

**CENTRO DE INVESTIGACIÓN CIENTÍFICA Y DE EDUCACIÓN
SUPERIOR DE ENSENADA**



**DIVISIÓN DE FÍSICA APLICADA
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN**

**Administración del Conocimiento como Soporte al
Proceso de Mantenimiento del Software.**

TESIS

Que para cubrir parcialmente los requisitos necesarios para obtener el grado de
MAESTRO EN CIENCIAS

Presenta:

OSCAR MARIO RODRÍGUEZ ELIAS

Ensenada, Baja California, México. Agosto del 2003.

RESUMEN de la tesis de **OSCAR MARIO RODRÍGUEZ ELIAS**, presentada como requisito parcial, para la obtención del grado de **MAESTRO EN CIENCIAS** en **CIENCIAS DE LA COMPUTACIÓN**. Ensenada, Baja California, México. Agosto del 2003.

Administración del Conocimiento como Soporte al Proceso de Mantenimiento del Software.

Resumen aprobado por:

Dra. Ana Isabel Martínez García
Directora de Tesis

El conocimiento es uno de los recursos más valiosos de una organización. En particular, suele ser el mayor capital con el que pueden contar muchas organizaciones de desarrollo de software. Esta importancia del conocimiento ha llevado a muchas organizaciones a buscar maneras de administrarlo con el fin de darle el mejor uso posible, evitando su pérdida y desaprovechamiento. Entre los beneficios que las organizaciones han obtenido con la aplicación de la administración del conocimiento, se encuentran la reducción del tiempo, costos y recursos invertidos en sus procesos, así como el aumento en la calidad de sus productos y servicios.

Dentro de la ingeniería del software, el mantenimiento es la etapa que consume la mayor cantidad de recursos y tiempo dentro de las organizaciones de desarrollo de software. Sin embargo, los esfuerzos que estas últimas han hecho para aplicar la administración del conocimiento en sus procesos, se han orientado al resto de las etapas, sobre todo a las relacionadas con el desarrollo de nuevos productos. Esto ha llevado a la existencia de problemas relacionados con la pérdida y el desaprovechamiento del conocimiento dentro del proceso de mantenimiento del software.

En este trabajo se analiza la manera de apoyar en la solución de los problemas de pérdida y desaprovechamiento del conocimiento en el proceso de mantenimiento del software. Esto se logra al establecer elementos teóricos y tecnológicos que pueden ser utilizados en el desarrollo de herramientas para dar soporte a la administración del conocimiento dentro de los grupos de mantenimiento de software. Dentro de estos elementos, se define un modelo para catalogar los tipos y fuentes de conocimiento existentes en estos grupos; se describe un conjunto de escenarios que muestran la manera en que la administración del conocimiento puede ayudar en la solución de diversos problemas que se presentan a los encargados del mantenimiento de software; y se propone una arquitectura basada en agentes, cuyo fin es servir de base para el desarrollo de sistemas de administración del conocimiento en el mantenimiento del software.

Palabras clave: administración del conocimiento, mantenimiento del software, agentes de software, ingeniería del software, ingeniería del conocimiento.

ABSTRACT of the thesis of **OSCAR MARIO RODRÍGUEZ ELIAS**, presented as partial requirement, to obtain the **MASTER IN SCIENCIAS** degree in **COMPUTER SCIENCES**. Ensenada, Baja California, Mexico. August of 2003.

Knowledge Management as Support in the Software Maintenance Process.

Knowledge is one of the most important organizational resources. In particular, it is the most valuable capital for most of the software development organizations. The latter has taken many organizations to search for ways to manage their knowledge in order to make a better use of it, reducing its loss and waste. Within the benefits that knowledge management has provided for organizations, we can include the reduction of time, costs and resources invested in their processes, as well as the increase in the quality of their products and services.

In software engineering, the maintenance process consumes the greatest amount of resources and time in the software organizations. However, the efforts that these latter have made to apply knowledge management in their processes, have been oriented to the other stages of the software life cycle, mainly to those related to the development of new products; taking the software organizations to the loss and waste of knowledge on this process.

In this work, we analyze how to address the problems of loss and waste of knowledge in the software maintenance process by providing support to manage them. We do this by establishing theoretical and technological elements that can be applied to develop tools for knowledge management in software maintenance groups. Between these elements, a model to characterize the knowledge types and sources that exists in these kinds of groups, is introduced; some scenarios that show how knowledge management can help to solve diverse problems of software maintenance, are described; finally an agent based architecture oriented to be used as a framework in the development of knowledge management tools for software maintenance, is proposed.

Keywords: knowledge management, software maintenance, software agents, software engineering, knowledge engineering.

Dedicatoria

*A mis padres: Lourdes Elias y Oscar Mario Rodríguez,
quienes me enseñaron a forjar mi camino.*

*A mis hermanos: Andrés, Lulú, Malti y Jesús,
cada quien tiene sus sueños,
quiero ayudarlos a alcanzar los suyos.*

Agradecimientos

A Dios por darme vida y fuerza para alcanzar esta meta.

Un agradecimiento muy especial a mi directora de tesis, Dra. Ana Isabel Martínez García, que sin su gran apoyo y guía este trabajo no sería lo mismo.

A los miembros de mi comité de tesis: Dr. Jesús Favela Vara, Dra. Josefina Rodríguez Jacobo y Dr. Roberto Conte Galván, por sus valiosas aportaciones y el tiempo dedicado en la revisión de este trabajo.

Al Dr. Jesús Favela, por sus ideas y aportaciones que ayudaron a dar rumbo a la realización de esta tesis.

A la Dra. Aurora Vizcaino que aportó las ideas que dieron inicio a este trabajo.

Al Departamento de Informática del CICESE, y a Intersel, por prestarme su experiencia, tiempo y confianza.

A mi familia, pues sin su apoyo y cariño este logro hubiera sido muy difícil.

A mi novia, Lupita, por los hermosos días a su lado.

A mis amigos de las rondallas femenil y del desierto que me apoyaron en esta etapa. Sin esa historia mi vida no sería igual.

A mis compañeros y amigos de generación, que hicieron mi estancia más llevadera.

Al Centro de Investigación Científica y de Educación Superior de Ensenada (CICESE).

Al Consejo Nacional de Ciencia y Tecnología (CONACYT), por su apoyo económico.

Contenido

	Página
Capítulo I. Introducción	1
I.1 Antecedentes.....	1
I.2 Planteamiento del problema	3
I.3 Objetivos.....	4
I.4 Metodología.....	5
I.5 Contenido de la tesis.....	6
Capítulo II. Mantenimiento del Software.....	8
II.1 Definición de mantenimiento del software.....	8
II.1.1 El proceso de mantenimiento del software.....	11
II.1.2 Actores y roles en el mantenimiento del software.....	13
II.2 Importancia del mantenimiento del software	15
II.3 Problemas del mantenimiento del software.....	16
II.3.1 Aspectos socio-culturales del mantenimiento	16
II.3.2 Aspectos de la documentación de los sistemas	18
II.3.3 Pérdida y desaprovechamiento del conocimiento	19
II.4 Soporte al mantenimiento del software	21
II.4.1 Tipos de herramientas de soporte al mantenimiento del software	23
II.5 Resumen	26
Capítulo III. Administración del Conocimiento en el Mantenimiento del Software ..	28
III.1 Administración del conocimiento: una definición	29
III.1.1 Una definición del conocimiento.....	31
III.1.2 Mecanismos de conversión del conocimiento	32
III.2 Conocimiento en el mantenimiento del software	34
III.2.1 Fuentes de conocimiento	34
III.2.1.1 Documentación.....	35
III.2.1.2 Sistema o producto	36
III.2.1.3 Personas	37
III.2.2 Tipos de conocimiento requerido por los encargados del mantenimiento del software	38
III.2.2.1 Propuesta de un modelo para identificar el conocimiento en el mantenimiento del software.....	40
III.2.3 Flujo del conocimiento en el mantenimiento del software.....	42
III.3 Beneficios de la administración del conocimiento	45
III.4 Tecnologías del Conocimiento: TI en la administración del conocimiento	47
III.4.1 Una categorización de las técnicas y herramientas para la administración del conocimiento	48
III.4.1.1 Falta de apoyo en la conversión de conocimiento explícito en tácito	51
III.4.2 Agentes de software en la administración del conocimiento	52

Contenido (continuación)

	Página
III.4.2.1 Agentes de Software: definición y características	52
III.4.2.2 Sistemas basados en agentes dentro de las áreas de la administración del conocimiento.....	53
III.5 Administración del conocimiento en las organizaciones de desarrollo de software	54
III.5.1 Administración del conocimiento en el proceso de mantenimiento del software	56
III.6 Resumen	57
Capítulo IV. Casos de Estudio en dos grupos de Mantenimiento del Software	59
IV.1 Descripción del ambiente de los casos de estudio.....	60
IV.1.1 Departamento de informática del CICESE.....	60
IV.1.2 Intersel	61
IV.2 Metodología seguida durante los casos de estudio.....	63
IV.2.1 Descripción de la captura de los procesos de los grupos estudiados.....	64
IV.2.2 Técnica empleada para el modelado de los procesos de los grupos estudiados	67
IV.3 Modelado y evaluación de los procesos llevados a cabo por los grupos estudiados	68
IV.3.1 Procesos identificados durante los casos de estudio.....	69
IV.3.1.1 Proceso de atención a clientes	69
IV.3.1.2 Proceso de realización y atención de solicitudes de modificaciones.....	72
IV.3.1.3 Proceso de implementación de cambios y control de versiones.....	75
IV.3.1.4 Diferencias y similitudes relacionadas con el conocimiento	78
IV.3.2 Aspectos técnicos y sociales más relevantes identificados	80
IV.3.2.1 Aspectos relacionados con la documentación	80
IV.3.2.2 Aspectos relacionados con el conocimiento	82
IV.3.2.3 Aspectos relacionados con la organización interna de los grupos estudiados	86
IV.4 Escenarios.....	89
IV.5 Características básicas para una herramienta de administración del conocimiento en el mantenimiento del software	93
IV.5.1 Hacia una arquitectura de agentes para la administración del conocimiento en el mantenimiento del software.....	96
IV.6 Resumen	96
Capítulo V. Arquitectura de Agentes para la Administración del Conocimiento en el Mantenimiento del Software.....	98
V.1 Arquitectura.....	99
V.1.1 Especificaciones de diseño de la arquitectura	100
V.1.2 Diseño de la arquitectura	102
V.2 Prototipo	104

Contenido (continuación)

	Página
V.2.1 Escenario	105
V.2.2 Análisis y diseño del prototipo	107
V.2.2.1 Características de implementación del prototipo.....	108
V.2.2.2 Casos de uso identificados en el escenario, y sus diagramas de secuencia.....	109
V.2.2.3 Diagrama de clases del prototipo.....	116
V.2.3 Implementación del prototipo.....	121
V.2.3.1 Tecnologías utilizadas para la implementación del prototipo	121
V.2.3.2 Aspectos de la implementación de los elementos de la arquitectura de agentes	123
V.2.3.3 Manejo de la información y de la base de conocimientos	128
V.2.4 Funcionalidad básica del prototipo.....	133
V.3 Resumen	139
Capítulo VI. Evaluación del prototipo	141
VI.1 Guía para la realización de las pruebas	142
VI.1.1 Relación de la guía de pruebas con las características del prototipo.....	144
VI.2 Metodología para la realización de las pruebas: preparación.....	145
VI.3 Descripción del proceso de pruebas	149
VI.3.1 Inicialización del prototipo.....	149
VI.3.2 Solicitud de atención de proyectos.....	152
VI.3.3 Recuperación de las fuentes de conocimiento encontradas, y los datos de las mismas	156
VI.3.4 Almacenamiento de la solución dada al error reportado	160
VI.4 Discusión de los resultados de la etapa de pruebas	163
VI.4.1 Solución de las fallas detectadas durante la etapa de pruebas.....	166
VI.5 Resumen	168
Capítulo VII. Conclusiones, Aportaciones y Trabajo Futuro	170
VII.1 Conclusiones.....	170
VII.2 Aportaciones.....	172
VII.3 Logros adicionales.....	173
VII.4 Trabajo futuro.....	174
Literatura Citada.....	176
Ligas de Internet citadas	188
Apéndice A. Casos de Estudio	190
A.1 Caso de Estudio: Departamento de informática del CICESE.....	190
A.2 Caso de estudio: Intersel.....	207

Contenido (continuación)

	Página
Apéndice B. Análisis y Diseño del Prototipo.....	216
B.1 Diagramas de casos de uso	216
B.2 Diagramas de secuencia	218
B.3 Diagramas y descripción de clases	221
Apéndice C. Funcionalidad del Prototipo	234
C.1 Interfaz gráfica del agente de directorio.....	234
C.2 Interfaz gráfica del agente de producto.....	237
C.3 Aplicación para el manejo de la base de conocimientos	242
Apéndice D. Instalación, Configuración y Ejecución del Prototipo.....	245
D.1 Procedimiento de instalación.....	245
D.2 Configuración	247
D.3 Ejecución del prototipo.....	250

Lista de figuras

	Página
Figura 1. Fases genéricas del proceso de desarrollo de software.....	9
Figura 2. Mecanismos de conversión del conocimiento	33
Figura 3. Dominios de conocimiento en la construcción de sistemas (Curtis <i>et al.</i> , 1988).	39
Figura 4. El ciclo del conocimiento (Choo, 1999).	42
Figura 5. Modelo del flujo del conocimiento en el mantenimiento del software.....	43
Figura 6. Conversión de conocimiento en el modelo del flujo de conocimiento en el mantenimiento del software.....	44
Figura 7. Organización interna de Intersel	62
Figura 8. Simbología de las gráficas ricas.....	67
Figura 9. Proceso de atención a clientes en Intersel.....	70
Figura 10. Proceso de atención a clientes en el DI.....	71
Figura 11. Proceso de realización de una solicitud de modificaciones en Intersel	73
Figura 12. Elaboración del proyecto de modificaciones en Intersel	74
Figura 13. Proceso de solicitud de modificaciones en el DI	75
Figura 14. Identificación de las partes del sistema a modificar	76
Figura 15. Arquitectura de agentes para la administración del conocimiento en el mantenimiento del software.....	102
Figura 16. Diagrama del caso de uso Inicializar sistema.	110
Figura 17. Diagrama del caso de uso Atender proyecto.....	111
Figura 18. Tipos de interacciones utilizadas en los diagramas de secuencia.	112
Figura 19. Primer parte del diagrama de secuencia del caso de uso Inicializar sistema. ...	112
Figura 20. Segunda parte del diagrama de secuencia del caso de uso Inicializar sistema.....	113
Figura 21. Primer parte del diagrama de secuencia del caso de uso Atender proyecto. ...	114
Figura 22. Segunda parte del diagrama de secuencia del caso de uso Atender proyecto. ...	115
Figura 23. Diagrama de clases del prototipo.....	117
Figura 24. Diagrama de las clases de las acciones de los agentes.....	119
Figura 25. Diagrama de clases que muestra la relación del agente de personal y sus actividades principales.....	119
Figura 26. Diagrama de clases de la ontología de fuentes y tipos de conocimiento.	120
Figura 27. Ejemplo del funcionamiento de la plataforma JADE.	123
Figura 28. Representación de la ejecución del prototipo dentro de la plataforma JADE. ...	124
Figura 29. Representación de la búsqueda del servicio de directorio.	125
Figura 30. Ejemplo del proceso realizado al enviar un mensaje solicitando el registro de un agente de personal con el agente de directorio.....	126
Figura 31. Ejemplo de la estructura de un mensaje en JADE.	127
Figura 32. Diagrama de clases de la ontología del agente de directorio.....	128
Figura 33. Distribución de las colecciones de la base de datos del prototipo.	129

Lista de figuras (continuación)

	Página
Figura 34. Diagrama con las clases de las entidades de información de la base de datos.	130
Figura 35. Representación de los datos de un miembro del personal en formato XML. ...	130
Figura 36. Ejemplo de la representación en XML de una fuente de conocimiento de tipo personal.	131
Figura 37. Ejemplo de una consulta en lenguaje XPath.	132
Figura 38. Representación de las acciones del sistema al momento de inicializar el subsistema local del IM.	133
Figura 39. Interfaz gráfica del agente de personal.	135
Figura 40. Ventana con los datos del reporte de error.	136
Figura 41. Ventana para la captura de las soluciones de los errores reportados.	137
Figura 42. Ventanas de captura de archivos modificados y consultados.	137
Figura 43. Despliegue del mensaje de total de fuentes encontradas.	138
Figura 44. Ventana para el despliegue de fuentes de conocimiento encontradas.	139
Figura 45. Interfaz del manejador remoto de agentes.	146
Figura 46. Interfaz de monitoreo de mensajes entre agentes.	147
Figura 47. Mensaje de falla en la autenticación del usuario.	150
Figura 48. Muestra el resultado de la inicialización del prototipo.	151
Figura 49. Mensajes enviados por los agentes durante la inicialización del prototipo.	151
Figura 50. Muestra el resultado de la selección de un proyecto de la lista de proyectos de la interfaz de usuario del agente de personal.	153
Figura 51. Muestra el resultado de solicitar atender un proyecto en la interfaz de usuario del agente de personal.	154
Figura 52. Muestra el intercambio de mensajes entre el agente de personal y el agente manejador de conocimiento al momento de que el IM solicita atender un proyecto.	155
Figura 53. Muestra las fuentes de conocimiento encontradas , y los conceptos o temas de conocimiento buscados.	157
Figura 54. Muestra los datos de una fuente de conocimiento de tipo reporte de error. ...	158
Figura 55. Muestra el despliegue de los datos del reporte de error.	159
Figura 56. Mensaje enviado por el agente jua_n_km al agente jua_n_ksm para solicitarle que despliegue los datos de una fuente de conocimiento.	160
Figura 57. Muestra los datos de la solución antes de guardarlos.	161
Figura 58. Muestra la forma en que se almacenó el reporte de error en la base de datos.	162
Figura 59. Muestra la manera en que es almacenada la fuente de conocimiento relacionada con el reporte de error.	163
Figura 60. Distribución geográfica de los sistemas a los que da mantenimiento el DI.	193
Figura 61. Proceso de mantenimiento en el DI del CICESE.	194
Figura 62. Representación del proceso que se lleva al recibir solicitudes del usuario.	198
Figura 63. Representación de la manera en que los técnicos logran un entendimiento del proceso al que da soporte el sistema que mantienen.	199

Lista de figuras (continuación)

	Página
Figura 64. Representación del proceso de Solución de errores del sistema.....	200
Figura 65. Realización de modificaciones que pueden resultar en cambios grandes.....	202
Figura 66. Modelo de capas en el control de versiones en el DI del CICESE.....	205
Figura 67. Proceso que muestra la manera en que se lleva el control de las versiones de los archivos fuente.	206
Figura 68. Organización interna de Intersel.....	208
Figura 69. Flujo de comunicación hacia y desde el usuario en Intersel.....	209
Figura 70. Proceso de atención a usuarios en Intersel.....	210
Figura 71. Proceso de realización de una solicitud de modificaciones en Intersel.....	212
Figura 72. Elaboración del proyecto de modificaciones en Intersel.....	212
Figura 73. Caso de uso Visualizar fuentes encontradas.....	216
Figura 74. Caso de uso Visualizar datos de fuentes encontradas.....	217
Figura 75. Caso de uso Guardar solución.....	217
Figura 76. Tipos de comunicaciones usadas en los diagramas de secuencia.....	218
Figura 77. Diagrama de secuencia del caso de uso Visualizar fuentes encontradas.....	219
Figura 78. Diagrama de secuencia del caso de uso Visualizar datos de fuentes encontradas.....	220
Figura 79. Diagrama de secuencia del caso de uso Guardar solución.....	221
Figura 80. Diagrama de clases del prototipo.....	222
Figura 81. Diagrama de clases de las actividades realizadas por los agentes.....	225
Figura 82. Diagrama de clases de la ontología de conocimiento.....	228
Figura 83. Diagrama de clases de la ontología del agente de personal.....	231
Figura 84. Ontología del agente manejador de conocimiento.....	232
Figura 85. Diagrama de clases de la ontología del agente de proyecto.....	233
Figura 86. Interfaz gráfica del agente de directorio.....	234
Figura 87. Muestra los datos de los agentes seleccionados.....	235
Figura 88. Muestra el catálogo de miembros del personal.....	236
Figura 89. Ventana de captura de datos de miembros del personal.....	236
Figura 90. Confirmación de la eliminación de un miembro del personal, del catálogo.....	237
Figura 91. Interfaz gráfica del agente de producto.....	237
Figura 92. Edición de los datos del producto.....	238
Figura 93. Ventana para la captura de participantes del producto.....	239
Figura 94. Ventana para la captura de los datos de los módulos del producto.....	239
Figura 95. Ventana para la captura de datos de los documentos del producto.....	240
Figura 96. Muestra el catálogo de proyectos.....	240
Figura 97. Ventana para la captura de datos de los proyectos.....	241
Figura 98. Muestra la lista de reportes de error.....	242
Figura 99. Muestra el catálogo de tipos de conocimiento.....	243
Figura 100. Ventana para captura de datos de temas de conocimiento.....	243
Figura 101. Muestra el módulo para la asignación de conocimiento a las fuentes.....	244
Figura 102. Ventana para elegir el tipo de conocimiento que tiene o necesita una fuente.....	244

Lista de tablas

	Página
Tabla I. Etapas y actividades del proceso de mantenimiento del software (IEEE Std. 1219-1998).....	11
Tabla II. Entradas, controles y salidas de las distintas etapas del proceso de mantenimiento del software (IEEE Std. 1219-1998).....	13
Tabla III. Trabajo actual en el área de administración de configuración (Estublier, 2000).....	24
Tabla IV. Tipos de herramientas para el mantenimiento del software (Bennett y Rajlich, 2000).....	24
Tabla V. Tipos de herramientas de soporte automatizado para mejorar el código (<i>Ibid.</i>).....	25
Tabla VI. Generalización de los tipos de conocimiento que requiere el personal de mantenimiento.....	40
Tabla VII. Motivación para la administración del conocimiento en la ingeniería del software (Rus y Lindvall, 2002).	45
Tabla VIII. Categorización de los tipos de tecnologías para la administración del conocimiento (Marwick,2001).....	48
Tabla IX. Guía para la realización de las pruebas.	142
Tabla X. Relación entre las características y las pruebas definidas en la guía.	144
Tabla XI. Lista de fallas encontradas durante las pruebas.	165

Capítulo I. Introducción

El conocimiento ha venido a jugar un papel muy importante en las organizaciones modernas, convirtiéndose en uno de los recursos más valiosos con el que éstas pueden contar (Davenport y Prusak, 2000; Tiwana, 2000). Estudios realizados en el área de la administración del conocimiento, muestran que el proporcionar mecanismos que permitan a las organizaciones reducir la pérdida y el desaprovechamiento del conocimiento, puede contribuir en un aumento en la productividad, al reducir los costos y tiempos invertidos en la realización de las tareas, a la vez que se aumenta la calidad de los productos (Basili y Caldiera, 1995; Althoff *et al.*, 1999).

Dentro de las organizaciones de desarrollo de software, el mantenimiento es la etapa que consume la mayor parte del tiempo y recursos. Entre los principales problemas del mantenimiento se encuentran la pérdida y el desaprovechamiento del conocimiento con el que pueden contar los encargados de este proceso. Aún cuando este tipo de problemas han sido tratados por diversas organizaciones de software, por lo general el enfoque se orienta más hacia las etapas del desarrollo de nuevos sistemas, como lo son la administración de proyectos, análisis y diseño, pruebas, etc., que a la del mantenimiento.

Es por lo anterior que en este trabajo se estudia la manera de proveer a los encargados del mantenimiento de los beneficios que puede dar la administración del conocimiento, al buscar reducir los problemas de pérdida y desaprovechamiento del conocimiento que se da en los grupos de mantenimiento del software.

1.1 Antecedentes

Actualmente no existe una definición generalizada del concepto de administración del conocimiento (Barquin, 2001). Sin embargo, podemos decir que es un conjunto de prácticas que buscan el mejor aprovechamiento del conocimiento con el que puede contar una organización, tanto del conocimiento expresado de manera tangible (explícito), como el

existente en las habilidades y experiencia de cada persona (tácito). Todo esto con el fin de hacerlas más competitivas.

Las organizaciones dedicadas al desarrollo del software hacen un uso intensivo del conocimiento (Robillard, 1999; Kucza y Komi-Sirviö, 2001; Rus y Lindvall, 2002); debido a esto, muchas de ellas han hecho esfuerzos por aprovechar los beneficios de la administración del conocimiento en sus actividades, con el fin de reducir los tiempos y costos de sus procesos, a la par de aumentar la calidad de sus productos. No obstante que el mantenimiento es la etapa que consume la mayor parte del tiempo y recursos en este tipo de organizaciones (Lientz *et al.*, 1978; Parikh, 1985; Pigoski, 1997; Barry *et al.*, 1999; Polo *et al.*, 2002; Pressman, 2002), los esfuerzos antes mencionados se han orientado más hacia las tareas de los administradores, analistas, diseñadores y desarrolladores de proyectos nuevos, no así a los encargados de la etapa de mantenimiento.

Entre los principales problemas del mantenimiento de software reportados por la literatura, está la falta de documentación (Dart *et al.*, 1993; Singer, 1998). Esta falta de documentación provoca que los encargados del mantenimiento requieran de otras fuentes, como pueden ser los desarrolladores originales, pero en la medida en que los sistemas envejecen, cada vez es más difícil contar con la ayuda de éstos para las labores de mantenimiento (Thomsett, 1998). Esto último es particularmente cierto si consideramos que muchos de los sistemas a los que se da mantenimiento tienen más de 10 años de haberse desarrollado (Osborne y Chikofsky, 1990).

Como consecuencia de lo anterior, los encargados del mantenimiento del software se basan en su experiencia personal. Sin embargo, con frecuencia ésta no es suficiente, debido a que es común que sean desarrolladores recién egresados de las escuelas o universidades quienes están a cargo de las labores de mantenimiento. Además, aun cuando pudiera pensarse que, una vez que los desarrolladores obtengan la experiencia suficiente para poder dar mantenimiento a un determinado sistema, los altos costos en tiempo y recursos asociados disminuirán, esto no resulta así. Entre las causas de lo anterior está la frecuente rotación de

personal que se da entre los grupos de desarrollo (Lientz, 1983). Esto ocasiona pérdida de conocimiento, así como el tener que lidiar con desarrolladores sin experiencia, ya que si los desarrolladores experimentados se van, el conocimiento se va con ellos: *“al mismo ritmo que la experiencia camina hacia afuera de la puerta, la inexperiencia camina hacia adentro”* (Rus y Lindvall, 2002).

Otro de los problemas relacionados con el conocimiento es el desaprovechamiento del mismo. *“Con frecuencia las organizaciones no saben lo que realmente saben”* (Tiwana, 2000). Es frecuente que dentro de una organización, existan personas con el conocimiento o experiencia suficiente para resolver un determinado problema, pero si no tenemos conocimiento de lo que los demás saben, difícilmente los consultaremos. Este tipo de escenarios es común que se presente dentro de los grupos de mantenimiento del software.

Todo lo anterior hace evidente la necesidad de poner a disposición de los encargados del mantenimiento del software, mecanismos que les permitan disminuir la pérdida y desaprovechamiento de todo aquel conocimiento que les pueda ser de utilidad para sus actividades. Viendo que este tipo de problemas son similares a los tratados por medio de la administración del conocimiento, se propone su utilización en el proceso de mantenimiento del software, como un medio para reducir algunos de los problemas existentes en dicho proceso.

1.2 Planteamiento del problema

Los trabajos desarrollados dentro de las áreas que comprende la administración del conocimiento permiten atacar distintos problemas asociados al conocimiento que se dan en el proceso de mantenimiento de software. Sin embargo, los trabajos existentes son generales, lo que provoca que varias de las características inherentes al mantenimiento del software no sean tomadas en cuenta. El mantenimiento del software presenta diferencias significativas con respecto al desarrollo de nuevos sistemas, por ejemplo, la necesidad de tratar con código desarrollado por otras personas (Banker *et al.*, 1993). Existen estudios que

muestran que los encargados de la etapa de mantenimiento requieren distintas herramientas a las del resto de las etapas de la ingeniería del software (Dart *et al.*, 1993). Esto nos lleva a la necesidad de una herramienta que de soporte a los problemas relacionados con el conocimiento en el proceso de mantenimiento del software, desde un punto de vista particular de las necesidades de los encargados del mismo.

Es por esto que vemos necesario buscar la manera de adaptar las nuevas tecnologías que están siendo desarrolladas en el área de administración del conocimiento para solucionar los problemas de pérdida y desaprovechamiento del conocimiento en el mantenimiento del software. Esto nos lleva a plantear las siguientes preguntas:

- ¿De qué manera la administración del conocimiento puede ayudar a facilitar el trabajo del personal encargado del mantenimiento del software?
- ¿Es posible desarrollar una herramienta que dé soporte a la administración del conocimiento en el mantenimiento del software?

Para responder estos cuestionamientos, se ha propuesto la realización de un estudio encaminado a establecer elementos que permitan identificar cómo la administración del conocimiento puede ayudar a dar soporte a algunos de los problemas que se les presentan a los encargados del mantenimiento del software. Con el fin de establecer estos elementos se definieron algunos objetivos que dieron rumbo a la realización del presente trabajo. Estos objetivos se listan a continuación.

1.3 Objetivos

El objetivo principal de este trabajo es analizar cómo la administración del conocimiento puede ayudar a disminuir algunos de los problemas existentes en el mantenimiento del software, para lo cual se han definido los siguientes objetivos específicos:

- Identificar cuáles son los problemas principales asociados al mantenimiento de software.
- Identificar qué métodos y técnicas se han utilizado dentro del área de la administración del conocimiento.
- Identificar escenarios que muestren la manera en que la administración del conocimiento puede ayudar en la solución de los problemas que se les presentan a los encargados del mantenimiento del software.
- Identificar de qué manera los avances en la administración del conocimiento pueden aplicarse a éste tipo de escenarios.
- Identificar qué tipo de soporte se puede proporcionar a los grupos de desarrollo encargados del mantenimiento de software para que tengan acceso al conocimiento que requieren para realizar sus tareas.

1.4 Metodología

La realización de este trabajo de tesis consistió de diversas etapas. Primeramente se hizo una revisión bibliográfica para identificar los principales aspectos relacionados, tanto con el mantenimiento del software, como con la administración del conocimiento y la relación entre estas dos áreas. Estos aspectos incluyeron la problemática existente, así como el estado actual en ambas áreas. Posteriormente se realizaron dos casos de estudio en dos grupos de mantenimiento de software, orientados a constatar y complementar la información obtenida durante la revisión de la literatura con un ambiente real de trabajo.

Los objetivos de la realización de los casos de estudio también incluyeron la identificación de escenarios para apoyar en la obtención de las características que debe cubrir una herramienta de administración del conocimiento en el mantenimiento del software. Con base en estas características se diseñó una arquitectura basada en agentes de software, cuyo propósito es servir como el punto de partida para el desarrollo de sistemas de soporte a la administración del conocimiento en el mantenimiento del software. Posteriormente se desarrolló un prototipo basado en la arquitectura propuesta, orientado a validar la viabilidad

de la arquitectura para servir como base en el desarrollo de sistemas de administración del conocimiento en el mantenimiento del software. Finalmente se realizó un conjunto de pruebas encaminadas a validar si el prototipo permite dar seguimiento al tipo de escenarios que dieron paso a los requerimientos, tanto para el diseño de la arquitectura, como para el diseño del prototipo.

1.5 Contenido de la tesis

El contenido de esta tesis se compone de siete capítulos y cuatro apéndices, que se explican brevemente a continuación.

En el capítulo II se introduce al área del mantenimiento del software con el fin de establecer un marco de referencia para entender el contexto de la realización de este trabajo. Primeramente se define el concepto de mantenimiento del software y se presenta una generalización del proceso, así como de los actores y roles del mantenimiento. En seguida se introduce la importancia del mantenimiento dentro de las organizaciones de desarrollo de software, para después mostrar algunos de los principales problemas que han sido reportados por la literatura. Por último, se presentan los distintos mecanismos, técnicas y herramientas que se han empleado para dar soporte al proceso de mantenimiento del software.

En el capítulo III se discuten distintos elementos que permiten entender la manera en que la administración del conocimiento puede ayudar a solucionar algunos de los problemas del mantenimiento del software. Primeramente se definen los conceptos principales involucrados dentro del área de administración del conocimiento. En seguida se propone un modelo para categorizar el tipo y fuentes de conocimiento requerido por los encargados del mantenimiento del software, así como para ilustrar el flujo del conocimiento dentro este proceso. Posteriormente se describen los principales beneficios que se busca obtener con los mecanismos de administración del conocimiento, así como los principales tipos de herramientas que son utilizadas para lograr estos beneficios. Por último se muestran

algunos de los trabajos sobre administración del conocimiento que se han desarrollado dentro del área de ingeniería del software.

En el capítulo IV se presenta la realización de dos casos de estudio. Primeramente se da una descripción de los dos grupos que estuvieron bajo estudio, así como del estado de sus procesos y problemática existente. Posteriormente se muestran una serie de escenarios que permiten identificar la manera en que una herramienta de administración del conocimiento puede ayudar a reducir algunos de los problemas encontrados. Finalmente se presentan las características identificadas como básicas para el desarrollo de una herramienta que de soporte a la administración del conocimiento en el proceso de mantenimiento del software.

En el capítulo V se presenta la propuesta de una arquitectura de agentes de software que sirva de base para el desarrollo de sistemas de administración del conocimiento en el mantenimiento del software. Además, se describe el análisis, diseño e implementación de un prototipo que tuvo el fin de validar la viabilidad del desarrollo de sistemas basados en la arquitectura propuesta.

En el capítulo VI se presentan las pruebas realizadas al prototipo para validar la funcionalidad del mismo; así como los resultados obtenidos.

En el capítulo VII se presentan las conclusiones de este trabajo, las principales aportaciones y algunas propuestas de trabajo futuro.

Finalmente, el apéndice A presenta información de los casos de estudio realizados, el apéndice B el diseño del prototipo desarrollado, el apéndice C la funcionalidad de los elementos implementados como apoyo para el funcionamiento del prototipo, y el apéndice D la instalación, configuración y ejecución del prototipo.

Capítulo II. Mantenimiento del Software

“El cambio es vida, y no hay vida sin problemas”

Ichak Adizes

Pressman menciona que *“el cambio es algo inevitable cuando se construyen sistemas basados en computadoras”* (Pressman, 2002). Los sistemas de software son conocidos por ser altamente flexibles. El cambio es una de las características fundamentales del software, y la etapa de la ingeniería del software que se encarga de lidiar con esta evolución es el mantenimiento. Por lo tanto, podemos decir que lo que permite que el software continúe funcionando, es el mantenimiento. Un sistema puede no sobrevivir por mucho tiempo si no evoluciona; si no se mantiene con las condiciones necesarias para su funcionamiento.

En este capítulo se define el concepto de mantenimiento del software, se presenta una generalización del proceso, así como de los actores y roles del mantenimiento. Se introduce la importancia del mantenimiento dentro de las organizaciones de desarrollo de software. Se identifican algunos de los principales problemas que han sido reportados por la literatura. Y por último, se presentan los distintos mecanismos, técnicas y herramientas que se han empleado para dar soporte al proceso de mantenimiento del software.

II.1 Definición de mantenimiento del software

En su libro sobre ingeniería de software, Pressman menciona que el proceso de ingeniería del software puede dividirse en tres fases genéricas (figura 1), donde la primera, que es la definición, consiste en determinar qué es lo que se desea realizar, y conlleva actividades de análisis y planeación. La segunda etapa consiste en el desarrollo, en determinar cómo se realizará el sistema. Esta etapa abarca actividades como el diseño, la codificación, implementación y pruebas del sistema. Por último, la tercera etapa es el mantenimiento, durante la cual se da soporte a los cambios que pueden surgir a lo largo del tiempo de vida

útil del sistema, como lo son la corrección de errores, adaptaciones, modificaciones y mejoras al sistema (Pressman, 2002).

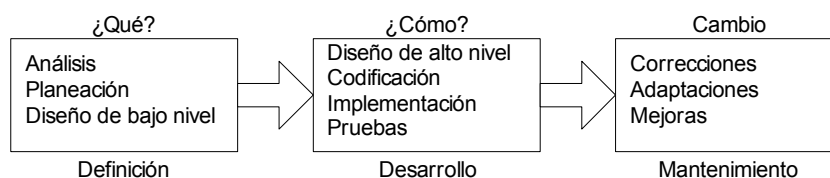


Figura 1. Fases genéricas del proceso de desarrollo de software.

Aunque esto da una idea de lo que es el mantenimiento del software, no lo define por sí solo. El estándar sobre mantenimiento del software de la IEEE lo define como: “*las modificaciones que se hacen al software después de haber sido liberado, para corregir fallas, mejorar el rendimiento, o para adaptarlo a los cambios del medio ambiente*” (IEEE Std. 1219-1998). Esto propone que el mantenimiento puede dividirse en tres tipos:

- *Mantenimiento correctivo*. Es el proceso de diagnosticar y corregir los errores que se encuentran durante la utilización del software.
- *Mantenimiento adaptativo*. Son actividades encaminadas a mantener el software a la par con los cambios del medio ambiente, como podrían ser los avances tecnológicos, nuevos paradigmas, hardware, equipo y sistemas en general.
- *Mantenimiento perfectivo*. Se encarga del mejoramiento del software: agregar nuevas características, mejorar las ya existentes, etc.

Esta definición no se preocupa por el mantenimiento del software sino hasta después de que éste ha sido totalmente terminado, sin embargo, como lo mencionan Osborne y Chikofsky, “*es esencial un modelo del ciclo de vida que considere el manejo y cambio de los sistemas de software. Esto significa tener en cuenta el mantenimiento en todos los aspectos del proceso de desarrollo*” (Osborne y Chikofsky, 1990). Es decir que es necesario considerar que el sistema tarde o temprano requerirá ser modificado, por lo que se requiere hacer que el software sea fácil de mantener. Esta idea ha llevado al nacimiento del concepto de mantenibilidad del software (Andersson y Eriksson, 1991; Ramage y Bennett, 1998), definido en (IEEE Std. 610.12-1990) como la facilidad con que un sistema o componente

de software puede ser modificado para corregir fallas, mejorar el rendimiento u otros atributos, o adaptarlo a los cambios del medio ambiente. A su vez, el hacer que el software existente cumpla con esta propiedad, ha llevado a un cuarto tipo de mantenimiento:

- *Mantenimiento preventivo.* Hartman y Robson lo definen como modificaciones hechas al sistema para facilitar el mantenimiento futuro (Hartmann y Robson, 1990). En (IEEE Std. 1219-1998) lo definen como el mantenimiento realizado con el propósito de prevenir problemas antes de que ocurran. El mantenimiento preventivo puede ser visto como un esfuerzo por predecir y corregir futuros posibles fallos o cambios en los requerimientos del software, antes de que estos ocurran. Son modificaciones hechas al software con una visión orientada a facilitar su futuro mantenimiento.

Por otra parte, el mantenimiento también puede catalogarse tomando en cuenta la urgencia de los cambios a realizar. Esto nos lleva a dividir el mantenimiento en dos categorías:

- *Mantenimiento urgente.* En (IEEE Std. 1219-1998) lo definen como mantenimiento correctivo no planeado, realizado para mantener un sistema en funcionamiento. Este tipo de mantenimiento se genera por situaciones que requieren una acción rápida, como errores graves en el sistema que deben ser corregidos al instante. Debido a esto no puede realizarse un plan, ya que las modificaciones deben ser llevadas a cabo en cuanto se presente el problema.
- *Mantenimiento no urgente.* Esta categoría abarca todas aquellas tareas de mantenimiento que no tienen fuertes restricciones de tiempo, por lo que pueden planearse antes de iniciarlas.

Analizando lo anterior, podemos decir que el mantenimiento del software se encarga de los cambios hechos al software después de su liberación, ya sea para corregir errores, mejorarlo, adaptarlo a los cambios del medio ambiente, o hacerlo más fácil de mantener. Una vez definido este concepto, nos enfocaremos a determinar cuales podrían ser las tareas principales, así como los actores y los roles que estos juegan durante el proceso de mantenimiento.

II.1.1 El proceso de mantenimiento del software

Debido a las diferencias que pueden existir en cada organización, es difícil definir un proceso de mantenimiento que sea análogo a cualquier tipo de organización. Por lo anterior, hemos tomado el proceso propuesto por el estándar sobre mantenimiento del software de la IEEE (IEEE Std. 1219-1998) para dar una idea de las actividades que pueden constituir el proceso de mantenimiento del software. Este estándar propone una serie de actividades agrupadas en siete etapas, las cuales se muestran en la tabla I.

Tabla I. Etapas y actividades del proceso de mantenimiento del software (IEEE Std. 1219-1998).

<p>Identificación, clasificación y asignación de prioridad al problema o modificación</p> <ol style="list-style-type: none"> 1. Asignar un número de identificación 2. Clasificar el tipo de mantenimiento 3. Analizar la modificación para determinar si se acepta, rechaza o se evalúa posteriormente 4. Hacer una estimación preliminar del tamaño y magnitud de la modificación 5. Asignar prioridad a la modificación 6. Asignar una solicitud de mantenimiento (SM) a un bloque de modificaciones calendarizadas para su implementación
<p>Análisis</p> <ol style="list-style-type: none"> 1. Realizar un análisis de factibilidad, y un reporte de factibilidad (RF) con los siguientes datos: <ul style="list-style-type: none"> Impacto de la modificación Solución alternativa, incluyendo prototipo Análisis de la conversión de requerimientos Implicaciones en la seguridad Factores humanos Costo a largo y corto plazo Beneficios de hacer la modificación 2. Análisis detallado, que incluye: <ul style="list-style-type: none"> Definir los requerimientos para la modificación Identificar los elementos a modificar Identificar aspectos de la seguridad Idear estrategias de pruebas Plan de desarrollo e implementación
<p>Diseño</p> <ol style="list-style-type: none"> 1. Identificar módulos del software afectados 2. Modificar la documentación de los módulos de software 3. Crear casos de prueba para el nuevo diseño, incluyendo aspectos de seguridad 4. Identificar y crear pruebas de regresión 5. Identificar documentación (del sistema y usuario) y actualizar requerimientos 6. Actualizar la lista de modificaciones

Continuación Tabla I. Etapas y actividades del proceso de mantenimiento del software (IEEE Std. 1219-1998).

Implementación
<ol style="list-style-type: none"> 1. Codificar modificaciones y probar 2. Integración 3. Análisis de riesgos 4. Revisión de pruebas preparadas
Pruebas del sistema
<ol style="list-style-type: none"> 1. Pruebas de funcionalidad del sistema 2. Pruebas de las interfaces 3. Pruebas de regresión 4. Revisar las pruebas de aceptación preparadas
Pruebas de aceptación
<ol style="list-style-type: none"> 1. Realizar pruebas de aceptación a nivel funcional 2. Realizar pruebas de interoperabilidad 3. Realizar pruebas de regresión
Liberación
<ol style="list-style-type: none"> 1. Conducir una auditoría de configuración física 2. Notificar a la comunidad de usuarios 3. Archivar una versión del sistema para respaldo 4. Realizar la instalación y dar entrenamiento para los encargados de atender al cliente

Además de las actividades, el estándar define una serie de elementos de entrada y salida, así como algunos controles para cada una de estas etapas. La tabla II presenta estos elementos.

Es necesario tener en cuenta que estas actividades y elementos del proceso de mantenimiento sólo son sugerencias. Es decisión de cada organización definir qué actividades son relevantes, así como los elementos que participarán en éstas. Sin embargo, lo que aquí se muestra es una idea de lo que resulta recomendable considerar con el fin de tener un mejor control del proceso de mantenimiento. Este estándar propuesto por la IEEE, puede servir de guía para todas aquellas organizaciones que deseen tener un proceso de mantenimiento bien definido, de manera que permita la mejora continua del mismo.

Tabla II. Entradas, controles y salidas de las distintas etapas del proceso de mantenimiento del software (IEEE Std. 1219-1998).

Etapas	Entradas	Controles	Salidas
Identificación del problema	Solicitud de modificaciones (SM)	Identificador único de SM Entrada de la SM al repositorio	SM validada Proceso determinado
Análisis	Documentos del proyecto o sistema Repositorio de información SM validada	Revisión técnica Verificación de Estrategias de prueba Documentación actualizada Identificación de aspectos de seguridad	Reporte de factibilidad (RF) Reporte de análisis detallado Requerimientos actualizados Lista de modificaciones preliminares Plan de implementación Estrategias de pruebas
Diseño	Documentación del proyecto o sistema Código fuente Bases de datos Salida de la etapa de análisis	Revisión e inspección del software Verificación de diseño	Lista de modificaciones revisada Análisis detallado revisado Plan de implementación revisado Línea base del diseño actualizada Plan de pruebas actualizado
Implementación	Código fuente Documentación del sistema o producto Resultados de la etapa de diseño	Revisión e inspección del software Verificación del control de la administración de configuración del software Verificación de la trazabilidad del diseño	Software actualizado Documentos de diseño actualizados Documentos de pruebas actualizados Documentos de usuario actualizados Material de entrenamiento actualizado Reporte de la revisión de la preparación de las pruebas actualizado
Pruebas del sistema	Documentación actualizada del software Reporte de la revisión de la preparación de pruebas Sistema actualizado	Control de la administración de configuración de: Código Listados SM Documentación de pruebas	Sistema probado Reportes de prueba
Pruebas de aceptación	Reporte de la revisión de la preparación de pruebas Sistema totalmente integrado Planes, casos y procedimientos de las pruebas de aceptación	Pruebas de aceptación Auditoría funcional Establecimiento del sistema base	Nueva línea de base del sistema Reporte de pruebas de aceptación Reporte de la auditoría de configuración funcional
Liberación	Sistema probado y aceptado	Auditoría de configuración física Documento de descripción de versión	Reporte de auditoría de configuración física Documento de descripción de versión

II.1.2 Actores y roles en el mantenimiento del software

Los actores y los roles que juegan durante el proceso de mantenimiento pueden variar considerablemente de una organización a otra. Sin embargo, hay estudios que permiten hacer una identificación general del tipo de actores y roles involucrados. Polo *et al.* se basan en el estándar sobre mantenimiento de software de la IEEE (IEEE Std. 1219-1998) para definir tres tipos de actores (Polo *et al.*, 1999):

- *Cliente.* El dueño del software, y quien hace la solicitud de mantenimiento.
- *Personal de mantenimiento.* Quienes realizan el mantenimiento.
- *Usuario.* Quien hace uso del sistema al que se dará el mantenimiento.

Ellos hacen una categorización de 8 perfiles o roles desempeñados por estos actores. Tres corresponden al cliente, cuatro al personal de mantenimiento, y uno al usuario. Antes de presentar estos perfiles es necesario aclarar que están basados en un esquema que considera que el cliente y el personal de mantenimiento son dos organizaciones distintas, y que el usuario se encuentra dentro de la organización cliente. Esto, sin embargo, no siempre resulta así, como se verá más adelante en esta tesis. Los tres perfiles desempeñados por el cliente son:

- *Solicitante del mantenimiento*. Es quien hace la petición de mantenimiento, e indica los requerimientos.
- *Organización del sistema*. Se refiere a la persona que tiene conocimiento del sistema que requiere mantenimiento. Conoce la manera en que éste debe funcionar.
- *Atención a usuarios*. Hay organizaciones que tienen personas encargadas de la atención a los usuarios de los sistemas con los que cuentan o distribuyen. Estas personas informan, de los incidentes reportados por los usuarios, a la persona encargada de hacer la solicitud de cambios al personal de mantenimiento.

