Los usuarios persisten en la base de datos la información necesaria para identificarse ante el sistema. Como se ilustra en la figura 40, cada usuario puede tener un rol diferente, el cuál le permite realizar las diversas funciones dentro de la plataforma. Los roles se comprenden por: Usuario Consultor, Usuario Administrador y Usuario Especialista. El usuario Especialista mantiene una sesión lo cuál le permite tener las sesiones persistentes si en algún momento cierra la aplicación sin haberlas terminado. En la figura 40 se ilustra el Modelo entidad relación de la base de datos.

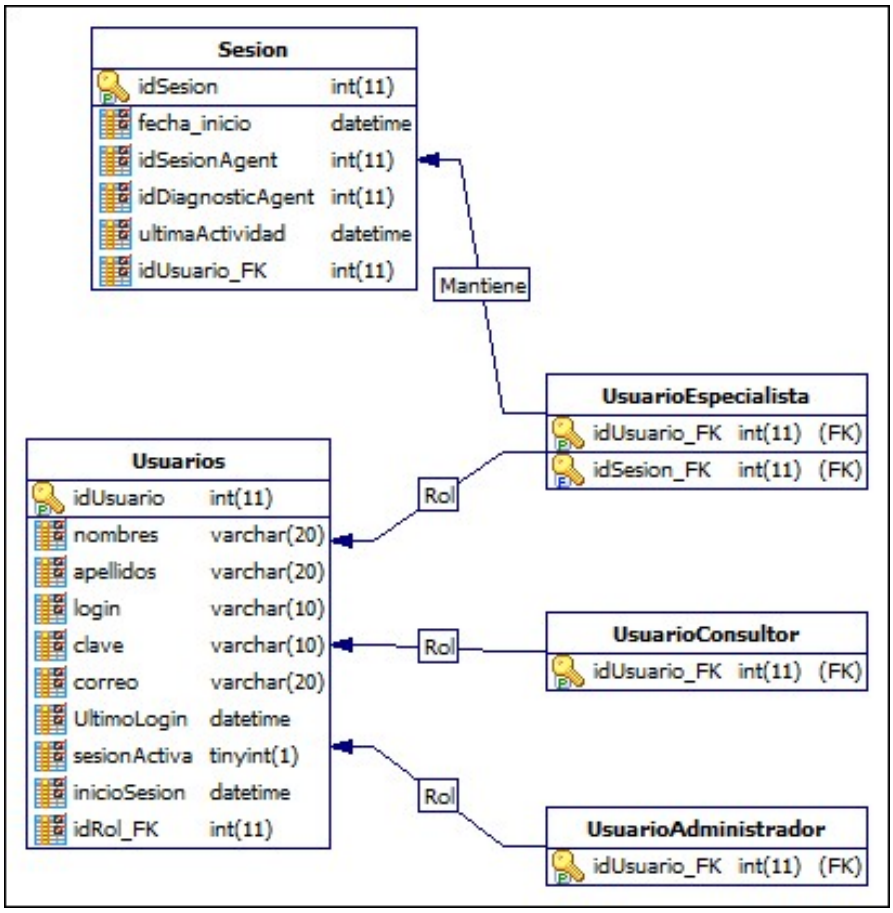

Figura 40 - Modelo entidad relación Usuarios

• 3.2 Base de Datos Lógica

La base de datos de la lógica es la encargada de persistir toda la información referente al funcionamiento de la plataforma. Es la información que hace referencia a los sujetos a diagnosticar, consultas, diagnósticos e historiales.

La base de datos se compone por las siguientes tablas:

- Sujeto: encargada de mantener la información principal del sujeto a diagnosticar o el acudiente del sujeto a diagnosticar.
- Diagnóstico: se encarga de persistir la información principal acerca de una consulta que tuvo el sujeto.
- Consultas Activas: mantiene la información de la consulta de un sujeto si todavía no se ha cerrado por parte del Usuario Consultor.
- Historia: encargada de persistir el historial del sujeto a diagnosticar para que pueda ser utilizado en futuras consultas por parte de los Usuarios Consultores.

La tabla Historia se compone por las siguientes tablas:

- Información Estática: es la información del sujeto a diagnosticar que no cambia a través del tiempo.
- Información Transversal: es la información del sujeto a diagnosticar que puede cambiar entre una consulta realizada y otra futura.
- Información Especifica: es la información del sujeto a diagnosticar que sólo hace referencia a una consulta en particular.

En la figura 41 se ilustra el Modelo entidad relación de la base de datos.