Los cuatro perfiles del personal de mantenimiento son:

- *Manejador de solicitudes de mantenimiento*. Se encarga de recibir las solicitudes, y determinar si éstas se aceptan o rechazan, así como el tipo de mantenimiento que se realizará. Posteriormente turna la solicitud al encargado de realizar el plan de trabajo.
- *Calendarizador*. Se encarga de establecer los planes a seguir para la realización de las modificaciones.
- *Equipo de mantenimiento*. Son las personas encargadas de implementar los cambios solicitados. El equipo de mantenimiento puede estar constituido, a su vez, de distintos roles, los cuales dependen del tipo de organización interna que tenga el mismo.
- *Jefe del equipo de mantenimiento*. Es la persona que establece los estándares o procedimientos a seguir en las tareas de mantenimiento.

Por último, el octavo perfil lo constituye el *usuario*, que es el que utiliza el sistema que requiere mantenimiento. Los usuarios se encargan de informar los incidentes con el sistema al encargado de la atención a usuarios.

II.2 Importancia del mantenimiento del software

Para definir el grado de importancia que puede tener el mantenimiento del software, una buena idea es el determinar cuanto puede costar a las organizaciones. Desde hace varios años han habido autores que han dado cifras al respecto, por ejemplo:

“Estimaciones del total de recursos de sistemas y programación consumidos varían en rangos tan altos como el 75-80 %” (Lientz et al., 1978).

“La mayoría de los programadores gastan el 50% (y en algunos casos el 80%) de su tiempo en mantenimiento... En el ciclo de vida del software, el nuevo desarrollo es solo el 33%, el restante 67% es mantenimiento” (Parikh, 1985).

“El mantenimiento del software existente puede dar cuenta de más del 60% de las inversiones efectuadas por una organización de desarrollo” (Pressman, 2002).

Para entender por qué se consumen esta gran cantidad de recursos durante el mantenimiento del software, podemos volver la vista a los aspectos que involucra el mantenimiento. Tomando en cuenta que el mantenimiento abarca todos los cambios que se le pueden hacer a un sistema una vez que este ha sido liberado, una pregunta lógica puede ser ¿cuánto tiempo puede durar en funcionamiento un sistema? Muchos de los sistemas que actualmente se encuentran en funcionamiento tienen un promedio de vida de más de 10 o 15 años (Osborne y Chikofsky, 1990); y para que estos sistemas hayan podido vivir tanto tiempo, ha sido necesario darles mantenimiento para adaptarlos a los cambios tecnológicos que se han dado a través de todo este tiempo. Es de aquí que se deriva la gran cantidad de recursos y esfuerzos dedicados al mantenimiento del software.

Lo anterior nos puede llevar a considerar que el mantenimiento debe ser un factor de suma importancia para las organizaciones, sin embargo, como se menciona a continuación, uno

de los problemas más comunes del mantenimiento, es que no se le da la suficiente importancia (Lientz y Swanson, 1978; Dart *et al.*, 1993; Thomsett, 1998).

II.3 Problemas del mantenimiento del software

No obstante la gran cantidad de recursos que consume el mantenimiento, es frecuente la falta de interés con respecto a éste (Dart *et al.*, 1993; Thomsett, 1998). Además, aún cuando es sabido que muchos sistemas pueden durar en funcionamiento más de una década (Osborne y Chikofsky, 1990), la mayoría son desarrollados sin pensar en que estos requerirán cambios con el tiempo. En este sentido existe una gran variedad de problemas relacionados al mantenimiento del software (Lientz *et al.*, 1978; Lientz y Swanson, 1981; Lientz, 1983; Vessey y Weber, 1983; Dart *et al.*, 1993; Singer, 1998; Thomsett, 1998). Entre los principales está la falta de documentación con la que tienen que lidiar los encargados del mantenimiento, y problemas como la rotación constante del personal, lo cual provoca la existencia constante de personal sin experiencia en las tareas de mantenimiento del software. Este tipo de problemas ocasiona que los conocimientos y experiencias que se van obteniendo durante las distintas etapas de mantenimiento por las que puede pasar un sistema, se pierdan con facilidad.

Esta sección presenta algunos de los problemas más comunes asociados al mantenimiento del software. Primeramente se describen algunos de los aspectos socio-culturales del mismo, posteriormente los relacionados con la documentación, y por último los problemas de pérdida y desaprovechamiento del conocimiento.

II.3.1 Aspectos socio-culturales del mantenimiento

Algunos de los problemas del mantenimiento se derivan de aspectos relacionados con la cultura dentro de las organizaciones de desarrollo de software. Parikh menciona algunos mitos que se tienen con respecto al mantenimiento, entre ellos está que algunos programadores piensan que si solo ellos entienden el código que generan, tendrán trabajo asegurado para siempre (Parikh, 1985). Quizá el miedo a perder el trabajo lleva a muchos

programadores a tomar una actitud egoísta con respecto al conocimiento que poseen, de manera que se niegan a dejar documentación, pensando que una vez que otros sepan lo que ellos, podrán perder su empleo. Por otro lado, Pressman menciona tres mitos con respecto a los desarrolladores (Pressman, 2002):

1. Una vez que escribimos el programa y hacemos que funcione, nuestro trabajo ha terminado.
2. Hasta que no tengo el programa “ejecutándose”, realmente no tengo forma de comprobar su calidad.
3. Lo único que se entrega al terminar el proyecto es el programa funcionando.

Este tipo de situaciones puede ser parte de las causas de que no se le dé al mantenimiento la importancia que requiere, y de no desarrollar los sistemas pensando en que deberán ser modificados por otros.

Existen diversos estudios donde se muestra la falta de interés que existe con respecto al mantenimiento del software. Dart *et al.*, en un estudio sobre herramientas para el mantenimiento del software, mencionan que la mayoría de las personas creen que el mantenimiento no es visto como un trabajo prestigioso (Dart *et al.*, 1993). Por otro lado, Thomsett afirma que desde los años sesenta y setenta, hasta finales de los noventa, el mantenimiento ha sido visto como trabajo de segunda clase (Thomsett, 1998); y podríamos pensar que hasta el momento esto no ha cambiado significativamente.

Por otra parte, la mayoría de los sistemas de software son desarrollados sin pensar en el tiempo que estos continuarán en servicio. Muchos sistemas no son diseñados para cambiar (Dart *et al.*, 1993). Un ejemplo muy claro de esto lo es el “bug del año 2000” o bug del milenio (Thomsett, 1998; Barry *et al.*, 1999), causado por el hecho de que la mayoría de los sistemas que existían en esa fecha se diseñaron para soportar solo los dos últimos dígitos del año. Esta decisión fue tomada debido a que cuando muchos sistemas fueron desarrollados (alrededor de los 60’s y 70’s) era muy caro el costo en espacio de

almacenamiento, sin embargo, se pensó que serían reemplazados posteriormente; nunca se llegó a pensar que pudieran seguir en servicio en el año 2000.

II.3.2 Aspectos de la documentación de los sistemas

“La documentación proporciona el fundamento para un buen desarrollo y, lo que es más importante, proporciona guías para la tarea de mantenimiento del software” (Pressman, 2002). La documentación de los sistemas de software es un factor clave que puede ayudar en gran medida en labores de mantenimiento. Cuando la documentación existe, es una de las principales fuentes de información consultadas, ya que es una forma de comunicación entre los miembros de un proyecto (Curtis *et al.*, 1988). Sin embargo, es evidente que uno de los principales problemas que se les presenta a los encargados del mantenimiento, es la inexistencia o la falta de actualización o calidad de la documentación.

La baja calidad en la documentación es uno de los principales problemas existentes en las organizaciones de software (Lientz y Swanson, 1978; Lientz y Swanson, 1981; Lientz, 1983). Dart *et al.* mencionan entre los hallazgos de su estudio, que muchas de las personas que entrevistaron expresaron que la creación y mantenimiento de la documentación es una de las actividades que causa mayores problemas, y que la documentación de soporte que acompaña a los sistemas que mantienen es muy poca (Dart *et al.*, 1993). Por su parte, Singer hace un estudio sobre mantenimiento del software donde presenta varios comentarios dados por diversos entrevistados, los cuales hacen ver que la documentación con frecuencia no es confiable debido a que no está actualizada (Singer, 1998).

Las causas de los problemas de la documentación pueden ser muy diversas, entre ellas está el que los administradores ejercen presión sobre el control de costos y el calendario, y esto ocasiona que la documentación de los cambios no se haga, o sea hecha de manera insuficiente (Lientz, 1983). Por ejemplo, Walz *et al.* mencionan: *“nos sorprendió que tanta información fue presentada al equipo y nunca capturada ... es importante reconocer que*

mucha de la información que necesita ser parte de las memorias del equipo no es capturada formalmente” (Walz et al., 1993).

Los problemas de la documentación provocan que mucho del conocimiento y experiencias que se van generando durante las labores de mantenimiento sólo se queden en la mente de las personas. Debido a esto, dentro del mantenimiento del software comúnmente se dan problemas de pérdida y desaprovechamiento del conocimiento.

II.3.3 Pérdida y desaprovechamiento del conocimiento

Los problemas de pérdida y desaprovechamiento del conocimiento tienen dos vertientes principales, el desaprovechamiento por no saber de la existencia de las fuentes del mismo, y la pérdida por causas como la rotación del personal, ya sea que este personal se mueva a otras empresas o puestos dentro de la misma empresa.

Para poder hacer uso eficiente del conocimiento que puede existir dentro de una organización, es necesario saber de la existencia de las distintas fuentes de conocimiento con que se puede contar, así como el tipo de conocimiento existente en cada una ellas. Sin embargo, es frecuente que los ingenieros de mantenimiento no conozcan todas las fuentes de conocimiento que les pueden ser de utilidad en un momento dado. Un ejemplo de esto es el siguiente fragmento de entrevista obtenido de (Curtis et al., 1988):

“Arquitecto de Software: Aun cuando la descripción del producto para este proyecto es un documento secreto, sólo para las personas que necesitan saber. Yo sé que hubo al menos tres revisiones de las que no supe hasta 6 meses después. Algunos de los cambios que nosotros estamos haciendo ahora, se pudieron haber evitado si hubiéramos tenido acceso a éstas con anterioridad.”

Así como este escenario hay otros, por ejemplo, la existencia de algún miembro del personal que tenga conocimientos para solucionar un determinado problema, pero si el

resto del personal no sabe de estos conocimientos, es muy probable que no sea consultado cuando el problema se presente.

Una de las principales causas de la pérdida del conocimiento en los grupos de mantenimiento del software, son los constantes cambios de personal que se dan a través del tiempo que puede durar en funcionamiento un sistema. Curtis *et al.* presentan un ejemplo de esto (*Ibíd.*):

“Ingeniero de sistemas: No tenemos suficiente documentación. No tenemos suficientes revisiones de código. No tenemos suficientes revisiones de diseño ... Vamos a sufrir porque todos los chicos listos que desarrollaron el sistema nos van a dejar ... y ¿qué va a ser de los pobres que tienen que mantener el sistema?...”

La rotación del personal es uno de los principales problemas del mantenimiento (Lientz y Swanson, 1978; Lientz y Swanson, 1991; Lientz, 1983). Lientz dice que *“la rotación del personal de mantenimiento, puede causar la reducción del soporte al sistema y puede incluso causar problemas como personal no entrenado o no familiarizado para hacer la implementación de una mejora o corrección”* (Lientz, 1983). Munson presenta un ejemplo de los problemas de la rotación del personal que puede darse en sistemas legados, en la actualidad es difícil encontrar personas que conozcan el lenguaje en el que algunos de éstos sistemas fueron desarrollados, ya que aquellos que lo conocían se encuentran retirados (Munson, 1998). Lientz y Swanson mencionan que al parecer el uso de ciertos lenguajes está asociado con problemas mayores en la efectividad de los programadores (Lientz y Swanson, 1981). Los lenguajes de programación han ido evolucionando con el tiempo. Los primeros lenguajes son de nivel más bajo y más difíciles de manejar que los que existen en la actualidad. Además, los nuevos profesionistas de la programación salen de las escuelas conociendo sólo los lenguajes de alto nivel actuales.

Por otro lado, los sistemas se hacen más complejos y difíciles de mantener conforme pasa el tiempo (Lientz, 1983). A través de los años, tanta gente ha trabajado en ellos, que la complejidad del código y los errores ocultos se acrecientan en gran medida. Al mantener sistemas existentes, los programadores por lo general agregan código más que reemplazarlo, parchan más que descartar y empezar de nuevo; entre otras cosas (Lewis, 1990). Es claro que el mantenimiento se ve severamente afectado por la complejidad del código (Banker *et al.*, 1993; Barry *et al.* 1999), ya que para mantener el software necesitamos entenderlo (Buckley, 1989). Considerando que la mayor parte del conocimiento de un sistema se obtiene más a través de la experiencia que por medio del entrenamiento (Curtis *et al.* 1988), queda claro que resulta muy costoso para las organizaciones la inversión que se hace en cada miembro del personal de mantenimiento, ya que estos requieren de tiempo para familiarizarse con el sistema que van a mantener, y una vez que lo conocen, con frecuencia se mudan a otros puestos u organizaciones. Esto ocasiona que el ciclo se vuelva a repetir, ya que por lo general son ingenieros sin experiencia los que son asignados para llenar los espacios dejados (Thomsett, 1998). Si consideramos que los sistemas se van haciendo más complejos con el tiempo, el resultado es que a estos ingenieros sin experiencia les costará más tiempo familiarizarse con el sistema que a sus antecesores.

“Ingeniero de sistemas: Alguien tuvo que gastar cien millones para poner ese conocimiento en mi cabeza. No vino gratis” (Curtis et al., 1988).

II.4 Soporte al mantenimiento del software

Los problemas del mantenimiento del software son tratados desde diversas perspectivas, las cuales pueden catalogarse en dos grandes grupos. El primero se enfoca en facilitar el manejo de sistemas legados o código ajeno, el segundo en la necesidad de hacer que el software sea fácil de mantener. Sin embargo, estos dos enfoques se relacionan mutuamente.

Como se menciona en la sección anterior, muchos de los problemas del mantenimiento se relacionan con la necesidad de tratar con sistemas legados. Dentro de estos, parece ser que la preocupación principal ha sido la dificultad de comprender código ajeno, esto ha llevado a buscar soluciones dentro del rubro de la comprensión de programas. La comprensión de programas se basa en el estudio de la manera en que los programadores comprenden el código de los sistemas a los que dan mantenimiento, sobre todo cuando estos sistemas han sido desarrollados por otros. Algunos de los estudios realizados en el área de la comprensión de programas comprenden el papel que juega la complejidad del software (Vessey y Weber, 1983; Banker *et al.*, 1993), así como la manera en que los programadores se forman modelos mentales del código con el fin de entenderlo (Mayrhauser y Vans, 1995).

Entre los mecanismos utilizados para hacer más entendible el código, se encuentran el desarrollo de estándares de codificación (Oman y Cook, 1990) y documentación interna del código (Basili y Mills, 1982; Basili y Abd-El-Hafiz, 1996). Otras técnicas empleadas en el mantenimiento del software son la reingeniería, ingeniería inversa y reestructuración de programas (Chikofsky y Cross, 1990; Pressman, 2002). Este tipo de técnicas buscan facilitar la identificación de las especificaciones de diseño de un sistema, obtener información del mismo, una representación del código que sea más entendible y manejable para los programadores, o ayudar en la reestructuración del mismo.

Todos estos estudios hechos para dar solución a los distintos problemas que se presentan en los grupos de mantenimiento del software, han dado origen a diversas herramientas de apoyo al proceso de mantenimiento. A continuación se describen este tipo de herramientas.

II.4.1 Tipos de herramientas de soporte al mantenimiento del software

Existen una gran cantidad de herramientas que han sido adoptadas por los grupos encargados del mantenimiento del software, algunas se desarrollaron con el propósito de solucionar problemas específicos del mantenimiento, mientras que otras se han tomado de las distintas etapas del proceso de desarrollo de software.

Tradicionalmente el mantenimiento del software se ha provisto de herramientas que pueden agruparse dentro de dos campos principales: la reingeniería e ingeniería inversa, y la administración de configuración del software:

- **Reingeniería e ingeniería inversa.** Las herramientas dentro de esta categoría, permiten a los ingenieros de mantenimiento obtener información del código fuente (Harandi y Ning, 1990), facilitan la navegación a través del mismo (Hausler *et al.*, 1990; Rich y Wills, 1990), y permiten la reestructuración del código (Choi y Scacchi, 1990; Griswold y Notkin, 1993), ya sea a otros lenguajes o modelos de programación, por ejemplo de un modelo estructurado a uno orientado a objetos.
- **Administración de configuración del software.** Alexis Leon propone que *“el trabajo del equipo de mantenimiento... puede ser facilitado en gran medida si el proyecto tiene un buen sistema de administración de configuración”* (Leon, 2000). Pressman menciona que las actividades de Administración de Configuración del Software (SCM por sus siglas en inglés) *“sirven para: 1) identificar el cambio, 2) controlar el cambio, 3) garantizar que el cambio se implemente adecuadamente, y 4) informar del cambio a todos aquellos que puedan estar interesados”* (Pressman, 2002). Por su parte, Estublier identifica una serie de áreas en las que se han enfocado los estudios sobre SCM (Estublier, 2000), éstas permiten observar el tipo de apoyo que proporciona la SCM al mantenimiento del software. La tabla III muestra estas áreas.

Tabla III. Trabajo actual en el área de administración de configuración (Estublier, 2000)

Repositorios de componentes
Control de versiones Modelos de datos y productos Modelos de sistemas
Ayuda a los ingenieros
En la construcción de sistemas Apoyo en la compilación Identificación de dependencias durante la compilación Soporte a los espacios de trabajo Soporte al trabajo cooperativo y remoto
Soporte a procesos
Control de cambios

Por otra parte, Huff *et al.* presentan una recopilación de los tipos de herramientas de soporte a la Ingeniería de Software Asistida por Computadora (CASE por sus siglas en inglés), donde agrupan cinco de estos tipos dentro de la etapa de mantenimiento, los cuales son: 1) ingeniería inversa, 2) administración de cambios, 3) traslación de lenguajes, 4) reestructuración de código, y 5) migración de aplicaciones (Huff *et al.*, 1992). Como complemento de lo anterior, en la tabla IV se presenta una recopilación de los distintos tipos de herramientas para dar soporte al mantenimiento del software hecha por Bennett y Rajlich (Bennett y Rajlich, 2000).

Tabla IV. Tipos de herramientas para el mantenimiento del software (Bennett y Rajlich, 2000)

Análisis de impacto y manejo de efectos secundarios
Despliegue de la estructura de programas
Pruebas de regresión
Mejores diseños de lenguajes de programación
Identificación, localización y representación de conceptos en el código
Administración de configuración y control de versiones

Entre los principales objetivos de los tipos de herramientas de soporte al mantenimiento están: ayudar a entender la estructura y especificaciones de diseño del código; apoyar en la realización de los cambios, ya sea identificando efectos secundarios, o facilitando la identificación de elementos en el código; y facilitar el control de versiones y configuración de los sistemas.

Mucho del trabajo sobre herramientas automatizadas para dar soporte al mantenimiento del software, se ha orientado a proporcionar medios que ayuden a mejorar el código de una manera más rápida, reduciendo el tiempo y el costo invertidos. Bennet y Rajlich presentan un conjunto de este tipo de herramientas, las cuales se muestran en la tabla V.

Tabla V. Tipos de herramientas de soporte automatizado para mejorar el código (Ibid.)

Migración de lenguajes de programación obsoletos a lenguajes modernos
Migración de bases de datos obsoletas a modernas
Reestructuración del código y datos para reducir la complejidad
Métricas y métodos de evaluación para identificar aspectos empíricos
Herramientas de documentación para el manejo de comentarios
Paquetes de servicio (service packs) para reducir el tamaño de la distribución del software al consumidor
Actualización del software de manera remota
Revisores de integridad en lenguajes de programación
Administración de nombres e identificadores

Estos tipos de herramientas automatizadas por lo general tienden a resolver problemas específicos, como puede ser el trasladar un determinado sistema de un lenguaje de programación a otro, por lo mismo son poco utilizadas, ya que existe una gran diversificación entre los grupos de mantenimiento que se podrían beneficiar de ellas. Por ejemplo, los modelos de programación empleados por cada uno de ellos, sobre todo si no se realizan mediante una metodología bien establecida, como resulta ser en la mayoría de los casos, hace difícil que una herramienta de este tipo pueda ser usada para cualquier situación. Además, si consideramos el costo que éstas suelen tener, su utilización resulta prohibitiva para muchas organizaciones.

Si bien existe una gran cantidad de trabajos que han logrado apoyar a los encargados del mantenimiento del software, estos trabajos se enfocan principalmente en facilitar el manejo del código de los sistemas a ser mantenidos, así como dar soporte a la administración de cambios y control de versiones. Podemos observar claramente una necesidad de trabajos que busquen reducir los problemas de pérdida y desaprovechamiento del conocimiento que comúnmente se da en los grupos de mantenimiento del software.

II.5 Resumen

Con el fin de entender el contexto dentro del cual se realiza el presente trabajo de tesis, en este capítulo se realizó una revisión de los principales aspectos relacionados con el mantenimiento del software. Entre estos, se proporcionó una definición de mantenimiento, así como una clasificación de las formas que puede tomar. Posteriormente, para dar una idea de las actividades que involucra el proceso de mantenimiento, se mostró el modelo propuesto por el estándar sobre mantenimiento del software de la IEEE (IEEE Std. 1219-1998). También se presentó una generalización de los actores y roles que participan dentro de este proceso. Además, se identificaron algunos de los principales problemas del mantenimiento del software, donde se observa la existencia de problemas asociados con la pérdida y desaprovechamiento del conocimiento. Por último, se presentaron los distintos mecanismos y herramientas utilizadas para apoyar en la solución de los problemas del mantenimiento. En este último punto se observa la necesidad de herramientas orientadas a disminuir los problemas de pérdida y desaprovechamiento del conocimiento existente en el proceso de mantenimiento del software.

Los problemas de pérdida y desaprovechamiento del conocimiento no son particulares al mantenimiento del software, al contrario, diversas organizaciones se han dado a la tarea de buscar mecanismos para solucionarlos, lo que ha llevado al nacimiento del concepto de administración del conocimiento, un concepto que agrupa diversos esfuerzos orientados a lograr el mejor aprovechamiento del conocimiento dentro de las organizaciones. Tomando en cuenta que la administración del conocimiento promete ser una solución a muchos de los problemas relacionados con el conocimiento dentro de las organizaciones, en este trabajo se considera a la administración del conocimiento como un medio para brindar soporte a los problemas de pérdida y desaprovechamiento del conocimiento, presentes en el proceso de mantenimiento del software.

Con el fin de entender el concepto de administración del conocimiento y dar una idea de cómo puede ayudar a dar soporte al mantenimiento de software, en el siguiente capítulo se

presenta su definición y algunos de los mecanismos de administración del conocimiento que se han desarrollado. También se describen los esfuerzos realizados para aplicar la administración del conocimiento en las organizaciones de desarrollo de software, donde se observa que estos esfuerzos se han orientado hacia las etapas de la ingeniería del software relacionadas con el desarrollo de nuevos sistemas.

Capítulo III. Administración del Conocimiento en el Mantenimiento del Software

“El conocimiento es experiencia.

Todo lo demás es solamente información”

Albert Einstein

“Todos aprendemos por la experiencia,

*y este conocimiento es usado posteriormente en tareas similares,
adaptando este conocimiento a las nuevas situaciones”*

(Kucza *et al.*, 2001)

Actualmente, muchas organizaciones se han dado cuenta de que el mayor capital con el que pueden contar es el conocimiento de sus miembros. Durante el trabajo diario de las organizaciones se genera conocimiento, y mucho de este conocimiento no se queda guardado en documentos o repositorios de información, sino que pasa a formar parte de la experiencia acumulada en sus miembros. Por lo tanto, si las personas que tienen este conocimiento se van, el conocimiento se va con ellos. Esto tiene enormes costos para las organizaciones, ya que deben volver a invertir para que quien ocupe el cargo vacante pueda adquirir la experiencia de quien lo abandona.

En el capítulo anterior se tratan algunos aspectos relacionados al mantenimiento del software, de donde se observa que entre los principales problemas existentes se encuentra la pérdida y el desaprovechamiento del conocimiento durante el proceso de mantenimiento. Este tipo de problemas no son exclusivos del proceso de mantenimiento ni de las organizaciones de desarrollo de software, como acabamos de mencionar. Es por lo anterior que diversas organizaciones han empleado mecanismos que les apoyen en la solución de los problemas que les presenta el manejo del conocimiento de sus miembros, y de la organización en general. Si bien a través de los años estos mecanismos y herramientas han contado con diversas nomenclaturas (Rus y Lindvall, 2002), en la actualidad hay una tendencia por agruparlos dentro de un solo campo, *la administración del conocimiento*.

En este capítulo se discuten distintos elementos que permiten entender la manera en que la administración del conocimiento puede ayudar a solucionar algunos de los problemas del mantenimiento del software. Primeramente se definen los conceptos principales involucrados dentro del área de administración del conocimiento. En seguida se propone un modelo para categorizar el tipo y fuentes de conocimiento requerido por los encargados del mantenimiento del software, así como para ilustrar el flujo del conocimiento dentro este proceso. Posteriormente se describen los principales beneficios que se buscan obtener con los mecanismos de administración del conocimiento, así como los principales tipos de herramientas que se han utilizado para lograr estos beneficios. Por último se muestran algunos trabajos sobre administración del conocimiento desarrollados dentro del área de ingeniería del software.

III.1 Administración del conocimiento: una definición

Actualmente no existe una definición generalizada del concepto de administración del conocimiento (McElroy, 2000; Barquin, 2001; Firestone, 2001; Frank, 2001). Algunas definiciones dadas por distintos autores son:

“Administración del conocimiento ... es un proceso implementado a través de un período de tiempo, que tiene que ver tanto con relaciones humanas como con prácticas de negocios y tecnologías de información” (Benjamins et al., 1998)

“... administración del conocimiento organizacional para crear valor de negocio y generar una ventaja competitiva” (Tiwana, 2000).

“Administración del conocimiento es el proceso a través del cual una empresa usa su inteligencia colectiva para completar sus objetivos estratégicos” (Barquin, 2001)

“Administración del conocimiento es una actividad humana que es parte del proceso de administración del conocimiento (PAC) de un agente o colectividad ... y el PAC es una red dirigida, persistente y multipropósito de interacciones entre agentes humanos a través de la cual, los agentes participantes administran (manejan, dirigen, gobiernan, controlan, coordina, planean, organizan) a otros agentes, componentes, y actividades participantes en el proceso de conocimiento básico (producción e integración de conocimiento) con el fin de producir un todo planeado, dirigido y unificado, manteniendo, mejorando, adquiriendo y transmitiendo la base de conocimiento de la empresa” (Firestone, 2001)

“La administración del conocimiento está emergiendo como una disciplina que provee de los mecanismos para administrar sistemáticamente el conocimiento que evoluciona con la empresa” (Abdullah et al., 2002)

“Administración del conocimiento ... es un medio para capturar, retener, y distribuir personal experto, procesos corporativos, y experiencias de negocios” (Hwang, 2002)

“Administración del conocimiento es una colección de metodologías y estrategias para la reingeniería de procesos de negocios para fomentar el flujo del conocimiento entre una organización y las personas” (Kolp, 2002)

Las definiciones sobre administración del conocimiento varían considerablemente de un autor a otro. Hay desde las muy generales y simplistas hasta las muy complicadas. Sin embargo, es necesario definir bajo qué términos será entendida la administración del conocimiento dentro de este trabajo. Por lo tanto, entenderemos el concepto de administración del conocimiento, como:

una disciplina que provee de mecanismos (modelos, procesos, tecnologías) encaminados a buscar el mejor aprovechamiento del conocimiento (captura, creación, identificación, almacenamiento, recuperación, transferencia, diseminación) existente dentro de una organización, con el fin de incrementarlo, y evitar su pérdida y sub-utilización.

III.1.1 Una definición del conocimiento

Para poder administrar el conocimiento, primero es necesario definir qué es lo que se considera como tal. Si bien todos tenemos una idea de lo que es el conocimiento, existen diferencias entre estas percepciones. En nuestro caso, adoptaremos la definición propuesta por Davenport y Prusak, la cual es una de las más utilizadas:

“El conocimiento es una mezcla fluida de experiencias enmarcadas, valores, información contextual, y pericia que provee de un marco para evaluar e incorporar nuevas experiencias e información. Esto se origina y es aplicado en la mente de quienes conocen. En las organizaciones, con frecuencia viene embebido no solo en documentos o repositorios, sino también en rutinas, procesos, prácticas o normas de la organización” (Davenport y Prusak, 2000).

De esta definición podemos identificar que el conocimiento puede existir en documentos o medios tangibles, así como en la mente de las personas. Estas dos características han llevado a que el conocimiento sea catalogado en dos clases: conocimiento explícito y tácito (Nonaka y Konno, 1998; Tiwana, 2000).

1. *Conocimiento explícito.* Este tipo de conocimiento es el que está expresado de una manera formal, y por lo tanto puede ser fácilmente comunicado y transferido. Nonaka y Konno establecen que *“el conocimiento explícito puede ser expresado en palabras y números, y compartido en forma de datos, fórmulas científicas, especificaciones, manuales, y cosas por el estilo”* (Nonaka y Konno, 1998). Por su parte, Tiwana lo define como *“aquel componente de conocimiento que puede ser codificado y*

transmitido en un lenguaje sistemático y formal: documentos, bases de datos, páginas web, correos electrónicos, cartas, etc.” (Tiwana, 2000).

2. *Conocimiento tácito.* Es el conocimiento personal utilizado para realizar las tareas y para obtener un entendimiento del medio ambiente; como la experiencia y las habilidades individuales de cada persona. Con frecuencia este tipo de conocimiento es difícil de articular de una manera formal, por lo que resulta difícil de transferir. Nonaka y Konno mencionan al respecto que *“el conocimiento tácito es altamente personal y difícil de formalizar, haciendo difícil comunicarlo o compartirlo con otros”* (Nonaka y Konno, 1998). Mientras tanto, Tiwana lo define como *“conocimiento específico del contexto, que es difícil de formalizar, grabar, o articular; está almacenado en la cabeza de las personas”* (Tiwana, 2000).

Como podemos ver, de las dos formas que puede adoptar el conocimiento, la explícita es la que puede ser manejada con mayor facilidad, sin embargo, el conocimiento tácito es el que da el grado de pericia en la realización de una tarea. Por lo tanto, para compartir la experiencia resulta necesario poder transferirla a los demás, por ejemplo, capturarla y hacerla explícita, y una vez hecho esto, obtener ese conocimiento explícito y convertirlo en experiencia o habilidades para su reutilización. Esto nos lleva a preguntarnos si es posible transferir el conocimiento tácito, así como convertirlo en explícito y viceversa, con el fin de compartir la experiencia con mayor facilidad.

III.1.2 Mecanismos de conversión del conocimiento

Existen distintos mecanismos para transferir o convertir el conocimiento entre sus diferentes formas. Específicamente, la combinación de las dos categorías del conocimiento hace posible cuatro mecanismos o patrones de conversión del conocimiento (Nonaka y Konno, 1998). Nonaka y Takeuchi mencionan que estos mecanismos son la socialización, la exteriorización, la combinación y la interiorización (*Ibid.*; Choo, 1999; Kucza y Komi-Sirviö, 2001). La figura 2 muestra el funcionamiento de estos mecanismos.

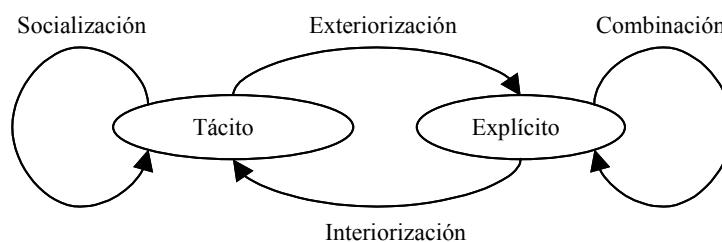


Figura 2. Mecanismos de conversión del conocimiento

*“La **socialización** involucra el compartir conocimiento tácito entre individuos”* (Nonaka y Konno, 1998). *“Es el proceso de compartir experiencias que crea conocimiento tácito como modelos mentales y habilidades técnicas que se comparten”* (Choo, 1999). Mediante la socialización, los individuos transmiten entre sí sus experiencias.

*“La **exteriorización** requiere la expresión del conocimiento tácito y su translación hacia formas comprensibles que pueden ser entendidas por otros”* (Nonaka y Konno, 1998). Según Choo *“la exteriorización es el proceso fundamental de la creación de conocimiento, en el cual el conocimiento tácito se vuelve explícito al compartir metáforas, analogías, modelos o anécdotas”* (Choo, 1999). Es decir, que este mecanismo permite convertir el conocimiento tácito en explícito al plasmar el primero en medios formales, tales como informes o documentos (entre otros), a través de palabras, gráficos, imágenes, video, audio, etc.

*“La **combinación** involucra la conversión de conocimiento explícito en conjuntos más complejos del mismo”* (Nonaka y Konno, 1998). *“Es el proceso de combinar o reconfigurar cuerpos desiguales de conocimiento explícito existentes que conduzcan a la producción de nuevo conocimiento explícito”* (Choo, 1999). Mediante este mecanismo es posible generar conocimiento explícito consultando fuentes formales de conocimiento. Un ejemplo de esto podría ser la elaboración de un informe de los errores más comúnmente reportados en un determinado sistema, tomando como base los reportes de errores que hayan sido capturados en una bitácora.

“La interiorización es la conversión de conocimiento explícito a conocimiento tácito” (Nonaka y Konno, 1998). *“Es el proceso de aprender y socializar al hacer reiteradamente una tarea de modo que el conocimiento explícito de los principios y procedimientos aplicados llegue a ser absorbido como el conocimiento tácito del estilo y el hábito del individuo”* (Choo, 1999). La interiorización permite a los individuos incrementar su experiencia mediante la consulta reiterada de conocimiento explícito en la realización de sus actividades. Un ejemplo de esto lo podemos ver cuando un programador, al empezar a programar con un determinado lenguaje, requiere de la constante consulta de libros o tutoriales que expliquen su utilización (léxico, semántica, etc.); pero, conforme pasa el tiempo, el programador requiere cada vez menos de la consulta de estos libros o tutoriales, debido al aumento de su experiencia en el manejo de dicho lenguaje.

III.2 Conocimiento en el mantenimiento del software

Para poder dar soporte a la administración del conocimiento en el mantenimiento del software, es necesario identificar cuál es el conocimiento que requieren los encargados del mismo, así como las fuentes que utilizan para obtenerlo. Con este fin, hemos hecho una analogía con los dos tipos de conocimiento descritos anteriormente: explícito y tácito, la cual nos permitió definir un modelo para catalogar el tipo de conocimiento requerido por los encargados del mantenimiento, así como las fuentes que usan para obtenerlo. En esta sección se presenta este modelo. Primeramente se definen las fuentes de conocimiento existentes en el mantenimiento, y posteriormente los tipos de conocimiento requeridos por los encargados de este proceso.

III.2.1 Fuentes de conocimiento

Waltz *et al.* mencionan que en general los miembros de un equipo de desarrollo de software no tienen todo el conocimiento requerido por el proyecto, y tienen que adquirir información adicional antes de completar su trabajo productivo. Las fuentes de esta información pueden ser documentación relevante, sesiones de entrenamiento formal, el resultado de realizar actividades de prueba y error, y la consulta a otros miembros del equipo (Waltz *et al.*,

1993). Las fuentes de información o conocimiento consultadas durante el mantenimiento del software pueden ser muy diversas. Cada organización es muy distinta y tiene diferentes modos de manejar su información. Sin embargo, existen estudios donde es posible obtener algunas generalizaciones respecto de las fuentes de conocimiento en el mantenimiento del software.

Diversos estudios hacen referencia a las fuentes de información de las que hacen uso los ingenieros de software encargados del mantenimiento (Dart *et al.*, 1993; Singer, 1998; Seaman, 2002), de donde uno de los más orientados a la obtención de las fuentes de información en el mantenimiento de software ha sido dirigido por Carolyn Seaman (Seaman, 2002). Seaman menciona que las fuentes de información utilizadas en el mantenimiento de software incluyen: artefactos del proceso de desarrollo (documentación de requerimientos, de diseño, del código, planes de pruebas y reportes, etc.); documentación para los usuarios (manual de usuario, de configuración, etc.); el sistema mismo (tanto el sistema ejecutable como el código fuente); y acceso directo a los desarrolladores y usuarios del sistema. De esto podemos identificar tres categorías: documentación, que incluye cualquier tipo de información en papel o documentos electrónicos; sistema, incluye el sistema ejecutable, las bases de datos y el código fuente; y personas, las personas que son consultadas por los ingenieros o programadores durante las distintas fases del mantenimiento.

III.2.1.1 Documentación

La documentación utilizada por el personal de mantenimiento es muy variante y depende en gran medida del tipo de actividad que se desea realizar. Por ejemplo, para el entendimiento de los procesos del usuario, se puede hacer uso de documentos de análisis de requerimientos que los describen; para implementar ciertos algoritmos en un determinado lenguaje, se pueden usar libros o tutoriales de dicho lenguaje, y así sucesivamente. Sin embargo, es posible clasificar esta documentación en cinco grupos principales:

1. *Documentación del sistema*. Este tipo de documentación corresponde a documentos que describen el sistema, y que son generados durante el proceso de desarrollo o durante el

mantenimiento. Específicamente puede ser información de los requerimientos del sistema; procedimientos y procesos que son apoyados por el mismo; la estructura interna del sistema, su diseño, arquitectura de bases de datos, etc.

2. *Documentación técnica*. Corresponde a la documentación de las herramientas y lenguajes utilizados por el personal de mantenimiento, como manuales, libros, tutoriales, etc.
3. *Documentación de usuarios*. Contiene información que va dirigida a los usuarios de los sistemas a ser mantenidos, como son los manuales de usuario, de configuración, de instalación, etc.
4. *Reportes de errores*. Contienen la información que se captura al momento de que un usuario reporta algún error o solicita ayuda. Este tipo de documentos pueden contener, las causas del error, la solución que se le dio, quién lo reportó, así como el nombre de la persona que le dio solución.
5. *Solicitudes de mantenimiento*. Este tipo de documentos son los que contienen los requerimientos que deben ser cubiertos por las modificaciones que se han solicitado. Con frecuencia van asociados con información que contiene las actividades que se realizaron para hacer las modificaciones solicitadas, quiénes las realizaron, etc.

III.2.1.2 Sistema o producto

Dentro de la categoría de fuentes de conocimiento del sistema podemos identificar tres clases principales:

1. *Sistema ejecutable*. El sistema ejecutable se utiliza para entender su funcionamiento de manera externa. En ocasiones se utiliza, al momento de corregir errores, para seguir el proceso realizado por el usuario; esto ayuda a identificar el error, en que condiciones se da, así como una aproximación al lugar exacto (dentro del sistema) donde se genera.
2. *Bases de datos*. Las bases de datos del sistema se utilizan para identificar si los errores generados por el sistema se deben a fallas en los datos. También les permiten a los ingenieros entender los flujos de datos dentro del sistema, así como la manera en que las diferentes entidades de datos se relacionan entre sí.

3. *Código fuente.* Definitivamente el código fuente es un factor de suma importancia, ya que es en éste donde se puede ver la estructura interna real del sistema. El código fuente representa el estado del sistema. La documentación muchas veces puede no corresponder con la realidad, por lo que resulta necesario estudiar el código para conocer como se están haciendo las cosas realmente.

III.2.1.3 Personas

Las personas a las que los ingenieros de mantenimiento consultan pueden agruparse en tres categorías

1. *Usuarios o Clientes.* Los usuarios resultan de gran ayuda a la hora de definir los requerimientos que deben cubrir las modificaciones, así como para identificar las causas y circunstancias que originan los errores en el sistema a la hora de corregirlos. También pueden ser consultados como apoyo para el entendimiento del funcionamiento del sistema cuando no existe documentación al respecto, o ésta resulta insuficiente.
2. *Otros miembros del personal.* El apoyo de otros miembros del equipo resulta de gran utilidad, sobre todo cuando estos han sido los desarrolladores o han trabajado previamente con el sistema que va a ser modificado. También se consulta a otros miembros del grupo cuando se conoce su grado de pericia en ciertas áreas (como puede ser la manipulación de bases de datos, conocimiento de las herramientas y lenguajes de desarrollo, etc.), y ese conocimiento es necesario para el trabajo que se pretende hacer.
3. *Otros expertos.* En ocasiones los encargados del mantenimiento consultan a personas que no forman parte del personal, pero que son expertas dentro de un dominio específico, como por ejemplo en el manejo de cierto lenguaje o herramienta, o dentro del dominio de la aplicación a la que se dará el mantenimiento, como pudiera ser un contador en el caso de un sistema de finanzas. Algunos medios para este tipo de consultas lo constituyen las listas de discusión sobre ciertos temas, páginas para consultas en Internet, entre otros.

III.2.2 Tipos de conocimiento requerido por los encargados del mantenimiento del software

La identificación del tipo de conocimiento, o conocimiento tácito, requerido por los encargados del mantenimiento del software puede ser complicada debido a su misma naturaleza (Choo, 1999); si a esto le agregamos la complejidad que puede existir en el proceso de mantenimiento de una organización en particular, la dificultad se incrementa en gran medida. Sin embargo, existen trabajos que pueden ayudar a identificar el tipo de conocimiento relacionado al mantenimiento del software.

Las personas encargadas del mantenimiento del software, *“deben tener un buen entendimiento de lo que tienen que hacer, cómo lo deben hacer, dónde deben hacer los cambios o modificaciones, y qué impacto tendrán estos cambios en otros programas”* (Leon, 2000). Por su parte, Mayrhauser y Vans mencionan que requieren dos tipos de conocimiento: 1) uno general e independiente de la aplicación a la que se va a dar mantenimiento, 2) y otro específico que representa el nivel de entendimiento que se tiene del sistema (Mayrhauser y Vans, 1995). Esto último está relacionado a lo que Curtis *et al.* llaman el *“conocimiento del dominio de la aplicación”*. Curtis *et al.* proponen que los dominios de conocimiento involucrados en la construcción de sistemas están compuestos por la conducta del usuario y el dominio de la aplicación, el cual abarca la arquitectura del sistema, y ésta a su vez incluye la arquitectura del hardware y software, así como conocimientos sobre los algoritmos y estructuras de datos (Curtis *et al.*, 1988). Estos dominios de conocimiento se muestran en la figura 3.

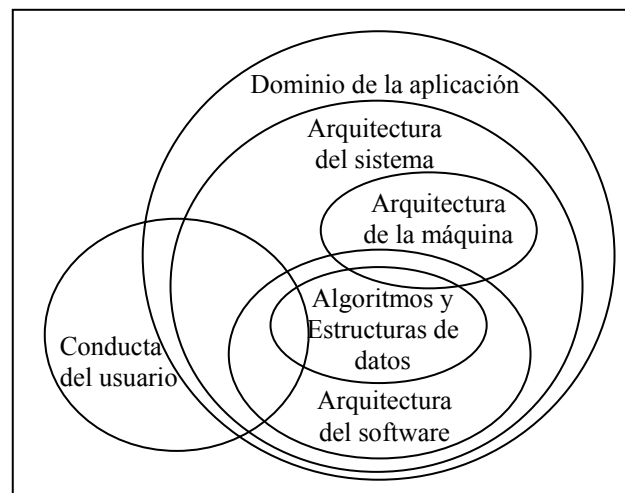


Figura 3. Dominios de conocimiento en la construcción de sistemas (Curtis *et al.*, 1988).

Por otro lado, Robillard propone que, considerando la manera en que el conocimiento es almacenado en la memoria de las personas, se pueden identificar dos tipos del mismo (Robillard, 1999):

1. *Procedural*, es dinámico e incluye habilidades psicomotoras, y su adquisición se basa principalmente en la práctica (el saber cómo);
2. *Declarativo*, es estático y está basado en hechos, se refiere a las propiedades de los objetos, personas y eventos, y sus relaciones. Este a su vez puede dividirse en dos clases:
 - a. *Tópico*, que se refiere a significados, como el significado de las palabras en un diccionario o libro de texto;
 - b. *Episódico*, el cual consiste en la experiencia de cada individuo con el conocimiento. Este último se obtiene mediante experiencias en la aplicación del primero.

Los trabajos presentados muestran que es posible hacer una generalización de los principales tipos de conocimiento existentes y necesarios en los grupos de mantenimiento del software. Con esta base, hemos desarrollado un modelo para identificar estos tipos de conocimiento.

III.2.2.1 Propuesta de un modelo para identificar el conocimiento en el mantenimiento del software

Para identificar el tipo de conocimiento requerido por los encargados del mantenimiento del software, hemos definido un modelo basado en los tipos de conocimiento antes mencionados. La identificación se ha hecho, primeramente, tomando como base los dos tipos de conocimiento establecidos por Mayrhauser y Vans (Mayrhauser y Vans, 1995). Para el conocimiento relacionado con el dominio de la aplicación hemos tomado como base los dominios de conocimiento propuestos en (Curtis *et al.*, 1988). Y finalmente, hemos dividido cada tipo de conocimiento de acuerdo a Robillard, en procedural y declarativo (Robillard, 1999). La tabla VI muestra este modelo.

Tabla VI. Generalización de los tipos de conocimiento que requiere el personal de mantenimiento.

Conocimiento	Procedural	Declarativo	
		Tópico	Episódico
General e independiente de la aplicación	Programación: desarrollo de algoritmos y estructuras de datos	Tipos y funcionamiento de estructuras de datos. Lenguajes de programación	Experiencias con el desarrollo de algoritmos y estructuras de datos
	Manejo y configuración de sistemas operativos	Tipos y características de los sistemas operativos	Experiencias con el manejo de sistemas operativos
	Manejo y configuración de bases de datos	Tipos de bases de datos, así como instrucciones y estructuras de las mismas	Experiencias con la utilización de instrucciones y manipulación de la estructura de bases de datos
	Manejo y configuración de redes de cómputo	Tipos de redes, características y dispositivos que las conforman	Experiencias con el manejo de redes de cómputo
Dominio de la aplicación	Seguimiento a los procesos del usuario	Procesos seguidos por el usuario	Experiencias con el desarrollo y modificación de los sistemas que dan soporte a esos procesos
	Identificación de problemas en el sistema	Funcionamiento del sistema. Errores más comunes	Uso del sistema. Experiencias con la corrección de errores
	Seguimiento y modificaciones al funcionamiento del sistema	Arquitectura del sistema: módulos, algoritmos, estructuras de datos que lo conforman	Uso del sistema. Experiencias con la modificación de los módulos del sistema
	Manipulación de la base de datos del sistema	Estructura de la base de datos	Experiencias con el desarrollo o modificación de la base de datos
	Manejo de las herramientas y lenguaje con el que se da mantenimiento al sistema	Características de las herramientas. Lenguaje en el que está codificado el sistema	Experiencias con la utilización de esas herramientas y lenguaje de programación; así como con otros similares

La tabla VI muestra una generalización de los principales tipos de conocimiento requeridos por los encargados del mantenimiento del software. Es posible observar que dentro de los conocimientos independientes de la aplicación se encuentran la programación, el manejo de

sistemas operativos, bases de datos y redes. De esta manera, por ejemplo, en el caso de la programación, los ingenieros requieren conocimiento de algoritmos y estructuras de datos; para esto les sirve el conocimiento que puedan tener sobre lenguajes de programación, así como las experiencias previas en el análisis, diseño y programación de algoritmos y estructuras de datos.

En relación al conocimiento en el dominio de la aplicación, tenemos que se requiere saber cómo seguir los procesos realizados por los usuarios, tanto los procesos a los que da soporte el sistema, como la manera en que el usuario hace uso del mismo. También se requiere conocer cómo identificar problemas en el sistema, así como dar seguimiento y saber cómo modificar el funcionamiento del mismo. Otros tipos de conocimiento necesarios son el saber manipular la base de datos del sistema, así como manejar las herramientas y lenguajes con los que se le da mantenimiento.

Si bien la tabla VI presenta solo temas generales, puede ser utilizada para realizar modelos con conocimientos más específicos, siguiendo hasta conseguir el nivel de detalle que se requiera. Por ejemplo, podemos considerar que es importante definir el conocimiento asociado al proceso de realización de cheques en el sistema de finanzas. En este proceso podríamos suponer que se requieren conocimientos sobre la elaboración de cheques (procedural), a la par, podríamos definir que para poder elaborar un cheque es preciso saber, primeramente qué es un cheque, para qué sirve, cuál es el formato que debe llevar, así como los campos que deben ser llenados (declarativo-tópico); lo cual se puede aprender mediante la experiencia que la persona haya tenido en el manejo o elaboración de cheques, o debido a pláticas con personas que hayan hecho un cheque con anterioridad (declarativo-episódico).

Hasta aquí se han definidos los tipos de conocimiento requeridos por los encargados del mantenimiento, así como las fuentes de las que se basan para obtenerlo. Sin embargo, es importante entender cómo estos elementos interactúan dentro del proceso de mantenimiento. Es decir, con el fin de poder identificar la manera en que el conocimiento

es creado, utilizado y transferido entre los miembros del grupo de mantenimiento, es necesario identificar cómo los ingenieros de mantenimiento crean, utilizan y transfieren el conocimiento al resto del equipo.

III.2.3 Flujo del conocimiento en el mantenimiento del software

Para comprender la manera en que el personal de mantenimiento genera y hace uso del conocimiento requerido para realizar sus actividades, es importante conocer el proceso seguido por el personal durante la realización de las mismas. Sin embargo, como ya lo hemos mencionado, las diferencias existentes entre distintas organizaciones, departamentos de una organización, y aun entre proyectos de un mismo departamento (sin incluir a las personas que participan en estos proyectos), hace difícil poder identificar un proceso genérico de mantenimiento de software.

Una primera aproximación al flujo del conocimiento se puede obtener analizando la propuesta de Choo, quien identifica tres campos de uso y creación de la información: 1) para lograr una percepción del medio ambiente; 2) para generar conocimiento; y 3) para tomar decisiones (Choo, 1999). La propuesta de Choo consiste en que éstos tres campos pueden vincularse para formar lo que él llama “*el ciclo del conocimiento*” (figura 4). En este ciclo de conocimiento, la percepción sirve para realizar una representación del medio ambiente; esta percepción se obtiene mediante el uso de la información y los flujos de experiencias. Esto lleva a la creación de significados compartidos o modelos mentales que son utilizados para planear y tomar decisiones.

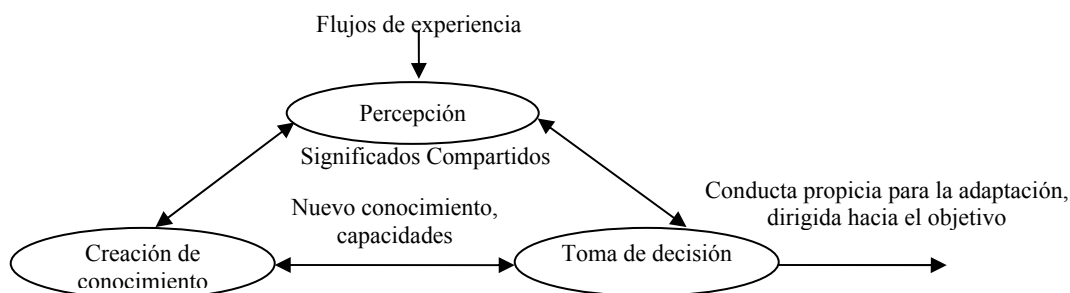


Figura 4. El ciclo del conocimiento (Choo, 1999).

Tomando la tesis de Choo, es posible adaptarla para identificar un modelo del ciclo que sigue el conocimiento dentro del mantenimiento de software. La figura 5 muestra este modelo. En ella podemos ver que el ingeniero de mantenimiento crea y hace uso de información de distintas fuentes que sirven para compartir experiencias.

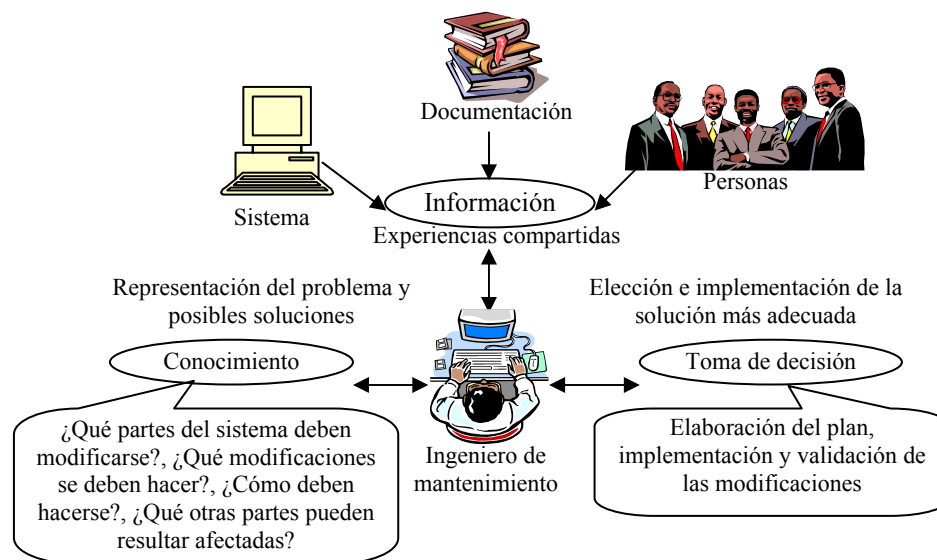


Figura 5. Modelo del flujo del conocimiento en el mantenimiento del software.

El ingeniero de mantenimiento utiliza su experiencia previa, así como los mecanismos de conversión de conocimiento para obtener el conocimiento necesario para hacer una representación del problema y plantear posibles soluciones, para posteriormente tomar la decisión de qué solución implementar. A la par de lo anterior, el ingeniero de mantenimiento crea nuevo conocimiento conforme utiliza, tanto su experiencia como las experiencias compartidas en la solución de los problemas, para posteriormente transmitirla como experiencias compartidas al resto del equipo. En la figura 6 se muestra cómo es llevado a cabo este proceso.

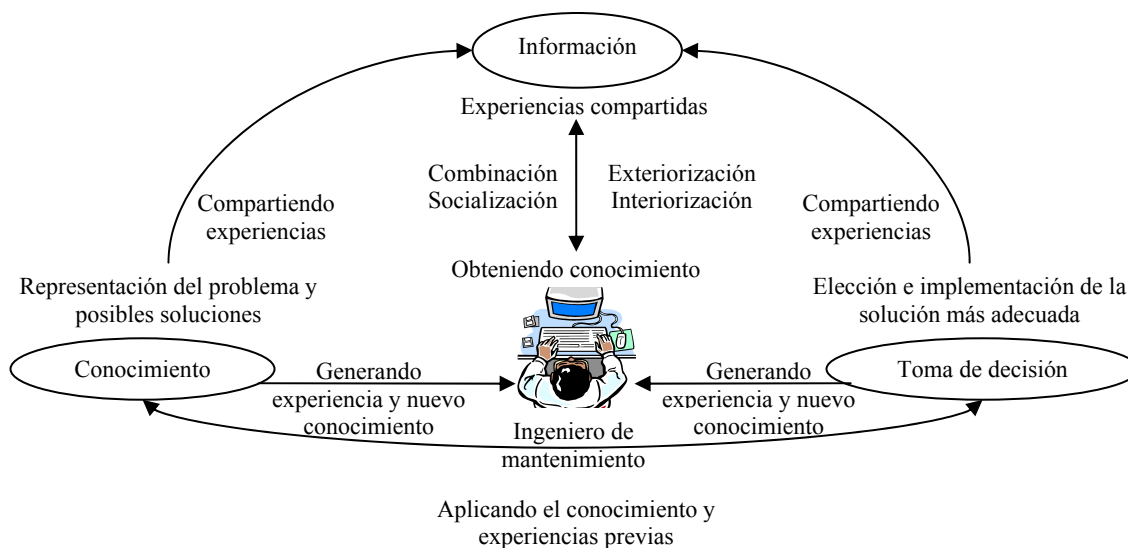


Figura 6. Conversión de conocimiento en el modelo del flujo de conocimiento en el mantenimiento del software.

Como se puede observar en la figura 6, el ingeniero de mantenimiento genera experiencia mediante la aplicación del conocimiento, tanto el propio como el compartido, en la toma de decisiones. A la par, utiliza los mecanismos de conversión del conocimiento para compartir sus experiencias obtenidas al aplicar el conocimiento y tomar decisiones, así como para obtener conocimiento de las experiencias de otros miembros del grupo.

Al identificar la manera en que los ingenieros de mantenimiento crean, usan y transmiten el conocimiento que requieren para sus actividades, se hace más fácil el encontrar formas de apoyarlos en estas tareas, por ejemplo mediante mecanismos que ayuden en la captura, almacenamiento del conocimiento y experiencias que son creadas por los mismos ingenieros de mantenimiento al momento en que identifican un determinado problema y proponen una solución, para posteriormente facilitar su transferencia y recuperación por otros miembros del grupo.

Los beneficios de proveer a los ingenieros de mantenimiento de mecanismos que les ayuden en las tareas antes mencionadas, pueden ser muchos, entre los principales está el

incrementar sistemáticamente el conocimiento con el que pueden contar, así como reducir el desaprovechamiento y el riesgo de pérdida del mismo.

III.3 Beneficios de la administración del conocimiento

Los beneficios de la administración del conocimiento pueden ser muy variados, por ejemplo, la tabla VII muestra un conjunto de 7 necesidades que Rus y Lindvall mencionan como motivo para la implementación de la administración del conocimiento en la ingeniería del software (Rus y Lindvall, 2002).

Tabla VII. Motivación para la administración del conocimiento en la ingeniería del software (Rus y Lindvall, 2002).

Necesidades del negocio
Disminuir tiempos y costos e incrementar la calidad
Tomar mejores decisiones
Necesidades de conocimiento
Adquirir conocimiento acerca de nuevas tecnologías
Acceder al dominio del conocimiento
Compartir conocimiento acerca de políticas y prácticas locales
Capturar conocimiento y saber quien conoce qué.
Colaborar y compartir conocimiento

Esta tabla permite ver que la administración del conocimiento puede ayudar a reducir el tiempo y costos invertidos en las actividades de una organización, a la par de que se incrementa el conocimiento y se reduce la pérdida del mismo, al permitir capturarlo y compartirlo entre los miembros de la organización. Rus y Lindvall también mencionan que la administración del conocimiento puede ser vista como una estrategia de prevención y mitigación de riesgos, ya que considera riesgos que son comúnmente ignorados, tales como:

- Pérdida de conocimiento debido a la reducción del personal.
- Falta de conocimiento y un largo período de tiempo para adquirirlo debido a largas curvas de aprendizaje.
- Personas repitiendo errores y trabajo, porque olvidaron lo que aprendieron en proyectos anteriores.
- Individuos que tienen conocimientos clave dejan de estar disponibles.

Además de los beneficios propuestos por Rus y Lindvall, existen muchos otros, por ejemplo, Tiwana propone 24 aspectos divididos en 6 categorías, que hacen a la administración del conocimiento necesaria para las organizaciones (Tiwana, 2000). Sin embargo, es posible reducir todos estos problemas tratados por la administración del conocimiento, a tres aspectos básicos:

- **Disminuir el riesgo de pérdida de conocimiento.** En la medida en que las organizaciones dependan más del conocimiento tácito de sus miembros, estas contarán con un mayor riesgo de pérdida del mismo, ya que si estas personas se van, se llevan el conocimiento con ellos. Además, las experiencias adquiridas pueden olvidarse con el tiempo, lo que conduce a cometer los mismos errores, o repetir el trabajo por no recordar lo que se hizo en situaciones anteriores. La administración del conocimiento permite reducir el riesgo de pérdida del conocimiento, al proporcionar mecanismos para capturar y convertir la mayor cantidad posible de conocimiento tácito, en explícito.
- **Disminuir el desperdicio del conocimiento existente.** *“Las compañías con frecuencia no saben lo que realmente saben”* (Tiwana, 2000). Se dan casos en que existen dentro de una organización fuentes de conocimiento que pueden ser de utilidad para solucionar ciertos problemas, pero por no saber que éstas existen, o cual puede ser el conocimiento que tienen, no se consultan, por lo que se desaprovecha el conocimiento que poseen. La administración del conocimiento proporciona mecanismos que facilitan la distribución, búsqueda, almacenamiento, identificación y recuperación de las distintas fuentes de conocimiento con que puede contar una organización.
- **Incrementar el conocimiento.** El conocimiento se ha convertido en un factor de competitividad entre organizaciones, de manera que se ha hecho necesario que éstas busquen conocer más que la competencia. Entre más conocimiento tenga una organización, mayores oportunidades tendrá. La administración del conocimiento provee de mecanismos que permiten el incremento sistemático del conocimiento existente dentro de una organización.

Al analizar estos beneficios se hace evidente la posibilidad de facilitar la solución de los problemas de pérdida y desaprovechamiento del conocimiento en el mantenimiento del software, por medio de mecanismos de administración del conocimiento. Con el fin de identificar qué tipos de mecanismos o herramientas podrían ayudar a obtener los beneficios aquí expuestos, en la siguiente sección se presentan las distintas tecnologías que se han empleado dentro de la administración del conocimiento.

III.4 Tecnologías del Conocimiento: TI en la administración del conocimiento

Actualmente nos encontramos en la era del conocimiento (Tiwana, 2000). Si la era de la información dio origen al nacimiento de las tecnologías de la información, podemos decir que la actual era del conocimiento está dando origen al nacimiento de las tecnologías del conocimiento.

La administración del conocimiento ha sido abordada desde diversos ángulos. Mientras algunos han propuesto modelos o procesos, (Nonnaka y Konno, 1998; Choo, 1999; McElroy, 2000; Firestone, 2001), la mayoría se han orientado hacia aspectos tecnológicos. Si bien la administración del conocimiento no es propiamente una tecnología, sino que involucra cambios en los procesos y modelos de negocios, no podría entenderse el auge que actualmente tiene, sin la tecnología. La tecnología ha proporcionado los medios para que las ideas que propone la administración del conocimiento puedan convertirse en realidad práctica. De hecho, muchas de las tecnologías que actualmente se manejan dentro del ámbito de la administración del conocimiento, han existido desde antes de que se acuñara el término, solo que bajo nombres distintos (Frank, 2001; Rus y Lindvall, 2002).

A continuación se presenta una categorización de las tecnologías que han sido usadas dentro del ámbito de la administración del conocimiento. Es necesario aclarar que, como se menciona anteriormente, algunas de estas tecnologías se desarrollaron bajo esquemas distintos al de administración del conocimiento, sin embargo, sus características las han

llevado a ser consideradas como parte esencial de los mecanismos o herramientas usadas dentro del área de administración del conocimiento.

III.4.1 Una categorización de las técnicas y herramientas para la administración del conocimiento

Marwick toma como base los mecanismos de conversión del conocimiento para hacer una categorización de las tecnologías de administración del conocimiento (Marwick, 2001). Tomando esta idea, las tecnologías de administración del conocimiento podrían dividirse en cuatro grupos, como se muestra en la tabla VIII.

Tabla VIII. Categorización de los tipos de tecnologías para la administración del conocimiento (Marwick,2001)

Conocimiento	Tácito	Explícito
Tácito	Socialización	Exteriorización
	Colaboración síncrona (mensajería instantánea) Tele/Video Conferencias y reuniones electrónicas Localización de expertos (Ackerman, 1998)	Soporte a la colaboración con mecanismos que permitan guardar los intercambios de información Reutilización de experiencias y lecciones aprendidas
Explícito	Internalización	Combinación
	Visualización de búsquedas de documentos Representación, y extracción de resúmenes de documentos	Búsqueda y recuperación de información Categorización de documentos

El problema de la categorización propuesta por Marwick, es que varias de las herramientas empleadas dentro de la administración del conocimiento caen dentro de dos o más de estas categorías, por lo que en este trabajo hemos decidido hacer una categorización con base en las áreas que comúnmente se han empleado para dar soporte a la administración del conocimiento. Estas áreas se listan a continuación:

Sistemas cooperativos y Groupware. Ellis *et al.* definen el término groupware como “*sistemas basados en computadora que dan soporte a grupos de personas que persiguen*

una meta o tarea común y que provee de una interfaz a un medio ambiente compartido” (Ellis *et al.*, 1991). Algunos ejemplos son sistemas de mensajería instantánea; sistemas de tele/video conferencias y reuniones electrónicas (Isaacs *et al.*, 1994; Han y Smith, 1996); localización de personas (Girgensohn, *et al.*, 1996); soporte a reuniones, por ejemplo mediante tableros electrónicos para expresar ideas que posteriormente podrán ser almacenadas (Moran *et al.*, 1998); apoyo a la colaboración entre individuos mediante mecanismos que permitan guardar el intercambio de información entre estos (Hindus y Schmandt, 1992) como pueden ser el correo electrónico, algunos sistemas de mensajería instantánea, entre otros.

Localización de expertos. Este tipo de herramientas busca facilitar el acceso a las personas que tienen el conocimiento para realizar ciertas tareas. Ejemplos de este tipo de sistemas son Answer Garden (Ackerman, 1998), un sistema que permite hacer preguntas, que son dirigidas a personas expertas que puedan dar respuesta a las mismas; el recomendador de expertos (McDonald y Ackerman, 2000); y el sistema personalizado de recuperación de información (Martínez Ramírez, 2003).

Memoria organizacional. Actualmente no existe una idea generalizada de lo que debe ser un sistema de memoria organizacional. Walsh y Ungson presentan la siguiente definición: “información histórica de una organización que se puede aplicar en las decisiones actuales ... la cual es almacenada como consecuencia de las decisiones tomadas con anterioridad y a las cuales se refiere, se recolecta de manera individual, y a través de interpretaciones compartidas” (Walsh y Ungson, 1991). Entre las características principales de este tipo de sistemas están el facilitar el almacenamiento, búsqueda y recuperación de grandes cantidades de documentos, por lo que se han usado frecuentemente dentro del ámbito de la administración del conocimiento (Abecker *et al.*, 1998; Ackerman, 1998; Kalfoglou, 2000).

Portales de conocimiento. Los portales de conocimiento aprovechan los beneficios de Internet para proporcionar un punto de acceso estándar para la búsqueda de información relevante dentro de algún dominio específico (Staab y Maedche, 2001; Marwick, 2001;

Barquin, 2001). Algunos ejemplos pueden ser vistos en (CUTTER, 2003; LinuxKP, 2003; KA2, 2003; TIME2RESEARCH, 2003).

Sistemas de soporte a la toma de decisiones. Este tipo de sistemas puede proporcionar apoyo en la toma de decisiones al contar con una base de conocimiento que permite identificar opciones a seguir según un determinado contexto. Un ejemplo de este tipo de sistemas, así como su relación con la administración del conocimiento, puede verse en (Pedersen y Larsen, 2000).

Reutilización de experiencias y lecciones aprendidas. Las historias son un medio muy eficiente para transmitir conocimiento entre los individuos. Por lo tanto, el permitir transmitir experiencias al contar las lecciones aprendidas en una determinada situación, se ha vuelto una técnica muy aceptada y utilizada dentro de la administración del conocimiento (Barquin, 2001), y en particular en ingeniería de software (Basili y Caldiera, 1995). La reutilización de experiencias permite identificar los errores y aciertos que se realizan durante el trabajo diario de una organización, con el fin de repetir los casos de éxito y evitar cometer los mismos errores. Como ejemplos podríamos mostrar el concepto de la fábrica de experiencias (Experience Factory) (Basili y Seaman, 2002), la realización de análisis de los resultados de un proyecto (análisis postmortem) (Birk, 2002), o la utilización de escenarios para plantear problemas y soluciones (Cheah y Abidi, 2000).

Ontologías. Las ontologías se han vuelto una tecnología muy recurrente en sistemas de administración del conocimiento (Benjamins *et al.*, 1998; Liao *et al.*, 1999; Kalfoglou, 2000; Sure, 2002). *“Las ontologías proveen un entendimiento común y compartido acerca de un dominio que puede ser comunicado entre personas y sistemas heterogéneos y ampliamente extendidos”* (Fensel, 2001), por lo que son una buena opción para compartir conocimiento (Gruber, 1993). Es por lo anterior que son ampliamente utilizadas para catalogar y recuperar documentos, información o conocimiento, y de esta manera crear bases de conocimiento que faciliten la captura, creación, almacenamiento, y recuperación de conocimiento dentro de un dominio específico.

Si comparamos las categorías de tecnologías aquí expuestas, con la de los tipos de herramientas para la administración del conocimiento hecha por Marwick, (Marwick, 2001), es posible ver que existe una carencia de herramientas que apoyen en la conversión de conocimiento explícito a tácito. La mayoría se orientan a apoyar el proceso de socialización, exteriorización, y combinación. Sin embargo, es necesario que una buena herramienta de administración del conocimiento busque incluir los cuatro tipos de procesos de conversión de conocimiento.

III.4.1.1 Falta de apoyo en la conversión de conocimiento explícito en tácito

Como ya mencionamos, es posible ver que la mayoría de las herramientas de administración del conocimiento se orientan a dar soporte a sólo tres de los mecanismos de conversión de conocimiento: socialización, exteriorización y combinación. Marwick afirma que, el desarrollo de *“las tecnologías para ayudar a los usuarios a obtener nuevo conocimiento tácito, por ejemplo mediante una mejor apreciación y entendimiento del conocimiento explícito, es un reto de particular interés en la administración del conocimiento”* (Marwick, 2001).

Las herramientas que caen dentro de esta última categoría, además de facilitar la identificación del conocimiento relevante a una determinada tarea, deben facilitar el digerir las grandes cantidades de información y conocimiento que pueden existir dentro de un repositorio. Algunos esfuerzos dentro de este ámbito se han hecho en mecanismos que permitan asimilar de una mejor manera estas grandes cantidades de información, como por ejemplo, técnicas de visualización que facilitan la identificación del contenido de algún almacén de datos o documentos (Robertson *et al.*, 1993). Además, en años recientes las tecnologías de agentes de software han permitido el desarrollo de sistemas que apoyen a los usuarios en estas tareas, por ejemplo, al hacer una preselección de la información o el conocimiento relevante, de manera tal que la carga de trabajo para el usuario sea menor.

Los beneficios que proporcionan los agentes de software a las herramientas de administración del conocimiento los han convertido en una de las principales opciones para el desarrollo de las mismas. En particular, en este trabajo los hemos considerado como la tecnología en la cual basar nuestra propuesta de soporte, es por ello que en la siguiente sección tratamos el uso que se ha dado a los agentes dentro de la administración del conocimiento.

III.4.2 Agentes de software en la administración del conocimiento

Los agentes de software se están convirtiendo en una tecnología que puede aportar muy variados beneficios a la administración del conocimiento. Para entender mejor el por qué de esto, es necesario entender primeramente qué son, y cuáles son sus características.

III.4.2.1 Agentes de Software: definición y características

Un agente es un componente de software capaz de percibir su ambiente y actuar en consecuencia en la búsqueda de la realización de una tarea o meta. Para esto, un agente debe cumplir con las siguientes características básicas (Wooldrige y Ciancarini, 2001):

- **Autónomo.** Pueden mantener un estado y tomar decisiones dependiendo de éste, sin la intervención de humanos.
- **Reactivo.** Deben ser capaces de percibir su medio ambiente mediante sensores, y de actuar dependiendo de los cambios que ocurran en el mismo.
- **Pro-activo.** Deben ser capaces no solo de actuar como respuesta a algún estímulo, sino actuar en la búsqueda de un objetivo, por lo que deben ser capaces de tomar la iniciativa.
- **Sociable.** Deben poder interactuar con otros agentes a través de algún lenguaje de comunicación.

Además de estas características, los agentes de software también pueden tener las capacidades de (Nwana, 1996; Bradshaw, 1997; Berger *et al.*, 2001; Bigus y Bigus, 2001):

- **Movilidad.** Capacidad para migrar de forma segura entre distintos lugares.

- **Adaptabilidad.** Habilidad de aprender y mejorar por medio de la experiencia (Generar conocimiento).
- **Inteligencia.** Capacidad para adaptarse al entorno, al razonar (por medio de mecanismos de inteligencia artificial) durante la realización de tareas.

Estas características de los agentes son las que los convierten en una buena opción para desarrollar sistemas que requieran actuar sin la constante intervención del usuario. Por lo tanto han comenzado a ser usados ampliamente en el desarrollo de sistemas dentro de las distintas áreas que abarca la administración del conocimiento.

Para mostrar la manera en que se ha usado a los agentes dentro de la administración del conocimiento, a continuación se presentan algunos ejemplos de sistemas, basados en agentes, desarrollados dentro de los distintos tipos de herramientas usadas para la administración del conocimiento.

III.4.2.2 Sistemas basados en agentes dentro de las áreas de la administración del conocimiento

Como mencionamos, las características que los agentes de software proporcionan a los sistemas los han convertido en una tecnología viable para el desarrollo de sistemas de administración del conocimiento (Lacher y Koch, 2000; Tacla y Barthès, 2002; Kolp, 2002). Algunos ejemplos de la utilización de agentes en las distintas áreas que dan soporte a la administración del conocimiento son:

- **Sistemas cooperativos y Groupware.** Un ejemplo es el trabajo de Alf Inge Wang, quien ha diseñado una arquitectura multi-agentes para dar soporte a la ingeniería del software de manera cooperativa (Wang, *et al.*, 1999). Este estudio trata de integrar el trabajo cooperativo asistido por computadora con tecnologías de apoyo a procesos de software mediante una arquitectura multi-agentes.
- **Memoria organizacional y manejo del sobre flujo de información.** Los agentes se han usados ampliamente para facilitar el manejo de grandes cantidades de información,

y evitar, de esta manera, el sobre flujo de información en el usuario. Ejemplos de este tipo de sistemas pueden ser los de filtrado de mensajes (Riecken, 1994), manejo de grandes cantidades de información distribuida (Moreau *et al.*, 2000), y sistemas de memoria organizacional (Bergenti *et al.*, 2000; Gandon, 2002; Poggi *et al.*, 2002; Abecker *et al.*, 2003).

- **Compartir conocimiento.** Por sus características, los agentes de software son una muy buena opción para desarrollar sistemas donde se requiere compartir conocimiento entre aplicaciones. Algunos ejemplos de sistemas basados en agentes, que caen dentro de este rubro pueden verse en (Singh *et al.*, 1995; Tamma y Bench-Capon, 2001; Mercer y Greenwood, 2002).

En esta sección se han expuesto los distintos mecanismos y herramientas que son empleados dentro de la administración del conocimiento. Analizando los beneficios que estos proporcionan, cabe preguntarnos el cómo ofrecer éstos a los encargados del mantenimiento del software. Es claro ver que muchas de las tecnologías mencionadas son de propósito general, por lo que pueden, e incluso son utilizadas por diversos grupos de mantenimiento del software, en particular las relacionadas con mecanismos de socialización, como pueden ser algunos sistemas cooperativos (mensajería, correo electrónico, etc.). Sin embargo, hace falta identificar cómo proveer al mantenimiento de los beneficios de la reducción de la pérdida y desaprovechamiento del conocimiento. Para esto, primeramente analizaremos la manera en que las organizaciones de desarrollo del software hacen uso de la administración del conocimiento dentro de sus procesos, lo cual se presenta en la siguiente sección.

III.5 Administración del conocimiento en las organizaciones de desarrollo de software

Las organizaciones desarrolladoras de software también se han dado cuenta de los beneficios que pueden traer a la ingeniería del software el poder capturar, almacenar, administrar y recuperar todo el conocimiento generado durante los distintos proyectos que

realizan. Por lo tanto, muchas de ellas han tomado la administración del conocimiento como un medio para ser más competitivas. Esto se puede ver en el artículo “Knowledge Management in Software Engineering” (Rus y Lindvall, 2002) publicado en la revista IEEE Software de Marzo de 2002, la cual está dedicada a la administración del conocimiento en la ingeniería del software; y en donde se presentan algunos esfuerzos realizados por diversas empresas desarrolladoras de software para aplicar la administración del conocimiento como apoyo en sus actividades.

Los principales objetivos buscados por los mecanismos, técnicas y tecnologías de administración del conocimiento empleadas dentro de la ingeniería del software son:

- **Localización de fuentes de conocimiento (memoria organizacional y localización de expertos).** La ingeniería del software se ha beneficiado de tecnologías que permiten hacer un mejor manejo de las distintas fuentes de conocimiento con las que puede contar. Ejemplos de esto pueden ser la utilización de sistemas de memoria organizacional y localización de fuentes de conocimiento experto. Un ejemplo de esto es el sistema “Answer Garden” (Ackerman, 1998), el cual, como se menciona anteriormente, permite la recuperación de conocimiento almacenado en medios tangibles, así como la localización de personas expertas.
- **Reutilización de experiencias.** Otro de los esfuerzos para aplicar mecanismos de administración del conocimiento en la ingeniería del software, se ha orientado a permitir el aprovechamiento de las lecciones aprendidas durante los distintos proyectos realizados dentro de las organizaciones de desarrollo software. Un ejemplo puede verse en el concepto de la fábrica de experiencias de Victor Basili (Basili y Seaman, 2002). El trabajo de Victor Basilli se ha orientado a la reutilización de las experiencias en los grupos de desarrollo de software, con el fin de mejorar el rendimiento, y aumentar la calidad del software (Basili y Caldiera, 1995; Mendonca *et al.*, 2001; Basili *et al.*, 2001; Basili y Seaman, 2002). Otro ejemplo es el trabajo de Klaus-Dieter Althoff, quien utiliza el razonamiento basado en casos para reutilizar experiencias (Althoff *et al.*, 1997; Tautz y Althoff, 1998; Althoff *et al.*, 1999; Althoff *et al.*, 2002).

- **Mejorar los procesos de desarrollo del software.** Una de las principales razones de ser de la administración del conocimiento en la ingeniería del software, es mejorar los procesos de software, con el fin de incrementar el rendimiento y la calidad, a la vez que se reducen los costos. Es por ello que muchos de los esfuerzos realizados en el desarrollo de sistemas de administración del conocimiento llevan este objetivo (Basili y Caldiera, 1995; Kucza y Komi-Sirviö, 2001; Schneider *et al.*, 2002).

Al analizar los sistemas presentados en los trabajos antes mencionados, es posible ver que la mayoría se concentra en capturar las experiencias en el proceso de desarrollo de sistemas nuevos, así como el conocimiento asociado a las tareas administrativas. Es claro ver la falta de aplicación de mecanismos de administración del conocimiento para dar soporte a los problemas particulares de los grupos de mantenimiento del software. Esto último se detalla un poco más a continuación.

III.5.1 Administración del conocimiento en el proceso de mantenimiento del software

No obstante los esfuerzos que se han realizado por aplicar la administración del conocimiento dentro de la ingeniería del software, aun no existen suficientes estudios de este tipo que busquen solucionar los problemas del mantenimiento de los sistemas existentes. Quizá esto pueda entenderse si consideramos que, con frecuencia, se le da menor importancia al mantenimiento que al resto de las fases de la ingeniería del software, como se discutió en el capítulo anterior. Por lo tanto, generalmente el interés en la solución de problemas en el desarrollo de software se orienta a apoyar las tareas de los administradores, analistas, diseñadores y desarrolladores de proyectos nuevos, no así a los encargados de la etapa de mantenimiento.

Aun cuando los trabajos desarrollados dentro de las áreas que comprende la administración del conocimiento permiten atacar distintos problemas asociados al conocimiento en el proceso de mantenimiento de software, estos trabajos son demasiado generales, lo que

provoca que varias de las características inherentes al mantenimiento del software no sean tomadas en cuenta.

Banker *et al.* mencionan que el mantenimiento del software presenta diferencias significativas con respecto al desarrollo de nuevos sistemas; por ejemplo, la necesidad de tratar con código desarrollado por otras personas (Banker *et al.*, 1993). Existen estudios que demuestran que los encargados de la etapa de mantenimiento requieren herramientas distintas a las usadas en las otras etapas de la ingeniería del software (Dart *et al.*, 1993). Esto nos lleva a la necesidad de herramientas que den soporte a los problemas relacionados con el conocimiento en el proceso de mantenimiento del software, desde un punto de vista particular de las necesidades de los encargados del mismo.

III.6 Resumen

En este capítulo se trató el tema de la aplicación de la administración del conocimiento en el mantenimiento del software. Se presentaron algunos conceptos generales, así como los mecanismos que han sido utilizados para dar soporte a la administración del conocimiento. Posteriormente se presentaron aspectos relacionados con el conocimiento en el mantenimiento del software. Dentro de este rubro, se propuso un modelo para catalogar las fuentes de conocimiento, así como para identificar el tipo de conocimiento, y el flujo que éste sigue dentro de los grupos dedicados al mantenimiento del software. Como se verá más adelante, este modelo nos permitió identificar algunos aspectos a considerar al momento de diseñar una herramienta de soporte a la administración del conocimiento en el mantenimiento del software. Por último, se presentaron los esfuerzos que han sido realizados para aplicar la administración del conocimiento dentro de las organizaciones de desarrollo de software.

De la información presentada en este capítulo es posible identificar tres aspectos importantes: 1) los estudios sobre administración del conocimiento dentro de la ingeniería del software se han orientado a la reutilización de experiencias para mejorar el rendimiento

de los desarrolladores, a la vez que se aumenta la calidad de los productos y se reducen los costos; 2) estudios recientes sobre herramientas para la administración del conocimiento, se han orientado en la utilización de agentes de software para reducir la carga de trabajo de los usuarios; y 3) hacen falta herramientas de administración del conocimiento que den soporte a las necesidades particulares de los grupos dedicados al mantenimiento del software.

Para identificar la manera de apoyar a los encargados del mantenimiento mediante una herramienta de administración del conocimiento, se realizaron dos casos de estudio en dos grupos de mantenimiento de software. Estos casos de estudio estuvieron orientados a identificar los requerimientos básicos que debe cubrir una herramienta como la mencionada, así como corroborar los tipos de conocimiento y fuentes del mismo que fueron definidas en este capítulo, con las utilizadas por los encargados del mantenimiento en un ambiente de trabajo real. El siguiente capítulo presenta los resultados de estos dos casos de estudio.

Capítulo IV. Casos de Estudio en dos grupos de Mantenimiento del Software

“Nunca perdáis contacto con el suelo, porque solo así tendréis una idea aproximada de vuestra estatura”.

Antonio Machado

Solo conociendo nuestra situación actual podemos saber como mejorarla. Con el fin de identificar las actividades del mantenimiento del software realizadas en un ambiente de trabajo real, se llevaron a cabo dos casos de estudio, los cuales estuvieron encaminados a identificar las fuentes de información que usan los ingenieros encargados del mantenimiento de software, los actores que participan en las actividades de mantenimiento, la forma en que interactúan para el logro de sus actividades, y la manera en que las llevan a cabo. Los casos de estudio se realizaron con el fin de buscar características que deben ser cubiertas por una herramienta de administración del conocimiento que pueda ser de apoyo a distintos grupos de mantenimiento del software.

Originalmente sólo se contempló la realización de uno de los caso de estudio. El objetivo fue identificar la manera en que la administración del conocimiento puede ayudar en la solución de los problemas que se presentan dentro de un grupo real de mantenimiento de software. El segundo caso de estudio se contempló una vez concluido el primero, y se realizó con el fin de contrastar la información obtenida, y ver que tan distintos o similares son los problemas que se presentan en dos grupos de mantenimiento con diferentes esquemas de trabajo.

En este capítulo se presentan los principales datos recabados durante la realización de los casos de estudio mencionados (mayores detalles se encuentran en el Apéndice A, y en Rodríguez y Martínez, 2003). Primero se da una descripción de los dos grupos estudiados. Posteriormente se describen los principales procesos llevados a cabo por estos, así como los aspectos técnicos y sociales que fueron considerados los más relevantes para el presente

trabajo. En seguida se muestra una serie de escenarios que permiten identificar la manera en que una herramienta de administración del conocimiento puede ayudar a solucionar algunos de los problemas encontrados. Por último se presentan las características identificadas como básicas para el desarrollo de una herramienta que de soporte a la administración del conocimiento en el proceso de mantenimiento del software.

IV.1 Descripción del ambiente de los casos de estudio

Los casos de estudio se realizaron en dos grupos dedicados al mantenimiento del software. El primero se realizó en el Departamento de Informática del Centro de Investigación Científica y de Educación Superior de Ensenada (CICESE); mientras que el segundo en Intersel, una empresa dedicada al desarrollo de sistemas de administración telefónica. A continuación se dan algunos datos específicos de la composición y el giro de cada uno de estos grupos.

IV.1.1 Departamento de informática del CICESE

El Departamento de Informática (DI) del CICESE se encarga de dar mantenimiento a los sistemas que dan soporte a las distintas áreas de las que está constituido el Centro (DI_CICESE, 2003). El CICESE es una institución dedicada a la investigación y formación de recursos humanos a nivel posgrado; para esto requiere de diversas áreas que dan apoyo tanto a las actividades administrativas como académicas del centro, mismas que utilizan diversos sistemas de software para poder realizar sus actividades. Las áreas principales a las que atiende el DI son:

- Subdirección de Administración Presupuestal;
- Subdirección de Recursos Humanos;
- Subdirección de Recursos Materiales;
- Subdirección de Recursos Financieros;
- Dirección de Estudios de Posgrado;
- así como a las distintas divisiones y departamentos académicos del Centro.

Al momento del estudio, el personal del DI estaba constituido de 14 personas: un Jefe de departamento, una secretaria, cinco técnicos, y 7 becarios y personal por honorarios o externo. Las distintas personas participantes en el departamento tienen perfiles muy variados. Dentro de los técnicos, el tiempo de experiencia dentro del departamento variaba en intervalos de 2 a 5 años. Mientras que entre los becarios era de 3 meses a 2 años. Algunos tenían experiencia previa en otras organizaciones, mientras que otros iniciaron su trabajo dentro del DI.

Cada uno de los técnicos del DI se encarga de atender a alguna de las áreas del CICESE. El técnico asignado a cada área es el responsable de los sistemas de software de la misma; se encarga de atender las solicitudes de cambios de los usuarios del área, así como de ver que estas solicitudes se realicen. Por lo general los técnicos son quienes llevan a cabo la identificación de requerimientos, y quienes realizan las modificaciones a los sistemas. En caso de que no cuenten con el tiempo, o el tipo de modificaciones así lo requieran, solicitan la ayuda de becarios o personal externo. Pero la responsabilidad de las modificaciones a los sistemas sigue estando a cargo del técnico.

IV.1.2 Intersel

Intersel es una empresa dedicada al desarrollo de software de administración telefónica. Actualmente desarrolla y mantiene diversos productos de software, donde el principal es el sistema Intertel. La empresa Intersel fue creada en 1989 con la versión inicial del sistema Intertel. El cual ha ido evolucionando durante mas de 10 años (INTERSEL, 2003). Actualmente la empresa Intersel mantiene distintas versiones del sistema Intertel, y da soporte a clientes distribuidos en toda la república Mexicana, y parte de Sudamérica.

Intersel se constituye de cinco departamentos: el departamento de administración, encargado de las tareas de gerencia, contabilidad, recepción y administración; ventas, que se encarga de la comercialización de los distintos productos manejados por la empresa;

soporte técnico, encargado de la atención a los distribuidores, clientes y usuarios del producto; producción, que realiza las copias del software, así como el empaquetado de las mismas; y por último, el departamento de investigación y desarrollo, donde se llevan a cabo las tareas de desarrollo y mantenimiento de los sistemas de software. La figura 7 muestra esta distribución.

Departamento de Ventas	Departamento de Soporte Técnico	Departamento de Producción	Departamento de Administración
Departamento de Investigación y Desarrollo			

Figura 7. Organización interna de Intersel

El área de desarrollo se divide en tres sub-áreas: una encargada de las labores propias del desarrollo y mantenimiento de sistemas, otra encargada de la documentación para los usuarios del sistema, y otra más que se encarga de mantener al día las tarifas telefónicas utilizadas por los sistemas de administración telefónica desarrollados y mantenidos por el departamento.

Al momento de la realización del caso de estudio, el área de investigación y desarrollo estaba constituida de 8 personas: el jefe del departamento y 5 ingenieros de software encargados del desarrollo y mantenimiento de los sistemas, una persona encargada del manejo de las tarifas telefónicas, y otra encargada de la documentación de usuarios. La experiencia del personal de mantenimiento, si bien variada, resultó ser considerable, cuatro de ellos, y el jefe del departamento, contaban con más de cuatro años laborando dentro del grupo, y con anterioridad trabajaron en otras empresas, también como desarrolladores; mientras que uno de ellos resultó ser de reciente ingreso (entre dos o tres meses), y no tenía más de un año de haber terminado sus estudios de licenciatura.

La organización del grupo de desarrollo es mediante proyectos. Cada proyecto es dirigido por un líder de proyecto, el cual se mantiene en contacto directo con el jefe del departamento, y se encarga de la elaboración del plan de proyecto. Normalmente se

manejan un máximo de dos proyectos durante un mismo período. Por lo general, cada ingeniero de software es asignado a uno de estos proyectos, pero esto no exenta que pueda dar apoyo en otros si así se requiere. Los proyectos pueden variar en cuanto a tiempo, desde dos o tres semanas, hasta años, pero se busca que no pasen de ocho o diez meses; en caso de ser así, se busca la manera de dividir el proyecto en proyectos más pequeños con períodos más cortos.

La descripción de los ambientes existentes en los dos lugares donde se realizaron los casos de estudio, da una idea de la composición y el giro de cada uno de los dos grupos estudiados. Una vez definidos estos ambientes, se describe la metodología que se siguió para la realización de los casos de estudio, así como los pasos que la componen.

IV.2 Metodología seguida durante los casos de estudio

Para lograr la mejora de cualquier proceso (en nuestro caso particular el proceso de mantenimiento de software) y brindar el soporte apropiado a una organización, es necesario primero lograr un entendimiento de lo que sucede dentro del mismo (Briand *et al.*, 1998). Existen diversos métodos y técnicas que se pueden utilizar para el estudio del proceso de mantenimiento de una organización (Singer y Lethbridge, 1996). Dentro de los métodos y técnicas más comunes para la captura de procesos, y que han arrojado buenos resultados no solo en el estudio del mantenimiento de software sino en estudios de ingeniería de software en general, se encuentran las entrevistas (Curtis *et al.*, 1988; Dart *et al.*, 1993; Singer, 1998; Briand *et al.*, 1998). Los dos casos de estudio se realizaron principalmente mediante entrevistas semi-estructuradas al personal encargado del mantenimiento; y en menor medida en la observación de la manera en que éstos trabajan. Estas observaciones tuvieron lugar durante la realización de las entrevistas.

El estudio del proceso de mantenimiento de software puede dividirse en dos fases principales: la primera corresponde a un análisis descriptivo de los procesos, con el fin de modelarlos para lograr su entendimiento; la segunda es la realización de una evaluación

cualitativa de los mismos (Briand *et al.*, 1998). Para llevar a cabo estas dos fases en los dos casos de estudio, se tomó como base un enfoque de ingeniería de procesos, específicamente se utilizó la metodología de análisis y diseño de procesos (PADM por sus siglas en inglés).

Existen diversas metodologías a seguir para el modelado de procesos de desarrollo de software, las cuales buscan objetivos diversos (Curtis *et al.*, 1992); dentro de éstas, la metodología PADM permite el modelado de procesos complejos y no estructurados (lo que resulta de utilidad en nuestro caso particular) (Wastell *et al.*, 1994). PADM toma en consideración dos enfoques en el estudio de procesos, el primero es lograr un entendimiento del proceso en términos generales, para posteriormente comparar esta representación con el proceso real, a fin de ir acercándonos más a éste en cada iteración. PADM también considera a los procesos como sistemas socio-técnicos, donde es necesario encontrar la manera en que interactúa el subsistema técnico con el subsistema social. Una de las principales ventajas de PADM es que es una metodología abierta y adaptable; permite complementarse con otras técnicas o metodologías. Esto último lo hemos considerado de vital importancia en la realización de nuestros casos de estudio.

PADM define cuatro etapas principales para el estudio de procesos:

- *Definición del proceso*, que incluye el identificar las entradas y salidas, así como los actores que interactúan y los roles que juegan dentro del mismo.
- *Captura y representación*, comprende el modelado del proceso; la realización de una representación gráfica del mismo.
- *Evaluación*, se centra en la identificación de las fallas y debilidades del proceso.
- *Rediseño*, comprende el diseño de un nuevo proceso a seguir.

En los casos de estudio aquí expuestos solo se cubrieron las tres primeras fases de PADM.

IV.2.1 Descripción de la captura de los procesos de los grupos estudiados.

Como mencionamos anteriormente, ambos estudios se basaron en entrevistas y observaciones. Para realizar las entrevistas y grabarlas en audio, se solicitó la aprobación,

del jefe del departamento en el caso del DI, y del director general en conjunto con el jefe del departamento de desarrollo en el caso de Intersel. En ambos casos, se acordó con el jefe del departamento el calendario a seguir para la realización de las entrevistas. Todas las entrevistas fueron grabadas en audio. Las entrevistas fueron realizadas principalmente en los espacios de trabajo de cada uno de los entrevistados. En ambos casos, solo una entrevista se realizó en un lugar distinto. En el caso del DI, la entrevista realizada al personal de apoyo del departamento (becarios) se hizo en una sala de reuniones, mientras que en el caso de Intersel, esto sucedió con la entrevista a uno de los encargados del área de soporte técnico o atención a clientes. En ambos casos, el jefe del departamento se encargó de informar al personal acerca del estudio a realizar.

En el caso del DI se realizaron un total de 12 entrevistas de entre 45 y 60 minutos cada una. En cada entrevista se les informó a las personas la intención de grabarlas en audio, por lo que se solicitó su consentimiento, comprometiéndonos a guardar su anonimato. Tres de las entrevistas se le realizaron al jefe del departamento, la primera con el fin de establecer el objetivo de este trabajo y obtener una primera visión de la estructura interna de los procesos y grupos de trabajo del departamento. En una primera instancia se consideró entrevistar a la totalidad de los miembros del departamento, sin embargo, sólo se entrevistó a 10 de ellos. Siete de las entrevistas se hicieron al personal técnico de base. A tres de ellos se les entrevistó dos veces, mientras que a otro no fue posible entrevistarlos ya que se encontraba en período vacacional. Otra de las entrevistas se le hizo a la secretaria del departamento. Por último, se realizó una entrevista conjunta con cuatro de los becarios, ya que tres de ellos no se presentaron ese día.

En el caso de Intersel se realizaron cuatro entrevistas de una hora cada una. La primera se hizo en conjunto con el director general de la empresa y el jefe del departamento de investigación y desarrollo. Esta entrevista fue dirigida a exponer el motivo del estudio, así como obtener una visión preliminar de los procesos principales que realiza el departamento y de la estructura interna del mismo. En esta sesión también se determinó qué personas serían entrevistadas posteriormente, lo cual se hizo tomando en consideración el

conocimiento que tienen con relación a los procesos que son llevados a cabo dentro del departamento, especialmente los relacionados con el mantenimiento del software que desarrolla la empresa. Esta selección se hizo con apoyo del jefe del departamento de desarrollo. Se determinó que se realizarían entrevistas a las siguientes personas: una al mismo jefe del departamento, otra a uno de los ingenieros de software con mayor experiencia dentro del departamento, y otra más al responsable del área de soporte técnico y atención a clientes. Cabe señalar que el objetivo principal de la realización de este caso de estudio fue hacer una comparación con los principales datos obtenidos en el primero, es por ello que no se realizó un estudio más a fondo. Otro punto a considerar es que parte de la información con la que se complementó a las entrevistas realizadas, fue la experiencia de un año del entrevistador como parte del grupo estudiado, por lo que las entrevistas se orientaron a aspectos más específicos que en el primer caso de estudio.