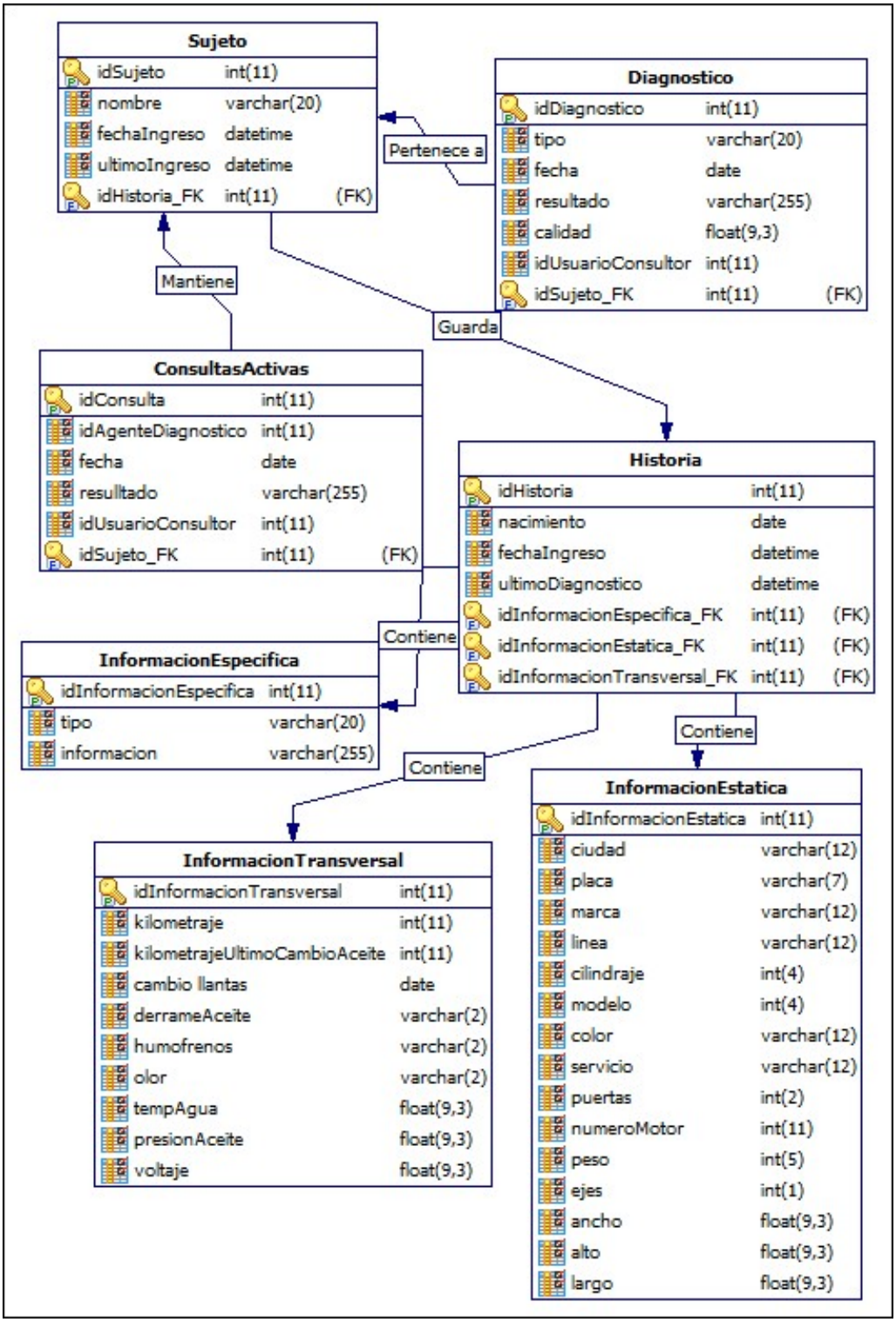

Figura 41 - Modelo entidad relación Lógica

Con la presentación de los componentes obtenemos una descripción más detallada de las capas, componentes y agentes que conforman la plataforma. Se presenta la descripción de los atributos y métodos que componen las clases de la plataforma. Con la presentación de los componentes del sistema se termina el capítulo 5 lo cuál nos permite continuar a realizar la implementación del prototipo para el diseño propuesto. En el capítulo siguiente se brinda la descripción de las herramientas utilizadas, implementación del código, pruebas del prototipo y resultados de las pruebas.

VII – APLICACIÓN PRUEBAS Y RESULTADOS

Para el modelo que se describió en el capítulo anterior, se realizó el desarrollo de un prototipo funcional de la plataforma con una interfaz gráfica para probar los componentes que se modelaron en el diseño. El alcance que se definió para el prototipo consiste, en probar los componentes principales de la capa de negocio definida anteriormente. La base principal de la plataforma es el tema de la cooperación de la federación de expertos.

1. Herramientas y Tecnologías Utilizadas

A partir de un análisis realizado a diferentes tecnologías y metodologías escogimos las que se acomodaron más a las necesidades de los objetivos pactados. Las herramientas que se seleccionaron para el desarrollo de la aplicación son descritas a continuación.

• 1.1 Lenguaje de Programación

Para el desarrollo del código de la plataforma FedExpert se utilizó el lenguaje de programación Java y se utilizó como entorno de desarrollo integrado (IDE) Netbeans 6.7.1. Para la ejecución de sistemas multi-agentes (SMA) se utilizó el Framework BESA, la cuál es una herramienta desarrollada en la facultad de Ingeniería de Sistemas de la Pontificia Universidad Javeriana. Como se mencionó en el capitulo VI para el modelo se decidió utilizar la plataforma de programación JEE que permite la fácil integración de los diferentes componentes desarrollados en Java.

• 1.2 Sistemas Expertos

Para la ejecución de los sistemas expertos se utilizó la herramienta JESS (Java Expert System Shell). JESS es una herramienta que permite desarrollar sistemas basados en reglas en el lenguaje de programación Java. Los sistemas expertos que se utilizaron están creados en el lenguaje CLIPS (Sistema de Producción Integrado en Lenguaje C). Se utilizó dicha herramienta debido a que se encontró una gran variedad de sistemas expertos que ya estaban desarrollados, los cuales pudimos utilizar con nuestra plataforma. CLIPS es totalmente adaptable a la herramienta JESS lo cuál nos permitió integrar nuestra programación hecha en el lenguaje Java y los Sistemas Expertos desarrollados en CLIPS. Para la adaptación de los sistemas expertos encontrados con la plataforma FedExpert se realizó una pequeña modificación. A los sistemas expertos implementados en CLIPS se le agregó un enrutador de salida. Este enrutador permite enviar las preguntas que realiza el sistema experto a la clase Java que se desarrolló para el agente de diagnóstico encargado.

• 1.3 Bases de Datos

Como se describió en el capítulo anterior, se desarrollaron dos bases de datos. Una base de datos para los usuarios que van a utilizar y administrar el sistema y otra base de datos para las funcionalidades de la plataforma. Las bases de datos fueron creadas en un servidor

MySQL utilizando la tecnología de almacenamiento *InnoDB* la cuál, permite la creación de llaves foráneas utilizando bases de datos *MySQL*. La base de datos se encuentra instalada remotamente en un servidor con sistema operativo Windows Server 2003 en la dirección 213.175.204.186:3306/fedexpert_bd. La base de datos se instaló en un servidor remoto para tener acceso a la información desde cualquier ubicación, lo que nos permite simular la situación ideal de la plataforma.

2. Implementación

La meta principal de la plataforma es permitir la cooperación de los sistemas expertos que conforman la federación. Para esto se decidió implementar en una gran parte la lógica de negocio del sistema, la cuál permite realizar la cooperación de los expertos. A continuación, se presenta la descripción de los componentes y las clases que se implementaron del diseño descrito anteriormente.

Las capas que se desarrollaron fueron las de Lógica y las de Datos. Los componentes que fueron desarrollados en su totalidad son los: agentes, comportamientos, estados, expertos y recursos. Los componente que fueron desarrollado parcialmente son el componente de guardas y el datos.

La capa que no fue desarrollada fue la capa de la Presentación. Para la realización del prototipo se realizó el diseño y la implementación de una interfaz gráfica de usuario. Esta implementación permite simular el uso que debería tener la capa de la presentación. La interfaz gráfica de usuario permite a los usuarios tener la interacción básica de las funcionalidades de la plataforma, sin necesidad de utilizar la consola.

3. Pruebas

A partir del prototipo funcional implementado y desarrollado para probar las principales características de la aplicación se diseñaron diferentes tipos de pruebas. El objetivo de las pruebas es ofrecer resultados con los cuáles se puede sustentar si los objetivos establecidos inicialmente y las metas pactadas se cumplieron o no se cumplieron.