Las entrevistas estuvieron encaminadas a identificar las actividades que realizan los entrevistados, y cómo las llevan a cabo; qué tipo de conocimiento requieren para realizarlas, y qué conocimiento consideran que debe tener una persona que recién inicia a trabajar con esas actividades; así como qué tipos de fuentes de información y conocimiento consultan, y cuáles consideran más importantes o consultan con mayor frecuencia.

Durante el estudio se hicieron algunas observaciones de la manera en que los distintos miembros de los grupos estudiados trabajaban e interactuaban entre ellos, así como del ambiente de trabajo; qué tipo de información o documentos tienen a la mano, cuáles tienen en sus computadoras y cómo los organizan. Esto se realizó mientras se llevaban a cabo las entrevistas, y durante los tiempos de espera cuando las personas que serían entrevistadas se encontraban ocupadas con sus actividades. Las observaciones permitieron complementar los datos obtenidos con las entrevistas. A continuación se describe la técnica empleada para el modelado de los procesos identificados mediante las entrevistas realizadas. Posteriormente se presenta el modelado y evaluación de los mismos.

IV.2.2 Técnica empleada para el modelado de los procesos de los grupos estudiados

Una de las etapas del modelado de procesos comprende la representación gráfica del mismo. Existen distintas técnicas diagramáticas para el modelado de procesos. En nuestro caso hemos decidido utilizar la técnica conocida como Gráfica Rica, desarrollada por Peter Checkland como parte de la Metodología de Sistemas Suaves (Monk y Howard, 1998; Checkland y Scholes, 1999), la cual ha sido tomada como base para el desarrollo de la metodología PADM antes mencionada.

Una gráfica rica permite obtener una visión global del proceso, así como representarlo a distintos niveles de detalle. Mediante esta técnica es posible mostrar a los actores o personas involucradas en el proceso, así como su propósito en el mismo, sus deseos, miedos e inquietudes. También es posible mostrar los conflictos o acuerdos existentes, así como los detalles del medio ambiente en el que se desarrolla el proceso. La figura 8 presenta ejemplos de los elementos de la simbología de la técnica de gráfica rica descritos a continuación:



Figura 8. Simbología de las gráficas ricas.

- **Roles**, representados mediante algún ícono (el que se considere más apropiado).
- **Artefactos**, salidas o entradas de las actividades, los cuales se pueden representar por cajas o íconos (lo que se considere más apropiado).
- **Actividades**, son tareas o interacciones de los roles y son representadas por nubes.
- **Acciones de los roles**, pueden ser representadas como nubes de pensamiento. También pueden servir para expresar las inquietudes, miedos, preocupaciones, molestias, etc. de las personas que ejecutan dichos roles.

- **Conexiones.** Existen dos tipos de conexiones o relaciones principales entre los elementos de una gráfica rica: los roles se conectan con las actividades mediante líneas, mientras que los artefactos lo hacen mediante flechas, cuya dirección indica si son entradas, salidas o son modificados por dicha actividad.

Sin embargo, debido a que la técnica de gráfica rica es abierta, permite ser complementada si así se requiere. En nuestro caso hemos utilizado dos agregados a estos elementos. El primero son las líneas interrumpidas o punteadas, las cuales indican posibilidad, es decir, que un determinado elemento o enlace puede existir o no, o que puede ser una de varias posibilidades. El segundo tiene el mismo fin, indicar que el elemento puede no existir al poner una cruz con líneas discontinuas sobre él.

IV.3 Modelado y evaluación de los procesos llevados a cabo por los grupos estudiados

Las entrevistas realizadas permitieron identificar los procesos principales que ambos grupos siguen durante el mantenimiento de software, por ejemplo, la atención de problemas de los usuarios; los tipos de solicitudes de cambios que manejan, así como la manera en que estas solicitudes son tratadas por cada uno de los grupos; y las actividades involucradas en las modificaciones dentro de los sistemas, como por ejemplo, los mecanismos de control de versiones de cada grupo. También se buscó identificar las fuentes de información y conocimiento que éstos consultan, y el tipo de conocimiento que requieren. Por último, se identificaron algunos de los aspectos técnicos y sociales más relevantes en ambos grupos, como son: problemas relacionados con la documentación de los sistemas a los que dan mantenimiento, y problemas de pérdida y desaprovechamiento del conocimiento.

A continuación se presentan los principales procesos que se llevan a cabo en ambos grupos estudiados. Posteriormente se presentan los principales aspectos técnicos y sociales que fueron encontrados. Toda esta información nos permitió identificar escenarios que facilitan la apreciación de las ventajas que puede proporcionar un sistema de administración de

conocimiento que apoye al personal encargado del mantenimiento del software, así como definir las características básicas que deben ser cubiertas por el mismo.

IV.3.1 Procesos identificados durante los casos de estudio

Entre los principales aspectos que se buscó identificar en ambos casos de estudio, está el proceso por el cual los usuarios o clientes son atendidos para solucionar sus problemas con un sistema dado, el proceso que se sigue para generar las solicitudes de cambios al mismo, la manera en que se realiza el proceso de implementación de los cambios solicitados, y la forma en que es llevado el control de versiones de los sistemas. También se buscó identificar el tipo de conocimiento requerido durante estas tareas, así como los mecanismos que utilizan para obtenerlo y compartirlo entre ellos.

A continuación se describen los procesos mencionados, contrastando las diferencias y similitudes existentes entre los dos casos de estudio. Estos procesos se identificaron tomando como base la información proporcionada por los entrevistados. En particular se tratan cuatro aspectos principales: 1) el proceso de atención de clientes o usuarios, 2) el proceso de realización y atención de solicitudes de mantenimiento, 3) el proceso de implementación de cambios y control de versiones, y 4) las diferencias y similitudes relacionadas con el conocimiento en cada grupo. Cabe hacer mención que en ambos grupos se refiere de distinta manera a los ingenieros de mantenimiento. En el caso del DI se les llama técnicos, mientras que en Intersel se les denomina ingenieros de software, es por ello que durante la descripción de los procesos aquí presentada, estos términos se utilizan indistintamente para referirse a los ingenieros de mantenimiento.

IV.3.1.1 Proceso de atención a clientes

Fue posible identificar que el proceso de atención de clientes en Intersel resulta muy distinto al que es llevado a cabo en el DI del CICESE; sin embargo también existen algunas similitudes. La primer diferencia radica en el hecho de que el DI trabaja para dar soporte a una sola organización que es el CICESE, mientras que Intersel distribuye sus servicios a

una gran cantidad de organizaciones dentro del país e incluso en el extranjero. También, a diferencia del DI, donde los ingenieros encargados del mantenimiento de los sistemas tienen una comunicación constante con los usuarios, los ingenieros de Intersel rara vez se comunican con alguno, ya que manejan un proceso que contempla intermediarios entre los ingenieros de mantenimiento y los usuarios finales.

Para dejar un poco más claro el proceso aquí mencionado, se presenta un escenario típico de atención al cliente en cada uno de los grupos estudiados. Debido a que la organización y procesos de cada una de las empresas que utilizan los sistemas mantenidos por los grupos estudiados, son distintos, hemos decidido agrupar, tanto al cliente como al usuario en un solo tipo de actor, al que hemos denominado cliente.

En Intersel se maneja un esquema de atención a clientes por estratos. Intersel maneja distribuidores para hacer llegar sus sistemas a los clientes. Estos distribuidores sirven como intermediarios entre los clientes e Intersel. La figura 9 muestra como se realiza este proceso. En ella se ve como el cliente hace su solicitud al distribuidor que le vendió el sistema. Si el distribuidor puede solucionar el problema lo hace inmediatamente, de no ser así, se comunica con el departamento de soporte técnico de Intersel para que se le apoye en la solución. Posteriormente, la solución sigue la misma ruta, del departamento de soporte técnico al distribuidor, y de éste al cliente.

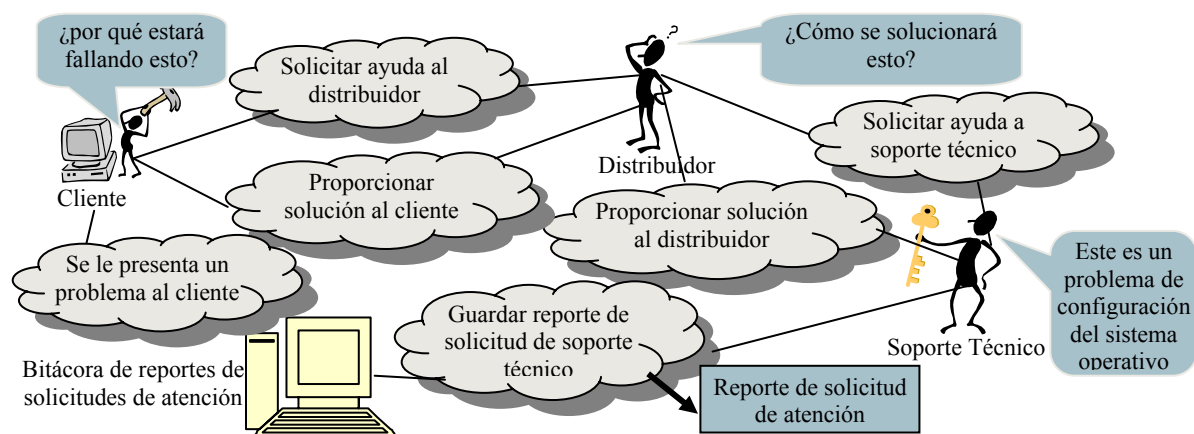


Figura 9. Proceso de atención a clientes en Intersel

Es importante mencionar que los encargados de soporte técnico mantienen una bitácora electrónica de las solicitudes atendidas, por lo que cualquier solicitud de atención, por insignificante que ésta sea, es almacenada en la misma. Entre los datos que son capturados en la bitácora se encuentra el nombre del cliente o distribuidor que solicita el soporte, la fecha en que se realiza la solicitud, el problema reportado, la solución que se le dio, así como la persona que realizó la atención. Esta bitácora es utilizada para identificar las solicitudes más comunes, así como las soluciones que se dieron a estas. Con base en estos datos, se desarrollan programas de inducción y capacitación para los nuevos miembros del departamento de soporte, y para los distribuidores. Otra de las aportaciones para las que ha servido esta bitácora, es la realización de un sistema de preguntas frecuentes que puede ser accedido, a través de la página de Internet de Intersel, por cualquier distribuidor o cliente.

Por otro lado, en el DI la atención a clientes puede ser realizada tanto por la secretaria como por los mismos técnicos. En este caso, podríamos decir que la secretaria juega un papel parecido al del grupo de soporte técnico de Intersel, sin embargo, existen problemas que no pueden ser atendidos por ella, por lo que los turna al técnico encargado. La figura 10 muestra este proceso.

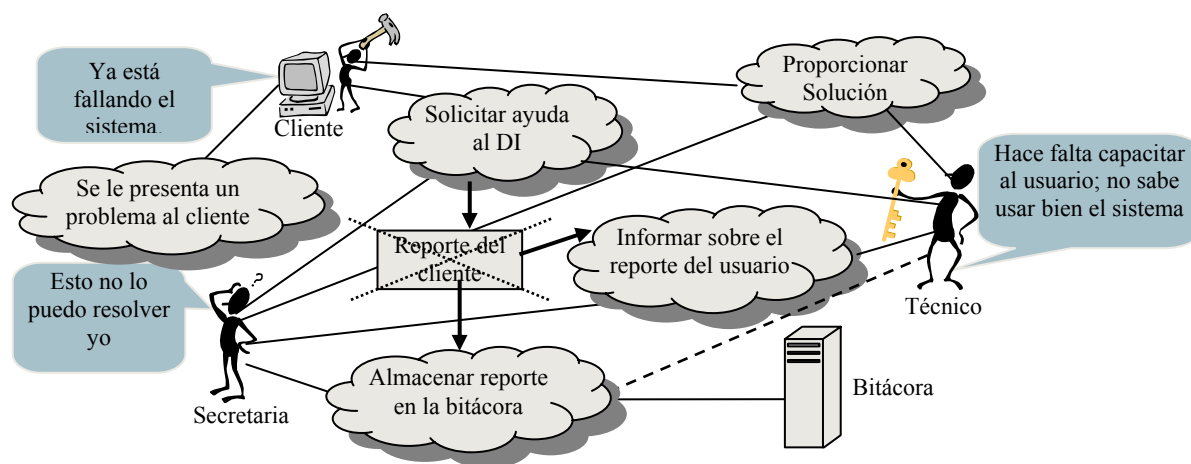


Figura 10. Proceso de atención a clientes en el DI

Es posible observar que en ambos casos existe una bitácora donde se capturan las solicitudes de atención, pero, mientras que en Intersel todas las solicitudes son capturadas,

en el DI por lo general únicamente las solicitudes hechas a través de la secretaria son capturadas. Los técnicos rara vez capturan estas solicitudes; sobre todo si el problema no es propio del sistema, sino que es falta de capacitación del usuario.

Otra de las similitudes identificadas, es que la mayoría de las fallas o problemas reportados por los clientes no son fallas del sistema, sino más bien problemas de desconocimiento por parte de estos; tanto desconocimiento del uso del sistema, como falta de capacitación en el uso de computadoras en general. Por ejemplo, desconocimiento del manejo y configuración del sistema operativo.

IV.3.1.2 Proceso de realización y atención de solicitudes de modificaciones

Con respecto a las solicitudes de cambios, las diferencias son todavía más marcadas. Primeramente, las solicitudes de cambio en Intersel surgen de una serie de ideas que pueden provenir de cualquiera de los miembros de la empresa. Para esto, se ha implementado un mecanismo para compartir ideas, conocimiento e intereses comunes. Intersel tiene un sistema en el cual cada uno de los miembros de la empresa puede suscribirse a una serie de listas de correo y carpetas utilizadas para compartir ideas entre ellos. En una de estas carpetas compartidas es donde los miembros de la empresa pueden aportar ideas para la mejora de los distintos productos que manejan. Estas ideas pueden provenir de comentarios hechos por los distribuidores o clientes, por problemas detectados en los sistemas, por las características de los productos de la competencia, en fin, por una gran cantidad de factores.

Cuando se decide desarrollar una nueva versión del producto, los líderes de las distintas áreas se reúnen para determinar las características que contendrá la misma. Una vez que se ponen de acuerdo, se genera un documento con los requerimientos de la nueva versión, el cual es firmado por todos los involucrados y enviado al departamento de desarrollo. En la figura 11 se muestra una representación de este proceso.

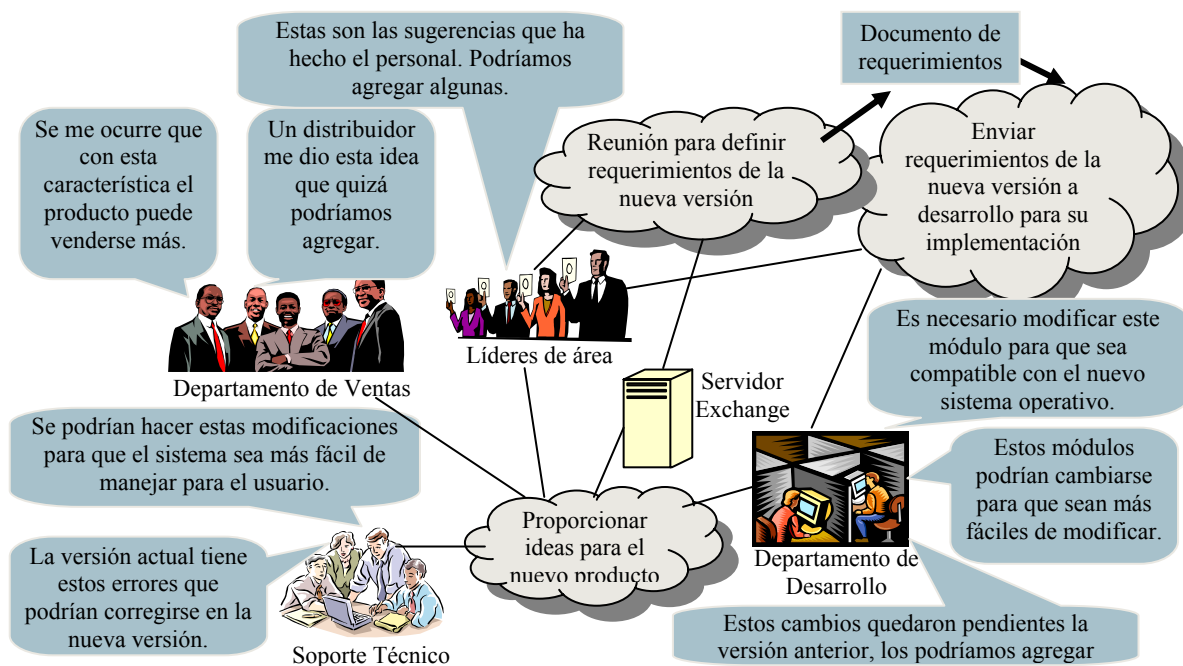


Figura 11. Proceso de realización de una solicitud de modificaciones en Interseal

Ya que se han definido los requerimientos y que estos han sido enviados al departamento de desarrollo, el jefe del departamento se reúne con la persona que se hará responsable del proyecto, y demás ingenieros de software que participarán en las modificaciones. Esto se hace con el fin de determinar las actividades que requerirán ser realizadas para cubrir los requerimientos planteados, así como para determinar los tiempos que estas actividades consumirán. Posteriormente se crea un proyecto en el cual se plasman estas actividades con sus tiempos requeridos, así como la persona responsable de llevar a cabo cada una de ellas. La asignación de las actividades se hace tomando como criterio la disponibilidad en tiempo, la carga de trabajo, así como la experiencia de cada ingeniero de software con los módulos que serán modificados. El líder del proyecto realiza también un documento donde establece las características del proyecto, sus objetivos, alcances y recursos que serán requeridos. Se identifican los principales riesgos que pudieran presentarse, así como la manera en que serán abordados en caso de que se presenten. La figura 12 muestra una representación gráfica de este proceso.

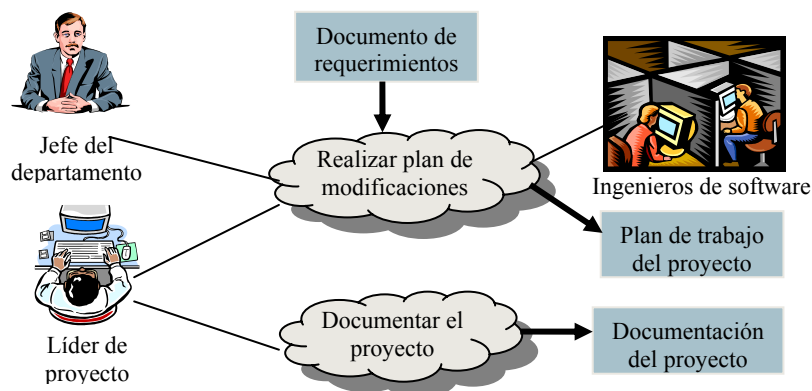


Figura 12. Elaboración del proyecto de modificaciones en Interseal

Cuando ya ha sido realizado el documento del plan de proyecto, se presenta a los líderes de área para su aprobación. Si el documento es aprobado, se inician las modificaciones, de lo contrario, se negocia con las personas que no estén de acuerdo con las condiciones. Estas negociaciones pueden resultar por la necesidad de terminar el proyecto en un menor tiempo o con una menor cantidad de recursos, entre otros factores. Con frecuencia, en estos casos, la solución resulta ser la disminución de los requerimientos a cubrir. Si durante la realización del proyecto alguna de las áreas decide agregar o eliminar algún requerimiento, es necesario que esto quede por escrito y que cada uno de los involucrados firme de enterado. Este documento se anexa al documento de requerimientos inicial.

En contraste con lo anteriormente expuesto, el proceso de solicitudes de modificaciones en el DI, en la mayoría de los casos es realizado entre el cliente y el técnico responsable del área. Además de que no queda mucha documentación al respecto, salvo los correos o memorandums enviados por el cliente, donde plasma las modificaciones que está solicitando. Sin embargo, sobre todo cuando los cambios son grandes o su implementación requiere una cantidad de tiempo considerable, los requerimientos son establecidos entre el técnico responsable de los sistemas del área, el jefe del departamento, así como el encargado del área que solicita las modificaciones. En estos casos, el DI desarrolla un proyecto donde define las actividades a realizar, así como el tiempo que éstas requerirán. Este proceso es mostrado en la figura 13.

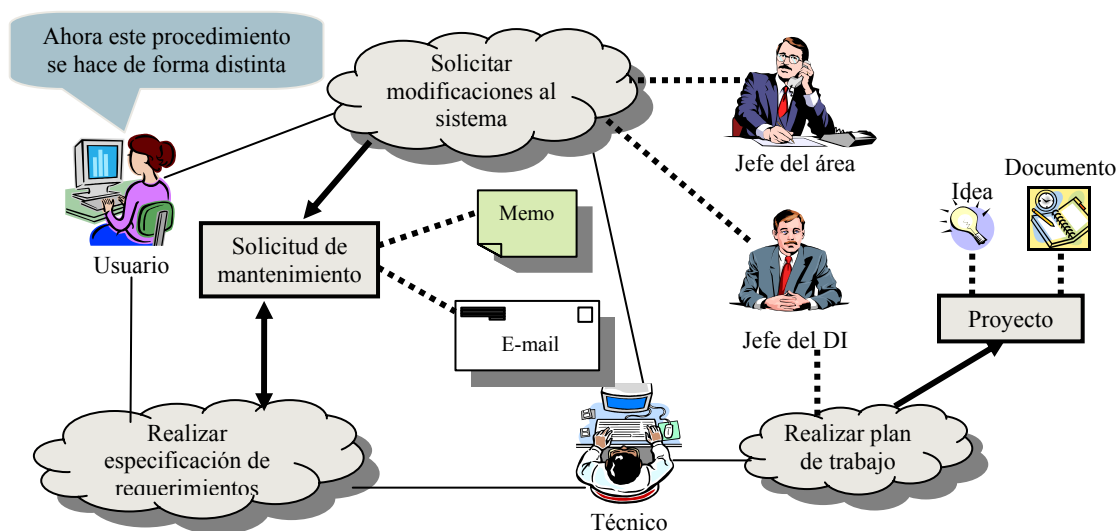


Figura 13. Proceso de solicitud de modificaciones en el DI

Como es posible ver, si bien los procesos en Intersel y en el DI son distintos, existen algunas similitudes entre ellos. Primeramente, existe un mecanismo mediante el cual son enviados los requerimientos a los encargados de llevar a cabo las modificaciones, aun cuando en Intersel este proceso es más formal y está mejor documentado. Otra similitud es la existencia de un proyecto con las actividades a realizar, los tiempos que estas requerirán, así como las personas asignadas a cada una de ellas. Sin embargo, en el DI no en todas las solicitudes de cambios se genera un proyecto de este tipo. Por último, también fue posible observar en ambos casos, que los proyectos generados sirven para que quienes solicitaron los cambios (los líderes de área en Intersel, y los usuarios o jefes de área en el DI) puedan dar seguimiento al estado de los mismos. Es decir, puedan conocer en un momento determinado el porcentaje de avance de cada proyecto.

IV.3.1.3 Proceso de implementación de cambios y control de versiones

El proceso de implementación de los cambios, en términos generales y a nivel individual, resulta significativamente similar; mas no el control de versiones. Para ver esto claramente supongamos lo que sucedería en un escenario típico en Intersel, haciendo comparaciones con lo que sucedería en el DI.

Primeramente, una vez que se han determinado los requerimientos y se ha definido el proyecto con las tareas que serán realizadas por los involucrados, el siguiente paso es llevar a cabo estas tareas. En el DI por lo general las modificaciones son implementadas por una sola persona: el responsable del área a la que pertenece el sistema a modificar; mientras que en Intersel son dos o más personas las involucradas. Sin embargo, a nivel individual, al momento de identificar las partes del sistema que deben ser modificadas, cada técnico o ingeniero de mantenimiento realiza un proceso similar. Como se muestra en la figura 14, primeramente es necesario identificar los programas, tablas, reportes, etc. que requerirán ser modificados. Para esto, en ambos casos los ingenieros se basan en la experiencia que cada quien tiene con el sistema. Si ésta no es suficiente, se consulta directamente el código del sistema, se recurre a la documentación existente, o se consulta a alguno de los compañeros que haya trabajado previamente con el sistema. La principal diferencia aquí, es que en el DI con frecuencia se consulta a los usuarios del sistema para resolver las dudas que pudiera tener el técnico, mientras que en Intersel esto no sucede.

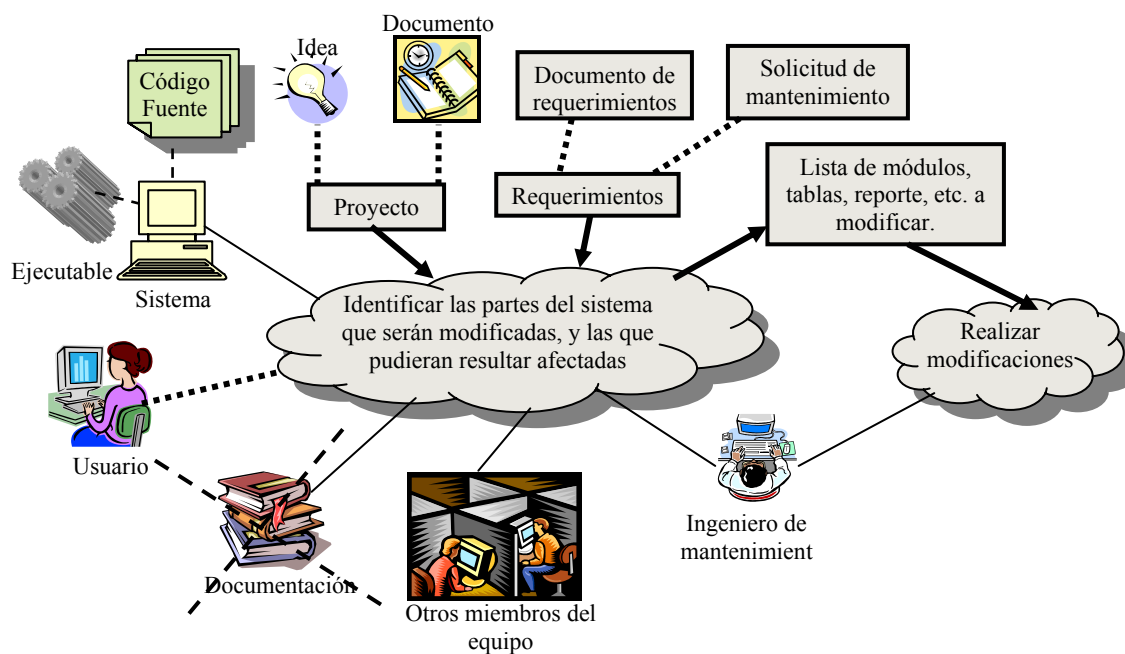


Figura 14. Identificación de las partes del sistema a modificar

En ambos casos fue posible observar mediante las entrevistas, que dos puntos importantes a considerar durante las modificaciones son la identificación de los archivos fuente que requerirán ser modificados, así como el prever qué otros programas, módulos o sistemas pueden ser afectados por los cambios. También en ambos casos se tiene un proyecto o plan a seguir para la realización de las modificaciones. Sin embargo, en el caso del DI, este plan o proyecto no siempre existe en un documento formal, ya que puede ser solo un conjunto de ideas en la mente del técnico encargado de realizarlo. Por último es posible ver que si bien la manera de establecer los requerimientos es distinta, en ambos casos existe, en menor o mayor grado, una serie de requerimientos que definen los cambios a realizar.

Una vez que el ingeniero ha identificado los archivos que requiere revisar, los baja a su máquina desde un contenedor central, realiza las modificaciones que se requieran, y regresa los archivos modificados al repositorio central. En este punto es donde se presentan las mayores diferencias entre el DI e Intersel, ya que en Intersel el control de versiones de los archivos fuente se hace mediante un sistema que puede considerarse semi-automatizado, debido a que utilizan un programa de software que se encarga de este control.

Cuando un ingeniero en Intersel requiere modificar algún archivo, solicita al programa que le proporcione permisos de escritura sobre el archivo, en caso de que este archivo esté siendo modificado por alguien más, el sistema se lo informa, de manera que es posible evitar, en la mayoría de los casos, que dos personas modifiquen un archivo al mismo tiempo. De esta manera reduce el riesgo de que al subir los archivos modificados se sobre escriban unos a otros. Sin embargo, en caso de que se presente esta situación, el programa mencionado maneja un historial de los cambios realizados a cada archivo, y datos como: la persona que hizo la modificación, la fecha en que se hizo y las diferencias en el código fuente de una versión a otra.

En contraste, en el DI no se maneja ningún programa para apoyar en estas tareas. Lo que se hace es asignar un directorio dentro del servidor de archivos a cada responsable de área, y se le proporcionan permisos de acceso sólo a este último. Los archivos fuente de los

sistemas que están a cargo de este técnico son almacenados en dicho directorio. Este técnico es el responsable de que las versiones de los archivos que se encuentren en el directorio mencionado sean las correctas. Si por alguna razón se sobre escribe algún archivo con una versión incorrecta, no habrá manera de recuperarlo a menos de que el técnico cuente con algún respaldo.

IV.3.1.4 Diferencias y similitudes relacionadas con el conocimiento

Fue posible identificar que los conocimientos generales requeridos por los ingenieros en Intersel son significativamente similares a los requeridos por los técnicos del DI. Ambos se asemejan a los tipos de conocimiento definidos en el modelo de tipos y fuentes de conocimiento descrito en el capítulo III. En ambos casos estos conocimientos pueden dividirse en dos grupos: independiente y dependiente de la aplicación.

Con respecto al conocimiento dependiente de la aplicación, las principales diferencias radican en los detalles de las aplicaciones o sistemas que manejan. Por ejemplo, los lenguajes de programación en los que están desarrollados, así como los ambientes de desarrollo. En el caso de Intersel se maneja principalmente el ambiente de desarrollo Visual Studio de Microsoft, mientras que en el DI se maneja el proporcionado por Oracle. Otro ejemplo son las bases de datos, ya que en Intersel se maneja principalmente FoxPro y SQL Server, mientras que en el DI es precisamente Oracle. Sin embargo, como es posible ver, en términos generales el tipo de conocimiento requerido es similar: programación, lenguajes de programación, manejo de bases de datos y de los sistemas operativos en donde corren las aplicaciones que manejan.

Donde se presentan las principales diferencias es en el conocimiento fuera del dominio de la aplicación que en ambos casos se considera importante para los ingenieros de software. Intersel ha tomado en cuenta el papel que juega el conocimiento dentro de la organización, por lo que ha hecho esfuerzos por aprovechar e incrementar el conocimiento de cada uno de sus miembros. Para lograrlo, la empresa ha desarrollado un sistema para compartir conocimiento en el que las personas pueden suscribirse a listas de correos y carpetas

compartidas, según sus intereses. A la par de lo anterior, se realizan cursos de capacitación para todo el personal, con temas de diversa índole. Algunos de los cursos son generales para todos los miembros de la empresa; por ejemplo, cursos de superación personal y liderazgo, entre otros. Mientras que otros están relacionados con las tareas propias del área a la que pertenece la persona; por ejemplo, cursos sobre el manejo de un determinado lenguaje de programación en el departamento de desarrollo.

Otra de las diferencias relacionadas con el conocimiento en ambos casos, es el riesgo de la pérdida del mismo. En el DI, cada técnico se encarga de los sistemas de un área específica, y por lo general es sólo él quien realiza las modificaciones a los sistemas de esa área. Esto conlleva que con el tiempo el técnico se haga experto en el manejo de esos sistemas, sin embargo, presenta el riesgo de que si el técnico deja el departamento, no exista personal con los conocimientos suficientes para dar soporte al sistema con la misma eficiencia. Aún cuando este problema también puede presentarse en Intersel, el riesgo es menor, ya que, aunque cada ingeniero está especializado en ciertos sistemas y módulos de los mismos, se busca que por lo menos dos personas se encuentren familiarizadas con los distintos módulos de los sistemas. Esto se logra al distribuir las tareas a la hora de realizar modificaciones en algún sistema, tratando de dejar las tareas complicadas a quien tenga la mayor experiencia, pero dejando ciertas tareas para ser realizadas por otro ingeniero, con el fin de que se vaya familiarizando con dicho sistema.

Durante el análisis de los distintos procesos mencionados, así como de la información proporcionada por los entrevistados, se identificaron varios aspectos relevantes, los cuales permitieron observar el estado actual de ambos grupos, así como algunos problemas existentes dentro de los mismos. A continuación se detallan los principales aspectos encontrados.

IV.3.2 Aspectos técnicos y sociales más relevantes identificados

Los aspectos técnicos y sociales identificados fueron muy variados, algunos resultaron ser comunes a las dos organizaciones, mientras que otros se presentaban solo en una de ellas. Principalmente se buscó identificar el conocimiento requerido por los encargados del mantenimiento, las fuentes que utilizan para obtener y compartir este conocimiento, así como la manera en que el entorno laboral influye en estos aspectos. En particular se buscó identificar la problemática y el estado actual que guardan ambos grupos con respecto a estos aspectos mencionados. El objetivo fue identificar las características que debe cubrir una herramienta que de soporte a los problemas de pérdida y desaprovechamiento del conocimiento.

Los distintos aspectos encontrados fueron clasificados en tres categorías: 1) los relacionados con la documentación, 2) los que tienen relación con el conocimiento tácito del personal de mantenimiento, y 3) los aspectos organizacionales que intervienen en esto último. Para la presentación de la información se han extraído algunos fragmentos de las entrevistas realizadas, esto con el fin de mostrar los puntos de vista de los entrevistados. Algunos fragmentos contienen agregados que solo buscan aclarar o poner en contexto la información proporcionada. Estos agregados se presentan entre corchetes cuadrados. Por último, no se hace distinción del caso de estudio de donde fueron obtenidos los fragmentos de entrevista; salvo que el punto tratado sólo se hayan presentado en uno de ellos.

IV.3.2.1 Aspectos relacionados con la documentación

Hay tres aspectos principales relacionados con la documentación. El primero es la falta o baja calidad de la documentación de los sistemas a los que se da mantenimiento, un problema que ya ha sido reportado con anterioridad, como se muestra en el capítulo II. El segundo punto fueron las distintas maneras en que el personal de mantenimiento documenta los cambios que realiza. Y por último, el efecto que tiene la documentación interna del código a la hora de hacer cambios en el mismo. A continuación se detalla cada uno de estos aspectos.

Falta o baja calidad de la documentación. Fue posible detectar que la documentación técnica de los sistemas resulta de gran ayuda a la hora del mantenimiento; cuando existe, es una de las primeras fuentes que se consulta. Sin embargo, es frecuente que esta documentación no exista, no esté completa, sea de difícil acceso, no esté actualizada, e incluso no corresponda con la realidad del sistema que fue desarrollado en un principio.

“...si no tengo experiencia en ese cambio que voy a hacer, recurro a la documentación que existe...”

“... hay manuales técnicos, pero no tan completos. Por ejemplo, cuando yo entré a trabajar con [este sistema], cuando se estaba desarrollando, estaba el documento de análisis de requerimientos y un poquito del diseño ..., ahora sí, que si me los pides, los documentos, no los tenemos a la mano ... es que, en un principio no se documentaba mucho ...”

“ ... la documentación que tenemos, deja tú que no esté actualizada, incluso la documentación del sistema no tiene mucha correspondencia. Tú puedes ver un análisis de lo que es recursos financieros desde hace cuatro años, y ves el sistema y te preguntas: ¿cómo derivaron este sistema del análisis? ... lo que propusiste desde el análisis preliminar, e incluso desde el análisis ya de alto nivel, no tiene mucho que ver ...”

Distintas maneras de documentar los cambios realizados. Pudimos identificar que los ingenieros tienen muy diversas maneras de documentar los cambios que hacen a los sistemas. Algunos documentan el código de los programas que modifican, otros llevan un registro en papel, mientras que otros utilizan los reportes de errores para indicar cuál fue la solución, como en el caso del DI, hay quienes utilizan los reportes de la bitácora para indicar ahí qué fue lo que se hizo para resolver el problema reportado.

Entrevistador.- ¿Cuándo realizas algún cambio dejas algo por escrito, por ejemplo, recibes la solicitud del cliente y dejas plasmado en algún lado qué fue lo que hiciste para solucionar ese problema que te están planteando?

Entrevistado.- En el documento a nivel código donde hago la modificación ... si son modificaciones no muy grandes las que tengo que hacer, entonces las comento a nivel de código...

“... aquí llevamos una bitácora ... de los cambios que se hacen ... explica desde qué fue el problema, hasta cual fue la solución y qué modulo se cambió; ya sea una forma, un reporte, una tabla. Qué fue lo que pasó ... se queda una constancia ahí ...”

“... llevo un registro, bitácora, notas; en este cuaderno ...”

La documentación del código afecta el entendimiento del mismo. Un punto interesante que se detectó fue que la documentación del código resulta de gran ayuda para entender lo que hace, sobre todo cuando éste ha sido desarrollado por otra persona. Esto también ayuda al programador a recordar lo que hace el código que él mismo pudo haber desarrollado. En contraparte, la falta de comentarios o documentación en el código de los programas dificulta su entendimiento.

Entrevistador.- Me comentaste que al principio te metías a los sistemas; te metiste al código para tratar de ver cómo funcionaba, cómo manejaba la base de datos. ¿Te fue fácil entender el código?

Entrevistado.- Sí, porque tenían comentarios ...

“... como es un lenguaje nuevo con el que estoy trabajando ... por eso empecé a poner comentarios, de repente se me olvidaba, me perdía ...”

“... también me pasa que los sistemas no están documentados [internamente], entonces, si necesito hacer una modificación, ... ahí me tienes revisando todo el código, tratando de seguir más o menos, ¡¡¡ah!!!, de aquí se va para acá, acá es acá ... ¡¡¡ah!!!!, entonces esta tabla la utiliza para esto ...”

IV.3.2.2 Aspectos relacionados con el conocimiento

Uno de los principales puntos de los casos de estudio fue el identificar el tipo de conocimiento que requieren los ingenieros de mantenimiento para realizar sus tareas, así como la manera en que estos lo obtienen. Durante el estudio fue posible observar una correspondencia con los tipos y fuentes de conocimiento que ya hemos reportado en el capítulo anterior, y los que fueron identificados en ambos casos de estudio. También fue posible observar que las experiencias previas juegan un papel muy importante a la hora de

dar mantenimiento a los sistemas de software. A continuación se presentan estos tres aspectos: los tipos de conocimiento requeridos, las maneras en que el personal de mantenimiento obtiene este conocimiento, y la importancia de las experiencias previas.

Tipos de conocimiento requerido por el personal de mantenimiento. Los casos de estudio permitieron identificar que el conocimiento requerido por el personal de mantenimiento corresponde a tres aspectos principales: 1) los procesos seguidos por el usuario, 2) el funcionamiento y arquitectura interna del sistema a ser modificado, y 3) las herramientas y lenguajes de programación empleados para dar mantenimiento al mismo.

- **Los procesos del usuario.** El conocimiento y entendimiento de los procesos que son seguidos por el usuario resultan de gran ayuda. Estos procesos incluyen la manera en que los usuarios trabajan, cómo utilizan el sistema, así como los procesos a los que da soporte éste último.

“...hay que entender cómo lo hace el usuario para también poder entender cómo lo hace el sistema...”

“... es muy importante, primero que nada, saber muy bien como se lleva el proceso [al que da soporte el sistema], y entonces, <<¡¡¡ah!!!, si yo hago este cambio puede afectar en tal parte>>. Tú ya tienes ese conocimiento, y entonces es más fácil que te metas a modificar algo en el código, en el desarrollo del sistema; o agregues algo...”

Entrevistador.- ¿Qué tipo de documentación piensas que te sería útil?

Entrevistado.- Los procesos, los procedimientos ..., por ejemplo en mi caso ... cómo hacer un cheque, suponiendo, tomas información de esto ... todo el flujo de los pasos ...

- **Funcionamiento y arquitectura interna del sistema.** El conocimiento de la estructura y el funcionamiento interno de las aplicaciones o sistemas a ser mantenidos es fundamental para poder determinar qué partes son las que serán cambiadas, así como también para saber con anticipación las repercusiones que podría tener un cambio determinado en el resto del sistema.

“...[a la hora de dar mantenimiento es importante] conocer como funciona el sistema básicamente, porque, modificaciones que hagas en una aplicación tal vez le pueda pegar a todo lo demás. Puede afectar muchas cosas más. Tienes que conocer toda la estructura de la base de datos, y conocer bien las aplicaciones que están a tu cargo para poder hacer una modificación, sin miedo a que vayas a afectar todo el funcionamiento del sistema...”

- **Herramientas y lenguajes utilizados para programar.** El conocimiento de las herramientas y los lenguajes utilizados por los programadores a la hora de realizar modificaciones a un sistema, es fundamental, ya que con este conocimiento es como deciden la manera en que los cambios serán implementados. El conocimiento de los procesos y la estructura interna del sistema les ayuda a identificar que partes del sistema se deben modificar, pero el conocimiento del lenguaje y las herramientas les ayuda a definir cómo realizar estos cambios. En ocasiones, el aprendizaje de estas herramientas y lenguajes cuesta mucho trabajo a los programadores, sobre todo si son nuevos al iniciar a trabajar con ellas, o habían trabajado con ambientes y lenguajes de desarrollo muy distintos. En contraste, el haber trabajado con lenguajes o herramientas parecidas facilita la adaptación y el aprendizaje de las nuevas.

Entrevistador.- Cuando iniciaste a trabajar aquí, ¿qué fue lo que te costó más trabajo?

Entrevistado.- La herramienta...porque no la conocía...

“...me costó mucho trabajo conocer la herramienta de desarrollo...”

“... yo trabajaba con lenguajes no visuales, no gráficos, y aquí es otro tipo de herramientas ... por experiencia como programador sé que existen ciertas funciones base, sé cómo hacer ciertas cosas, mas no cómo es la sintaxis, cómo se compila o qué ...”

“... como es una herramienta visual, yo ya traía experiencia en Delphi, que también es visual, entonces como que ... los programas visuales son muy similares, cambian en algunas cosas, pero si ya sabes uno no te es tan difícil aprender el otro...”

Diversas formas de obtener el conocimiento necesario. El personal de mantenimiento, por lo general se basa en su propia experiencia, así como en el conocimiento previo que puedan tener para la realización de sus tareas. Pero, cuando esto no es suficiente, utiliza diversos métodos y técnicas para obtener el conocimiento que requieren, como lo es la experimentación personal con el funcionamiento de los sistemas, así como dar seguimiento al código de los mismos. Si esto no es suficiente, recurren a diversas fuentes de conocimiento, como la documentación que pudiera existir, compañeros que hayan trabajado en el sistema con anterioridad, o los usuarios del mismo.

“...me baso en el conocimiento general que tengo sobre los sistemas que manejo, y de ahí en ... por ejemplo, si no tengo experiencia en ese cambio que voy a hacer, recorro a la documentación que existe...”

“...me puse a ver todas las opciones que había por sistema, me puse a utilizarlas yo...”

“...me metía a las formas, me metía al código y veía como trabajaba la forma, y ya en base al formato, sacaba las relaciones de las tablas y veía que tablas se afectaban, en donde se registraba la información, y a veces una compañera era la que me explicaba, ella ya había trabajado con algunas aplicaciones...”

“...si yo le paso un sistema a alguien le explico más o menos el funcionamiento, entonces, con eso y la documentación, muchas veces ya es suficiente, ya si con eso no es suficiente, entonces sí con el usuario...”

Experiencias previas. Como es posible ver en las técnicas que el personal de mantenimiento utiliza para obtener el conocimiento necesario a la hora de dar mantenimiento a un sistema específico, las experiencias previas, tanto personales como de otros, resultan ser un factor muy importante. Una de las principales razones para la utilización de las experiencias previas parece ser el tomar ejemplos que puedan ser adaptados para resolver el nuevo problema. La utilización de ejemplos puede agilizar en gran medida el desarrollo de ciertas modificaciones, al proporcionar la posibilidad de reutilizar lo que ya existe. Sin embargo, es necesario conocer la existencia de las experiencias previas para poder utilizarlas.

“...[la bitácora] me sirvió más que nada para movimientos que se hacían, no de todos los días, pero que sí se hacían una o dos veces al año. Si no lo conocía muy bien, lo buscaba en la bitácora, veía la descripción de cómo se hacía y ya lo hacía yo...”

“... si en algún momento me mandan otra solicitud, en algún tiempo me dicen: <<¡¡¡ah!!!, tengo este problema>>, <<¡¡¡ah!!!, yo me acuerdo que en algún momento me pidieron eso>> ... que digo: <<es el mismo problema>>, entonces checas la bitácora: <<¿qué solución le di?>>; entonces es más rápida la solución...”

“... te sirve el mismo código que te ponen de ejemplo, nomás lo copias y le cambias las variables o cosas así, y ya...”

“... uno no tiene mas que implementarlo, o sea, tú nomás tomas la idea (del programa ejemplo) ... tú lo adaptas a tu proyecto...”

IV.3.2.3 Aspectos relacionados con la organización interna de los grupos estudiados

Los aspectos organizacionales identificados son básicamente dos, el primero relacionado con el riesgo de pérdida del conocimiento existente en ambos casos estudiados, y el segundo relacionado con el papel que juega el trabajo en equipo para reducir este riesgo.

Riesgo de pérdida de conocimiento. Si el conocimiento no está expresado de manera formal, y además se reduce la interacción entre los miembros del equipo de trabajo, los mecanismos de transmisión y conversión de conocimiento no pueden funcionar. Por ejemplo, en ocasiones la única fuente de conocimiento son las personas que hayan trabajado en los sistemas con anterioridad, pero si no es posible contar con el conocimiento de un determinado miembro del grupo y, peor aun, sólo él es quien conoce un determinado sistema, será muy difícil para otro retomar el trabajo que haya quedado pendiente.

“...el que conoce el sistema es porque el que estaba antes le pasó el conocimiento que tenía...”

En relación con lo anterior, se detectó en ambos casos de estudio que cada miembro del personal se especializa en ciertos sistemas o módulos. Por ejemplo, en el caso del DI, cada ingeniero se responsabiliza de un área, y de los sistemas que den soporte a esa área; mientras que en Intersel, los ingenieros se especializan en ciertos módulos de los sistemas.

“...todos los sistemas los vas a tener a tu cargo, y conforme haya nuevas aplicaciones o nuevos cambios, entonces ahí se toman decisiones si lo haces tú como tal, o lo hace un grupo de programadores, o lo haces tú con otras dos personas, pero la responsabilidad sigue siendo de la persona que está a cargo del área. ... [las personas se asignaron tomando en cuenta] quien tuviese mas experiencia con la mayor parte de los sistemas que están [en el área]...”

“...entonces el equipo que tenemos ahorita integrado, cada quien tiene, por decirlo así, su especialidad en ciertos módulos, o mayor experiencia en ciertos módulos...”

En el caso del DI se detectó que el riesgo de pérdida de conocimiento puede ser grave, ya que la separación que se ha hecho de las áreas provoca una disminución del intercambio de experiencias entre los miembros del grupo.

*“...antes sí la utilizaba (la bitácora) porque estaba distribuido mi trabajo en diferentes áreas, entonces, todas las llamadas llegaban por parte de [la secretaria], ella anotaba el problema, me la pasaba a mi, lo resolvía y yo lo documentaba, pero ahora tengo yo un trato directo con el usuario ... es un área muy particular que por lo general no tiene relación directa con casi nadie. Entonces yo no estoy documentando en esa bitácora, que debería de hacer ... sí lo hacía, pero ahorita ya no. **Como ya me quedé así como un poco aislada ...**”*

Por su parte, en el caso de Intersel se vio que se ha buscado disminuir este problema, al tomar conciencia de la necesidad de que más de una persona tenga conocimientos sobre los distintos módulos de los sistemas que manejan.

“...lo que hemos aprendido, que yo he aprendido, es no pongas todos los huevos en una sola canasta ... tampoco puedo dejar de ver que la experiencia que tiene cada persona es invaluable, pero no puedo asignar siempre a la misma persona en todos los proyectos los mismos módulos, se trata un poco de repartir. Cuando repartes un módulo, o repartes una sección de todo el

proyecto a una persona que no tiene tanta experiencia, por lo general esas tareas son tareas sencillas, pero sirven para que la otra persona se vaya involucrando más en el producto, de tal manera que ahorita, te puedo decir que el Smart Manager ya está disperso ... ya no es de que nosotros dos nada más podemos meterle mano a toda esa sección, sino que todos en mayor o menor medida están ya involucrados con la estructura y como está compuesto todo esto ... gracias a que cada uno de ellos empezó con tareas sencillas para modificar esa parte del código, y ya después entraron más a fondo, y ahorita ... podemos hablar el mismo canal todos...”

Importancia del trabajo en equipo. El trabajo en equipo permite un mejor intercambio de experiencias y conocimiento entre el personal, así como complementar las carencias de uno, con las habilidades y experiencia de los otros. El trabajo en equipo también se da cuando una persona sabe de alguien que tiene conocimiento que pueda ayudarla a resolver un determinado problema. Sin embargo, al igual que con las experiencias previas, es necesario tener una idea del conocimiento que pueda tener una persona para consultarla. Si no sabemos que esa persona pueda ser de ayuda por el conocimiento que tiene, lo más probable es que no sea consultada.

“...cuando yo llegué a trabajar aquí, me tocó trabajar con [un compañero], entonces, él tenía conocimiento sobre la herramienta, y yo sobre los procedimientos que se seguían en el área [en la] que estábamos trabajando ... pasaba que yo tenía más experiencia en ciertos ... programas y él en otros, entonces así era como de repente nos repartíamos el trabajo...”

“..., hay unos DBLink, que creó [mi compañera], ella es DBA (Administradora de la base de datos), y ella iba a hacer un cambio de quitar unos DBLink, entonces marcaba un error porque yo estaba usando en un programa esos DBLink, entonces vi con ella cómo se podía hacer para que ya no marcara ese error...”

Como es posible observar en los aspectos mostrados en esta sección, el personal de mantenimiento por lo general se basa en su propia experiencia para resolver los problemas que se le presentan. Cuando esta experiencia no es suficiente, recurre a otras fuentes para obtener el conocimiento necesario, como por ejemplo la documentación. Sin embargo, vimos que con frecuencia la documentación no existe o es de baja calidad. Por lo que tienen

que recurrir a otras fuentes, como por ejemplo sus compañeros. Sin embargo, esto no siempre es posible. También se observó que las experiencias previas pueden ser de gran ayuda para ejemplificar la manera de solucionar un determinado problema. Sin embargo, al igual que con otras fuentes de conocimiento, es necesario saber que existen y dónde encontrarlas para poderlas consultar.

Toda la información mostrada nos lleva a considerar dos problemas que ya han sido planteados: la existencia del riesgo de pérdida del conocimiento y el desaprovechamiento del mismo dentro de los grupos de mantenimiento del software. Además, es posible identificar algunos aspectos que deben ser tomados en cuenta si se desea dar soporte a estos problemas por medio de una herramienta de administración del conocimiento. Para ejemplificar estos aspectos a tomar en cuenta, se han definido algunos escenarios, mediante los cuales se busca identificar los requerimientos que debe cubrir una herramienta que dé soporte a los problemas aquí planteados. En la siguiente sección se presentan estos escenarios, para posteriormente definir las características básicas que debe tener un sistema que busque dar soporte en el desaprovechamiento y pérdida del conocimiento en el mantenimiento del software.

IV.4 Escenarios

Al analizar los resultados de los casos de estudio, es claro ver que existen aspectos que pueden ser apoyados por medio de la administración del conocimiento. Uno de los objetivos de los casos de estudio fue el identificar cómo la administración del conocimiento puede ayudar a solucionar algunos de los problemas que pueden presentarse durante el mantenimiento del software, y de esta manera obtener los requerimientos básicos para una herramienta de soporte. Para esto hemos extraído algunos escenarios observados durante ambos estudios. El uso de escenarios es una técnica que permite la identificación de especificaciones de diseño de sistemas de software (Carrol y Rosson, 1992).

A continuación se presentan los escenarios que ayudaron a identificar los requerimientos básicos que, consideramos deben ser cubiertos por un sistema de administración del conocimiento que de soporte al mantenimiento del software. Es necesario aclarar que estos escenarios son situaciones reales identificadas durante los dos casos de estudio, pero, los nombres han sido cambiados para guardar el anonimato de los sujetos de estudio.

Escenario 1: Localización de expertos. Rosario, un ingeniero de mantenimiento (IM), debe implementar ciertos cálculos dentro del sistema de finanzas. Ya que su conocimiento en el área no es suficiente, las modificaciones le han tomado alrededor de una semana más del tiempo programado. Al cabo de esta semana, Susana, la jefa del departamento (JD), en una revisión del avance de los proyectos, detecta el retraso. Cita a Rosario para preguntarle la razón de dicho retraso. Cuando Rosario le comenta a Susana cuál es el problema, Susana se da cuenta de que es algo en lo que ella tiene experiencia, por lo que pudo ayudar a Rosario a solucionar el problema ese mismo día.

Discusión. Al analizar este caso de estudio, podemos ver que si Rosario hubiera tenido la manera de saber desde un principio, que Susana tenía la experiencia para ayudarlo a resolver el problema, posiblemente el retraso nunca se hubiera dado. La siguiente frase resume este problema: *“pero ella como iba a saber si yo no se lo digo”*.

Escenario 2: Administración de documentos. Dentro del departamento de mantenimiento del software, existe un servidor de archivos donde se almacenan las versiones electrónicas de la documentación de los sistemas, así como otros documentos del departamento. Cierta día, José debía realizar cambios en un sistema que no se modificaba hacía tiempo. José sabía que existía un documento donde se describía dicho sistema: su estructura interna y funcionalidad. José sabía que ese documento le podía ser de mucha ayuda para la realización de las modificaciones, así que decidió buscarlo en el servidor de archivos. Al hacer esto, José se dio cuenta de que existían una gran cantidad de documentos. Debido a que José no sabía el nombre del que buscaba, tuvo que revisar prácticamente cada documento existente hasta que dio con el correcto.

Discusión. En este escenario podemos ver que si José hubiera tenido un sistema donde los documentos pudieran contar con cierta información que indicara su contenido y dónde localizarlos, el trabajo y el tiempo de búsqueda se habrían disminuido bastante.

Escenario 3: Reutilización de experiencias. Manuel era el encargado de realizar las modificaciones al sistema de nómina. Él sabía que por los cambios que año con año se dan en la legislación Mexicana, hay ciertas modificaciones que deben realizarse de manera periódica. Por esto, Manuel había estado documentando sus modificaciones en una bitácora que es llevada en el departamento. Cada vez que requería hacer este tipo de cambios, Manuel revisaba la bitácora para recordar, y darse una idea de qué es lo que debía hacer. Por desgracia, Manuel tuvo que dejar el departamento, y Yolanda tomó su lugar. Yolanda es un IM que acaba de ingresar al personal de mantenimiento. Como parte de su trabajo, Yolanda requiere hacer estas modificaciones relacionadas con el cálculo de impuestos en el sistema de nóminas. Debido a que Yolanda no sabe de los reportes que Manuel había estado documentando en la bitácora, no los consulta. Por lo tanto, las modificaciones le toman a Yolanda más del triple del tiempo que le tomaban a Manuel.

Discusión. Como podemos ver en este escenario, en el departamento existía información que pudo haber guiado a Yolanda en la realización de las tareas. Esta documentación podría haberle indicado a Yolanda dónde y cómo realizar los cambios. Sin embargo, debido a que Yolanda no tenía conocimiento de la existencia de la misma, no la consultó. Si Yolanda hubiera tenido la posibilidad de enterarse de alguna manera de la existencia de esta información, el tiempo consumido por las modificaciones seguramente hubiera sido menor.

Escenario 4: Identificación de archivos a modificar. Miguel, un IM sin mucha experiencia requiere modificar el formato de impresión del reporte de calificaciones por estudiante, el cual se encuentra dentro de la ruta: “subsistema de la dirección de estudios de postgrado-> módulo de datos de estudiantes-> impresión de reportes-> reporte de calificaciones-> por estudiante”. Para identificar cuáles son los archivos que requerirán ser

modificados, Miguel busca dentro de los archivos fuente si existe alguno cuyo nombre indique que se trata del “reporte de calificaciones por estudiante”. Al ver los nombres de los archivos, Miguel se da cuenta de que son pocos los que reflejan lo que pudiera contener el archivo, y entre estos no se encuentra el que busca. Por lo tanto, Miguel tiene que entrar desde el archivo que corresponde al menú principal, y, haciendo un recorrido a través del sistema ejecutable, a la par que revisa el código, analiza cada uno de los archivos que son llamados a través de la ruta, hasta identificar el que corresponde con la opción “reporte de calificaciones por estudiante”.

Discusión. En contraste con este escenario, cuando el IM que debe hacer las modificaciones tiene el conocimiento suficiente de la estructura interna del sistema, sabe cuáles son los archivos fuente que corresponden con la opción que se quiere modificar, por lo que no requiere hacer todo el seguimiento que hizo Miguel (el IM inexperto). De aquí podemos observar que si Miguel tuviera la posibilidad de saber quienes han modificado con anterioridad un determinado módulo, y mejor aún, cuáles son los archivos que, por lo general son modificados cuando deben hacerse cambios a éste, podría reducir el tiempo que dedica a la realización de las modificaciones.

Escenario 5: Identificación de archivos afectados por los cambios. Sergio, haciendo modificaciones a una de las tablas del sistema que estaba a su cargo, se dio cuenta que siempre que hacía cambios en dicha tabla se afectaban los mismos archivos. Por esta razón, Sergio escribió, en el reporte de error que dio origen a la modificación, cuáles eran los módulos afectados por los cambios en dicha tabla. Al paso del tiempo, Miguel tuvo que realizar modificaciones en un módulo donde se utilizaba la misma tabla. Estos cambios requirieron modificar parte de la estructura de la misma. Miguel hizo los cambios, los probó y verificó que funcionaban. Con esto Miguel dio por hecha su tarea. Cuando el sistema se puso en ejecución, resultó que habían otros dos módulos que comenzaron a fallar. Al revisar dichos módulos se dieron cuenta de que habían sido afectados por los cambios hechos por Miguel.

Discusión. Con anterioridad Sergio ya había previsto cuáles eran los módulos afectados por dicha tabla, e incluso lo había escrito en un reporte de error que estaba a disposición del resto del equipo. Sin embargo, como Miguel no sabía de esto, no lo revisó, y por lo tanto no se percató de lo que otro miembro del equipo ya sabía. Al analizar este caso, nos damos cuenta de que si hubiera existido la posibilidad de que Miguel tuviera acceso a dicho reporte de error, el problema se habría evitado.

Es posible ver, en los escenarios anteriores, que existe la necesidad de proporcionar a los encargados del mantenimiento del software, herramientas que les permitan identificar fuentes de conocimiento que les ayuden a realizar sus tareas. Además, se ve la necesidad de que estas herramientas los apoyen aun cuando los IM no sepan siquiera que estas fuentes existen. En la siguiente sección se plantean las principales características con las que, consideramos, deberá contar una herramienta de este tipo.

IV.5 Características básicas para una herramienta de administración del conocimiento en el mantenimiento del software

De los escenarios planteados es posible identificar la manera en que una herramienta de administración del conocimiento, puede ayudar a solucionar algunos problemas que se presentan en los grupos encargados del mantenimiento del software. En esta sección se presentan las características básicas obtenidas mediante el análisis de estos escenarios y del resto de la información capturada durante los casos de estudio. Estas características se han complementado con información reportada por la literatura sobre administración del conocimiento.

De la literatura es posible ver que las herramientas de administración del conocimiento deben considerar los siguientes aspectos básicos:

- Apoyo en la captura y recuperación de conocimiento y experiencias.

- Apoyo en el manejo de las fuentes de información.
- Apoyo en el flujo de trabajo.
- Apoyo en la comunicación y coordinación.

Sin embargo, estos aspectos son demasiado amplios como para poder cubrirlos en su totalidad. Por lo tanto, hemos decidido centrar nuestro interés en una serie de características más específicas, de manera que el desarrollo de un sistema que las cubra sea más factible. Considerando los datos recabados en ambos casos de estudio, hemos identificado las siguientes características específicas:

Mecanismos para el reporte de problemas y fallas por parte de los clientes o usuarios. En ambos casos de estudio se observó que existe un mecanismo para hacer llegar a los IM los problemas que los usuarios o clientes tienen con el sistema. Por lo tanto, consideramos importante que una herramienta de apoyo al mantenimiento de software de soporte a este proceso.

Mecanismos para la solicitud de modificaciones. Es necesario que la herramienta permita hacer llegar las solicitudes de modificaciones a las personas que deben llevarlas a cabo.

Apoyo en la identificación y acceso a fuentes de conocimiento relacionado con las tareas a realizar. Con frecuencia en una organización existen documentos o personas con el conocimiento suficiente para servir de apoyo en la solución de un problema. Sin embargo, si no tenemos idea de la existencia de estos documentos o personas, o del conocimiento que puedan tener, muy probablemente no los consultaremos. Debido a esto, consideramos importante poder identificar estas fuentes de conocimiento, aun cuando las mismas personas no sepan que existen.

Apoyo en la captura y recuperación de experiencias y casos similares. Es común que los IM recurran a los ejemplos o casos similares para darse una idea de cómo solucionar cierto problema. Con frecuencia estos casos similares existen dentro del mismo sistema. Por

ejemplo, la existencia de un módulo que llama a la misma base de datos a la que debemos acceder. Otro ejemplo puede ser la solución de un error en un determinado programa. Si el error ya se ha dado con anterioridad en el mismo o en algún otro programa, el poder consultar la solución que se le dio en esa ocasión podría ser de mucha ayuda. Es por esto que consideramos importante permitir la captura y recuperación de experiencias y casos similares para que puedan ser aprovechados por los IM. Dentro de este marco, las experiencias previas y casos similares deben poder apoyar en la identificación de los archivos que requerirán ser modificados, así como los que pudieran resultar afectados por los cambios. Esto se detalla a continuación:

- *Apoyo en la identificación de archivos fuentes a modificar.* Se detectó en ambos casos de estudio, que una de las partes principales del proceso de modificaciones a un sistema, es el poder identificar qué archivos son los que se requiere modificar. Este proceso puede consumir un tiempo considerable, sobre todo si la persona no cuenta con la suficiente experiencia y conocimiento sobre la estructura interna del sistema. Ya que en muchos casos no existe documentación que pueda apoyar en estas tareas, con frecuencia es necesario analizar el código fuente de una gran parte del sistema, para poder así identificar en que lugar del mismo se deben hacer las modificaciones.
- *Apoyo en la identificación de módulos o archivos que pudieran resultar afectados por los cambios.* El no prever qué otras partes del sistema pueden resultar afectadas al realizar una modificación en algún módulo, puede llegar a ocasionar retrasos considerables en los tiempos programados. Este problema se hace más probable entre menor experiencia tenga la persona encargada de implementar las modificaciones. Por lo mismo, consideramos que un sistema de apoyo a la administración del conocimiento en el mantenimiento de software, debe poder apoyar a los IM en este proceso.

El sistema debe ser pro-activo y autónomo. Para que el sistema sea realmente de utilidad no debe representar sólo una carga de trabajo extra para los IM, por lo tanto, es necesario que pueda trabajar sin una intervención constante del usuario. Otro factor que implica esta necesidad, es que con frecuencia las personas no saben de la existencia de fuentes de conocimiento que les pudieran servir para las tareas que realizan, por lo mismo no se

preocupan en buscarlas. Si queremos que todo este conocimiento sea de utilidad, el sistema debe ser capaz de buscarlo aun cuando el usuario no lo solicite de manera explícita.

IV.5.1 Hacia una arquitectura de agentes para la administración del conocimiento en el mantenimiento del software

Como es posible observar, es necesario que el sistema sea capaz de apoyar en la generación e identificación de conocimiento y fuentes del mismo, sin la necesidad de una constante intervención del usuario, ya que, si a los IM se les solicita que para cada tarea que realicen, capturen información para incrementar la base de conocimiento, estos difícilmente encontrarán una verdadera utilidad en el sistema. Además, vemos necesario que, aun cuando los IM no soliciten directamente la búsqueda de fuentes de conocimiento, por ejemplo, por no saber que existen, el sistema sea capaz de anticiparse e informarles de la existencia de fuentes de conocimiento que pudieran ser relevantes para la tarea a realizar. Por lo anterior, el sistema debe contar con cierta autonomía para actuar bajo determinadas circunstancias, por lo que hemos considerado los agentes de software como una tecnología viable para implementar un sistema de administración del conocimiento que de soporte al tipo de problemas anteriormente planteados, ya que estos cuentan con características que los hacen una buena opción en el desarrollo de sistemas autónomos y pro-activos, como se vio en el capítulo anterior.

IV.6 Resumen

En este capítulo se presentaron los resultados de dos casos de estudio, realizados con el fin de identificar la manera en que la administración del conocimiento puede aplicarse para resolver algunos de los problemas que se presentan en el mantenimiento del software. Para esto, también se definieron distintos escenarios que permitieron ver cómo la administración del conocimiento puede ser de utilidad. Estos escenarios permitieron hacer una identificación de las características básicas que debe cubrir una herramienta que de soporte a la administración del conocimiento en el mantenimiento del software. Al analizar estas características, vimos que los agentes de software prometen ser una buena opción para el

desarrollo de la herramienta, ya que entre las principales características que fueron definidas está la autonomía y pro-actividad, mismas que pueden ser obtenidas por medio de la utilización de agentes.

Tomando en cuenta los requerimientos identificados en este capítulo, se diseñó una arquitectura basada en agentes, que permita el desarrollo de sistemas que den soporte a la administración del conocimiento en el proceso de mantenimiento del software. Para validar la factibilidad de la implementación de sistemas basados en dicha arquitectura, se diseñó e implementó un prototipo. El diseño del prototipo se hizo mediante la definición de un escenario de uso, buscando que el prototipo desarrollado fuera capaz de dar soporte al escenario planteado. En el siguiente capítulo se presenta la descripción y el diseño de la arquitectura de agentes mencionada, así como los detalles del diseño, implementación y parte de la funcionalidad básica del prototipo desarrollado.

Capítulo V. Arquitectura de Agentes para la Administración del Conocimiento en el Mantenimiento del Software

*"¿Por qué esta magnífica tecnología científica,
que ahorra trabajo y nos hace la vida mas fácil,*

nos aporta tan poca felicidad?

*La respuesta es esta, simplemente:
porque aún no hemos aprendido a usarla con tino".*

Albert Einstein

Como ya se ha visto en capítulos anteriores, existen distintos trabajos que han buscado aplicar la administración del conocimiento dentro de los procesos de ingeniería de software. Sin embargo, estos trabajos se orientan a dar soporte a las etapas de la ingeniería del software relacionadas con el desarrollo de nuevos sistemas. Es por esto, y por la necesidad de atender los problemas de pérdida y desaprovechamiento del conocimiento existentes en el mantenimiento del software, que se buscó identificar las características básicas que debe cubrir una herramienta de administración del conocimiento que dé soporte a estos problemas. Lo anterior nos llevó a la realización de dos casos de estudio presentados en el capítulo anterior. De estos casos de estudio, así como de la revisión bibliográfica realizada en los capítulos previos, se obtuvieron algunas características generales, dentro de las cuales, los agentes de software prometen ser una buena opción para su implementación. Con esto en mente, nos enfocamos a diseñar una arquitectura basada en agentes que sirva como base para el desarrollo de sistemas de administración del conocimiento que den soporte al mantenimiento del software.

En este capítulo se presenta la arquitectura mencionada, así como un sistema prototipo que fue desarrollado tomando como base dicha arquitectura. Primeramente se listan las especificaciones que fueron consideradas para el diseño de la arquitectura, para después

detallar los elementos que la componen, y la funcionalidad que deben cumplir estos elementos. Posteriormente se presentan los detalles del desarrollo del prototipo, iniciando con un escenario de uso que fue definido con el fin de obtener requerimientos, así como las especificaciones de diseño, seguidamente se presenta el análisis y diseño del mismo, para finalmente presentar algunos detalles de su implementación y funcionalidad básica.

V.1 Arquitectura

Como ya se ha visto en capítulos previos, la revisión bibliográfica, así como la realización de dos casos de estudio, dieron paso a una serie de características que debe cubrir una herramienta de soporte a la administración del conocimiento en el mantenimiento de software, las cuales contemplan aspectos generales a cualquier tipo de organización, así como algunos propios de los grupos de mantenimiento de software que fueron estudiados. Con base en estas características se diseñó la arquitectura de agentes presentada en este capítulo. Para el diseño de la arquitectura fue necesario tomar en cuenta dos aspectos básicos, los actores participantes en el proceso de mantenimiento, y los elementos principales de éste proceso.

Actores. Como se menciona en el capítulo II, los actores participantes en el proceso de mantenimiento del software pueden dividirse en tres: 1) cliente, 2) personal de mantenimiento, y 3) usuario. Sin embargo, durante los casos de estudio realizados, pudimos observar que los roles asignados a los clientes y al usuario suelen variar considerablemente de una organización a otra. Por ejemplo, en el caso de Intersel, con frecuencia quienes solicitan las modificaciones resultan ser las demás áreas de la empresa, por otro lado, en el DI, los usuarios y los clientes suelen ser las mismas personas. Debido a esto, hemos considerado agrupar al cliente y al usuario en un solo actor que será el cliente, quien es el encargado de realizar los reportes de errores, o solicitudes de mantenimiento. A la par, este actor se interesa en conocer el estado de avance que tienen las solicitudes o reportes que ha realizado. Además, en muchos casos este actor cuenta con conocimientos que pueden ser

útiles para el personal de mantenimiento al momento de hacer modificaciones al sistema. Es por esto que la arquitectura debe considerar dos tipos de actores principales:

- *Cliente, y*
- *Personal de mantenimiento.*

Elementos del proceso de mantenimiento. Los casos de estudio también permitieron identificar dos elementos básicos que deben ser considerados:

- *Producto.* Es el sistema al que se da mantenimiento. Se compone de elementos tales como el código fuente, los programas ejecutables, documentación que lo describe, bases de datos que utiliza, etc.
- *Proyectos de mantenimiento.* A través de éstos es que se definen las tareas que deberán realizarse, quién las debe realizar, el tiempo que consumen, y finalmente las acciones que se requirieron para darles solución.

A continuación se describen las especificaciones que fueron consideradas para el diseño de la arquitectura propuesta, posteriormente se muestra el diseño, y una descripción de los distintos elementos que la conforman.

V.1.1 Especificaciones de diseño de la arquitectura

La arquitectura debe ser capaz de proporcionar la base para el desarrollo de sistemas que cumplan con las siguientes especificaciones:

1. Un medio para hacer llegar los reportes de problemas y fallas del sistema al personal de mantenimiento.
2. Un medio para hacer llegar las solicitudes de modificaciones al personal de mantenimiento.
3. Facilitar la identificación y acceso a fuentes de conocimiento relacionado con las tareas a realizar.
4. Permitir la captura y recuperación de experiencias previas y casos similares, apoyando en:

- la identificación de archivos fuente a modificar, y
 - de módulos o archivos que puedan resultar afectados por los cambios.
5. Permitir que los sistemas basados en ella puedan trabajar sin una intervención constante del usuario, es decir, que sean pro-activos y autónomos, lo cual lleva a considerar los agentes de software como la base para su diseño.

Además de lo anterior, es necesario definir los elementos que deben ser considerados por la arquitectura. Dentro de estos elementos fueron identificados los principales actores participantes en el proceso de mantenimiento y los elementos más relevantes del mismo:

6. Actores:
- *Cliente.*
 - *Personal de mantenimiento.*
7. Elementos del proceso de mantenimiento:
- *Producto.*
 - *Proyectos de mantenimiento.*
8. Debe contemplar la existencia de tres niveles de almacenamiento y acceso a la información y conocimiento existentes:
- *Local.* Permitir que los miembros del personal de mantenimiento compartan recursos o fuentes de conocimiento, almacenadas localmente, al resto del grupo.
 - *Global.* Permitir el acceso a un repositorio de conocimiento global.
 - *A nivel producto.* Permitir que los documentos que correspondan con un determinado producto puedan ser agrupados y administrados en conjunto.
9. Deben considerarse mecanismos de control de acceso, donde cada elemento participante dentro del sistema debe estar identificado con un nombre único.
10. Debe existir un directorio donde los distintos agentes de software puedan registrarse.

A continuación se presenta la arquitectura que se diseñó tomando en cuenta las especificaciones aquí listadas.

V.1.2 Diseño de la arquitectura

Tomando en consideración las especificaciones mencionadas anteriormente, se diseñó la arquitectura mostrada en la figura 15. Esta arquitectura se divide en cuatro contenedores de agentes de software principales. Estos contenedores son:

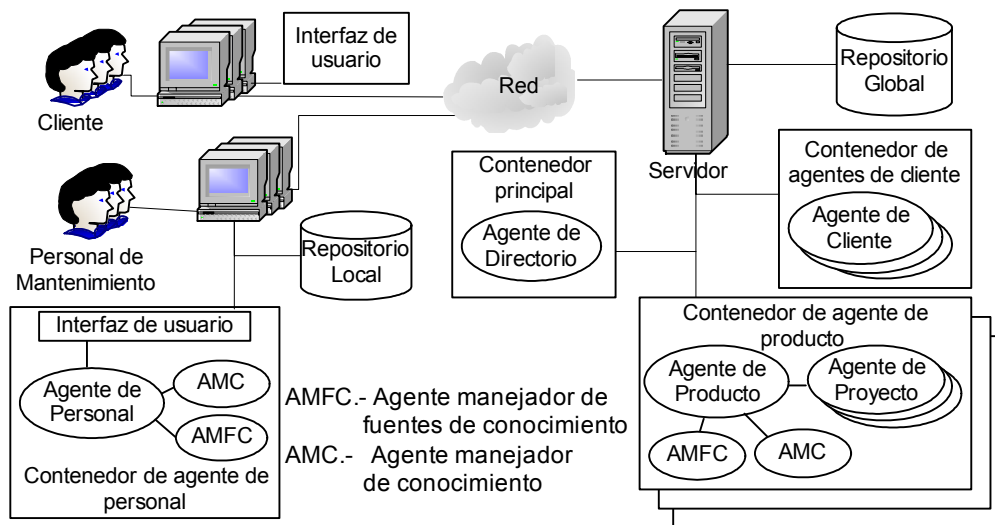


Figura 15. Arquitectura de agentes para la administración del conocimiento en el mantenimiento del software.

Contenedor principal. En el contenedor principal es donde se aloja el agente de directorio. Este agente es el encargado de llevar un registro de los distintos agentes que se encuentran trabajando en el sistema.

Contenedor de agentes de cliente. Este contenedor aloja a los agentes de los distintos clientes que se encuentren reportando errores, o solicitando modificaciones de los sistemas a los que da mantenimiento el personal. Los clientes contarán con una interfaz gráfica mediante la cual podrán conectarse al sistema y comunicarse con el agente correspondiente.

Contenedor de agente de personal. Este contenedor aloja a los agentes que dan servicio a un determinado miembro del personal. Cada miembro del personal cuenta con un contenedor de este tipo en su máquina local. Este tipo de contenedor aloja a tres agentes:

- *Agente de personal*, a través del cual el miembro del personal interactuará con el sistema, esto mediante una interfaz de usuario que será monitoreada por el agente.
- *Agente Manejador de Fuentes de Conocimiento (AMFC)*. Se encarga de llevar el control de los distintos documentos compartidos por el usuario hacia los demás miembros del personal. También es el encargado de recuperar las fuentes de conocimiento que se encuentren en otras localidades. Por ejemplo, podrá comunicarse con otros AMFCs para solicitar documentos que estén bajo el control de éstos últimos.
- *Agente Manejador de Conocimiento (AMC)*. Este agente se encarga de la búsqueda de conocimiento que pueda ser de utilidad para las tareas que realice el miembro del personal. También apoya en la creación de conocimiento de las tareas realizadas, así como de compartirlo con el resto de los miembros del personal.

Contenedor de agente de producto. Existe un contenedor de este tipo por cada sistema o producto al que se da mantenimiento. En este contenedor se alojan cuatro tipos de agentes:

- *Agente de producto*. Lleva el control de la información del producto o sistema al que se da mantenimiento. Entre otras cosas, tiene conocimiento sobre los miembros del personal que trabajan en él, así como de los reportes de error y solicitudes de cambios que se realizan sobre el producto.
- *Agente Manejador de Fuentes de Conocimiento (AMFC)*. Ya que cada producto está constituido por una serie de documentos, desde el código fuente hasta documentos de requerimientos, análisis, diseño, etc., se requiere un agente que se encargue del control de estos documentos, y que sirva de intermediario para la búsqueda y recuperación de los mismos, al igual que en el contenedor de agente de personal.
- *Agente Manejador de Conocimiento (AMC)*. Se encarga de la generación de conocimiento con base en las tareas que se realicen sobre el producto. Por ejemplo, puede identificar patrones durante las modificaciones hechas sobre los distintos módulos, y de esta manera, tratar de identificar cuáles módulos o programas pueden resultar afectados cuando algún otro es modificado.
- *Agente de Proyecto*. Para cada reporte de error o solicitud de cambios en el producto existe un agente de proyecto. Este agente tiene conocimiento sobre el reporte o solicitud

que lo originó, así como de las personas asignadas a cada una de las tareas que se requieran realizar. También conoce sobre el estado de avance de las tareas y del proyecto en general.

Con relación a los niveles de acceso a las fuentes de conocimiento, se ha definido el siguiente esquema:

- *Global*. Todos los miembros del personal tendrán acceso a los documentos y conocimiento dentro de este nivel. Podrán agregar y consultar todo lo que se encuentre dentro de este.
- *Local*. El acceso a la información y conocimiento local se hará a través del agente de personal, el cual turnará la solicitud al AMC o al AMFC según el servicio que se solicite.
- *A nivel producto*. El agente de producto llevará control de este tipo de recursos, de manera que los recursos que caigan dentro de este nivel sean accedidos por medio del agente de producto correspondiente, el cual turnará la solicitud al AMC o AMFC dentro de su contenedor, según el servicio solicitado.

Con el fin de mostrar el funcionamiento de la arquitectura propuesta, así como la factibilidad de que sirva como base para el desarrollo de sistemas de soporte para la administración de conocimiento en el mantenimiento de software, se desarrolló un prototipo, el cual se describe en la siguiente sección.

V.2 Prototipo

El diseño del prototipo está orientado a proporcionar una visión general de la manera en que funciona un sistema desarrollado con base en la arquitectura propuesta. El objetivo es demostrar la factibilidad del desarrollo de sistemas de administración del conocimiento con base en esta arquitectura, así como también, analizar el apoyo que este sistema puede proveer a los encargados del mantenimiento del software.

El desarrollo del prototipo se hizo tomando como base un escenario de trabajo para un ingeniero de mantenimiento (IM). Este escenario se obtuvo de los datos recabados en los casos de estudio presentados en el capítulo anterior, añadiendo los aspectos de la arquitectura que se pretende apoyen al IM en la realización de las actividades descritas por el escenario.

En seguida se describe el escenario mencionado, el cual se presenta numerando secuencialmente cada uno de sus pasos, con el fin de que pueda ser analizado con facilidad. Posteriormente se presenta el análisis y diseño del prototipo, donde se definen los requerimientos que fueron considerados para su desarrollo, seguidos por los principales casos de uso identificados en el escenario, los diagramas de secuencia de estos, y el diagrama de clases. Finalmente se presentan los detalles de la implementación y la funcionalidad básica del prototipo desarrollado.

V.2.1 Escenario

1. Juan, un IM, decide iniciar el sistema. Al hacerlo, se le solicita su nombre de usuario y clave de acceso, Juan los proporciona y solicita que sean validados.
2. Una vez que los datos de Juan han sido validados, el sistema inicializa los agentes de personal, AMC y AMFC que le darán apoyo a Juan en sus tareas. Estos agentes son nombrados tomando como base el nombre de usuario proporcionado por Juan. Específicamente juan para el agente de personal, juan_km para el agente AMC, y juan_ksm para el agente AMFC.
3. Al iniciar, el agente de Juan solicita al agente de directorio que lo registre, y muestra la interfaz gráfica de usuario.
4. El agente de directorio registra al agente juan, y les informa de esto a todos los agentes de producto registrados en los cuales el IM Juan se encuentre trabajando.
5. Ya que Juan está asignado al sistema de recursos financieros (SIREFI), el agente de producto SIREFI, una vez que el agente de directorio le informa que el IM Juan ha entrado al sistema, revisa entre sus proyectos si existe alguno que deba ser atendido por

- Juan. Al detectar que efectivamente el proyecto SIREFI_p_12 debe ser atendido por Juan, le envía un mensaje al agente de proyecto SIREFI_p_12 informándole que Juan se ha conectado al sistema.
6. El agente SIREFI_p_12 envía un mensaje al agente Juan para que le informe al IM Juan que tiene un proyecto pendiente de ser atendido.
 7. El agente Juan le muestra al IM Juan este proyecto en la interfaz de usuario.
 8. Los proyectos que se le muestran a Juan en su interfaz de usuario, están ordenados por el producto al que pertenecen.
 9. Al revisar los proyectos, Juan ve el proyecto SIREFI_p_12, y elige atenderlo.
 10. Juan ve que este proyecto se originó por un reporte de error hecho por un usuario del sistema de finanzas.
 11. Cuando Juan elige este proyecto, el agente Juan se lo indica al agente Juan_km.
 12. El agente Juan_km identifica los datos del reporte de error que dio origen al proyecto, específicamente el sistema o producto al que pertenece el módulo donde se generó el error, el mismo módulo, y el tipo de error.
 13. Con base en los datos anteriores, el agente Juan_km busca fuentes de conocimiento que le pudieran ayudar a Juan a resolver el error reportado. Principalmente trata de identificar qué tipo de conocimiento se requiere para poder trabajar con el sistema y módulo mencionados, y busca las fuentes que pudieran tener este conocimiento.
 14. Una vez hecho lo anterior, el agente Juan_km envía un mensaje de notificación al agente Juan indicándole las fuentes de conocimiento detectadas.
 15. El agente Juan le indica a Juan, a través de la interfaz gráfica de usuario, del total de fuentes de conocimiento que fueron encontradas.
 16. Al recibir la notificación, Juan decide ver cuáles son estas fuentes de conocimiento.
 17. El agente Juan envía un mensaje al agente Juan_km para que muestre a Juan las fuentes de conocimiento que encontró.
 18. El agente Juan_km muestra estas fuentes ordenadas por su tipo.
 19. Al ver los resultados, Juan decide revisar si existe algún reporte de error que se haya dado en el mismo módulo donde se presentó el error que pretende solucionar.

20. Juan encuentra que uno de los reportes de error recuperados por el agente `juan_km` cumple con esta condición, y decide ver cuál fue la solución que se dio a dicho error.
21. Al hacer esto, el agente `juan_km` le solicita al agente `juan_ksm` que le muestre a Juan los datos del reporte de error elegido.
22. El agente `juan_ksm` recupera el reporte de error y muestra los datos a Juan.
23. Juan identifica cuáles fueron los archivos que se modificaron para dar solución al error previo, y decide revisar esos archivos para darse cuenta de que el nuevo error se está generando en uno de ellos.
24. Posteriormente, Juan decide dar solución a dicho error, pero se encuentra con el problema de que el error se debe a un cálculo incorrecto.
25. Debido a que los conocimientos sobre finanzas de Juan no son muy amplios, decide revisar si dentro de las fuentes de conocimiento identificadas por el agente `juan_km`, se encuentra algún compañero que conozca sobre el módulo a modificar, o sobre finanzas.
26. Al hacer lo anterior, Juan se da cuenta de que Miguel, uno de sus compañeros, tiene ese tipo de conocimiento, por lo que decide consultarlo.
27. Juan revisa el correo electrónico, teléfono y localización física de Miguel, datos que le son proporcionados por el sistema, y decide consultarlo telefónicamente.
28. Juan le comenta a Miguel cuál es el problema
29. Debido a que Miguel ya había tenido experiencia con ese tipo de cálculos, le indica a Juan qué es lo que puede hacer para solucionar el error.
30. Finalmente Juan resuelve el error y guarda la solución indicando el archivo que se modificó, cuál fue la causa del error, cuál la solución, y como observaciones, Juan menciona que los conocimientos sobre finanzas de Miguel le ayudaron a resolver el problema. De esta manera Juan deja su aportación por si en un futuro a alguien más se le presenta un problema similar al de él.

V.2.2 Análisis y diseño del prototipo

Como ya mencionamos, el escenario definido anteriormente sirvió como base para el análisis y diseño del prototipo, lo cual se hizo utilizando el Lenguaje Unificado de

Modelado (UML, por sus siglas en inglés) (Booch *et al.*, 1999). En particular, se utilizaron diagramas de casos de uso, de secuencia y de clases. Los diagramas de casos de uso permiten definir el comportamiento esperado del sistema, sin la necesidad de entrar en detalles de cómo se logrará este comportamiento. Los diagramas de secuencia permiten identificar el orden temporal de las interacciones entre los objetos del sistema. Finalmente, los diagramas de clases permiten vislumbrar la estructura interna del sistema, mostrando las clases con las cuales será construido, así como las relaciones existentes entre las mismas.

El primer paso de la etapa de análisis y diseño fue el identificar las características necesarias para la implementación del prototipo. Estas características se listan a continuación. Posteriormente se definen los principales casos de uso, así como sus diagramas de secuencia identificados en el escenario. Finalmente se describe el diagrama de las principales clases que dieron paso a la implementación del prototipo.

V.2.2.1 Características de implementación del prototipo

Las características para la implementación del prototipo que fueron identificadas en el escenario son las siguientes:

1. Se debe proporcionar un mecanismo de control de acceso, con base en una clave y nombre de usuario único, con el cual identificar los agentes de cada IM.
2. Se deben implementar los elementos que permitan el funcionamiento básico de los agentes del contenedor de agente de personal. Esto implica implementar:
 - a. El funcionamiento básico de los agentes de directorio, producto y proyecto, así como los contenedores correspondientes.
 - b. Una base de datos para almacenar la información utilizada por los agentes.
 - c. Una base de conocimientos donde el AMC pueda buscar fuentes de conocimiento relacionadas con el reporte de error que será atendido.
3. Se debe proporcionar un mecanismo para hacer llegar los reportes de error al IM.
4. Se debe permitir ver los datos de los proyectos y reportes de error asignados al IM.
5. Se debe dar apoyo en la identificación de fuentes de conocimiento que pudieran estar relacionadas con el error reportado. Esto implica implementar:

- a. Una ontología para definir los tipos y fuentes de conocimiento.
 - b. Un mecanismo para definir el conocimiento que tiene cada fuente.
 - c. Un mecanismo para definir el conocimiento que se requiere para trabajar con cada producto o elemento del mismo (módulo, base de datos, etc.).
6. Se debe permitir recuperar las fuentes de conocimiento encontradas por el AMC.
 7. Se debe permitir la captura y almacenamiento de la solución dada al error reportado.

Otras características que deben ser implementadas con el fin de que el sistema pueda trabajar para dar soporte al escenario son las siguientes:

8. Mecanismo para dar de alta IM's en el sistema.
9. Mecanismo para dar de alta productos.
10. Mecanismo para crear reportes de error.
11. Mecanismo para crear proyectos. Los proyectos no manejarán tareas, sólo se limitarán a definir el reporte de error que dio origen al mismo, así como la persona encargada de atenderlo.

Los detalles de los elementos del prototipo que implementan estos últimos requerimientos pueden ser vistos en el apéndice C.

A continuación se describen los principales casos de uso identificados en el escenario, y sus correspondientes diagramas de secuencia.

V.2.2.2 Casos de uso identificados en el escenario, y sus diagramas de secuencia

Un caso de uso especifica un servicio que proporciona el sistema a sus usuarios, es decir, una forma específica de utilizar el sistema que es visible desde el exterior, así como las respuestas ofrecidas por el sistema (Booch et al, 1999). Al analizar el escenario planteado anteriormente es posible identificar cinco casos de uso principales, los cuales son: 1) la inicialización del sistema, que incluye la validación del usuario; 2) la elección del proyecto que será atendido; 3) la recuperación de las fuentes de conocimiento encontradas por el sistema; 4) la visualización de los datos de las fuentes de conocimiento encontradas; y 5) el

almacenamiento de la solución dada al error reportado. En esta sección sólo se tratan los dos primeros, el resto son descritos en el Apéndice B.

La figura 16 muestra el diagrama del **caso de uso Inicializar sistema**, el cual es ejecutado por el IM. A continuación se dan los detalles de este caso de uso.

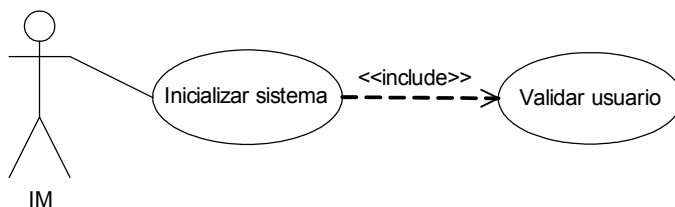


Figura 16. Diagrama del caso de uso Inicializar sistema.

Caso de uso: Inicializar sistema.

Actor: Ingeniero de mantenimiento.

Objetivo: Realizar la inicialización del sistema para que el IM pueda revisar los proyectos a los que está asignado.

Condición inicial: El IM debe tener un nombre y clave de usuario para poder ingresar al sistema. El nombre de usuario debe ser único para cada IM, ya que éste será su identificador dentro del sistema.

Descripción: En este caso de uso se realiza la validación del IM. Primeramente se realiza la inicialización de los agentes que le darán soporte al IM. A la par, se realiza el registro del agente de personal del IM con el agente de directorio, y se le muestran al IM los proyectos a los que está asignado. Este caso de uso corresponde con los puntos 1 al 8 del escenario.

La figura 17 muestra el diagrama del **caso de uso Atender proyecto**, el cual también es ejecutado por el IM. Los detalles se presentan a continuación.

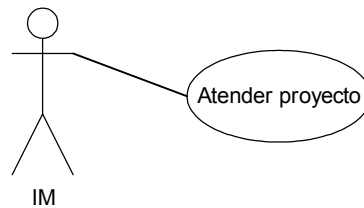


Figura 17. Diagrama del caso de uso Atender proyecto.

Caso de uso: Atender proyecto.

Actor: Ingeniero de mantenimiento.

Objetivo: Permitir al IM atender un proyecto.

Condición inicial: El sistema debe proporcionar al IM la lista de los proyectos en los que se encuentra trabajando, para que éste pueda elegir el proyecto que desea atender.

Descripción: En este caso de uso, el IM elige el proyecto que desea atender de una lista de proyectos a los que está asignado. A su vez, el sistema le muestra los datos del proyecto y del reporte de error que lo generó. A la par, el sistema busca fuentes de conocimiento que pudieran ayudar al IM en la solución del error reportado, así como de notificar el total de las fuentes encontradas. Este caso de uso corresponde con los puntos 9 al 15 del escenario.

Para obtener un mejor entendimiento del comportamiento descrito por los casos de uso, se desarrollaron diagramas de secuencia de los mismos. Estos diagramas son utilizados para modelar aspectos dinámicos del sistema, muestran la interacción temporal entre los objetos que participan para lograr el comportamiento esperado del mismo (Booch et. al, 1999). Es necesario aclarar antes de la presentación de los diagramas, que se han empleado tres tipos de interacciones entre los objetos mostrados en los mismos, las cuales pueden observarse en la figura 18. El primer tipo de interacción indica la llamada a una función del objeto al que apunta la flecha, solicitada por el objeto donde ésta inicia. El segundo tipo de interacción indica el envío de un mensaje al objeto al que apunta la flecha. Este tipo de interacción es la que se da entre los agentes del sistema, ya que, debido a su autonomía, estos no solicitan llamadas directas a funciones de otro agente, sino que envían mensajes indicando o solicitando una determinada acción, y es responsabilidad del agente que recibe

el mensaje si actúa, o no, en consecuencia. Por último, el tercer tipo de interacción se utiliza para indicar la respuesta generada por una llamada a función.

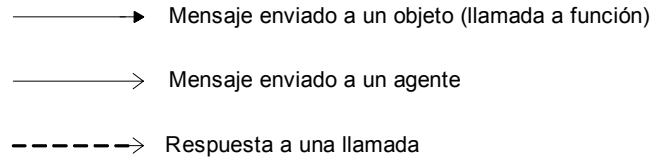


Figura 18. Tipos de interacciones utilizadas en los diagramas de secuencia.

El **diagrama de secuencia del caso de uso Inicializar sistema** se muestra en dos partes para facilitar su ilustración, la figura 19 muestra la primer parte, donde se define la secuencia de las interacciones que se dan entre los objetos participantes durante la validación del usuario del sistema, y el registro del agente de personal en el directorio. Es posible observar que una vez que el IM Juan ha sido validado, el contenedor de agentes del personal inicia los agentes correspondientes. Posteriormente, el agente Juan solicita al agente de directorio ser registrado, y crea la interfaz gráfica de usuario para que ésta sea desplegada al IM Juan.

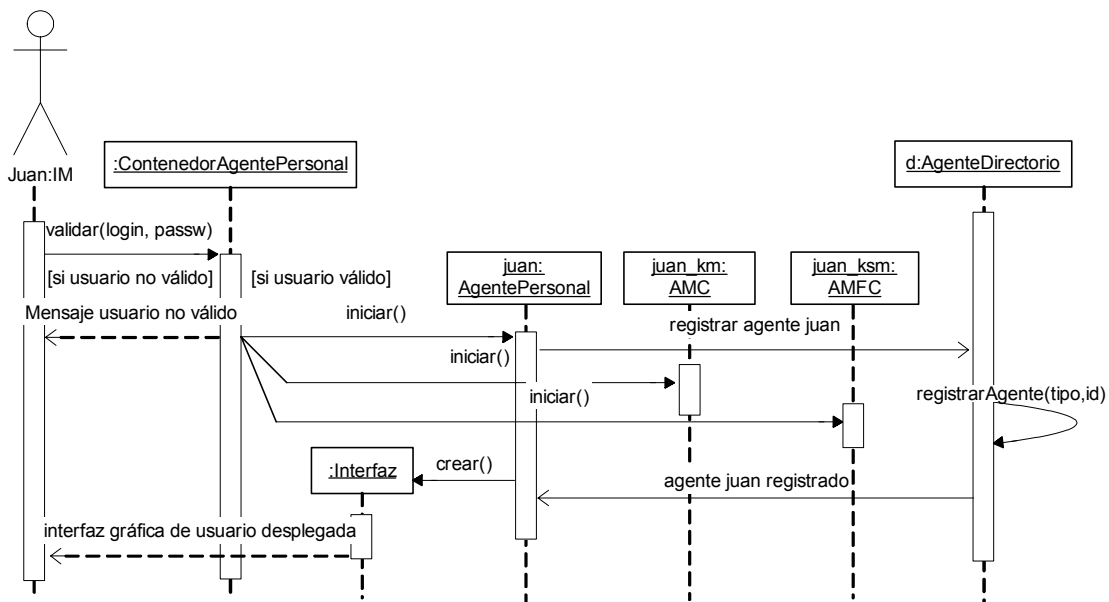


Figura 19. Primer parte del diagrama de secuencia del caso de uso Inicializar sistema.

La figura 20 muestra la **segunda parte del diagrama del caso de uso Inicializar sistema**. Es posible observar que, una vez que se ha llevado a cabo el intercambio de mensajes que se mostró en la figura anterior, el agente de directorio busca los productos donde el IM Juan se encuentra trabajando, en este caso SIREFI, y envía un mensaje al agente de producto SIREFI informándole que el IM Juan ha entrado al sistema. Posteriormente, el agente SIREFI verifica si tiene proyectos que estén asignados al IM Juan, al ver que el proyecto SIREFI_p_12 cumple con esto, le informa al agente correspondiente mediante un mensaje que indica que el agente juan ha sido registrado. El agente de proyecto SIREFI_p_12 envía un mensaje al agente juan donde le informa que el proyecto requiere ser atendido por el IM Juan. Una vez que el agente juan recibe el mensaje enviado por el agente SIREFI_p_12, le muestra al IM Juan, a través de la interfaz de usuario, los proyectos a los que está asignado y que requieren ser atendidos.