Las pruebas realizadas se realizaron utilizando Sistemas Expertos existentes en el área automotriz. Se utilizaron 4 Sistemas Expertos todos con conocimiento acerca de automóviles pero cada uno con conocimientos específicos diferentes. FedExpet está diseñado para ser utilizado en cualquier dominio de conocimiento en general. Lo importante es que todos los sistemas expertos tengan conocimiento similar. Para utilizar el sistema FedExpert con otro dominio de conocimiento diferente al implementado en el prototipo, es necesario adaptar los Sistemas Expertos como fue mencionado al principio de este capitulo. En este caso los Sistemas Expertos utilizados fueron basados en CLIPS. El sistema esta diseñado para ser utilizado con cualquier Sistema Experto que soporte el lenguaje de programación Java, solo hay que modificar la salida del Sistema Experto y la entrada de una de las clases que hacen parte del sistema.

• 3.1 Pruebas unitarias

Las pruebas unitarias consisten en realizar comprobaciones de funcionalidad de cada una de las guardas con las que cuenta el sistema. Para esto se diseñó e implementó una clase llamada *StartAction*, la cuál contiene un método para probar la funcionalidad de cada guarda que conforman los agentes. Como se ilustra en las figuras 42 y 43 se crearon métodos de prueba para cada guarda que se implementó en el prototipo. En la figura 42 se ilustra el ejemplo del código para el evento *UserInterfaceQuestion* el cuál, se encarga de enviar una pregunta a la interfaz de usuario. En la figura 43 se ilustra el ejemplo del código para el evento *SesionRequestAgentFail,* el cuál se encarga de notificarle al agente de sesión que un agente ha fallado. Para el nombre del método se utilizó como prefijo *sendEvent* y como sufijo el nombre de la guarda que se quiere probar.

Figura 42 - Método sendEventUserInterfaceQuestion

```
public void sendEventUserInterfaceQuestion(){
    try{
        String question = "What's the oil pressure?";
        DataUserInterfaceQuestion dataToSend = new DataUserInterfaceQuestion();
        dataToSend.setQuestion(question);
        admLocal.getHandlerByAlias("agentUserInterface01").sendEvent(dataToSend.createEvent());
        TraceConsole.trace("Se envio DataUserInterfaceQuestion al agente: agentUserInterface01.");
    } catch (Exception e) {
         TraceConsole.trace(e.getMessage());
    }
}
```

```
public void sendEventSesionRequestAgentFail() throws ExceptionBESA{
    String idAgent = "agentDiagnosis01";
    DataSesionRequestAgentFail dataToSend = new DataSesionRequestAgentFail();
    dataToSend.setIdAgente(idAgent);
    admLocal.getHandlerByAlias("agentSesion01").sendEvent(dataToSend.createEvent());
    TraceConsole.trace("Se envio DataSesionRequestAgentFail al agente: agentSesion01.");
}
```

Figura 43 – Método sendEventSesionRequestAgentFail

Para realizar las pruebas a las guardas primero se cargan las variables con las que cuenta el *DataBESA* asociado a la guarda en cuestión. En el caso de la figura 43 como se requiere enviar el ID de un agente que ha tenido problemas, se carga la variable *idAgent* con un String el cuál contiene el nombre de un agente al azar. Se llama al constructor por defecto del *DataBesa* asociado a la guarda a la que queremos probar. Se carga la variable que lleva el ID del

agente al *DataBESA* que construimos anteriormente. Por medio de una llamada al administrador local del contenedor donde se está trabajando, se invoca el método *sendEvent* enviándole el parámetro *idAgent* que se habia creado anteriormente.

```
[QuemesAPP]: Se envio DataUserInterfaceQuestion al agente: agentUserInterface01..
Pregunta recibida: What's the oil pressure?
```

Figura 44 - Salida de la consola cuando se envía el evento UserInterfaceQuestion

Para comprobar si el mensaje ha sido enviado correctamente a través de los agentes después de haber enviado el evento hacemos una impresión en la consola con un mensaje indicando que el evento ha sido enviado con éxito. Como se ilustra en la figura 44 para el evento *UserInterfaceQuestion* se imprime el rastreo en la consola y después se hace una impresión del contenido del *DataBESA*. Esto demuestra el éxito del mensaje y la correcta información contenida en el *DataBESA* que se envío a través del evento.

```
[QuemesAPP]: Se envio DataSesionRequestAgentFail al agente: agentSesion01..
El agente a restaurar es: agentDiagnosis01
```

Figura 45 - Salida de la consola cuando se envía el evento SesionRequestAgentFail

La figura 45 muestra la impresión por consola cuando se envía con éxito el evento *SesionRequestAgentFail*. Como se ilustra en los dos ejemplos anteriormente descritos, para cada guarda que se implementó en el sistema se realizaron las pruebas unitarias.

• 3.2 Pruebas de Circuitos Típicos

Las pruebas de circuitos típicos consisten, en no sólo probar una guarda en especial sino, probar un conjunto de guardas que dependan unas de otras. Como se observo anteriormente en el Capítulo VI, los comportamientos de cada agente se componen, de envío de eventos entre diferentes agentes de los cuáles se conforma el sistema. Las pruebas que se describen en esta sección tienen como objetivo comprobar la funcionalidad de un comportamiento.

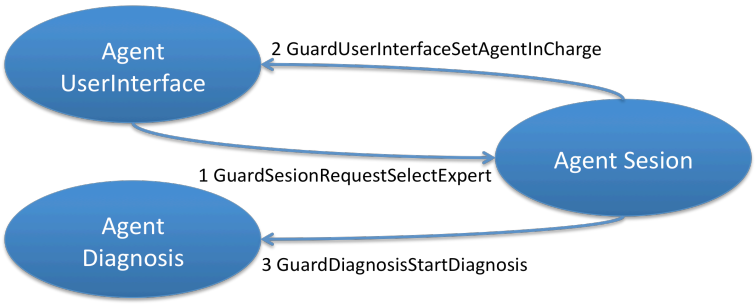

Figura 46 - Circuito entre agentes

La figura 46 muestra un circuito típico que se utiliza en la ejecución del sistema. El circuito es el encargado de decidir cual es el agente que se debe escoger para realizar el diagnóstico. El agente de sesión recibe un evento del agente de la interfaz gráfica indicándole que el usuario consultor ha decidido iniciar el diagnóstico a partir de un filtro que seleccionó. El agente de sesión activa la guarda asociada al evento y realiza la selección del experto. Una vez que

el agente de sesión tiene seleccionado el experto encargado de realizar el diagnóstico éste, le informa al agente de la interfaz de usuario con cuál agente va a realizar la interacción y le indica al agente de diagnóstico seleccionado que puede iniciar el diagnóstico.