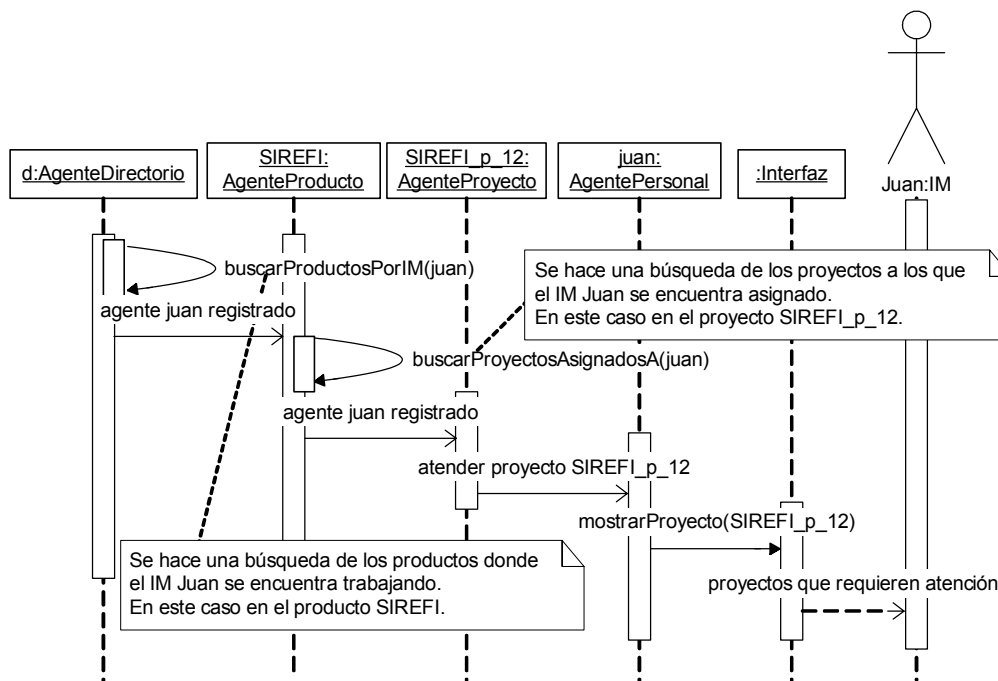


Figura 20. Segunda parte del diagrama de secuencia del caso de uso Inicializar sistema.

El **diagrama de secuencia del caso de uso Atender proyecto** también se presenta en dos partes. La figura 21 muestra la primer parte de este diagrama. Como se puede ver, una vez que se le han mostrado la lista de proyectos al IM Juan, este elige uno de ellos, lo cual es

notificado al agente juan por medio de un evento generado en la interfaz gráfica. Posteriormente el agente juan muestra los datos del proyecto. En seguida el IM Juan decide atender el proyecto, por lo que el agente juan muestra los datos del reporte de error que dio origen al mismo, a la vez que informa al agente juan_km que el IM Juan se encuentra atendiendo este proyecto.

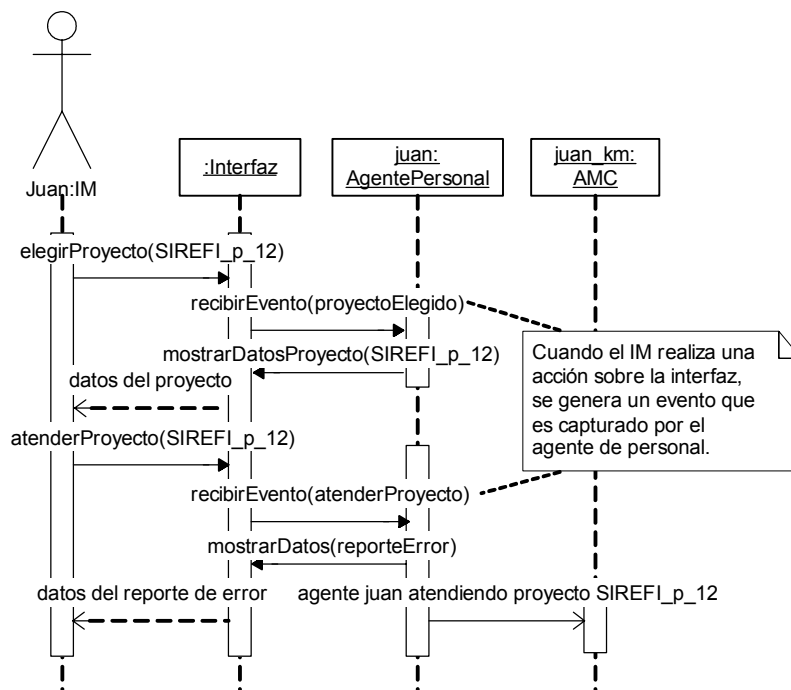


Figura 21. Primer parte del diagrama de secuencia del caso de uso Atender proyecto.

La segunda parte del diagrama de secuencia del caso de uso Atender proyecto, es presentado en la figura 22. En esta figura es posible ver que, una vez realizadas las acciones mostradas en la figura anterior, el agente juan_km extrae algunos datos del reporte de error, específicamente el sistema y módulo donde se generó el error, así como el tipo de error. Estos datos se agregan a una lista con temas de conocimiento a buscar, mediante la función "agregarTemaA_Buscar". Posteriormente, el agente juan_km se encarga de buscar, en la base de conocimientos (kBase), los temas de conocimiento que estén definidos como necesarios para trabajar con el producto y módulo donde se generó el error. Cada uno de estos temas necesarios, son agregados a la lista de temas de conocimiento a buscar. Una vez hecho esto, el agente juan_km busca las fuentes que conozcan sobre alguno de estos temas,

para posteriormente informar al agente juan el total de fuentes encontradas, y que éste, a su vez, se lo informe al IM Juan a través de la interfaz gráfica de usuario.

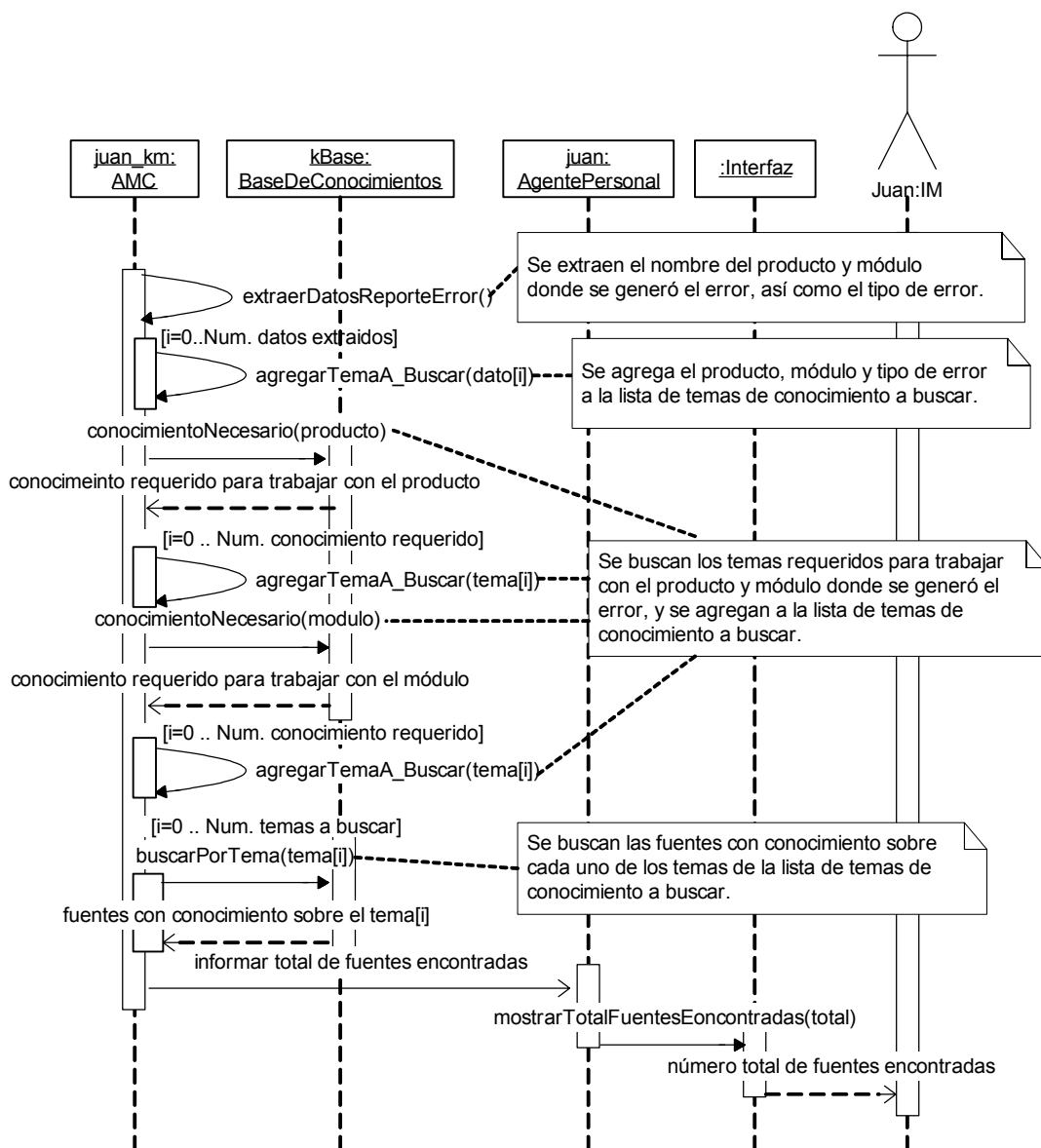


Figura 22. Segunda parte del diagrama de secuencia del caso de uso Atender proyecto.

Como ya mencionamos, aquí solo se presentan los dos primeros casos de uso, y sus diagramas de secuencia, extraídos del escenario, el resto se encuentran en el Apéndice B.

Una vez definido el comportamiento esperado del prototipo por medio de los diagramas de casos de uso y secuencia aquí mostrados, lo siguiente es definir su estructura interna, lo cual se hace a continuación a través de un diagrama de clases.

V.2.2.3 Diagrama de clases del prototipo

El diagrama de clases es una guía para la implementación del sistema, ya que presenta la estructura interna que éste tendrá. Esta sección presenta el diagrama de clases del prototipo implementado, las cuales han sido divididas en tres grupos: el primero corresponde a las clases que componen los elementos de la arquitectura implementados; el segundo a las clases que representan las acciones realizadas por los agentes para poder comunicarse entre ellos; mientras que el tercero lo constituyen las clases de la ontología que ha sido desarrollada para representar a los elementos de la base de conocimientos. La descripción de la totalidad de las clases mostradas en los diagramas se presenta en el Apéndice B.

Para el desarrollo del diagrama de clases fue necesario considerar algunos elementos de la tecnología utilizada, ya que éstos fueron la base para la implementación de varias de las clases del prototipo. Aún cuando estos elementos se describirán más adelante, en esta sección se menciona cuáles de ellos forman parte de los componentes del diagrama.

La figura 23 muestra el diagrama de las clases que corresponden a los elementos de la arquitectura implementados. Este diagrama presenta las relaciones de herencia y agregación entre las clases. Un punto importante es que se han agregado dos clases pertenecientes a la plataforma de agentes sobre la cual se implementó el prototipo, estas clases son Agent y Behaviour. La clase Agent es la base para la implementación de los distintos agentes, mientras que la clase Behaviour, es el mecanismo que proporciona la plataforma para implementar las acciones que ejecutan los agentes. Como es posible observar, las clases que definen los agentes heredan de la clase AbstractGuiAgent, la cual es una clase abstracta que define métodos para los agentes que manejan una interfaz gráfica. Es importante recalcar que existe una clase llamada RegisteredAgent, derivada de AbstractGuiAgent. Esta clase es la base para todos los agentes que son registrados en el directorio de agentes, y

proporciona algunas funciones básicas para esto. Otra de las clases de apoyo implementadas es *AgentsContainer*, la cual sirve como base para los contenedores de agentes, esta clase proporciona algunas funciones estándar para los contenedores, como la creación del contenedor, y la inicialización de agentes. A continuación se describen brevemente el resto de las clases mostradas en la figura.

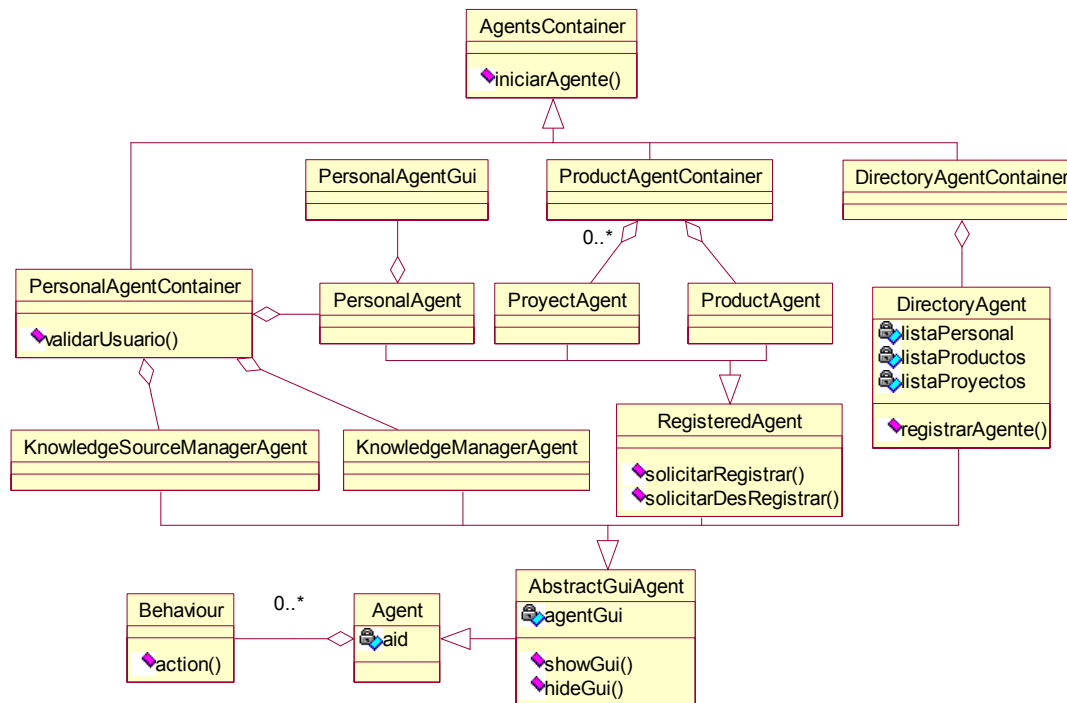


Figura 23. Diagrama de clases del prototipo.

- *PersonalAgentContainer*, implementación del contenedor de agente de personal. Se encarga de la validación del nombre y clave de usuario del IM, así como de la inicialización de los agentes que dan soporte al mismo.
- *PersonalAgent*, implementación del agente de personal. Se encarga de monitorear las acciones del IM.
- *PersonalAgentGui*, interfaz gráfica del agente de personal.

- *KnowledgeSourceManagerAgent*, implementación del agente manejador de fuentes de conocimiento. Se encarga de recuperar los datos de las fuentes de conocimiento consultadas por el IM.
- *KnowledgeManagerAgent*, implementación del agente manejador de conocimiento. Se encarga de hacer las búsquedas de fuentes de conocimiento.
- *ProductAgentContainer*, implementación del contenedor de agente de producto. Cada producto cuenta con un contenedor de este tipo, el cual se encarga de la inicialización del agente de producto, así como de los agentes de los proyectos del producto.
- *ProductAgent*, implementación del agente de producto. Lleva el control de los proyectos del producto. Cuando recibe la notificación de que un IM ha ingresado al sistema, revisa si tiene proyectos asignados a este IM, de ser así, notifica a los agentes de estos proyectos que el IM ha sido registrado.
- *ProjectAgent*, implementación del agente de proyecto. Cuando recibe la notificación de que el IM al que está asignado se ha registrado, envía un mensaje al agente de personal de este IM informándole que este proyecto requiere ser atendido.
- *DirectoryAgentContainer*, implementación del contenedor principal. Se encarga de inicializar la plataforma de agentes, así como el agente de directorio.
- *DirectoryAgent*, implementación del agente de directorio. Se encarga de recibir solicitudes de registro de los agentes de personal, producto y proyecto. Cuando detecta que el agente que solicita el registro es un agente de personal, envía un mensaje a los agentes de los productos donde el IM al que representa el agente de personal se encuentra trabajando, para notificar que el IM ha sido registrado.

Otra de las partes importantes del diseño del prototipo, fue la definición de las acciones mediante las cuales los agentes se comunicarán entre ellos. En la figura 24 se presenta la parte del diagrama que muestra estas acciones. Como es posible observar, todas ellas parten de la clase *Behaviour*, la cual, como ya mencionamos, es proporcionada por la plataforma de agentes para definir las acciones de los mismos. La plataforma también proporciona la clase *SimpleBehaviour* para dar un nivel un poco más alto de abstracción en la definición de acciones. La descripción de las clases mostradas en esta figura se presenta en el apéndice

B, sin embargo, para ejemplificar su utilización, en la figura 25 se muestra cuáles de estas actividades son realizadas por el agente de personal.

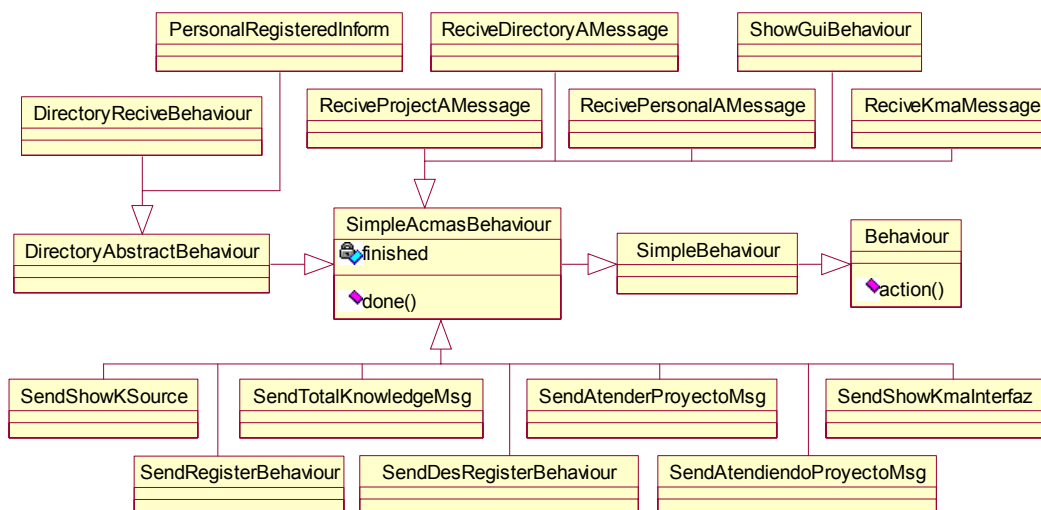


Figura 24. Diagrama de las clases de las acciones de los agentes.

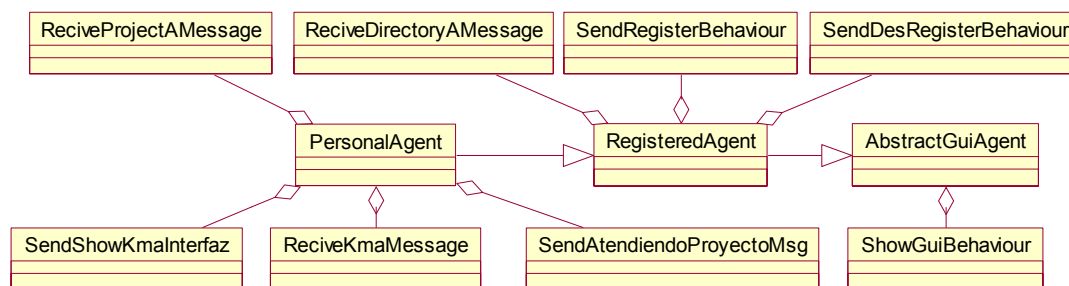


Figura 25. Diagrama de clases que muestra la relación del agente de personal y sus actividades principales.

En la figura 25 se puede ver como el agente de personal contiene actividades para solicitar el registro y la eliminación de su registro al agente de directorio (SendRegisterBehaviour, SendDesRegisterBehaviour), así como para recibir mensajes provenientes de éste último (ReciveDirectoryAMessage), las cuales hereda de la clase RegisteredAgent. Por otra parte, de la clase AbstractGuiAgent hereda una actividad para recibir mensajes solicitándole que despliegue su interfaz gráfica (ShowGuiBehaviour). Por su cuenta, este agente tiene actividades para recibir mensajes enviados por los agentes de proyecto

(ReciveProjectMessage), o el agente manejador de conocimiento (ReciveKmaMessage), así como para informar a éste último que el IM se encuentra atendiendo un proyecto (SendAtendiendoProyectoMsg), o solicitarle que despliegue la interfaz donde se muestran las fuentes de conocimiento encontradas (SendShowKmaInterfaz).

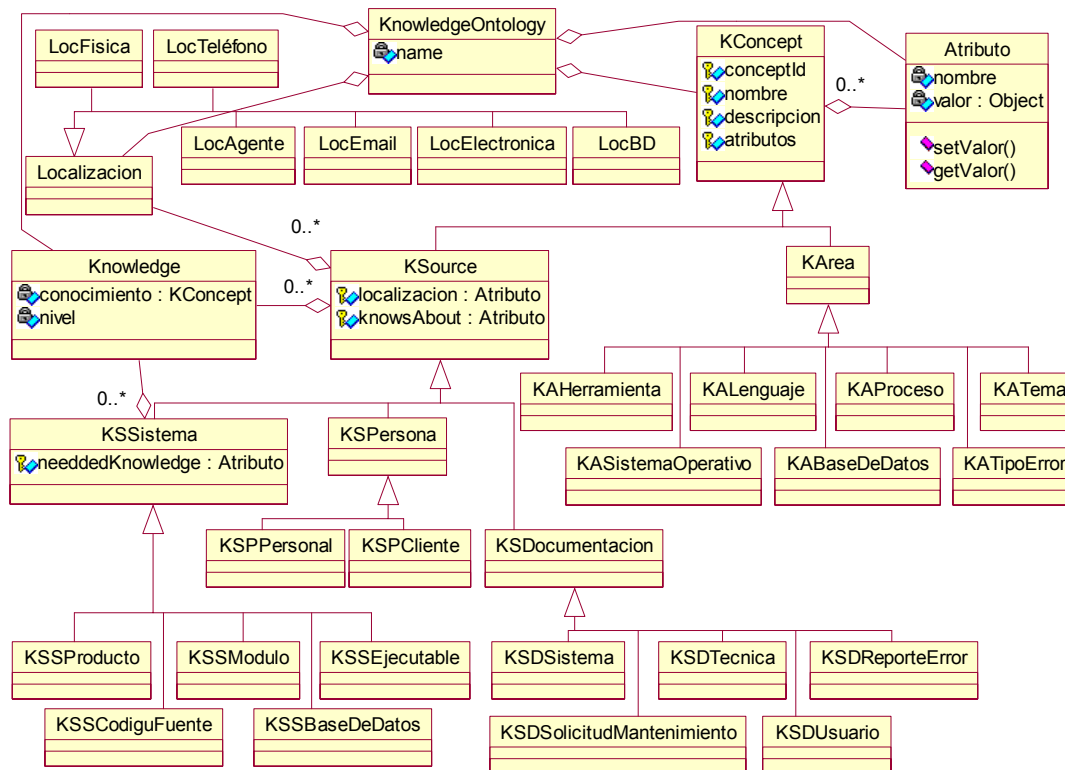


Figura 26. Diagrama de clases de la ontología de fuentes y tipos de conocimiento.

Durante el diseño también se consideró la forma en que serían manejados los elementos de la base de conocimientos. Para ello se decidió diseñar una ontología para representar las fuentes y tipos de conocimientos, tomando como base los definidos en el capítulo III. La figura 26 muestra el diagrama de las clases que componen esta ontología. Como es posible observar todos los elementos básicos de la ontología se derivan de la clase KConcept, la cual tiene dos sub-clases, KSource, utilizada para definir, a partir de ella, las distintas fuentes de conocimiento, y KArea, utilizada para definir áreas de conocimiento. Otras tres clases importantes son Knowledge, utilizada para definir el nivel de conocimiento que tiene

una determinada fuente sobre algún concepto de la ontología; Localización, utilizada para definir los mecanismos por los cuales puede ser consultada una determinada fuente de conocimiento; y Atributo, la cual se ha proporcionado para que las subclases de KConcept tengan un mecanismo estándar para la definición de atributos. El resto de las clases se describen en el apéndice B.

El diagrama de clases aquí mostrado, representa la estructura interna del prototipo que fue implementado. Los detalles de la implementación se presentan a continuación.

V.2.3 Implementación del prototipo

En esta sección se presentan los detalles de la implementación del prototipo. Estos detalles comprenden la tecnología utilizada, así como los elementos que fueron implementados para adaptar el diseño del prototipo a las características de las distintas tecnologías. Si bien se consideraron algunos aspectos de la tecnología al momento de realizar el diseño, hay otros que fueron considerados una vez que el prototipo se encontraba en la etapa de implementación.

A continuación se describen las tecnologías utilizadas para la implementación del prototipo; posteriormente se dan los detalles relacionados con la implementación de los elementos de la arquitectura, particularmente de los agentes y los mecanismos de comunicación entre ellos; para finalmente presentar los detalles de la implementación de la base de conocimientos.

V.2.3.1 Tecnologías utilizadas para la implementación del prototipo

La tecnología por medio de la cual se implementó el prototipo fue definida tomando en cuenta los siguientes requerimientos:

- **Posibilidad de integrar el sistema resultante a distintos ambientes de trabajo basados en agentes.** Esto nos llevó a considerar la necesidad de utilizar un estándar definido para el desarrollo de agentes. En particular, se decidió adoptar el estándar de la

Fundación para Agentes Físicos Inteligentes (FIPA por sus siglas en inglés), el cual define una arquitectura abstracta para el desarrollo de sistemas basados en agentes (Berger *et al.*, 2001; Burg, *et al.*, 2001; Dale y Mamdani, 2001; FIPA, 2003). El objetivo de FIPA es el de definir estándares que permitan la interacción entre sistemas heterogéneos basados en agentes. Entre las aportaciones de FIPA se encuentran una arquitectura básica para el desarrollo de plataformas de agentes, así como de sistemas basados e agentes; también define estándares para el lenguaje y los mecanismos de comunicación entre agentes.

- **Sistema fácilmente portable entre distintas plataformas.** El hecho de que en la actualidad el lenguaje Java se ha convertido en el más extendido para el desarrollo de sistemas que sean fácilmente portables entre plataformas, nos llevó a considerarlo como el lenguaje con el cual desarrollar el sistema prototipo.

Tomando en cuenta estos dos aspectos planteados, se consideró el Ambiente de Desarrollo de Agentes en Java (JADE por sus siglas en inglés) como la base para la implementación del prototipo. JADE proporciona una plataforma para el desarrollo de agentes que está basada en el estándar FIPA (Bellifemine *et al.*, 1999; Berger *et al.*, 2001; Bigus y Bigus, 2001; Vitaglione *et al.*, 2002; JADE, 2003). Entre sus características esta el que implementa prácticamente la totalidad del estándar, además de que se distribuye bajo una licencia de código abierto y está totalmente desarrollada en Java. A parte de JADE existen otra serie de plataformas para el desarrollo de agentes basadas en el estándar FIPA (Bigus y Bigus, 2001; FIPA, 2003), sin embargo, consideramos que JADE es la más completa de las que cubre los requerimientos arriba listados.

Por último, una vez definido el ambiente y la plataforma en que se desarrollaría el prototipo, el siguiente punto fue definir el esquema con base en el cual se almacenaría la base de conocimientos, así como la base de datos que se utilizaría para tal efecto. Esta elección se realizó tomando en cuenta que la información sería manejada utilizando el Lenguaje de Mercado eXtensible (XML por sus siglas en inglés). Se decidió utilizar XML debido a la flexibilidad que presenta para definir estructuras de datos complejas (Young,

2001). Como servidor de base de datos se decidió utilizar Xindice (XINDICE, 2003). Xindice es una base de datos nativa XML desarrollada bajo un proyecto dirigido por la fundación de software Apache (APACHE, 2003). Con esto queremos decir que Xindice almacena los datos con formato XML sin necesidad de realizar transformaciones a otros esquemas de base de datos, por ejemplo el esquema relacional (Liotta, 2003). La razón de utilizar Xindice radica en que su distribución está bajo el esquema de código abierto, por lo que resulta gratuita. Otro punto que se consideró importante es que está totalmente desarrollada en Java, lo que la hace fácilmente portable, además de que proporciona un conjunto de interfaces que permiten su fácil manejo mediante programas desarrollados en Java.

V.2.3.2 Aspectos de la implementación de los elementos de la arquitectura de agentes

Como ya mencionamos, la implementación de los agentes se realizó con base en las especificaciones de JADE. La plataforma JADE maneja un esquema basado en contenedores de agentes, la figura 27 muestra un ejemplo de lo que podría ser una plataforma JADE.

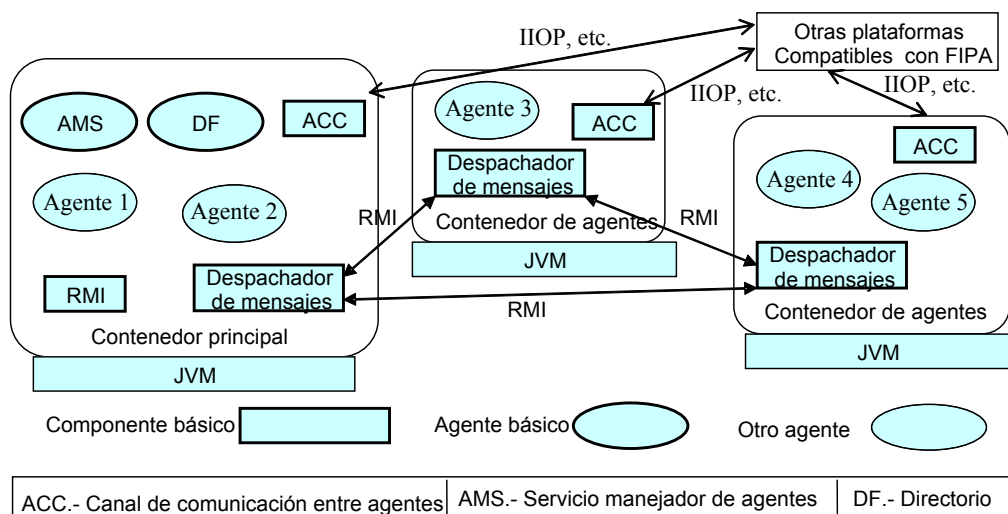


Figura 27. Ejemplo del funcionamiento de la plataforma JADE.

Como se puede observar en la figura 27, cada contenedor en la plataforma tiene un despachador de mensajes, encargado del envío de mensajes a agentes en otros

contenedores, así como un canal de comunicación entre agentes (ACC), mediante el cual es posible establecer comunicación con otras plataformas de agentes. Cada plataforma JADE tiene un contenedor principal. En éste se encuentra un agente de servicios encargado de la administración de los agentes de la plataforma (AMS), el cual mantiene un registro de los identificadores y estados en que se encuentran estos agentes; y un agente de directorio de servicios (DF), en el cual los agentes pueden registrar servicios. Por otro lado, JADE utiliza el mecanismo de Java para la Invocación de Métodos Remotos (RMI por sus siglas en inglés) como mecanismo de comunicación interna a la plataforma, es decir, para la comunicación que se da entre los agentes dentro de la misma plataforma. RMI es un mecanismo que ha proporcionado Java para poder invocar métodos entre objetos que se ejecutan en máquinas virtuales distintas (Froufe, 2000). Debido a que JADE ejecuta cada uno de sus contenedores en una máquina virtual de Java (JVM) diferente, utiliza RMI para poder invocar métodos en objetos distribuidos entre estas diferentes máquinas virtuales. El protocolo de comunicación RMI utiliza un registro para poder localizar los objetos distribuidos, el cual, como lo muestra la figura 27, también se encuentra en el contenedor principal de la plataforma.

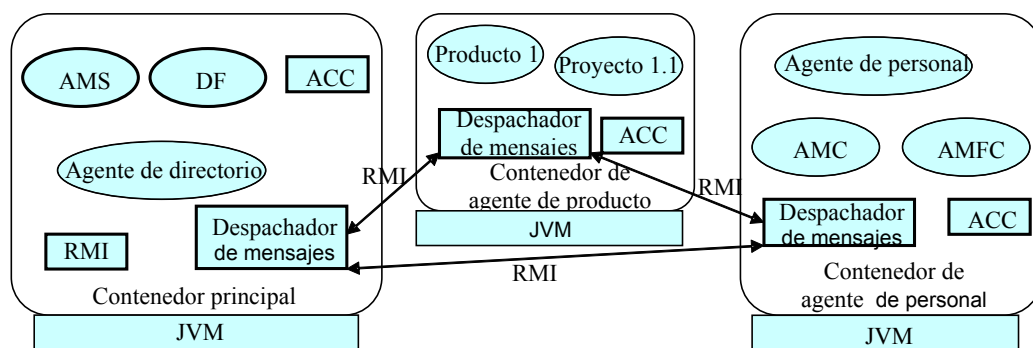


Figura 28. Representación de la ejecución del prototipo dentro de la plataforma JADE.

Tomando en cuenta la estructura interna de la plataforma de JADE, la figura 28 muestra un ejemplo de la manera en que se organizan los agentes del prototipo desarrollado, dentro de la plataforma. Como podemos ver, el agente de directorio se ejecuta dentro del contenedor principal. Para esto, se definió que el contenedor del agente de directorio se encargará de

iniciar la plataforma. La figura también muestra la representación de un contenedor de agente de personal, con el agente de personal, el AMC y el AMFC; así como un contenedor de agente de producto, donde se encuentra el agente de producto y uno de proyecto.

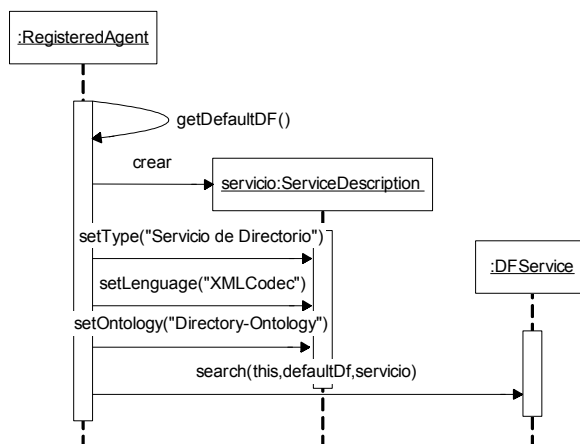


Figura 29. Representación de la búsqueda del servicio de directorio.

Es importante mencionar que para hacer posible que los agentes de personal puedan encontrar al agente de directorio dentro de la plataforma, se ha utilizado el directorio de servicios (DF) proporcionado por JADE, de manera que, el agente de directorio registra un servicio en el mismo al momento de activarse. Cuando algún agente derivado de RegisteredAgent es activado, solicita el servicio registrado por el agente de directorio al DF, este proceso se realiza mediante dos clases definidas por JADE: ServiceDescription que sirve para describir servicios, y DFService, que proporciona funciones para el registro y búsqueda de servicios, la figura 29 presenta un diagrama de secuencia que muestra un ejemplo de cómo se realiza esto. Como es posible observar, se utilizan la función setLanguage() y setOntology() para establecer el lenguaje y ontología que definirán el lenguaje de comunicación que entenderá el agente que registra el servicio, estos dos conceptos se tratan más adelante. El proceso del registro de servicios es muy similar al mostrado en este diagrama, la diferencia es la llamada que se hace a la clase DFService; en el caso del registro de servicios la función llamada es “register()”. Cabe mencionar que la

respuesta dada a la función “search()”, incluye los datos del agente de directorio, entre ellos, su identificador (directorioId), utilizado posteriormente para poder enviarle mensajes.

Para ilustrar el envío de mensajes entre agentes, se presenta un ejemplo del registro de un agente de personal en el directorio de agentes, lo cual se hace una vez obtenidos los datos del servicio de directorio (esto se mostró en la figura anterior). Para esto se crea una nueva actividad basada en la clase SendRegisterBehaviour. En esta clase, se crea un objeto de tipo Registrar y otro de tipo AgentePersonal, usados para crear el contenido del mensaje. Estos objetos se definen más adelante, en la ontología de directorio. Es importante mencionar que todos los mensajes se crean mediante la clase ACLMessage proporcionada por JADE. La figura 30 presenta un diagrama de secuencia que ejemplifica esto. El proceso mostrado en este diagrama, genera un mensaje similar al de la figura 31.

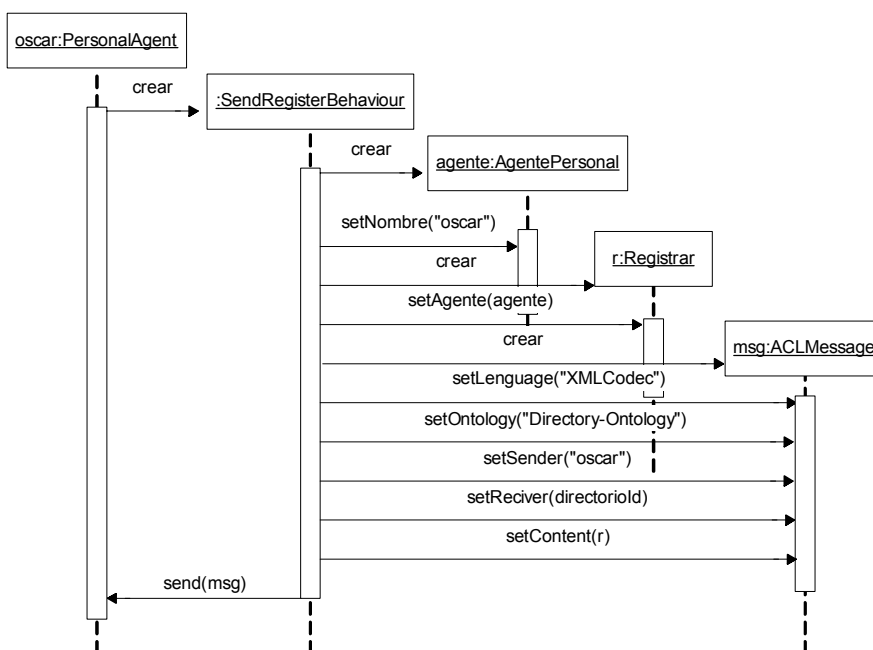


Figura 30. Ejemplo del proceso realizado al enviar un mensaje solicitando el registro de un agente de personal con el agente de directorio.

Otro punto importante a considerar dentro de la plataforma, es el lenguaje de comunicación utilizado por los agentes. FIPA ha definido una estructura de mensajes basada en lenguajes

de contenido y ontologías. El lenguaje de contenido indica la forma que tendrá el cuerpo del mensaje que será enviado, mientras que la ontología indica el tipo de elementos que pueden ser usados para la comunicación entre los agentes. La figura 31 muestra un ejemplo de la estructura de un mensaje en JADE. Como es posible observar, en el mensaje se indica el tipo de mensaje, en este caso es una petición (REQUEST); el agente que lo envía; el agente al que va dirigido; el contenido del mensaje; el lenguaje en el que va codificado el contenido; así como la ontología en la cual están definidos los elementos utilizados en el contenido del mensaje.

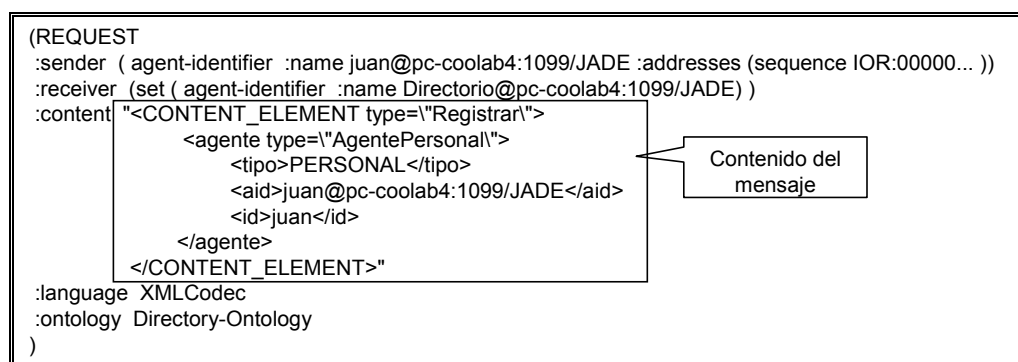


Figura 31. Ejemplo de la estructura de un mensaje en JADE.

JADE proporciona un mecanismo mediante el cual es posible utilizar distintos codificadores de lenguajes de contenido, los cuales pueden convertir un objeto de la ontología al formato definido por el lenguaje y viceversa. En nuestro caso, hemos utilizado un codificador para XML (XMLCodec), el cual, como se muestra en la figura 31, define una estructura para representar los elementos de la ontología en formato XML. Además, se desarrollaron cuatro ontologías para definir los elementos necesarios para la comunicación con los distintos agentes implementados, de las cuales, una es para el agente de directorio, otra para los agentes de proyecto, otra para el agente de personal y una más para el agente manejador de conocimiento. Estas ontologías se componen de tres tipos de elementos básicos: conceptos, predicados y acciones. Los predicados sirven para indicar hechos, mientras que las acciones sirven para solicitar a los agentes que realicen alguna tarea, por último, los conceptos permiten definir los elementos del dominio de la ontología. Para ejemplificar esto, la figura 32 muestra el diagrama de clases de la ontología del agente de

directorio, el resto de las ontologías se presentan en el apéndice B. En este diagrama se muestran tres interfaces proporcionadas por JADE para definir acciones de los agentes (AgentAction), predicados (Predicate) y conceptos (Concept); así como la clase Ontology que sirve para derivar las clases base de las ontologías. Como es posible observar en la figura, los componentes de la ontología del agente de directorio son: dos acciones principales de tipo DirectoryAction: Registrar y DesRegistrar, utilizadas para solicitar al agente de directorio que registre o quite el registro de algún otro agente; dos predicados, utilizados para indicar que un agente ha sido registrado o se ha eliminado su registro; y cuatro conceptos, utilizados para identificar a los agentes que solicitan ser registrados o que se elimine su registro.

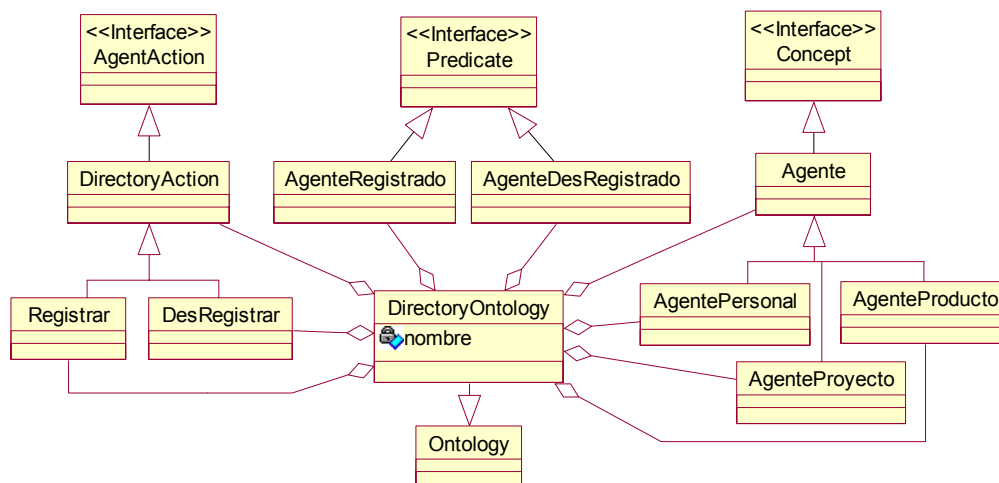


Figura 32. Diagrama de clases de la ontología del agente de directorio.

Una vez definidos los detalles de la implementación de los agentes, así como de los mecanismos de comunicación empleados, lo siguiente es detallar los aspectos relacionados con el manejo de la información utilizada por los agentes, así como de la base de conocimientos.

V.2.3.3 Manejo de la información y de la base de conocimientos

Como ya hemos mencionado, se utilizó Xindice como servidor de base de datos, el cual maneja XML como el lenguaje para definir las estructuras de datos. Xindice almacena la

información mediante un esquema estructurado en base a colecciones y documentos, donde los documentos son las estructuras de datos que se almacenan, y las colecciones la manera en que estos documentos se agrupan. Este esquema puede verse como una analogía de la estructura arborescente de archivos que manejan los sistemas operativos. En particular, la información manejada por el prototipo ha sido dividida de la manera mostrada por la figura 33, donde acmas es el nombre que se ha dado a la colección base, dentro de la cual existe una colección para los datos de los miembros del personal, otra para los de los clientes, otra para la base de conocimiento, otra para los reportes de error, y una más para los productos. En esta última se encuentra una colección para cada producto, en la cual se almacena información de los elementos que constituyen al producto, como los módulos, archivos, documentos, etc.

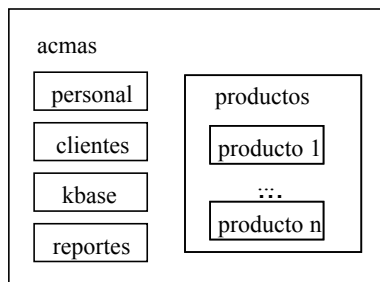


Figura 33. Distribución de las colecciones de la base de datos del prototipo.

Para el manejo de las entidades de información se desarrollaron una serie de clases mostradas en la figura 34 (estas clases se describen en el apéndice B). Como es posible observar, se definió una interfaz llamada XMLObjectInterface con dos funciones: “toXML()”, que regresa los datos del objeto en formato XML; y “fromXML()”, la cual obtiene los datos de una cadena con formato XML. Estas dos funciones son utilizadas para guardar y extraer los datos de la base de datos. De manera que, si se quieren convertir los datos de un miembro del personal al formato XML, por ejemplo, el resultado de la función toXML() sería algo parecido a lo que muestra la figura 35.

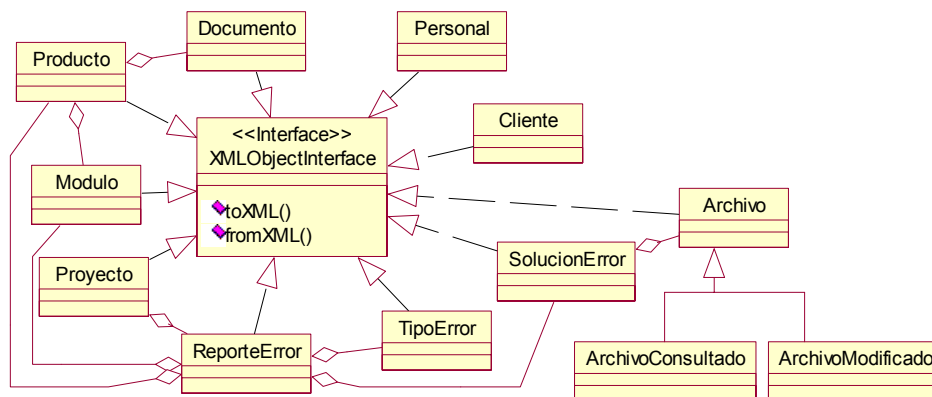


Figura 34. Diagrama con las clases de las entidades de información de la base de datos.

```

<PERSONAL login="oscar">
  <NOMBRE>Oscar Mario</NOMBRE>
  <APELLIDO>Rodríguez Elias</APELLIDO>
  <EMAIL>orodrigu@cicese.mx</EMAIL>
  <TELEFONO>25412</TELEFONO>
  <DIRECCION>Laboratorio de Cómputo Colaborativo.</DIRECCION>
</PERSONAL>

```

Figura 35. Representación de los datos de un miembro del personal en formato XML.