Como se realizó en las pruebas unitarias inicialmente se utiliza de nuevo la clase *StartAction,* la cuál contiene los métodos necesarios para realizar las pruebas. En este caso sólo se utiliza un método para iniciar el circuito. El resto de interacciones entre los agentes se realiza utilizando las respuestas que estos tienen por defecto. Para el ejemplo que se muestra a continuación se utiliza el método presentado en la figura 47.

```
public void sendEventRequestSelectExpert() throws ExceptionBESA{
    DataSesionRequestSelectExpert dataToSend = new DataSesionRequestSelectExpert();
    Filter filter = new Filter();
    for(int i=0; i<5; i++)
         filter.getValue().set(i, 1.0);
    dataToSend.setFilter(filter);
    admLocal.getHandlerByAlias("agentFallos01").sendEvent(dataToSend.createEvent());
```

```
[QuemesAPP]: Se envio DataSesionSelectExpert  al agente: agentSesion01..
El filtro que llego es:
1.0
1.0
1.0
1.0
1.0
1.0
1.0
Agente: agentDiagnosis1 Score: 6.0
El agente de diagnostico que se escojio es: PCEM_127.0.0.1_9876_4
[QuemesAPP]: Se envio DataUserInterfaceSetAgentInCharge al agente: agentUserInterface01..
[QuemesAPP]: Se envio DataDiagnosisStartDiagnosis al agente: PCEM_127.0.0.1_9876_4..
El agente con el que se debe comunicar es: PCEM_127.0.0.1_9876_4
```

Como se ilustra en los métodos que se explicaron en las pruebas unitarias, se crean los atributos que se enviaran en el *DataBESA*. En este caso, se crea un filtro de prueba. Se carga el filtro en el *DataBESA* y se envía el evento como se realizo en las pruebas anteriores. En la figura 48 se observa la salida generada cuando se cumplió el circuito explicado. Como se

Figura 48 - Salida de la consola cuando se cumple el circuito

puede observar, la salida de la consola se inicia con el envío del evento *SesionSelectExpert* al agente de sesión con el filtro creado como ejemplo. El *AgentSesion* imprime la lista del filtro. Se comprueba que la lista que recibió el *AgentSesion* es la misma que se creo para la prueba. Una vez que el *AgentSesion* seleccionó el experto, él envía el evento *UserInterfaceSetAgentInCharge* al *AgentUserInterface*. El *AgentUserInterface* recibe el *DataBESA* e imprime el nombre del *AgentDiagnosis* que se escogió. Al realizar el envío del evento al *AgentUserInterface*, el *AgentSesion* le envía el evento *DiagnosisStartDiagnosis* al agente seleccionado.

• 3.3 Pruebas de Concurrencia

Las pruebas tienen como meta principal evaluar la funcionalidad total del prototipo que se implementó para la plataforma FedExpert. Para dichas pruebas se necesitó poner en funcionamiento la totalidad del prototipo. Para esto se necesitaron subir las bases de datos, correr el contenedor de *BESA*, y utilizar la interfaz gráfica de Usuario que se diseñó para el prototipo. La interfaz gráfica de usuario nos permite manejar todos los servicios que ofrece el prototipo.

De esta forma, podemos ingresar la información requerida por el sistema. A continuación, se hace una breve descripción de las pruebas concurrentes que se realizaron al prototipo. La prueba completa se encuentra en un video localizado en la página Web asignada del trabajo de grado.

```
run:
[QuemesAPP]: Iniciando sistema...
[QuemesAPP]: -> Iniciando contenedor PCEM...
CARGANDO ARCHIVO: /Volumes/Files HD/rickel/NetBeansProjects/PCEM/res/ConfigBESA.properties
[QuemesAPP]: -> Contedor BESA [OK].
[QuemesAPP]: -> Iniciando Agentes...
CARGANDO ARCHIVO: /Volumes/Files HD/rickel/NetBeansProjects/PCEM/res/ConfigBESA.properties
CARGANDO ARCHIVO: /Volumes/Files HD/rickel/NetBeansProjects/PCEM/res/ConfigBESA.properties
OK Conexion Base de datos Usuarios.
CARGANDO ARCHIVO: /Volumes/Files HD/rickel/NetBeansProjects/PCEM/res/ConfigBESA.properties
[QuemesAPP]: -> Agentes [OK].
```

Figura 49 - Salida de consola inicio de main

Inicialmente antes de realizar despliegue de la interfaz gráfica de usuario, tenemos que correr el *main* el cuál permite iniciar la estructura BESA. La estructura BESA es el componente más importante de nuestra aplicación. Es la que nos permite utilizar el modelo de agentes diseñado y desarrollado para el prototipo. Primero se debe inicializar el contenedor BESA, luego se debe crear la estructura de los agentes en la cuál se asocian sus comportamientos y guardas, y finalmente se crean los agentes. Como se ilustra en la figura 49, se observa la salida de la consola en el momento de iniciar el *main* de la aplicación.

Las salidas nos muestran la confirmación de cada uno de los procedimientos que se llevan a cabo al inicio del *main*. Las salidas "Iniciando sistema" e "Iniciando contenedor" nos indican que el sistema y el contenedor se están iniciando. El sistema carga el archivo *ConfigBESA.properties*, el cuál trae las configuraciones por defecto que va a utilizar el contenedor BESA. También puede observar la confirmación de que se realizó con satisfacción la conexión con la base de datos y la correcta inicialización de los agentes.

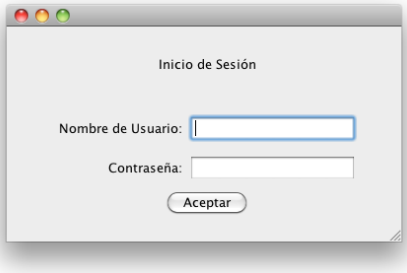

Figura 50 - Interfaz de inicio de sesión

Una vez que tenemos el contenedor de la estructura de BESA corriendo podemos iniciar la interfaz gráfica de usuario. Como se observa en la figura 50, la primera opción que nos ofrece la interfaz gráfica de usuario es la alternativa de hacer el inicio de sesión de usuario. Dependiendo del rol que el usuario tenga asignado, se le pondrán a disposición todos los servicios que este puede hacer.

Figura 51 - Interfaz del administrador del sistema

Cuando se realiza el inicio de sesión con el rol de administrador del sistema, el usuario tiene la posibilidad de realizar los servicios que se muestran el la figura 51. Como se explicó en el capítulo IV, el usuario consultor es el encargado de administrar las configuraciones de la plataforma. El usuario administrador del sistema tiene la capacidad de agregar y eliminar usuarios consultores, usuarios especialistas, sistemas expertos y, áreas de conocimiento.