Al igual que con las clases anteriores, los elementos de la ontología de fuentes y tipos de conocimientos, mostrada en la figura 26, también implementan la interfaz `XMLObjectInterface`, específicamente es la clase `KConcept` donde se implementan los métodos de esta interfaz. Un ejemplo de la manera en que un concepto de conocimiento (`KConcept`) es representado mediante XML, se muestra en la figura 36. En este ejemplo se ha tomado un objeto de tipo `KSPPersonal`, esta es una fuente de conocimiento que corresponde con el miembro del personal mostrado en la figura 35. En este caso es posible ver que la fuente contiene tres medios por los cuales puede ser consultada (correo electrónico, teléfono y dirección física), así como un tema sobre el cual conoce (Java).



Figura 36. Ejemplo de la representación en XML de una fuente de conocimiento de tipo personal. Esta fuente tiene tres medios por los cuales localizarla, o consultarla, así como un tema sobre el cual conoce.

Para ejemplificar la manera en que son agregadas las fuentes de conocimiento, tomaremos como ejemplo lo que sucede cuando se agrega un nuevo miembro del personal. Cuando lo anterior sucede, se crea un objeto de tipo `KSPPersonal`, el cual define a una fuente de conocimiento de tipo miembro del personal. Posteriormente, se toman los datos del miembro del personal y se agregan al objeto de tipo `KSPPersonal`, los datos principales son el identificador o login del miembro del personal, y los medios por los cuales es posible localizarlo (e-mail, teléfono, dirección física). Este proceso es realizado en el agente de directorio, ya que es en este agente donde se implementaron las funciones que permiten agregar clientes, miembros del personal, productos, reportes y tipos de error; esto se describe con mayor detalle en el apéndice C.

El último aspecto a considerar de la base de datos, es el mecanismo mediante el cual se realizan las búsquedas de las fuentes de conocimiento, lo cual se hace utilizando el lenguaje de consultas definido para XML, XPath (XPAT, 2003). El prototipo utiliza XPath como lenguaje de consultas. La figura 37 muestra un ejemplo de una consulta en lenguaje XPath. La consulta mostrada en la figura, es una búsqueda de las fuentes que conozcan sobre el producto SIREFI. Es necesario aclarar que todos los conceptos de conocimiento guardados en la base de datos tienen un identificador único. En el caso de los miembros del personal su identificador es el nombre de usuario, mientras que para los productos, el nombre del producto. El resto de los elementos de la base de conocimientos tienen un identificador que es generado por el sistema al momento de ser almacenados.

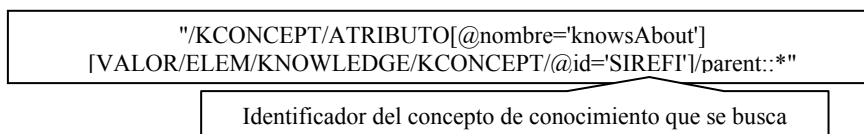


Figura 37. Ejemplo de una consulta en lenguaje XPath.

La consulta mostrada en la figura 37 indica que se están buscando todos aquellos conceptos de conocimiento (KCONCEPT) que cuenten con un atributo de nombre “knowsAbout”, en cuyo valor exista un elemento de tipo KNOWLEDGE, donde se defina un concepto de conocimiento (KCONCEPT) con identificador igual a “SIREFI”. Es decir, que se están buscando todas las fuentes que tengan conocimiento sobre el producto SIREFI.

Como es posible observar en toda esta sección, son bastantes los detalles que se tomaron en cuenta al momento de la implementación del prototipo. Además, como ya hemos mencionado, fue necesario desarrollar funciones de apoyo, de manera que se hiciera posible el funcionamiento de los elementos principales del mismo. Esta funcionalidad se presenta en la siguiente sección, donde se describe el funcionamiento de los principales elementos implementados para dar soporte al escenario definido para el diseño del prototipo.

V.2.4 Funcionalidad básica del prototipo

Esta sección presenta una descripción del funcionamiento básico del prototipo desarrollado. En particular, se muestra la funcionalidad del subsistema local a cada Ingeniero de Mantenimiento (IM), lo que incluye el funcionamiento de la interfaz de usuario, así como la de los agentes que apoyan las tareas del IM. Esta funcionalidad es mostrada por medio de un escenario de uso. Los datos mostrados en esta sección serán complementados a través de la descripción de las pruebas realizadas que se encuentran en el siguiente capítulo. Como ya hemos mencionado, para hacer posible el funcionamiento de los elementos del prototipo mostrados en esta sección, fue necesario implementar algunas funciones de apoyo, tanto en el agente de directorio, como en el de proyecto, además, se desarrolló una aplicación externa para el manejo de la base de conocimientos, la descripción de la funcionalidad de estos elementos se encuentra en el apéndice C.

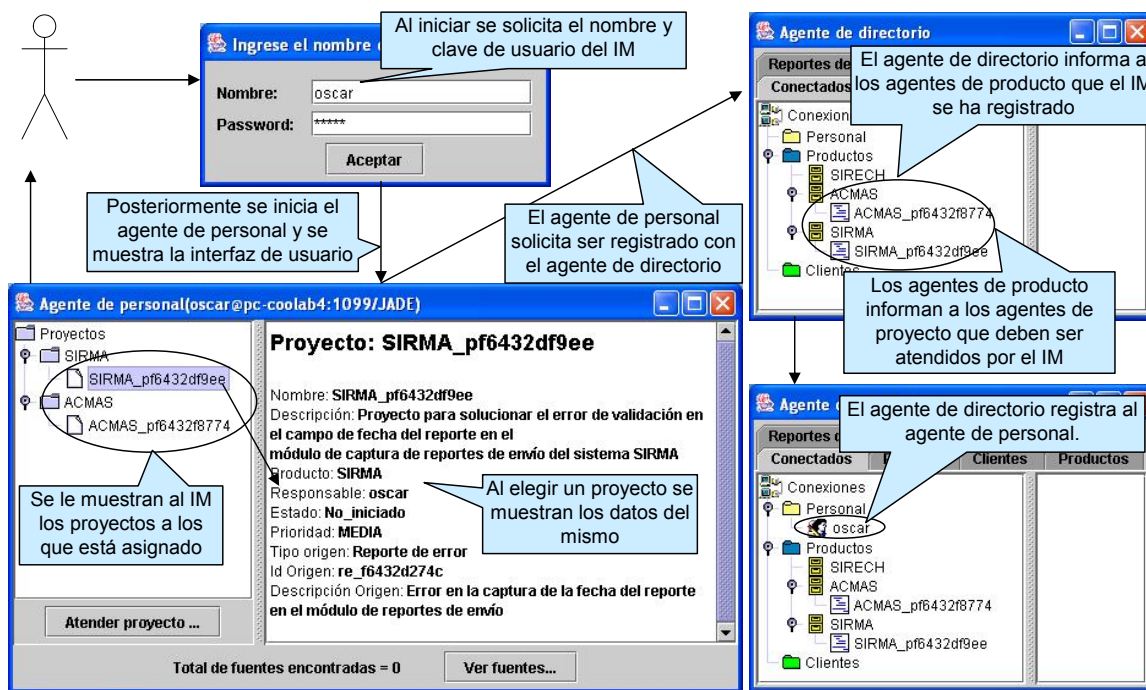


Figura 38 Representación de las acciones del sistema al momento de inicializar el subsistema local del IM.

Primeramente, la figura 38 muestra una representación de lo que sucede al momento en que el IM inicializa el sistema. Esta figura es una representación del caso de uso Inicializar

sistema. Como se puede ver, una vez que el nombre y clave de usuario del IM han sido validados, se despliega la interfaz de usuario del agente de personal, a la vez que este último solicita al agente de directorio que lo registre. Como se muestra en la figura 38, el agente de directorio también cuenta con una interfaz donde es posible ver los agentes que se encuentran registrados. Estos agentes están ordenados por su tipo, iniciando con los agentes de personal, seguido por los agentes de producto, y finalmente los de cliente. Al seleccionar cada uno de los agentes de producto, es posible ver los agentes de los proyectos de dicho producto.

Como también se muestra en la figura 38, se ha tomado un ejemplo donde el IM “oscar” está asignado a dos proyectos, uno perteneciente al producto SIRMA, y otro al producto ACMAS. Es posible observar que cuando el IM elige un determinado proyecto, los datos de éste son mostrados por medio de la interfaz gráfica.

La figura 39 muestra los elementos de que se compone la ventana principal de la interfaz gráfica del agente de personal. Primeramente está el título de la ventana, donde se muestra el identificador que el agente de personal tiene dentro de la plataforma de agentes. La parte central de la interfaz gráfica se compone de dos áreas grandes: la de despliegue de proyectos, y la de despliegue de datos. El área de despliegue de proyectos cuenta con un botón que permite indicar que se desea atender el proyecto seleccionado. Por último, en la parte inferior se presenta una leyenda que indica el número de fuentes encontradas por el agente manejador de conocimiento (AMC). El total de fuentes encontradas es informado por el AMC una vez que se le ha indicado que el IM decidió atender un determinado proyecto, y que ha concluido la búsqueda de fuentes que pudieran ayudar a realizar las tareas definidas por dicho proyecto. En esta última parte de la interfaz gráfica también se encuentra el botón que le permite al IM solicitar que se le muestren las fuentes de conocimiento encontradas.

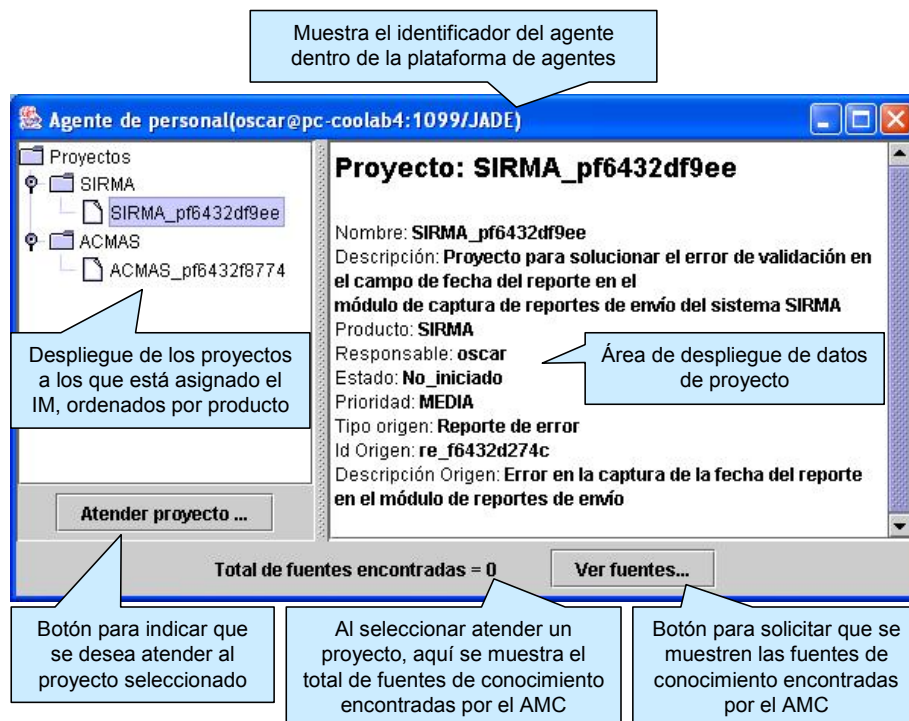


Figura 39. Interfaz gráfica del agente de personal.

Cuando el IM presiona el botón *Atender proyecto...*, el agente de personal muestra una ventana con los datos del reporte de error que dio origen al proyecto, a la par que informa al AMC del proyecto que el IM está atendiendo, para que busque fuentes de conocimiento que pudieran ayudar a solucionar el error reportado.

La figura 40 muestra la ventana con los datos del reporte de error. Para capturar la solución que se le dio al error reportado, se debe presionar el botón *Asignar Solución...* que se encuentra en la parte inferior de la pantalla. Este botón provoca que se abra la ventana que es mostrada en la figura 41.

The screenshot shows a window titled "Reporte de error" with the following fields and controls:

- Identificador:** re_f6439c27ed
- Nombre de Cliente:** Oscar Rodríguez
- Fecha:** Tue Jul 08 13:31:02 PDT 2003
- Sistema:** ACMAS (dropdown menu)
- Módulo:** PersonalAgentInterface (dropdown menu)
- Clase de error:** Error de interfaz de usuario (dropdown menu)
- Tipo de error:** Error cosmético (dropdown menu)
- Descripción del error:** La interfaz presenta alguna falla de tipo cosmética
- Descripción del reporte:** El ancho de la ventana del agente de personal es muy poco
- Detalles del error:**
 - Datos Capturados:** Ninguno
- Observaciones:** El ancho de la ventana principal de la interfaz gráfica del agente de personal es muy reducido, por lo que Es necesario incrementarlo para que los datos sean mostrados de una mejor manera.
- Asignar Solución...** (button)
- Aceptar** (button)
- Cancelar** (button)

Two callout boxes are present: one pointing to the top section with the text "Muestra los datos del reporte de error." and another pointing to the "Asignar Solución..." button with the text "Botón para mostrar la ventana para capturar los datos de la solución".

Figura 40. Ventana con los datos del reporte de error.

La ventana para capturar la solución de los errores reportados, mostrada en la figura 41, contiene diversos campos donde se proporciona información que puede servir en un futuro para ayudar en la solución de errores similares al reportado. En particular, se debe proporcionar el identificador de la persona que solucionó el error, el tiempo que se requirió para solucionarlo, la causa que dio origen al error, la manera en que se le dio solución, así como los archivos que fueron consultados o modificados. Las ventanas que permiten la captura de los archivos modificados y consultados son mostradas en la figura 42.

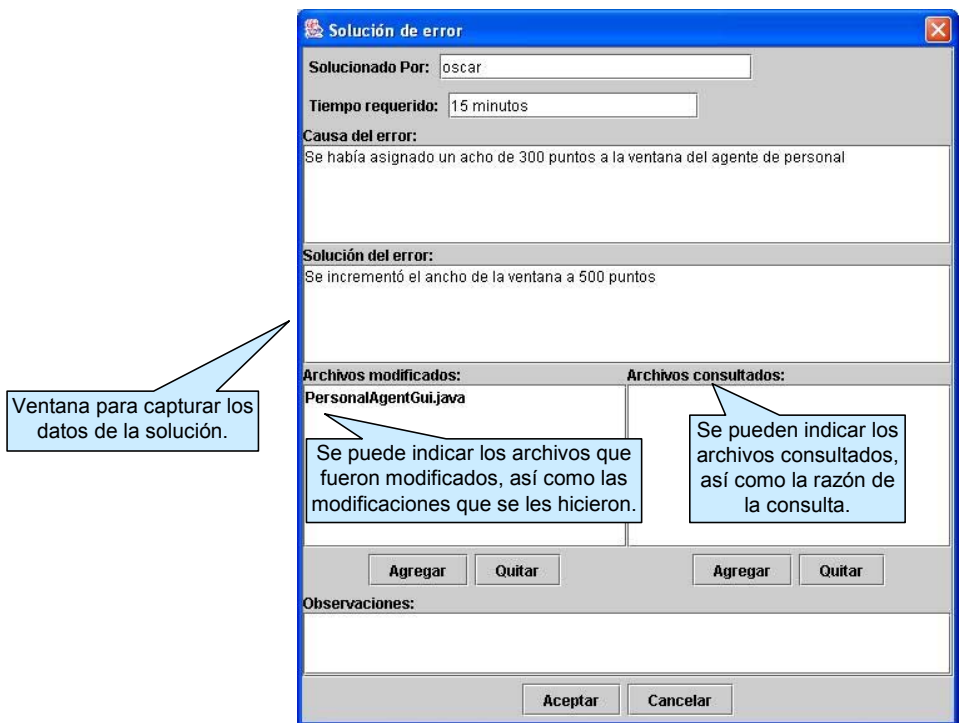


Figura 41. Ventana para la captura de las soluciones de los errores reportados.

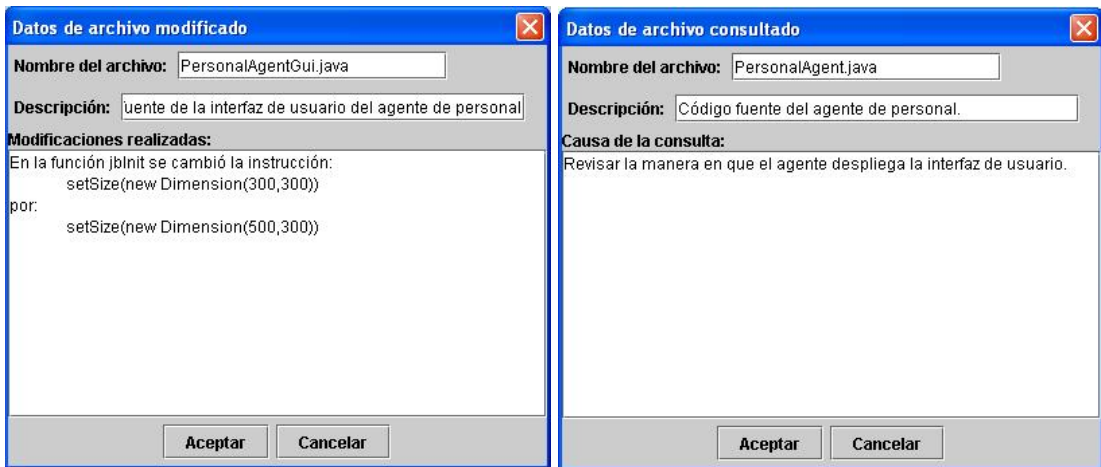


Figura 42. Ventanas de captura de archivos modificados y consultados.

Por otra parte, cuando el agente de personal recibe el mensaje del total de fuentes encontradas, enviado por el AMC, despliega una ventana de diálogo para comunicárselo al IM, a la par de que actualiza la leyenda en la parte inferior de la interfaz de usuario, tal y como lo muestra la figura 43.

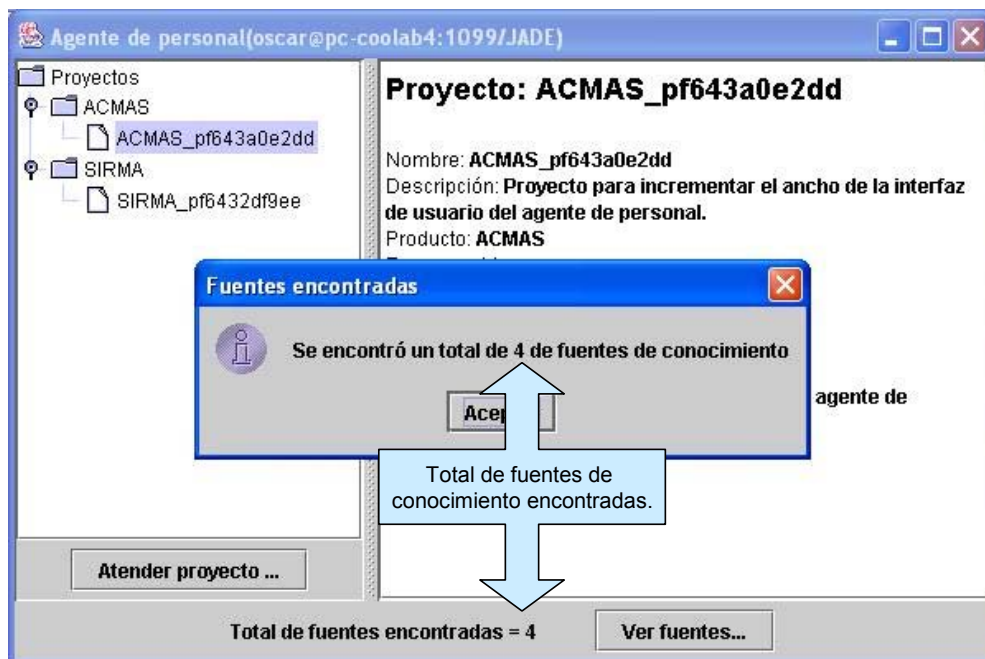


Figura 43. Despliegue del mensaje de total de fuentes encontradas.

Finalmente, al presionar el botón *Ver fuentes...* se despliega la ventana mostrada en la figura 44. Esta ventana es donde se presentan las fuentes de conocimiento que fueron encontradas. Los principales elementos que componen esta ventana son: la lista de fuentes encontradas, donde estas últimas se presentan ordenadas por su tipo, indicando el total de fuentes que hay en cada categoría; el área de despliegue de datos, donde se muestran los mecanismos para localizar una determinada fuente, así como los temas buscados sobre los cuales tiene conocimiento; y por último, la lista de temas de conocimiento a buscar, la cual presenta opciones para agregar o quitar temas, así como para solicitar una nueva búsqueda. Cabe hacer mención que, aún cuando se ha contemplado la posibilidad de asignar pesos que determinen la relevancia de un determinado tema a buscar, con el fin de poder ordenar las fuentes encontradas tomando como base esta relevancia, estos mecanismos no han sido implementados.

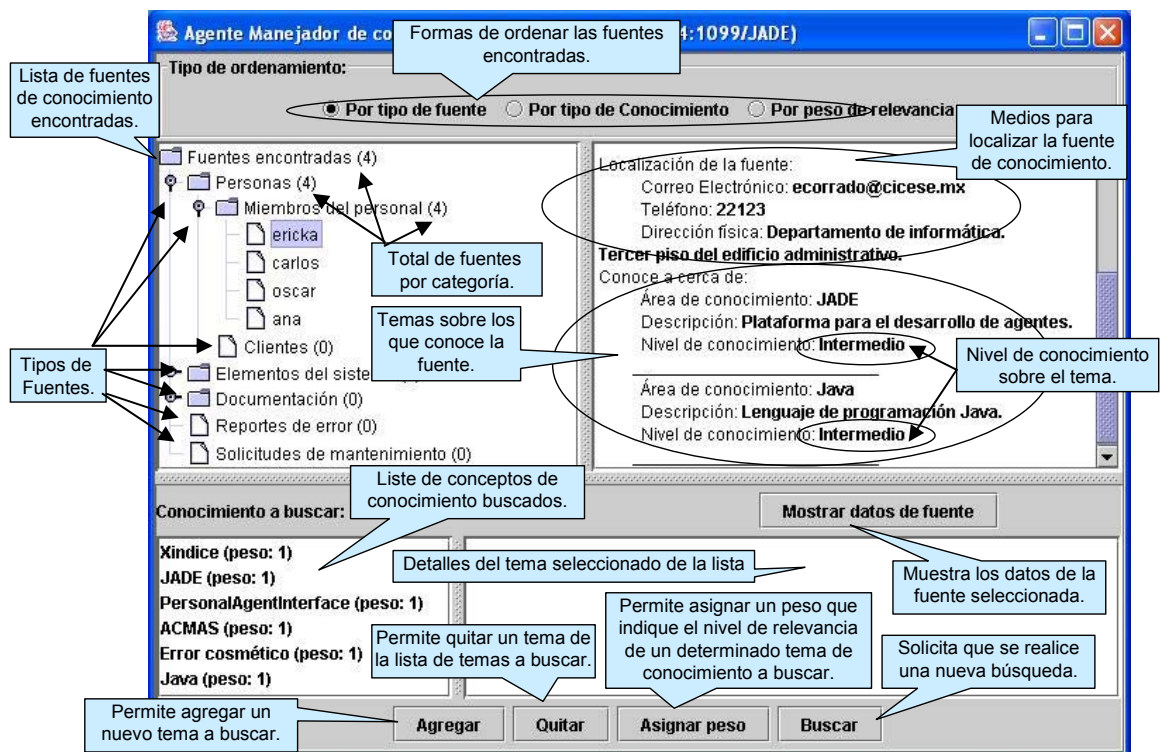


Figura 44. Ventana para el despliegue de fuentes de conocimiento encontradas.

V.3 Resumen

La arquitectura presentada en este capítulo, es el resultado de un conjunto de características obtenidas de la necesidad de apoyar a los encargados del mantenimiento del software en el manejo de los problemas de pérdida y desaprovechamiento del conocimiento que comúnmente tienen, como ya se ha visto en capítulos anteriores. La búsqueda de estas características llevó a la realización de dos casos de estudio, y finalmente al diseño de una arquitectura basada en agentes, cuyo objetivo es servir como la base del desarrollo de sistemas de administración del conocimiento que den soporte a los grupos encargados del mantenimiento del software.

Diversos trabajos sobre administración del conocimiento han tratado de solucionar el tipo de problemas que hemos planteado, sin embargo estos esfuerzos no han sido aplicados para tratar las necesidades particulares de los encargados del mantenimiento del software. La

arquitectura presentada en este capítulo contempla aspectos que ya han sido tratados por diversos sistemas de administración del conocimiento, pero también involucra aspectos propios del mantenimiento, con el fin de adoptar la administración del conocimiento desde un punto de vista particular de las necesidades de los encargados de este último.

Con el objetivo de validar que la arquitectura permite la implementación de sistemas de administración del conocimiento, se diseñó e implementó un sistema prototipo basado en la misma. Los detalles del diseño, implementación y funcionalidad del prototipo también fueron tratados en este capítulo. Si bien estos detalles permiten dar una primera impresión de que realmente es posible desarrollar sistemas de administración del conocimiento tomando como base la arquitectura, es necesario hacer un análisis más extenso para determinar que efectivamente el prototipo implementado cumple con esto. Es necesario validar que el prototipo cumple con las características que se definieron para una herramienta de administración del conocimiento en el mantenimiento del software, y con las especificaciones definidas por la arquitectura en la cual está basado. Con esto en mente se realizaron un conjunto de pruebas sobre el prototipo, orientadas a verificar si éste permite dar soporte al escenario definido para su diseño.

En el siguiente capítulo se presentan las pruebas que fueron realizadas, así como los resultados obtenidos. Estas pruebas fueron preparadas tomando como base el escenario definido en este capítulo, el cual sirvió como punto de partida para el diseño del prototipo.

Capítulo VI. Evaluación del prototipo

*“Un computador hará lo que le digas,
pero ello puede ser muy diferente a lo que tengas en mente”.*

Joseph Weizenbaum

*“Las pruebas del software...
representan una revisión final de las especificaciones,
del diseño y de la codificación”.*

(Pressman, 2003)

Existe una gran variedad de pruebas de software, entre las cuales, las encargadas de verificar que el sistema desarrollado cumple con los requerimientos establecidos, son las pruebas de validación (Pressman, 2003). Es por esto que, para validar que el prototipo presentado en el capítulo anterior cumple con los requerimientos funcionales que se definieron para su diseño, se realizaron un conjunto de pruebas de validación. Las pruebas se orientaron a validar si el prototipo permite dar seguimiento al escenario que se definió para su diseño. Dado que el prototipo se desarrolló con base en la arquitectura propuesta en el capítulo anterior, los resultados de ésta validación nos permitirán conocer si un sistema desarrollado con base en esta arquitectura, puede dar soporte al tipo de escenarios de los cuales se obtuvieron las características que dieron paso al diseño de la misma.

Las pruebas realizadas, así como los resultados obtenidos se presentan en este capítulo. La presentación de las pruebas y los resultados se ha dividido de la siguiente manera: primero se describe el plan de pruebas que se siguió y se define la relación que existe entre las pruebas realizadas y las características que se buscó probar; en seguida se muestra la metodología que se estableció para la realización de las pruebas, así como la preparación de éstas; para posteriormente presentar el proceso de realización de las mismas; y finalmente los resultados obtenidos.

VI.1 Guía para la realización de las pruebas

El plan de pruebas se realizó siguiendo un escenario similar al planteado para el diseño del prototipo, presentado en el capítulo anterior. La tabla IX presenta la guía de las pruebas a realizar, así como el comportamiento que se espera realicen los elementos del prototipo como resultado de éstas. Como es posible observar, se ha asignado un identificador a las actividades presentadas en la tabla. El identificador se compone de una letra que indica el caso de uso que se pretende validar con cada serie de pasos, iniciando con la letra “A” para la inicialización del sistema, “B” para la atención de un proyecto, “C” para la recuperación de las fuentes de conocimiento encontradas, “D” para la visualización de los datos de las fuentes de conocimiento, y “E” para el almacenamiento de la solución dada al error reportado. Además se ha asignado un número consecutivo que define la secuencia de los pasos en cada caso de uso.

Tabla IX. Guía para la realización de las pruebas.

Id	Pruebas/Resultado esperado
A1.-	El ingeniero de mantenimiento (IM) inicia la ejecución del prototipo. En este punto las actividades que el prototipo realizará son:
a.-	Mostrar una ventana para la captura del nombre y clave de usuario.
A2.-	El IM proporciona su nombre y clave de usuario. Este punto considera dos aspectos a probar:
a.-	Si el nombre o clave de usuario son incorrectos, el prototipo debe:
i.-	indicar al IM que los datos no son correctos,
ii.-	no permitir la inicialización de los agentes, y
iii.-	debe terminar su ejecución.
b.-	Si el nombre y clave de usuario son correctos. El prototipo debe:
i.-	inicializar los agentes que darán soporte al IM, tomando como identificador para los agentes el nombre de usuario del IM, y agregando el posfijo “_km” para el agente manejador de conocimiento, y “_ksm” para el agente manejador de fuentes de conocimiento,
ii.-	el agente de personal debe solicitar su registro en el directorio, al agente de directorio,
iii.-	el agente de directorio debe registrar al agente de personal,
iv.-	el agente de directorio debe informar al agente de personal que ha sido registrado
v.-	el agente de directorio debe informar a los agentes de producto que se encuentren registrados, y a los que esté asignado el IM, que éste último se ha registrado,
vi.-	los agentes de producto que reciban el mensaje anterior, deberán buscar si tienen proyectos a los que esté asignado el IM, de ser así, informarán a los agentes de proyecto correspondiente que el IM se ha registrado,

Tabla IX. Guía para la realización de las pruebas (continuación).

	vii.-	los agentes de proyecto que reciban el mensaje anterior, enviarán un mensaje al agente de personal del IM para informar que requieren ser atendidos,
	viii.-	el agente de personal mostrará al IM la lista de proyectos que requieren su atención,
	ix.-	la lista de proyectos se presentará ordenada por el producto al que pertenecen los proyectos.
B1.-		El IM selecciona un proyecto de la lista de proyectos a los que está asignado.
	a.-	El agente de personal deberá mostrar los datos del proyecto.
B2.-		El IM presiona el botón de atender proyecto. El agente de personal deberá:
	a.-	Mostrar una ventana con los datos del reporte de error que dio origen al proyecto.
	b.-	Enviar un mensaje al AMC indicándole el proyecto que se está atendiendo.
B3.-		El AMC recibe el mensaje enviado por el agente de personal. EL AMC deberá:
	a.-	Extraer los datos del sistema y módulo donde se generó el error, así como el tipo de error.
	b.-	Obtener el conocimiento requerido para trabajar con el sistema y módulo anterior.
	c.-	Realizar la búsqueda de fuentes que conozcan sobre los conceptos de conocimiento identificados.
	d.-	Enviar un mensaje al agente de personal indicándole el total de fuentes encontradas.
B4.-		El agente de personal recibe el mensaje que indica el total de fuentes encontradas. Esto lo lleva a realizar lo siguiente:
	a.-	Enviar un mensaje al IM indicándole el total de fuentes que fueron encontradas,
	b.-	Actualizar la leyenda en la parte inferior de la interfaz de usuario.
C1.-		El IM presiona el botón para ver las fuentes de conocimiento encontradas
	a.-	El agente de personal deberá enviar un mensaje al AMC para que muestre la interfaz de usuario.
C2.-		EL AMC recibe el mensaje enviado por el agente de personal para mostrar su interfaz.
	b.-	El AMC muestra la interfaz de usuario con las fuentes de conocimiento encontradas ordenadas por el tipo de fuente, así como la lista con los conceptos de conocimiento buscados.
D1.-		El IM selecciona una de las fuentes de conocimiento mostradas.
	a.-	El AMC muestra los datos de la fuente de conocimiento (mecanismos para localizarla, y el conocimiento que tiene.
D2.-		El IM presiona el botón de mostrar fuente para ver mayor información de la misma.
	a.-	El AMC debe enviar un mensaje al AMFC para indicarle que recupere los datos de la fuente de conocimiento elegida.
D3.-		EL AMFC recibe el mensaje para mostrar los datos de una fuente de conocimiento.
	a.-	EL AMFC muestra una ventana con los datos de la fuente de conocimiento.
E1.-		El IM presiona el botón de asignar solución en la ventana de datos de reportes de error.
	a.-	Se abre una ventana para capturar los datos de la solución dada al error reportado.
E2.-		El IM captura los detalles de la solución del error, y presiona el botón de aceptar.
	a.-	El agente de personal actualiza el reporte de error en la base de datos para asignarle la solución.
	b.-	El agente de personal agrega el reporte de error a la base de conocimientos como una nueva fuente de conocimiento.

Las actividades mostradas por la tabla son solo una guía, es por ello que no presentan detalles sobre el tipo de información que será capturada o desplegada. Estos datos se definen durante la descripción de la metodología establecida y la preparación que se hizo

para la realización de las pruebas, así como durante la descripción del proceso de pruebas. Estos puntos serán presentados a continuación, pero antes de pasar a ellos, hacemos una correlación de la guía de pruebas aquí descrita, con los requerimientos que se definieron para el desarrollo del prototipo.

VI.1.1 Relación de la guía de pruebas con las características del prototipo

Para verificar que las pruebas realizadas permiten validar que el prototipo cuenta con las características establecidas para su desarrollo, se identificó cuáles pruebas se relacionan con cada una de estas características. La relación es mostrada en la tabla X, la cual presenta las características, seguidas de los identificadores de las pruebas que permitirán validar si el prototipo cumple con ellas.

Tabla X. Relación entre las características y las pruebas definidas en la guía.

Características	Pruebas
Mecanismo de control de acceso.	A1.a; A2.a; A2.b
Elementos de la arquitectura implementados.	A2.b; B2.b; B3; B4; C1.a; C2; D2.a; D3
Hacer llegar los reportes de error al IM.	A2.b.vi; A2.b.vii; A2.b.viii; A2.b.ix
Permitir ver los datos de los proyectos y reportes de error asignados al IM.	B1.a; B2.a
Apoyo en la identificación de fuentes de conocimiento.	B2.b; B3; B4
Permitir recuperar las fuentes de conocimiento encontradas por el AMC.	C1; C2; D1; D2
Permitir la captura y almacenamiento de la solución dada al error reportado.	E1; E2

Como es posible observar en la tabla, cada característica se relaciona con más de una de las pruebas definidas en la guía, sin embargo, existe una secuencia definida por las actividades que corresponden con cada uno de los casos de uso identificados en el escenario que dio pie, tanto al desarrollo del prototipo, como a la definición de la guía de pruebas.

Una vez establecida la relación existente entre las pruebas definidas en la guía, y las características establecidas para el desarrollo del prototipo, pasaremos a describir la metodología que se siguió para preparar la realización de las pruebas.

VI.2 Metodología para la realización de las pruebas: preparación

Como ya mencionamos, las pruebas se realizaron tomando como base un escenario similar al definido para el diseño del prototipo. Para la realización de este escenario, se tomó información de uno de los sistemas a los que da soporte el personal del departamento de informática (DI) del CICESE, uno de los grupos donde se realizaron los casos de estudio presentados en el capítulo IV. En particular, el sistema del cual se obtuvieron los datos, es el de manejo de recursos financieros (SIREFI). El objetivo de lo anterior fue el verificar que el prototipo pudiera trabajar con información de un ambiente real de trabajo. Los datos que se tomaron de SIREFI fueron los principales módulos de que está compuesto; los principales documentos técnicos, de usuario y de sistema relacionados con el mismo; algunos reportes de errores hechos sobre el sistema, así como sus soluciones; los perfiles de conocimiento de dos miembros del personal de mantenimiento que han trabajado con él; y finalmente el tipo de conocimiento requerido para poder dar mantenimiento a los distintos módulos del sistema.

Para la captura de la información sobre la cual se realizaron las pruebas, se implementaron interfaces gráficas para los agentes de directorio y producto, las cuales proporcionan funciones para la captura de datos (estas interfaces, así como su funcionalidad básica, son descritas en el apéndice C). Entre las principales funciones que se implementaron en la interfaz de usuario del agente de directorio están: el permitir dar de alta a miembros del personal, clientes, productos, reportes de error, y tipos de error. Por su parte, en la interfaz de usuario del producto es posible crear proyectos y asociarlos a un determinado reporte de error. Para facilitar la captura de los datos de los reportes de error, se definió un catálogo de tipos de errores tomando como base la guía para la clasificación de anomalías del software, desarrollada por la IEEE (IEEE Std. 1044.1-1995).

Para el manejo de los conceptos de conocimiento, así como para asignar el conocimiento asociado a cada fuente, y el que se pudiera necesitar para trabajar con un determinado

elemento del sistema (módulo, archivo fuente, etc.), se desarrolló una aplicación que también es descrita en el apéndice C.

Por otro lado, se utilizaron las herramientas que proporciona JADE para poder verificar el funcionamiento de los agentes dentro de la plataforma. En particular, se utilizó el manejador remoto de agentes, el cual permite visualizar los agentes que se encuentran activos dentro de la plataforma, como lo muestra la figura 45. Este manejador de agentes proporciona diversas funciones que permiten, entre otras cosas, activar y desactivar agentes, así como enviar mensajes a determinados agentes, con el fin de probar el funcionamiento de éstos. Mediante este manejador de agentes fue que se validó que el prototipo realmente estaba inicializando los agentes establecidos.

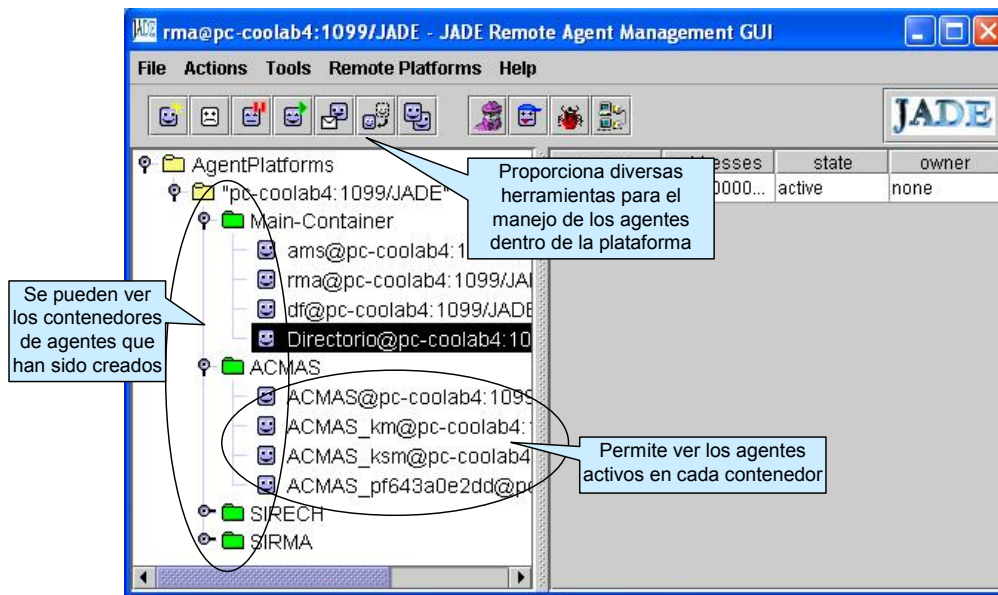


Figura 45. Interfaz del manejador remoto de agentes.

Para verificar la comunicación entre los distintos agentes se utilizó un agente "Sniffer", proporcionado por JADE, el cual permite ver el intercambio de mensajes entre los distintos agentes, como se muestra en la figura 46. A través de este mecanismo se validó que los agentes del prototipo enviaran correctamente los mensajes definidos.

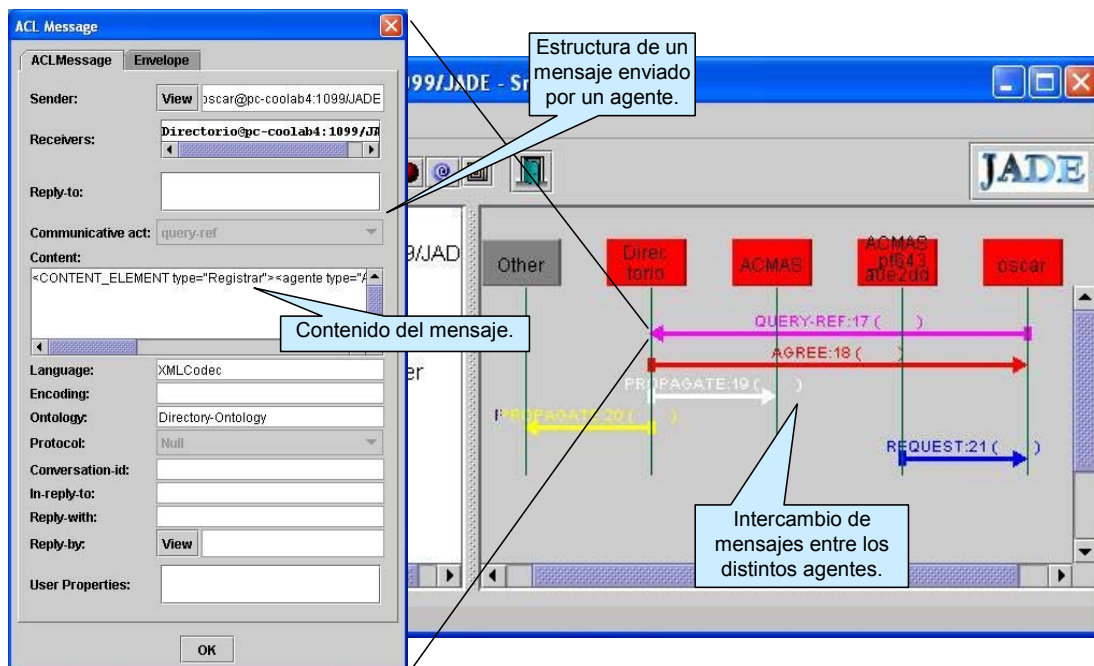


Figura 46. Interfaz de monitoreo de mensajes entre agentes.

Finalmente, se decidió que para hacer una correlación más directa con el escenario definido en el capítulo anterior, Juan será el usuario mediante el cual se realizarán las pruebas. Para esto, se decidió que dentro del sistema se contará con tres productos (ACMAS, SIREFI y SIRMA), y que Juan estará asignado a dos de ellos (ACMAS y SIREFI). Además se determinó que un proyecto en cada uno de estos productos también estará asignado a Juan. El proyecto que Juan deberá atender durante las pruebas, es un reporte de un error en el sistema SIREFI. Este reporte se refiere a un error en el cálculo de impuestos en los reportes de honorarios. Como conocimientos requeridos para trabajar con este módulo se han definido: manejo de los lenguajes SQL, C, y PL/SQL; de las herramientas de desarrollo de Oracle, específicamente Developer Forms y Developer Reports; conocimientos sobre contabilidad y cálculo de impuestos; y sobre el proceso de pago de honorarios. También se ha definido que Susana, un miembro del personal, es una experta en el manejo de las herramientas de Oracle; que Felipe, otro miembro del personal, tiene nivel medio de conocimiento sobre contabilidad e impuestos; y que Sofía, el cliente que hizo el reporte, conoce cómo se realiza el proceso de pago de honorarios. Dentro de la base de conocimientos se han agregado referencias a varios documentos, entre ellos, un manual

técnico del producto SIREFI, en el cual se describe el funcionamiento y estructura interna del sistema y la base de datos del mismo, así como los procesos a los que da soporte; además de la documentación electrónica de las herramientas de Oracle. Por último, también se han agregado tres reportes de errores previos sobre el sistema SIREFI, donde uno de ellos se refiere a un error anterior en el módulo de reportes de honorarios.

Los pasos a seguir durante la realización de las pruebas han sido agrupados en cuatro secciones. A continuación se presenta la secuencia de los mismos.

1. Probar la inicialización del prototipo.
 - a) Iniciar el prototipo y proporcionar un nombre o clave de usuario incorrecto.
 - b) Volver a iniciar el prototipo y proporcionar el nombre y clave de usuario de Juan.
2. Probar la solicitud de atención de proyectos.
 - a) Elegir el proyecto dentro del producto SIREFI.
 - b) Solicitar atender el proyecto seleccionado y esperar a que se despliegue el mensaje con el total de fuentes encontradas.
3. Probar la recuperación de las fuentes de conocimiento encontradas, y los datos de las mismas.
 - a) Solicitar que se muestren las fuentes de conocimiento encontradas.
 - b) Revisar que la lista de conocimientos buscados concuerde con los esperados.
 - c) Revisar los reportes de error que se hayan encontrado.
 - d) Solicitar que se muestre la información del reporte de error previo del módulo de reportes de honorarios.
 - e) Regresar a la ventana de fuentes de conocimiento y revisar las personas que se hayan encontrado.
4. Probar el almacenamiento de la solución dada al error reportado.
 - a) Ir a la ventana del reporte de error, y solicitar asignar solución.
 - b) Capturar los datos de la solución al error, así como archivos modificados, y aceptar los cambios.
 - c) Verificar que se haya actualizado el reporte de error, y que se haya agregado como una nueva fuente de conocimiento dentro de la base de conocimientos.

Una vez definida la mecánica de las pruebas a realizar, así como la manera en que estas han sido preparadas, en seguida se presenta la descripción del proceso de pruebas realizado.

VI.3 Descripción del proceso de pruebas

Esta sección presenta el proceso de pruebas realizado. Este proceso se ha dividido siguiendo la agrupación de la secuencia de pasos definida anteriormente. La descripción del proceso se hace de la siguiente manera: primeramente se menciona la sección correspondiente a los pasos que se están realizando, posteriormente se presenta cada uno de los pasos de prueba, así como el resultado esperado, seguido de una descripción del resultado obtenido. Principalmente se muestran los datos resultantes en la interfaz de usuario, así como los mensajes enviados por los agentes. La definición del resultado esperado se ha establecido tomando en cuenta la funcionalidad esperada del sistema, mostrada en la guía de pruebas descrita al inicio de este capítulo, así como los datos proporcionados al prototipo durante la preparación, presentada anteriormente. Dentro de los resultados esperados, se consideran principalmente aquellos que son visibles para el usuario.

VI.3.1 Inicialización del prototipo

Este conjunto de pruebas están encaminadas a validar que el prototipo realiza correctamente las funciones definidas durante su inicialización. Al iniciar el prototipo, el sistema debe mostrar una ventana para solicitar el nombre y clave de usuario. Esta prueba considera dos variantes, cuando el nombre o clave de usuario proporcionados son incorrectos, y cuando ambos son correctos.

Prueba: el nombre o clave de usuario son incorrectos.

Resultado esperado: el prototipo debe indicar al usuario que los datos son incorrectos, y terminar su ejecución sin permitir la inicialización de los agentes.

Esta prueba resultó satisfactoria desde el punto de vista de que el prototipo terminó su ejecución sin permitir la inicialización de los agentes, tal y como se esperaba. Sin embargo, el mensaje desplegado para indicar esto, no es muy claro para el usuario, además se pierde dentro del resto de la información que es desplegada, como se muestra en la figura 47.