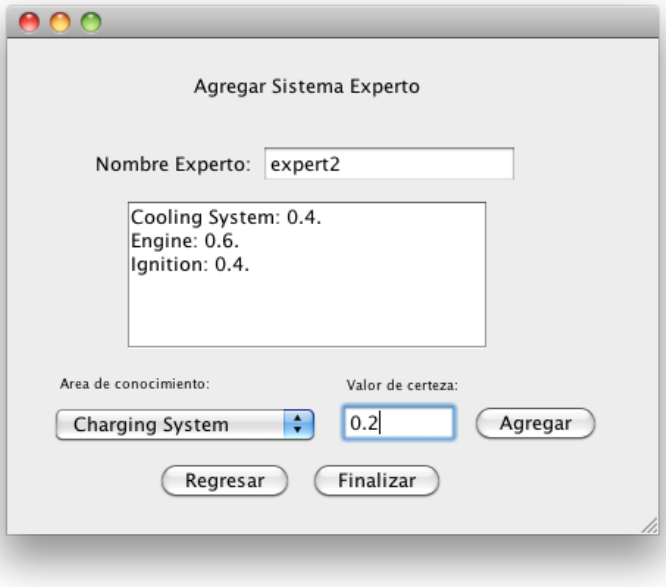

Figura 52 - Interfaz para agregar sistema experto

Una de las funciones más importantes del usuario administrador del sistema es la posibilidad de agregar sistemas expertos dinámicamente a la federación. El administrador del sistema debe asignar el nombre del experto. El nombre del experto es el nombre del archivo donde está almacenado el experto físicamente. En el caso que se ilustra en la figura 52 el nombre del

archivo CLIPS que contiene el sistema experto es *expert2.clp.* Archivo que debe estar físicamente ubicado en la carpeta principal de la aplicación. Además de asignarle el nombre del archivo al experto el administrador tiene la opción de escoger a partir de las áreas de conocimiento que están en la plataforma el puntaje que tiene el experto en cada área de conocimiento en particular.

La escala de puntuación que se le asigna a cada área de conocimiento es de 0.0 a 1.0. Donde 0.0 significa que el experto no sabe nada acerca del área de conocimiento y 1.0 que el sistema experto está totalmente calificado para responder preguntas acerca del tema. La puntuación que el administrador le da a las áreas de conocimiento es utilizada posteriormente para crear el filtro del agente de diagnóstico. Cuando se le han asignado todas las puntuaciones a las áreas de conocimiento que el experto tiene, se procede a crear el agente de diagnóstico. El agente de diagnóstico tiene almacenado en su estado el sistema experto seleccionado y el filtro que se le asignó.

Cuando se inicia la de sesión como usuario consultor la tarea más importante es empezar el diagnóstico. Una vez que ya seleccionó el sujeto a diagnosticar y ha llenado su historia, el usuario consultor debe seleccionar las áreas de conocimiento que necesita para realizar el diagnóstico como se ilustra en la figura 53. A partir de las áreas de conocimiento que el usuario seleccionó se creara un filtro el cuál será utilizado para seleccionar el sistema experto más adecuado para realizar el diagnóstico.

Figura 53 - Interfaz para iniciar diagnóstico

Cuando el usuario consultor inicia el proceso de diagnóstico el agente de sesión selecciona el *AgentDiagnosis* más adecuado para realizar el diagnóstico. Después de realizar la selección del experto, se inicia la interacción entre el *AgentDiagnosis* y el *AgentUserInterface*. Como se lustra en la figura 53 el *AgentDiagnosis* realiza preguntas efectuadas por el sistema experto al *AgentUserInterface* que es controlado por el usuario consultor. El usuario consultor debe responder las preguntas realizadas hasta que el sistema experto dictamine el diagnóstico.

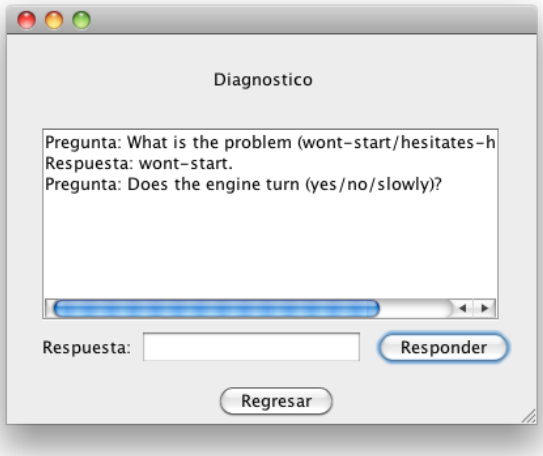

Figura 54 - Interfaz de diagnóstico

El proceso que se describió anteriormente utiliza todos los agentes del sistema. Esto implica la ejecución de las guardas más importantes cubriendo los objetivos principales de la plataforma. El proceso que se realizó para la prueba concurrente y que fue descrito anteriormente se ilustra en la figura 55. Podemos observar la creación de 3 agentes de diagnóstico con 3 sistemas expertos diferentes. Cada agente de diagnóstico con un filtro diferente. Los agentes de diagnóstico están identificados cada un con un ID diferente. Además, se muestra la conexión con la base de datos realizada por parte del agente historial. El agente de historial realiza la búsqueda del historial del sujeto a diagnosticar en la base de datos.

Las salidas que se muestran en la figura 55 demuestran el éxito del envío de mensajes entre los agentes que componen el sistema. También, se de muestra el correcto envío de la información necesaria para la funcionalidad del sistema. Las pruebas de concurrencia mostraron el correcto funcionamiento de la aplicación, utilizando todas las funciones en conjunto.

• 3.4 Pruebas de Estrés

La primera intención de las pruebas de estrés consiste en demostrar qué tan robusto, confiable y veloz es el sistema. Las pruebas que se diseñaron consisten principalmente en demostrar qué tan escalable es el sistema en permitir agregar agentes de diagnóstico a la federación. Las pruebas consisten en indicar cuánto tiempo se demora en realizar los diagnósticos dependiendo del número de agentes de diagnóstico que se encuentran registrados en la federación de la plataforma.

Como se menciono anteriormente, la variable que se va a incrementar es el número de agentes de diagnóstico. Mantendremos fijo el número de agentes de historial, de fallos, de sesión y de interfaz de usuario. Para comprobar la velocidad de respuesta del sistema, se contó el tiempo desde el momento que el agente de sesión recibe la petición del agente de la interfaz de usuario hasta, el momento que el agente de sesión le devuelve el diagnóstico.

```
run:
[QuemesAPP]: Iniciando sistema...
[QuemesAPP]: -> Iniciando contenedor PCEM...
CARGANDO ARCHIVO: /Volumes/Files HD/rickel/NetBeansProjects/PCEM/res/ConfigBESA.properties
[QuemesAPP]: -> Contedor BESA [OK].
[QuemesAPP]: -> Iniciando Agentes...
OK Conexion Base de datos Usuarios.
[QuemesAPP]: -> Agentes [OK].
[QuemesAPP]: Se envio DataSesionRequestAreas al agente: agentSesion01..
[QuemesAPP]: Se envio DataUserInterfaceResponseAreas al agente: agentUserInterface01..
[QuemesAPP]: Se envio DataSesionInsertAgentDiagnosis al agente: agentSesion01..
Se agrego nuevo Agente de diagnostico con area: Motor y id: PCEM_127.0.0.1_9876_4
Se creo el agente de diagnostico con el Alias: agentDiagnosis1
Sistema experto encontrado.
[QuemesAPP]: Se envio DataSesionInsertAgentDiagnosis al agente: agentSesion01..
Se agrego nuevo Agente de diagnostico con area: Motor y id: PCEM_127.0.0.1_9876_5
Se creo el agente de diagnostico con el Alias: agentDiagnosis2
Sistema experto encontrado.
[QuemesAPP]: Se envio DataSesionRequestAreas al agente: agentSesion01..
[QuemesAPP]: Se envio DataUserInterfaceResponseAreas al agente: agentUserInterface01..
[QuemesAPP]: Se envio DataSesionInsertAgentDiagnosis al agente: agentSesion01..
Se agrego nuevo Agente de diagnostico con area: Motor y id: PCEM_127.0.0.1_9876_6
Se creo el agente de diagnostico con el Alias: agentDiagnosis3
Sistema experto encontrado.
[QuemesAPP]: Se envio DataServiceLocatorRequestPatient al agente: agentServiceLocator01..
El paciente que se quiere cojer es: 1234
[QuemesAPP]: Se envio DataSesionRequestCreateAgentHistorial al agente: agentSesion01..
[QuemesAPP]: Se envio DataHistorialLoad al agente: agentHistorial01..
OK Conexion Base de datos Historia.
OK Conexion Base de datos Historia. Estatica
OK Conexion Base de datos Historia. Transversal: 1
[QuemesAPP]: Se envio DataServiceLocatorPatientInfo al agente: agentUserInterface01..
[QuemesAPP]: Se envio DataSesionSelectExpert  al agente: agentSesion01..
Agente: agentDiagnosis1 Score: 2.5
Agente: agentDiagnosis2 Score: 2.2
Agente: agentDiagnosis3 Score: 2.2
El agente de diagnostico que se escojio es: PCEM_127.0.0.1_9876_6
[QuemesAPP]: Se envio DataUserInterfaceSetAgentInCharge al agente: agentUserInterface01..
El agente con el que se debe comunicar es: PCEM_127.0.0.1_9876_6
[QuemesAPP]: Se envio DataDiagnosisStartDiagnosis al agente: PCEM_127.0.0.1_9876_6..
[QuemesAPP]: Se envio DataSesionRequestAgentFail al agente: agentSesion01..
[QuemesAPP]: Se envio DataHistoryRequestOnHistory al agente: agentHistorial01..
[QuemesAPP]: Se envio DiagnosticAnswer al agente: PCEM_127.0.0.1_9876_6..
[QuemesAPP]: Se envio DataHistoryRequestOnHistory al agente: agentHistorial01..
[QuemesAPP]: Se envio DataSesionRequestAgentFail al agente: agentSesion01..
[QuemesAPP]: Se envio DiagnosticAnswer al agente: PCEM_127.0.0.1_9876_6..
[QuemesAPP]: Se envio DataHistoryRequestOnHistory al agente: agentHistorial01..
[QuemesAPP]: Se envio DataSesionRequestAgentFail al agente: agentSesion01..
```

Figura 55 - Salida de la prueba concurrente

Para dicha prueba se tomó el tiempo actual del sistema en el momento de llegado de la petición y el tiempo actual del sistema en el momento de generar la respuesta. Luego, hacemos la resta del tiempo final con el tiempo inicial. De esta manera, obtenemos el tiempo total que se demora el sistema en devolver el diagnóstico. Para obtener el tiempo actual del sistema en cualquier momento se llamó el método de Java *System.currentTimeMillis()*.

Para evitar la interacción humana y así permitir el envío de ráfagas automáticas de mensajes a la velocidad máxima que el computador puede procesar, se decidió generar respuestas por defecto entre el agente de diagnóstico y el agente de la interfaz de usuario. Con esta solución podemos calcular con exactitud el tiempo de respuesta del sistema. Para la ejecución de dicha prueba se utilizó un sólo sistema experto con muchas instancias. Como el agente de sesión es el encargado de recibir la petición y devolver la respuesta, se modificó éste agente para que calcule el tiempo de respuesta. Los resultados de las pruebas son presentados en la siguiente sección.

Las pruebas se realizaron en un computador MacBook Pro, con procesador Intel Core2Duo de 2.26 Ghz y 4Gb de memoria RAM.

No. de *AgentDiagnosis*	Tiempo de Respuesta
10	944 MS
100	1915 MS
1000	4024 MS
10000	16908 MS
100000	59213 MS
1000000	329441 MS
10000000	1221409 MS
100000000	7288732 MS
1000000000	45992032 MS
10000000000	129833092 MS
100000000000	No respondió el Sistema

A medida que se iba incrementando el numero de *AgentDiagnosis* a la federación, el tiempo de respuesta del *AgentSesion* se incrementaba notablemente. A partir de la prueba que se utilizo 100,000,000 *AgentDiagnosis,* el equipo se volvió lento y las otras aplicaciones no respondían con la rapidez habitual. El consumo de RAM se incremento hasta un 43% y el procesador se mantuvo en 72%. Cuando se realizo la prueba con 100,000,000,000 *Agent-Diagnosis,* después de unos minutos el *NetBeans* dejo de funcionar. Se recurrió a cerrar la aplicación a la fuerza. Los resultados muestran que la aplicación es optima hasta 100,000,000 *AgentDiagnosis,* de ahí en adelante se vuelve inestable.

4. Resultados

A partir de las pruebas, las realizadas en la sección anterior, se presentan los resultados con respecto a la implementación de los diferentes agentes. Se realizan justificaciones sobre si

existe la necesidad o no de la implementación de cada uno de los agentes que hacen parte de la funcionalidad del sistema.

• 4.1 AgentSesion

La implementación del *AgentSesion* permitió tener un mecanismo de selección del Sistema Experto más apropiado para realizar la tarea designada. A partir de los filtros enviados por cada *AgentDiagnosis* al momento del registro de su registro, el *AgentSesion* realiza la selección del agente más apropiado para el diagnóstico por medio de un algoritmo de puntuación. Este agente le ahorra tiempo al sistema, debido a que no tiene que enviarle el diagnóstico a todos los agentes que estén registrados, solamente le envía el diagnóstico a los agentes que tienen conocimiento sobre ese tema. Los resultados de desempeño para el *AgentSesion* se muestran a continuación:

	No. de *AgentDiagnosis*	Tiempo de Respuesta
Consulta a todos	4	755 MS
Consulta al de mayor puntaje	4	315 MS

Tabla 1 - Desempeño AgentDiagnosis

Como se ilustra en la tabla 1, con un número de 4 *AgentDiagnosis* si se le envía la petición de diagnóstico a todos los agentes registrados el tiempo de respuesta del diagnóstico es de 755 milisegundos. Si sólo se le envía la petición de diagnóstico al agente con la puntuación más alta calculada por el *AgentSesion* el tiempo de respuesta del diagnóstico es de 315 milisegundos, 440 milisegundos menos. Estos resultados demuestran las ventajas al implementar el *AgentSesion,* el cuál permite realizar los diagnósticos en un menor tiempo.

• 4.2 AgentDiagnosis

Un *AgentDiagnosis* está encargado de controlar cada Sistema Experto que hace parte de la federación. Este agente se encarga de mantener la instancia del Sistema Experto permitiendo la adaptabilidad con el, realizando mínimas modificaciones para su funcionalidad con la plataforma. Para integrar un Sistema Experto con la plataforma solo hay que agregarle unos input readers, los cuales están encargados de atrapar el flujo emitido por un Sistema Experto y así el *AgentDiagnosis* puede utilizar ese flujo.

• 4.3 AgentHistory

La implementación del *AgentHistory* le permite al sistema reducir el tiempo de respuesta en los casos que exista información ya captada en la base de datos. Como el cuello de botella de la aplicación es la conexión de Internet y la base de datos está conectada remotamente a la aplicación, el *AgentHistory* solo hace un llamado a la base de datos para recolectar la historia del sujeto a diagnosticar. Para comprobar la funcionalidad del *AgentHistory* se realizó una

prueba realizando un diagnóstico. El Sistema Experto seleccionado tiene que hacerle 3 consultas al historial del sujeto. Si el *AgentHistory* no tuviera en su memoria la historia del sujeto se vería obligado a hacer las consultas en la base de datos una y otra vez, incrementando de esta manera el tiempo de respuesta de la plataforma. Los resultados del desempeño se muestran a continuación:

	No. De Consultas	Tiempo de Respuesta
Consulta Base de datos	3	11382 MS
Consulta en la memoria	3	315 MS

Tabla 2 – Desempeño AgentHistory

Como se observa en la tabla 1, cuando el *AgentHistory* realiza las consultas al historial directamente en la base de datos, el tiempo de respuesta del diagnóstico es de 11382 milisegundos. Esto se debe a que el *AgentHistory* no tenia cargado en su memoria el historial del sujeto y cada vez que el Sistema Experto realiza una pregunta, el agente de historial revisa en la base de datos si se encuentra la respuesta. El tiempo de respuesta de la consulta cuando el *AgentHistory* tiene la historia en su memoria es de 315 mili segundos, 11067 milisegundos menos que la otra prueba. Esto se debe a que el *AgentHistory* cargo de la base de datos el historial del sujeto al momento de ingresar el número de documento al inicio de la consulta. Este paso evito una futura consulta a la base de datos, lo cuál reduce notablemente el tiempo de respuesta de la consulta.

• 4.4 AgentUserInterface

El *AgentUserInterface* se encarga de controlar las interacciones de la interfaz gráfica de usuario con los otros agentes que hacen parte del sistema. El uso principal del *AgentUserInterface* es evitar que los otros agentes tengan control sobre la interfaz. El agente se encarga de enviar los mensajes y las interacciones que el usuario realiza y despliega las preguntas y resultados que los agentes emiten.

• 4.5 Agente de Fallos

La implementación del *AgentFault* le permite al sistema ser tolerable a fallos a nivel de los agentes. La principal meta de este agente es reducir el tiempo de caída de los componentes principales sistema. El *AgentFault* está constantemente enviándole mensajes a los demás agentes del sistemas, preguntándoles si están vivos. Cuando un agente no le responde al *AgentFault,* este toma las medidas apropiadas para reactivar el agente caído.

CONCLUSIONES

• Conclusiones del Estado del Arte:

La importancia de tener mecanismos de integración para los Sistemas Expertos actualmente existentes o futuros, lo cuál permite ofrecer áreas de experticia más grandes a la hora de solucionar problemas grandes o complejos.

La selección de los Agentes para la solución del problema propuesto, lo permite tener programas que puedan tomar decisiones por medio de mecanismos de razonamiento sencillos o complejos, comunicarse con otros agentes para obtener información, y actuar sobre el medio en el que se está desenvolviendo a través de sus ejecutores.

Los requerimientos más importantes de un sistema enfocado a la integración de Sistemas Expertos son: la posibilidad de tener un mecanismo apropiado para la selección del mejor Sistema Experto para la solución de un problema en particular, ofrecer mecanismos para la transferencia de información que ya está captada, y permitir la recuperación de historias.

• Conclusiones del Modelo:

La ventaja de utilizar un modelo basado en capas, nos permite tener las tres capas que conforman el sistema separadas una de otras.

La importancia de incluir en el modelo la posibilidad del manejo del historial de los sujetos, esto permite re-utilizar información acerca de los sujetos que ha sido captada anteriormente en otras consultas. Esto evita tener que re-capturar de nuevo información.

La importancia de incluir en el modelo la posibilidad del manejo del filtro de los Sistemas Expertos, le permite al sistema realizar una mejor selección del especialista más adecuado para realizar el diagnóstico al sujeto.

La importancia de incluir un módulo para utilizar un especialista humano. Se considera importante incluir este módulo debido a que no todos los Sistemas Expertos pueden dar respuesta a todos los diagnósticos. Esto le permite al sistema incluir especialistas humanos a la hora de responder a consultas.

• Conclusiones de la Implementación y el Prototipo

La utilización de la plataforma *BESA* para el desarrollo de los agentes del sistema. El uso de la plataforma simplificó la tarea de la implementación de los agentes que hacen parte del sistema.

La importancia de la implementación del *AgentSesion*. Se realizaron pruebas para medir el tiempo de respuesta del *AgentSesion* al momento de seleccionar el Sistema Experto más apropiado, donde se destacó que al implementar este agente, el sistema tuvo una notoria disminución en el tiempo de respuesta en comparación de que si no tuviera el mecanismo de selección.

La importancia de la implementación del *AgentHistory*. Se realizaron pruebas para medir el tiempo de respuesta de los diagnósticos cuando se utiliza. Las pruebas destacaron que al utilizar el agente permite tener un mecanismo el cuál disminuye notablemente el tiempo de respuesta del diagnóstico realizado por un *AgentDiagnosis*.

• Cumplimiento de los Objetivos:

Con base en los objetivos específicos que fueron establecidos para el trabajo de grado, se presentan a continuación los objetivos y se explica lo que fue realizado para conseguirlo.

Realizar un análisis del estado del arte de la aplicación de los sistemas expertos en el área médica y de modelos de integración y cooperación.

Se realizó el análisis del estado del arte, se hizo un análisis integral comparativo y se generaron una serie de oportunidades y conclusiones sobre él. Los cuales se describen en los Capítulos II y III.

Diseñar el modelo arquitectónico de la plataforma de colaboración entre sistemas expertos.

Se diseño el modelo arquitectónico de la plataforma de colaboración. El modelo incluye el modelo general de los componentes y el modelo de los agentes que integran el sistema. Se inicia con la descripción del modelo de negocio, el cual se describe en el capítulo IV y finaliza con la descripción de los modelos que se encuentran en los capítulos V y VI.

Desarrollar el programa implementando el diseño de la plataforma.

Se desarrolló un prototipo funcional a partir del diseño realizado. El cual se describe en el capítulo VII.

Validar en forma experimental la plataforma implementada.

Por medio de un plan de pruebas que incluye: pruebas unitarias, pruebas de circuitos típicos, pruebas concurrentes y pruebas de estrés. Se valido el funcionamiento del prototipo funcional. Las pruebas y los resultados son descritos en el capítulo VII.

• Contribuciones y Trabajos Futuros:

A partir del trabajo de grado propuesto se generó un modelo capaz de cumplir con los objetivos propuestos para la creación de una federación de Sistemas Expertos y especialistas humanos. El modelo aporta un valioso inicio a un campo que permite la integración de dife-

rentes especialistas lo que permite generar aplicaciones con un alto poder de cómputo a la hora de resolver tareas complejas en cualquier tipo de área.

El trabajo de grado propuesto generó serie de contribuciones y quedan abiertas una serie de oportunidades y perspectivas para el desarrollo de nuevas tecnologías que no sólo tienen que ver con la cooperación de Sistemas Expertos si no con otros temas que pueden cobijar diferentes herramientas. Los objetivos del trabajo de grado que no fueron realizados en su totalidad y quedan como perspectivas para trabajos futuros son:

Cooperación entre especialistas humanos y especialistas artificiales: brindar herramientas y servicios para facilitar la integración de diferentes fuentes de conocimiento, para así obtener una herramienta capaz de ofrecer servicios prestados por agentes o sistemas expertos artificiales y humanos especialistas como una sola gran herramienta con una gran variedad de conocimiento y diferente juzgamiento. Poder determinar cuando un experto o especialista pueda necesitar ayuda y así cooperar entre si.

Coordinación de tareas: capacidad de utilizar diferentes sistemas expertos para solucionar tareas en paralelo que forman una tarea principal y así reducir los tiempos de razonamiento y aumentar la productividad. Lo que permite repartirle una tarea a un experto que este especializado en un tema en particular y pueda resolver la tarea de manera mas optima.

Recuperación de historiales: poder ofrecer un servicio que se encargue de obtener la información de sujetos a diagnosticar de la base de datos, ordenarlos de la manera mas adecuada priorizando la información que pueda tener mas utilidad para los encargados de realizar los diagnósticos, y ponerlos a disposición a los interesados, para que puedan obtener de una manera mas eficaz información que pudo ser obtenida anteriormente por algún otro proceso o agente de diagnóstico.

Adaptación de Sistemas Expertos: ofrecer mecanismos para generar una fácil adaptación de sistemas expertos que quieran ser agregados a la federación. La principal idea de este trabajo de grado es permitir la cooperación entre sistemas expertos sin importar en que lenguaje estén programados o como estén programados. En este trabajo de grado se mostró como fue adaptado un sistema experto programado en CLIPS para que funcionara con los agentes programados en JAVA. Esto demuestra que puede haber una gran variedad de sistemas expertos para adaptar y ponerlos a funcionar.

Las perspectivas mencionadas anteriormente pueden ayudar a mejorar el diseño de la aplicación y expandir las funcionalidades y servicios que esta presta para ofrecer una herramienta capaz de brindar cooperación entre expertos y así cumplir los objetivos que fueron planteados inicialmente.

VIII - REFERENCIAS Y BIBLIOGRAFÍA

1. Referencias

[SHAA2004] K. Shaalan, M. El-Badry y A. Rafea. "**A multiagent approach for diagnostic expert systems via the internet**". Department of Computer Science, Cairo University. SHAA2004.

[SHEK1989] S. Shekhar. "**Coop: A Shell for Cooperating Expert Systems**". Computer Science division, University of California.

[HUNG2005] C. Wu. "**A multi-agent framework for distributed theorem proving**" Department of Electrical Engineering, National University of Kaohsiung.

[SHA2007] H. Qiu, X. Shao, P. Li y L. Gao. "**An agent- and service-based collaborative design architecture under a dynamic integration environment**". School of Mechanical Science and Engineering, Huazhong University of Science and Technology.

[RUFA1990] J. Rufat, G. Watts, O. Carey y W. Parrott. "**Distributed cooperative systems for advanced automation: Tradeoffs**". McDonnell Douglas Space Systems Company.

[SMIT2001] R. Smith, R. Davis. "**Frameworks for Cooperation in Distributed Problem Solving**". Defense Research Establishment Atlantic, Artificial Intelligence Laboratory.

[CHUN2007] H. Chu y G. Hwang. "**A Delphi-Based approach to developing expert systems with the cooperation of multiple experts".** Department of Information and learning Technology, National University of Tainan.

[GROS1990] C. Grossner y T. Radhakrishnan. "**Organizations for Cooperating Expert Systems**". Computer Science Dept. Concordia University.

[BASK1989] A. Baskin, S. Lu, R. Stepp y M. Klein. "**Integrated Design as a Cooperative Problem Solving Activity**". University of Illinois.

[GONZ2007] E. González y C. Bustacara. "**Desarrollo de aplicaciones basadas en sistemas multiagentes**". Pontificia Universidad Javeriana.

[KOUF2008] Koufi, V."**A medical diagnostic and treatment advice system for the provision of home care***"*. ACM.

[CAFE] http://www.cafesalud.com.co/marcos/productos.html

[BUCH2008] B. Buchanan y E. ShortLife. "**Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project** "

[GIAR2001] J. Giarratano. "**Sistemas expertos principios y programación***"*. International Thomson.

[KELE2008] A. Keles. ESTDD: **"Expert system for thyroid diseases diagnosis"**. Pergamon Press.

[CANO2003] C. Canosa. **"An intelligent system for the detection and interpretation of sleep apneas.***"* Elsevier Science.

[WANG2003] X.Wang. **"A self-learning expert system for diagnosis in traditional Chinese medicine".** Elsevier Science.

[MANG2005] A. Mangalampalli. **"A neural network based clinical decision-support system for efficient diagnosis and fuzzy-based prescription of gynecological diseases using homoeopathic medicinal system***"*. Elsevier Science.

[DOWI1988] J. Dowie. **"Progessional judgement: A reader in clinical decision making"** Cambridge University Press.

[JOHN1985] L. Johnson, E. Keravnou. **"Expert Systems Technology: A Guide"**. Cambridge: Abacus Press.

2. Bibliografía

[FRIE2003] E. Firedman. "**JESS In Action**". Manning