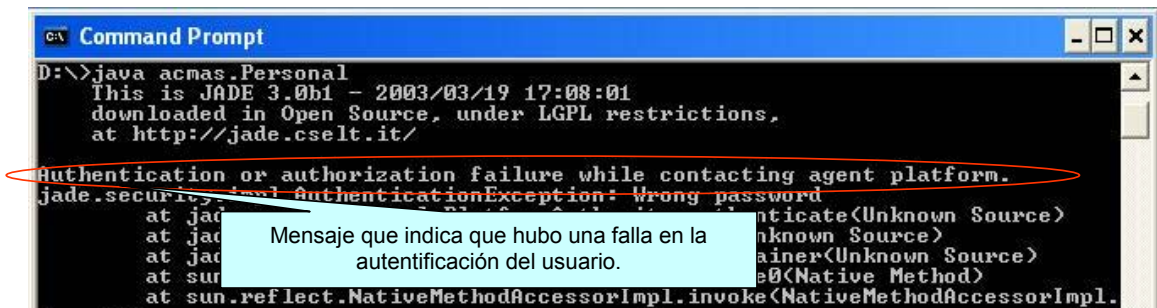


Figura 47. Mensaje de falla en la autenticación del usuario.

Prueba: el nombre y clave de usuario son correctos.

Resultado esperado: el prototipo debe iniciar los agentes que darán soporte al usuario, y presentar la interfaz gráfica de usuario. A la par, se debe realizar el registro del agente de personal en el directorio de agentes del prototipo. Posteriormente, se deben mostrar los dos proyectos a los que está asignado el usuario Juan. Estos proyectos estarán agrupados por el producto al que pertenecen. Los productos serán: ACMAS y SIREFI.

Durante esta prueba los resultados obtenidos fueron conforme a los esperados. En la figura 48 se observan los resultados en las interfaces de usuario. Se puede ver que la interfaz de usuario del miembro de personal Juan ha sido desplegada, y en ella se muestran los dos proyectos a los que Juan está asignado, uno del producto SIREFI, y otro de ACMAS. A su vez, es posible ver en la interfaz de usuario del agente de directorio, que el agente Juan ha sido registrado.

Por otro lado, la figura 49 muestra el intercambio de mensajes que se dio entre los distintos agentes. Como es posible observar, se intercambiaron cuatro tipos de mensajes, el primero fue para solicitar al agente de directorio que registrara al agente Juan; el segundo para

indicar tanto al agente de personal, como a los de producto y proyecto, que el agente juan había sido registrado; el tercer tipo de mensaje, fue para informar al agente juan que el proyecto ACMAS_pf6684febc8 requiere ser atendido, mientras que el cuarto mensaje fue similar al anterior, solo que el proyecto a ser atendido fue SIREFI_pf66c21f130.

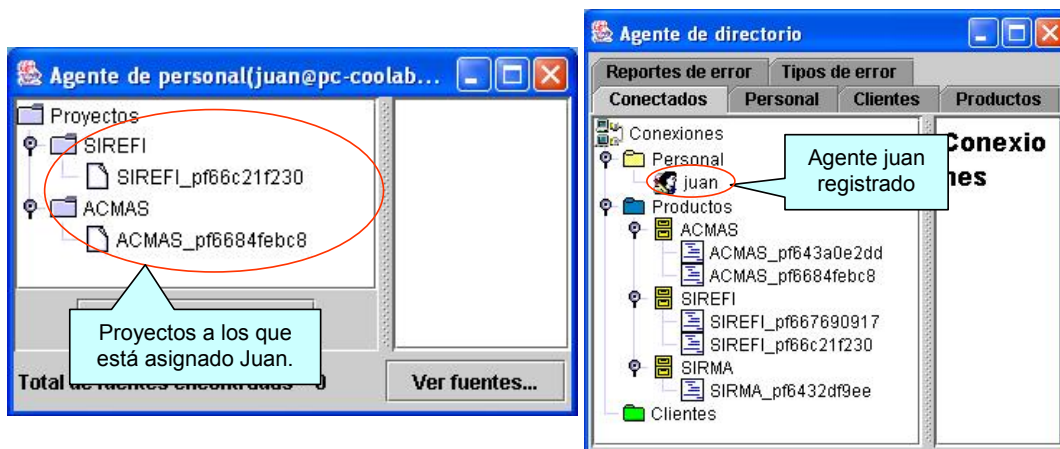


Figura 48. Muestra el resultado de la inicialización del prototipo.

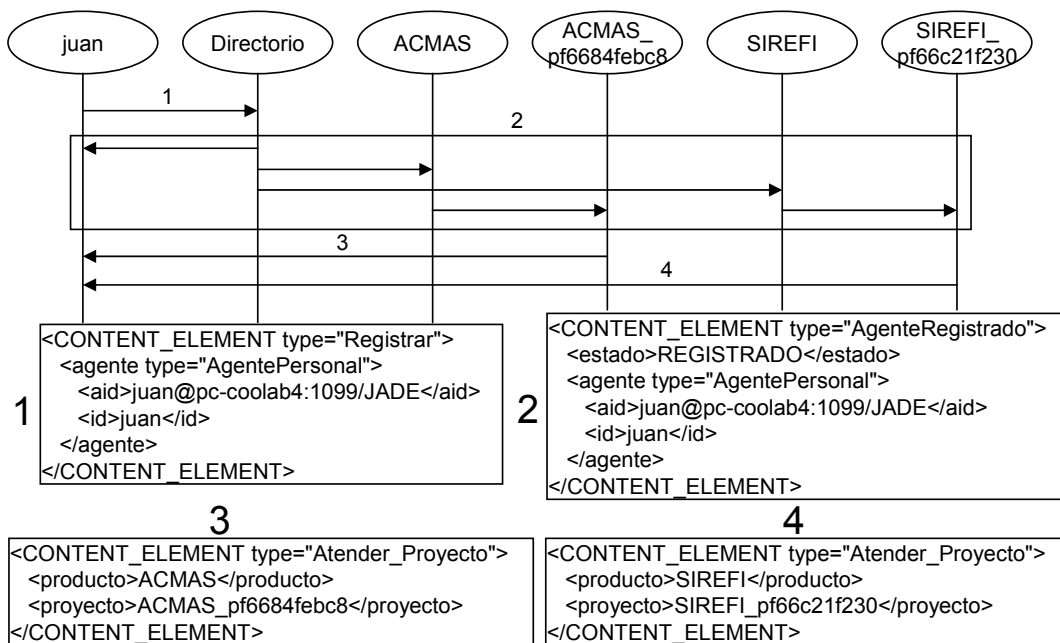


Figura 49. Mensajes enviados por los agentes durante la inicialización del prototipo.

Durante estas pruebas se detectó que el orden en el que se presentan los productos y proyectos en la lista de proyectos, varía dependiendo del orden en que se reciben los mensajes provenientes de éstos últimos. Si bien éste pudiera no ser un problema, valdría la pena considerar que estuvieran ordenados con base en algún criterio específico, como podría ser el nombre del producto o proyecto. Otro punto detectado en estas pruebas, fue la dificultad para identificar un proyecto específico por medio del código que se le dio al mismo. Como es posible observar en la figura 48, en la lista de proyectos se muestra el código que se les ha asignado, con el cual es difícil para el usuario identificar un proyecto específico. Este problema pudiera ser más grave en la medida que se tenga un mayor número de proyectos dentro de un mismo producto.

Una vez probada la inicialización del prototipo, cuyos resultados acabamos de presentar, se probó la atención de proyectos, lo cual se muestra a continuación.

VI.3.2 Solicitud de atención de proyectos

Mediante este conjunto de pruebas se busca validar que el prototipo permite ver los datos de los proyectos, así como los reportes de error que dieron origen a dichos proyectos. A la par, se valida que el prototipo realiza la búsqueda de fuentes de conocimiento e informa el total de las que fueron encontradas.

Prueba: elegir el proyecto dentro del producto SIREFI.

Resultado esperado: el prototipo debe mostrar los datos del proyecto en el área de despliegue de datos.

El resultado de esta prueba fue conforme a lo esperado. Los datos del proyecto que fue seleccionado se mostraron en el área de despliegue de datos de la interfaz del agente de personal, como se ilustra en la figura 50.

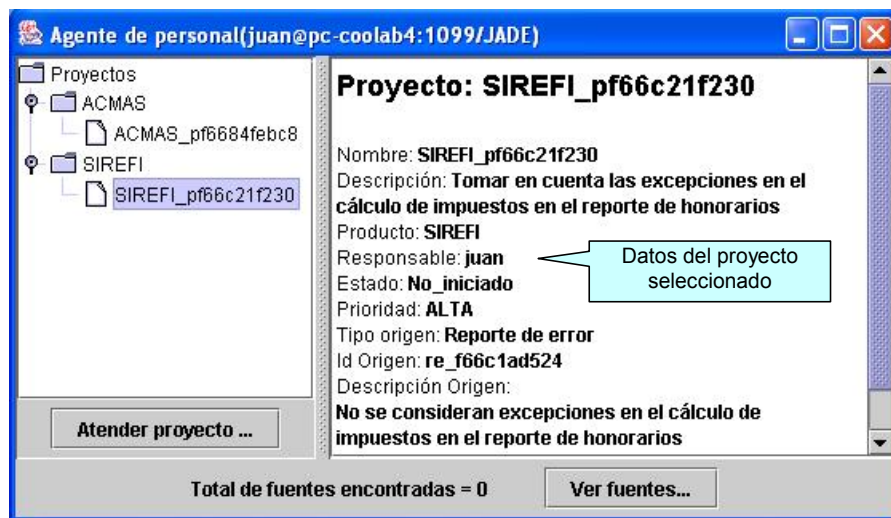


Figura 50. Muestra el resultado de la selección de un proyecto de la lista de proyectos de la interfaz de usuario del agente de personal.

Prueba: solicitar atender el proyecto seleccionado y esperar a que se despliegue el mensaje con el total de fuentes encontradas.

Resultado esperado: el prototipo debe abrir una ventana donde se muestren los datos del reporte de error que dio origen al proyecto del producto SIREFI. Posteriormente, se debe mostrar una ventana de diálogo indicando el total de fuentes de conocimiento encontradas, las cuales deben ser 8.

Esta prueba también resultó conforme a lo esperado, el prototipo desplegó la ventana donde se muestran los datos del reporte de error. Posteriormente desplegó un mensaje informando del total de fuentes encontradas, tal y como se ilustra en la figura 51. Como se muestra en la figura, el total de fuentes encontradas por el agente juan_km fueron 8, tal y como se esperaba.

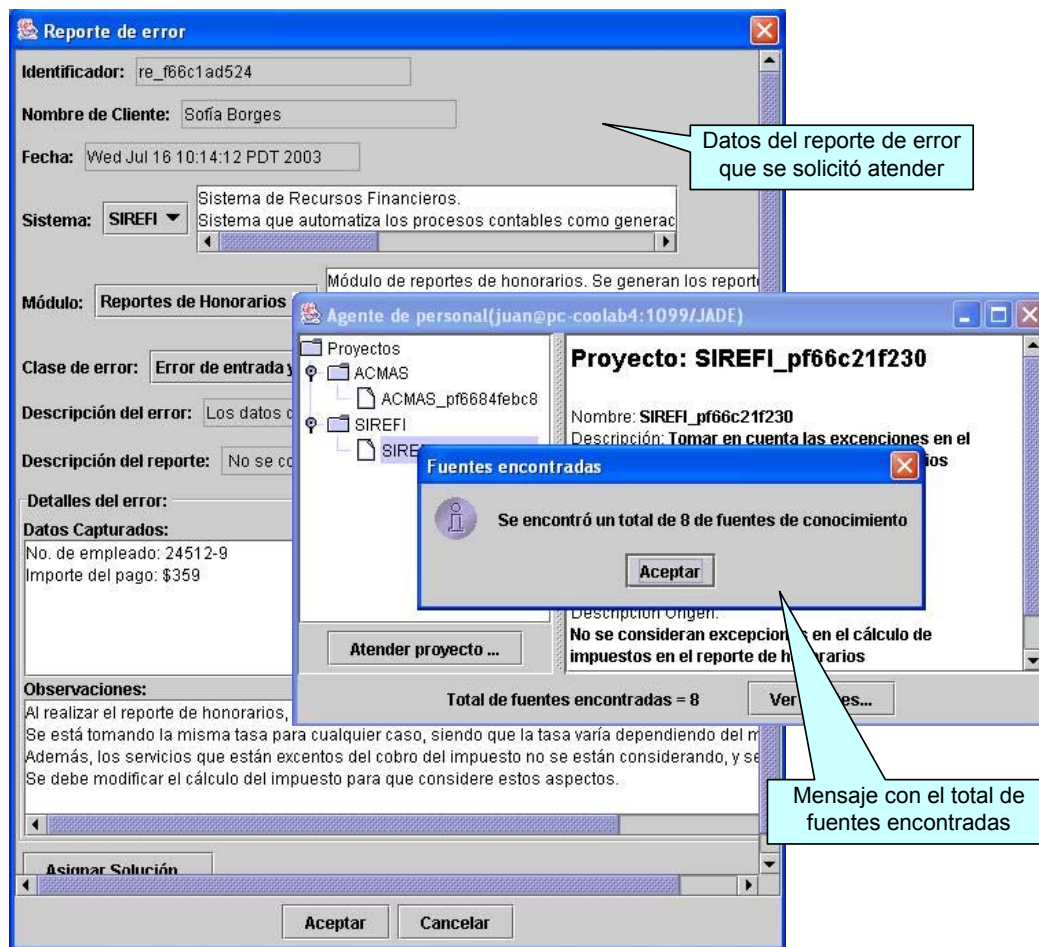


Figura 51. Muestra el resultado de solicitar atender un proyecto en la interfaz de usuario del agente de personal.

En la figura 52 se presenta el intercambio de mensajes que se dio entre el agente juan y el agente juan_km durante esta prueba. Como es posible observar, el agente juan envió los datos del reporte de error al agente juan_km. Por simplificación, en la figura sólo se muestran los datos más relevantes del mensaje, como el sistema o producto y el módulo donde se presentó el error, así como el tipo de error. La figura también muestra el mensaje enviado por el agente juan_km para informar el total de fuentes encontradas.

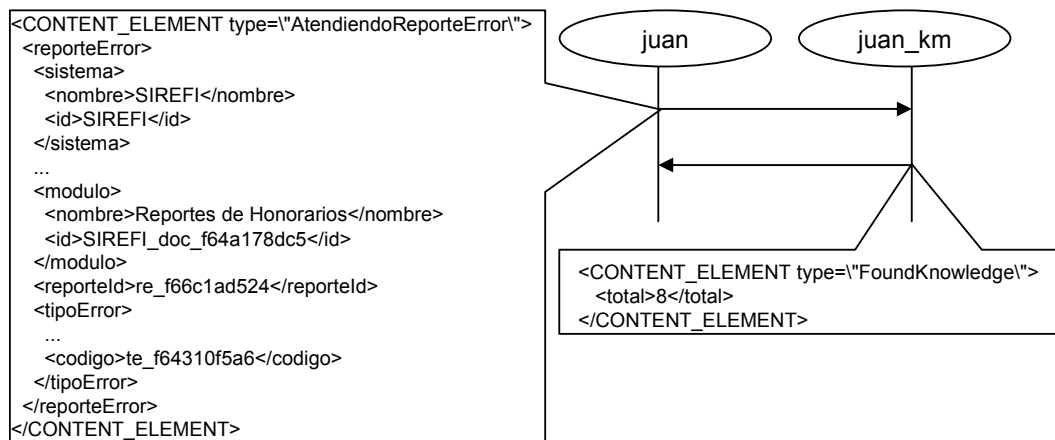


Figura 52. Muestra el intercambio de mensajes entre el agente de personal y el agente manejador de conocimiento al momento de que el IM solicita atender un proyecto.

Algunos problemas observados durante estas pruebas, fue que cuando el mensaje del total de fuentes encontradas es desplegado, interfiere con la ventana de los datos del reporte de error, lo que puede causar molestia si un usuario se encuentra trabajando con ella. Al momento de desplegarse el mensaje, tanto el mensaje como la ventana de interfaz de usuario del agente de personal se posicionan sobre la ventana con los datos del reporte de error, como lo muestra la figura 51. Otros detalles son que el espacio dado a los campos donde se muestra, tanto la descripción del reporte de error, como del tipo de error, en ocasiones resulta insuficiente, ya que hay descripciones que sobrepasan el espacio asignado.

Una vez que se validó que el agente `juan_km` informara del total de fuentes de conocimiento encontradas, se probó si éste desplegaba la interfaz donde estas fuentes pueden ser visualizadas, así como el despliegue de los datos de las mismas, lo que se presenta a continuación.

VI.3.3 Recuperación de las fuentes de conocimiento encontradas, y los datos de las mismas

Este conjunto de pruebas tiene el fin de permitir validar si el sistema recupera las fuentes de conocimiento esperadas. La presentación de los resultados de los cinco pasos definidos para esta sección de pruebas, se ha agrupado en dos, primeramente se verificará que el prototipo recupere todas las fuentes de conocimiento esperadas, por lo que se han agrupado los incisos a), b), c) y e) en un solo paso. Posteriormente se validará que el prototipo permita ver los datos de las fuentes de conocimiento recuperadas.

Prueba: solicitar que se muestren las fuentes de conocimiento encontradas.

Resultado esperado: se debe desplegar la ventana con las fuentes de conocimiento que fueron encontradas, donde deben aparecer un total de 8 fuentes de conocimiento, de las cuales tres serán personas: dos miembros del personal y un cliente (Felipe, Susana y Sofía respectivamente); dos documentos: uno de sistema, y otro técnico; y tres reportes de error. Esta ventana también debe contener la lista de los conceptos o temas de conocimiento que fueron buscados, los cuales deben corresponder con SIREFI; el módulo de Reportes de Honorarios; el tipo de error Salida de datos incorrecta; las herramientas Oracle, Developer Forms y Developer Reports; los lenguajes C++, SQL y PL/SQL; el proceso pago de honorarios, y las áreas contabilidad y análisis financiero.

Los resultados obtenidos de esta prueba fueron los esperados. Como se puede observar en la figura 53, se muestran las ocho fuentes de conocimiento, así como la lista de conceptos o temas de conocimiento que fueron buscados, los cuales concuerdan con los esperados.

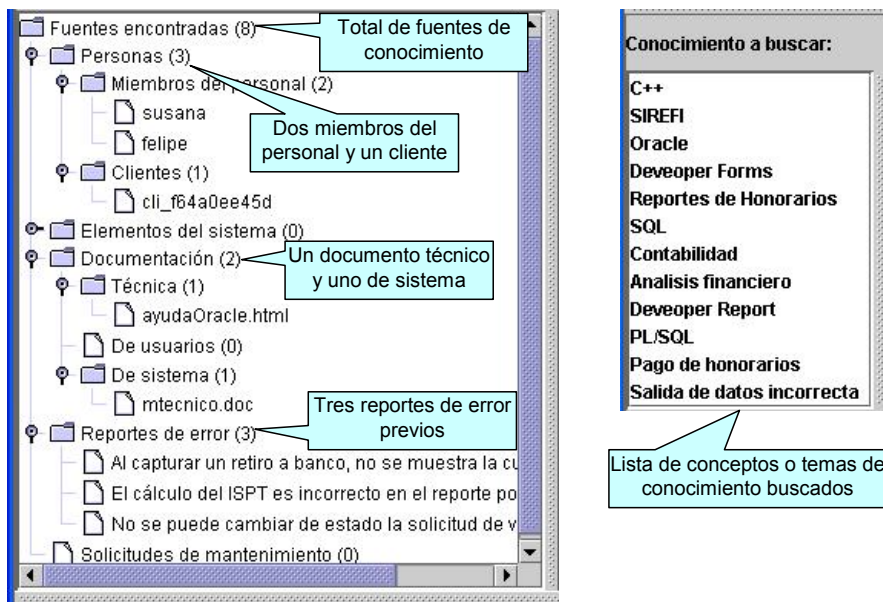


Figura 53. Muestra las fuentes de conocimiento encontradas , y los conceptos o temas de conocimiento buscados.

El principal problema encontrado durante esta prueba, es la manera en que se identifica al cliente dentro de las fuentes de conocimiento. Como se puede apreciar en la figura 53, el cliente que fue encontrado se representa por el código de cliente que se le ha asignado, lo cual no es lo más idóneo, ya que es difícil saber que cliente es. Una recomendación es manejar el nombre del cliente, en vez del código.

Prueba: solicitar que se muestre la información del reporte de error previo del módulo de reportes de honorarios.

Resultado esperado: al seleccionar la fuente de conocimiento que corresponde con el reporte de error, los datos de la misma se deben desplegar en el área de datos de la ventana de fuentes de conocimiento, estos datos indicarán en dónde se localiza el reporte, así como los temas de conocimiento con los que está asociado. Al oprimir el botón ver fuente, se debe abrir una ventana donde se muestren los datos del reporte de error.

Durante estas pruebas, el resultado visible para el usuario fue según lo esperado. Primeramente, al elegir el reporte de error previo, los datos de la fuente de conocimiento

asociada a éste se desplegaron en el área de datos de la ventana, como se muestra en la figura 54. En la figura se puede observar que el reporte de error está asociado, o tiene conocimiento sobre el sistema o producto al que pertenece el módulo donde se presentó el error, el mismo módulo, y el tipo de error, que en este caso son SIREFI, Reportes de Honorarios y Salida de datos incorrecta, respectivamente. Sin embargo, no se indica la localización del reporte de error, es recomendable indicar en qué parte de la base de datos se encuentra almacenado.

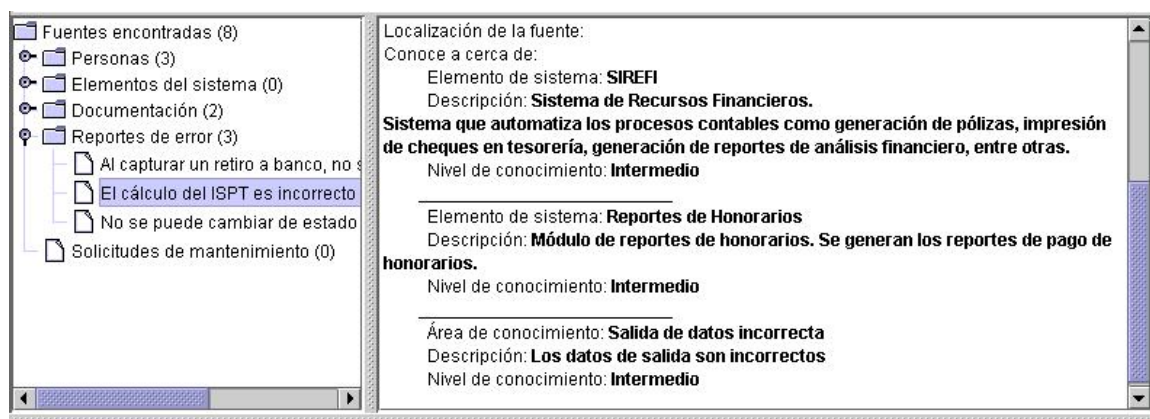


Figura 54. Muestra los datos de una fuente de conocimiento de tipo reporte de error.

Por otro lado, al solicitar el despliegue de los datos del reporte de error, el funcionamiento del prototipo fue el esperado, ya que el prototipo desplegó una ventana con los datos del reporte de error, como se muestra en la figura 55. Es posible observar que, además de los datos del reporte, se muestra la solución que se le dio, la cual, entre otros datos contiene: quien estableció la solución, cuanto tiempo se llevó, la causa del error, la solución que se dio, así como los archivos modificados, y cual fue la modificación de los mismos.

En la figura 56 se muestra el mensaje enviado por el agente manejador de conocimientos (juan_km) al agente manejador de fuentes de conocimiento (juan_ksm) al momento de solicitar que se muestren los datos del reporte de error. Este fue el mensaje que causó que los datos mostrados en la figura 55 fueran desplegados.

Reporte de error: re_f667487fee

Identificador: **re_f667487fee**

Fecha del reporte: **Tue Jul 15 11:45:59 PDT 2003**

Cliente que hizo el reporte: **Gregorio Castillo Peraza**

Sistema donde se generó el reporte: **SIREFI**

Descripción: **Sistema de Recursos Financieros.**

Sistema que automatiza los procesos contables como generación de pólizas, impresión de cheques en tesorería, generación de reportes de análisis financiero, entre otras.

Módulo donde se generó el reporte: **Reportes de Honorarios**

Descripción: **Módulo de reportes de honorarios. Se generan los reportes de pago de honorarios.**

Tipo del error: **Salida de datos incorrecta**

Código del error: **te_f64310f5a6**

Clase del error: **Error de entrada y salida de datos**

Descripción del error: **Los datos de salida son incorrectos**

Descripción del reporte: **El cálculo del ISPT es incorrecto en el reporte por honorarios**

Datos capturados:

Observaciones: **Al generar un reporte de honorarios, el cálculo total del ISPT no concuerda con el esperado.**

Solucion del error:

Solucionado por: **felipe**

Tiempo requerido: **3 horas**

Causa del error: **La función donde se realiza el cálculo del ISPT total en el reporte de honorarios, no se había actualizado para que reflejara el cambio en las disposiciones fiscales de este año.**

Solución que se dió: **Se modificó el cálculo del impuesto siguiendo las normas establecidas en el diario oficial de la federación del 9 de Enero del 2003.**

Archivos modificados: Archivo: **rep_hono.pc**

Descripción del archivo: **Programa de cálculo de honorarios.**

Modificaciones: **Semodificó la función "calcula_ISPT()" para que reflejara las nuevas**

disposiciones fiscales.

Archivos consultados: Archivo: **rep_hono.frx**

Descripción del archivo: **Forma del reporte de honorarios**

Causa de la consulta: **Se revisó que los cambios hechos en el archivo del programa no afectaran la forma de captura de datos de honorarios. También se revisó que los cambios realizados se reflejaran en la forma.**

Archivo: **rep_hono.rtf**

Descripción del archivo: **Reporte de honorarios**

Causa de la consulta: **Se verificó que los cambios no afectaran el despliegue del reporte.**

Observaciones: **Fue necesario consultar con la encargada del área de finanzas para que nos facilitara el documento donde se establecen las disposiciones fiscales que debieron ser tomadas en cuenta, así como para que nos explicara la manera en que debía realizarse el cálculo del impuesto.**

Figura 55. Muestra el despliegue de los datos del reporte de error.

Algo interesante en esta prueba, es que, como es posible observar en la figura 55, los datos presentados, pueden ser de ayuda para solucionar el reporte de error que se solicitó atender, ya que ambos errores sucedieron en el mismo módulo, están catalogados como del mismo tipo, y podría decirse que son similares debido a que ambos se refieren a un error en el cálculo de impuestos al momento de realizar el reporte de honorarios. De hecho, para solucionar el error del reporte que se está atendiendo, fue necesario modificar el mismo archivo que fue modificado al solucionar el error mostrado en la figura. Esto indica que hay casos en que el prototipo puede dar apoyo en la identificación de archivos fuente a modificar.

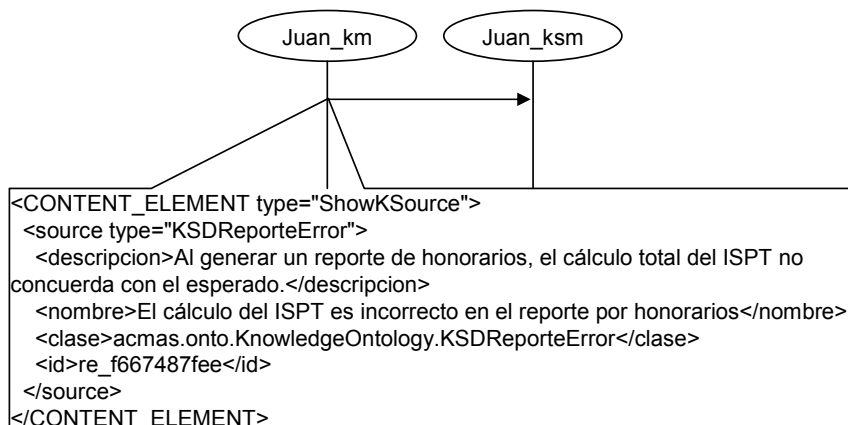


Figura 56. Mensaje enviado por el agente `juan_km` al agente `juan_ksm` para solicitarle que despliegue los datos de una fuente de conocimiento.

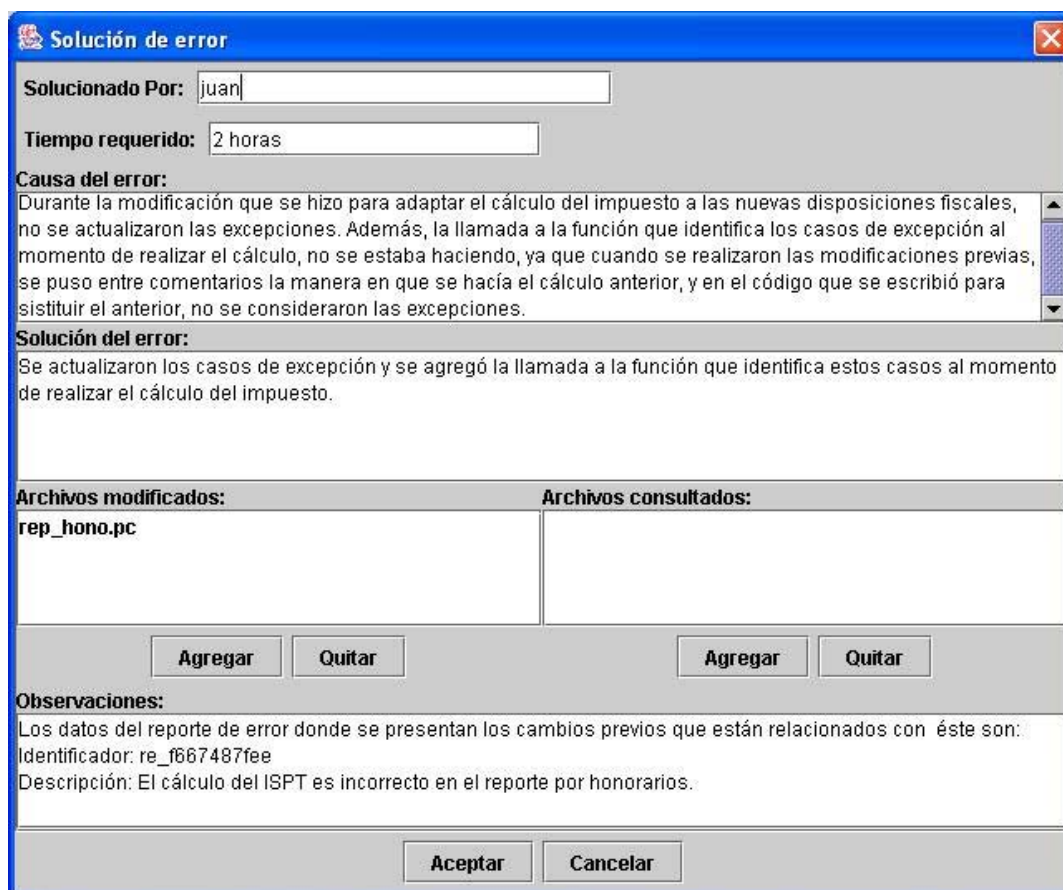
Ya que han sido realizados los pasos para probar la visualización de las fuentes de conocimiento encontradas, así como sus datos, lo siguiente es probar el almacenamiento de la solución dada al error reportado, lo cual se presenta a continuación.

VI.3.4 Almacenamiento de la solución dada al error reportado

Esta prueba tiene el objetivo de validar que el prototipo actualiza el reporte de error al guardar la solución que se le dio. Además, permite validar que se agregue una nueva fuente en la base de conocimiento, la cual es una referencia al reporte de error. Esta fuente debe indicar la localización del reporte, así como los conceptos o temas de conocimiento con los cuales está asociado, en particular, el producto, módulo y tipo de error del reporte. Debido a que los tres pasos descritos para la realización de esta prueba están relacionados, se han agrupado como uno solo.

Prueba: ir a la ventana del reporte de error y solicitar asignar solución; capturar los datos de la solución al error, así como archivos modificados, y aceptar los cambios; finalmente verificar que el reporte de error esté actualizado, y que se haya agregado una nueva fuente de conocimiento, que haga referencia al reporte, dentro de la base de conocimientos.

Resultado esperado: el prototipo debe desplegar la ventana para la captura de los datos de la solución. Una vez capturados los datos, al momento de aceptar los cambios, se cerrará la ventana de captura de los datos de la solución, quedando la ventana con los datos del reporte de error. Posteriormente, al oprimir el botón de aceptar en ésta última, el prototipo debe cerrar la ventana y actualizar los datos del reporte en la base de datos, agregando los datos de la solución, a la par que crea una nueva fuente de conocimiento relacionada con el reporte de error.



The screenshot shows a window titled "Solución de error" with the following content:

- Solucionado Por:** Input field containing "juan".
- Tiempo requerido:** Input field containing "2 horas".
- Causa del error:** Text area containing: "Durante la modificación que se hizo para adaptar el cálculo del impuesto a las nuevas disposiciones fiscales, no se actualizaron las excepciones. Además, la llamada a la función que identifica los casos de excepción al momento de realizar el cálculo, no se estaba haciendo, ya que cuando se realizaron las modificaciones previas, se puso entre comentarios la manera en que se hacía el cálculo anterior, y en el código que se escribió para sustituir el anterior, no se consideraron las excepciones."
- Solución del error:** Text area containing: "Se actualizaron los casos de excepción y se agregó la llamada a la función que identifica estos casos al momento de realizar el cálculo del impuesto."
- Archivos modificados:** List containing "rep_hono.pc".
- Archivos consultados:** Empty list.
- Buttons:** "Agregar" and "Quitar" buttons are present under both "Archivos modificados" and "Archivos consultados".
- Observaciones:** Text area containing: "Los datos del reporte de error donde se presentan los cambios previos que están relacionados con éste son:
Identificador: re_f667487fee
Descripción: El cálculo del ISPT es incorrecto en el reporte por honorarios."
- Bottom Buttons:** "Aceptar" and "Cancelar" buttons.

Figura 57. Muestra los datos de la solución antes de guardarlos.

Durante estas pruebas, el prototipo funcionó según lo esperado. La figura 57 muestra los datos de la solución antes de solicitar que estos fueran guardados. En la figura se aprecia que se capturó el identificador de quien dio solución al error, el tiempo que se requirió, la causa del error, la solución que se dio, el archivo que fue modificado, así como algunas observaciones.

Una vez que los datos de la solución fueron aceptados, el reporte de error fue actualizado correctamente. La figura 58 muestra la manera en que los datos del reporte de error fueron guardados en la base de datos. Como es posible observar, se han agregado los datos de la solución que se dio al error.

```

<REPORTE_ERROR id="re_f66c1ad524">
<CLIENTE>Sofia Borges</CLIENTE> <FECHA>Wed Jul 16 10:14:12 PDT 2003</FECHA>
<SISTEMA id="SIREFI"> <NOMBRE>SIREFI</NOMBRE>
  <DESCRIPCION>Sistema de Recursos Financieros.
  Sistema que automatiza los procesos contables como generación de pólizas, impresión de cheques en tesorería, generación de reportes de análisis financiero, entre otras.</DESCRIPCION>
</SISTEMA>
<MODULO id="SIREFI_doc_f64a178dc5"> <NOMBRE>Reportes de Honorarios</NOMBRE>
  <DESCRIPCION>Módulo de reportes de honorarios. Se generan los reportes de pago de honorarios.</DESCRIPCION>
</MODULO>
<TIPO_ERROR codigo="te_f64310f5a6"> <ERROR>Salida de datos incorrecta</ERROR>
  <CLASE_ERROR>Error de entrada y salida de datos</CLASE_ERROR>
  <DESCRIPCION>Los datos de salida son incorrectos</DESCRIPCION>
</TIPO_ERROR>
<DESCRIPCION> No se consideran excepciones en el cálculo de impuestos en el reporte de honorarios </DESCRIPCION>
<DATOS_CAPTURADOS>No. de empleado: 24512-9  Importe del pago: $359</DATOS_CAPTURADOS>
<OBSERVACIONES>Al realizar el reporte de honorarios, no se están considerando las excepciones al momento de calcular el impuesto. Se está tomando la misma tasa para cualquier caso, siendo que la tasa varía dependiendo del monto del pago. Además, los servicios que están exentos del cobro del impuesto no se están considerando, y se les aplica la tasa al igual que al resto. Se debe modificar el cálculo del impuesto para que considere estos aspectos.</OBSERVACIONES>
<SOLUCION_ERROR> <SOLUCIONADO_POR>juan</SOLUCIONADO_POR>
  <TIEMPO_REQUERIDO>2 horas</TIEMPO_REQUERIDO>
  <CAUSA_ERROR>Durante la modificación que se hizo para adaptar el cálculo del impuesto a las nuevas disposiciones fiscales, no se actualizaron las excepciones. Además, la llamada a la función que identifica los casos de excepción al momento de realizar el cálculo, no se estaba haciendo, ya que cuando se realizaron las modificaciones previas, se puso entre comentarios la manera en que se hacía el cálculo anterior, y en el código que se escribió para sustituir el anterior, no se consideraron las excepciones.</CAUSA_ERROR>
  <SOLUCION>Se actualizaron los casos de excepción y se agregó la llamada a la función que identifica estos casos al momento de realizar el cálculo del impuesto.</SOLUCION>
  <ARCHIVOS_MODIFICADOS> <ARCHIVO tipo="modificado">
    <NOMBRE>rep_hono.pc</NOMBRE>
    <DESCRIPCION>Archivo con el programa para el cálculo de impuestos en el reporte de honorarios.</DESCRIPCION>
    <CAUSA>Se modificó la función &quot;calcula_ISPT()&quot; para que hiciera la llamada a la función &quot;casosExcepcion()&quot; la cual identifica los casos de excepción al momento de calcular el impuesto. También se actualizó esta última función para que reflejara las nuevas excepciones.</CAUSA>
  </ARCHIVO> </ARCHIVOS_MODIFICADOS>
  <ARCHIVOS_CONSULTADOS/>
  <OBSERVACIONES>Los datos del reporte de error donde se presentan los cambios previos que están relacionados con éste son:
  Identificador: re_f667487fee
  Descripción: El cálculo del ISPT es incorrecto en el reporte por honorarios.</OBSERVACIONES>
</SOLUCION_ERROR></REPORTE_ERROR>

```

Figura 58. Muestra la forma en que se almacenó el reporte de error en la base de datos.

Finalmente, fue posible verificar que el prototipo efectivamente creó una nueva fuente de conocimiento asociada a dicho reporte de error, tal y como se esperaba. Los datos de la fuente de conocimiento que fue creada son mostrados en la figura 59. Se puede observar que la fuente de conocimiento creada contiene la localización del reporte de error, el cual se encuentra en una bitácora de reportes de error; y los conceptos o temas de conocimiento a los que está asociado el reporte de error, que en este caso es el producto SIREFI, el módulo Reportes de Honorarios y el tipo de error Salida de datos incorrecta.

```

<KCONCEPT id="re_f66c1ad524" clase="acmas.onto.KnowledgeOntology.KSDReporteError">
  <NOMBRE> No se consideran excepciones en el cálculo de impuestos en el reporte de honorarios </NOMBRE>
  <DESCRIPCION>Al realizar el reporte de honorarios, no se están considerando las excepciones al momento de calcular el impuesto.
  Se está tomando la misma tasa para cualquier caso, siendo que la tasa varía dependiendo del monto del pago.
  Además, los servicios que están exentos del cobro del impuesto no se están considerando, y se les aplica la tasa al igual que al resto.
  Se debe modificar el cálculo del impuesto para que considere estos aspectos.</DESCRIPCION>
  <ATRIBUTO nombre="localizacion"> <VALOR tipo="list" clase="jade.util.leap.ArrayList">
    <ELEM tipo="XMLObjectInterface" clase="acmas.onto.KnowledgeOntology.LocBD">
      <LOCALIZACION tipo="Base de datos" clase="acmas.onto.KnowledgeOntology.LocBD">
        <DIR>Bitácora de reportes de error.</DIR></LOCALIZACION> </ELEM>
      </VALOR></ATRIBUTO>
    <ATRIBUTO nombre="knowsAbout"> <VALOR tipo="list" clase="jade.util.leap.ArrayList">
      <ELEM tipo="XMLObjectInterface" clase="acmas.onto.KnowledgeOntology.Knowledge">
        <KNOWLEDGE> <NIVEL>Intermedio</NIVEL>
      <KCONCEPT id="SIREFI" clase="acmas.onto.KnowledgeOntology.KSSProducto"> <NOMBRE>SIREFI</NOMBRE>
      <DESCRIPCION>Sistema de Recursos Financieros. Sistema que automatiza los procesos contables como generación de pólizas,
      impresión de cheques en tesorería, generación de reportes de análisis financiero, entre otras.</DESCRIPCION>
    </KCONCEPT></KNOWLEDGE></ELEM>
    <ELEM tipo="XMLObjectInterface" clase="acmas.onto.KnowledgeOntology.Knowledge">
      <KNOWLEDGE> <NIVEL>Intermedio</NIVEL>
    <KCONCEPT id="SIREFI_doc_f64a178dc5" clase="acmas.onto.KnowledgeOntology.KSSModulo">
      <NOMBRE>Reportes de Honorarios</NOMBRE>
      <DESCRIPCION>Módulo de reportes de honorarios. Se generan los reportes de pago de honorarios.</DESCRIPCION>
    </KCONCEPT></KNOWLEDGE></ELEM>
    <ELEM tipo="XMLObjectInterface" clase="acmas.onto.KnowledgeOntology.Knowledge">
      <KNOWLEDGE> <NIVEL>Intermedio</NIVEL>
    <KCONCEPT id="te_f64310f5a6" clase="acmas.onto.KnowledgeOntology.KATipoError">
      <NOMBRE>Salida de datos incorrecta</NOMBRE>
      <DESCRIPCION>Los datos de salida son incorrectos</DESCRIPCION>
    </KCONCEPT></KNOWLEDGE></ELEM>
  </VALOR> </ATRIBUTO></KCONCEPT>

```

Figura 59. Muestra la manera en que es almacenada la fuente de conocimiento relacionada con el reporte de error.

Como es posible observar, durante los resultados de las pruebas mostradas en esta sección se detectaron algunas fallas en el prototipo, sin embargo, éstas no son severas, ya que la funcionalidad esperada del prototipo se está cumpliendo. Aún así, es necesario analizar más detalladamente los resultados aquí presentados, lo cual se hace a continuación.

VI.4 Discusión de los resultados de la etapa de pruebas

Las pruebas realizadas tuvieron como fin el validar que el prototipo funcionara según lo esperado, específicamente que permitiera dar seguimiento a escenarios similares al que fue definido para su diseño. Es por lo anterior que las pruebas realizadas se basaron en un escenario muy parecido.

Como ya hemos visto, durante las pruebas fue posible detectar algunas fallas. Estas fallas han sido catalogadas en tres tipos: funcionalidad, usabilidad, y falla estética.

- **Error de Funcionalidad.** Este tipo de falla indica que el problema se debe a que el prototipo no está realizando alguna función, o lo está haciendo de manera incorrecta.
- **Error de Usabilidad.** Estas fallas se refieren a problemas con la utilización del prototipo, como puede ser la dificultad para manejar algún elemento de la interfaz de usuario.
- **Error estético.** Este tipo de fallas se refieren a aspectos como el orden de los campos de captura, elementos de la interfaz que no se ven de la manera en que se espera. En términos generales, aspectos que podrían mejorar la apariencia del prototipo.

Además de estas categorías, las fallas fueron clasificadas según su severidad en alta, media y baja.

- **Alta.** La severidad del error o falla evita que el prototipo funcione correctamente, y que sea posible obtener el resultado esperado.
- **Media.** La falla puede evitar que ciertas funciones del prototipo no se realicen de manera correcta, sin embargo es posible obtener el resultado esperado.
- **Baja.** La falla no interfiere con el buen funcionamiento del prototipo.

La tabla XI muestra la lista de las fallas que fueron encontradas durante las pruebas. En ella se presenta la descripción de la falla, así como la prueba donde se presentó, la severidad que tiene y su tipo.

Para indicar la prueba en la que se presentó la falla, se ha tomado en cuenta la secuencia de pasos definida al principio de este capítulo, primeramente se proporciona el número de la sección de pasos, y posteriormente el inciso que define el paso específico donde se detectó la falla.

Tabla XI. Lista de fallas encontradas durante las pruebas.

No	Descripción	Prueba	Severidad	Tipo
1	El mensaje de usuario no válido no es claro y se pierde con el resto de la información desplegada.	1,a	Baja	Estético Funcionalidad
2	La ventana del total de fuentes encontradas se sobrepone a la ventana de datos del reporte de error.	2,b	Media	Usabilidad
3	El tamaño de los campos de descripción de reporte de error y tipo de error, en la ventana de datos del reporte de error, no es suficiente.	2,b	Media	Usabilidad Estético
4	Los productos y proyectos se ordenan según el orden en que el agente de personal recibe los mensajes enviados por los agentes de proyecto.	1,b	Media	Funcionalidad Usabilidad
5	Se usa el código del proyecto en la lista de proyectos.	1,b	Baja	Usabilidad
6	Se usa el código del cliente al desplegar las fuentes de conocimiento encontradas.	3,e	Baja	Usabilidad
7	No se muestra la localización de los reportes de error	3,d	Baja	Funcionalidad
8	Cuando se muestran algunas de las ventanas del prototipo, no es posible pasar a otras sin cerrarlas.	3,d 4,b	Media	Funcionalidad Usabilidad

Como se puede observar en la tabla, las fallas encontradas no son severas. Tomando en cuenta esto, podemos decir que en términos generales el prototipo funciona según lo esperado, ya que fue posible dar seguimiento al escenario definido para las pruebas, el cual es muy similar al que permitió obtener el diseño del prototipo.

Por otra lado, durante las pruebas fue posible observar situaciones que permiten prever que el prototipo puede apoyar en varias de las necesidades que fueron tomadas en cuenta para obtener los requerimientos que llevaron al diseño de la arquitectura. En particular, es posible observar que uno de los reportes de error que fueron recuperados al buscar las fuentes de conocimiento, contenía información que ayudó en la solución del error que se atendió durante las pruebas. Por ejemplo, el archivo modificado en este error previo, y en el que se atendió, era el mismo. En este ejemplo podemos ver que al facilitar el acceso a experiencias previas, en este caso un reporte de error, fue posible dar apoyo en la identificación de los archivos a modificar. Otro ejemplo se relaciona con la posibilidad de encontrar personas expertas a quienes consultar. En el escenario de pruebas fueron identificadas tres personas, donde una de ellas tenía conocimientos sobre el cálculo de impuestos, otra sobre el manejo de las herramientas con las cuales se iban a realizar las modificaciones, y por último, el cliente conocía sobre el proceso que debe realizarse para el

pago de honorarios, proceso relacionado con la funcionalidad del módulo que se iba a modificar.

Ya para finalizar, hemos identificado la causa de los errores detectados durante la etapa de pruebas, y hemos propuesto su posible solución. Esto se presenta a continuación.

VI.4.1 Solución de las fallas detectadas durante la etapa de pruebas

En esta sección se presentan la posible solución y la causa de las fallas encontradas. A continuación se listan las fallas en el orden en que aparecen en la tabla XI, seguida cada una de la causa y la posible solución.

1.- El mensaje de usuario no válido no es claro y se pierde con el resto de la información desplegada.

Causa: este problema se debe a que la autenticación del usuario se está haciendo directamente por la plataforma de agentes (JADE), la cual envía una excepción al momento de detectar que los datos del usuario no son válidos. Esta excepción no es transferida a la aplicación que implementa el mecanismo de validación, sino que es desplegada directamente a la consola, como se mostró en la figura 47.

Posible solución: La solución de esta falla puede que no sea sencilla, ya que al parecer es necesario modificar el código del mecanismo de autenticación utilizado por JADE, con el fin de permitir que la excepción enviada, cuando se detecta que el usuario no es válido o no tiene autorización, sea tratada por el prototipo. Ya que el código de JADE y el del mecanismo de autenticación que éste implementa es abierto, esta solución es posible, sin embargo, es necesario analizar el impacto que podría tener este cambio en el funcionamiento de la plataforma.

2.- La ventana del total de fuentes encontradas se sobrepone a la ventana de datos de reporte de error.

Causa: debido a que ambas ventanas son abiertas por el agente de personal, al momento en que éste abre la ventana con el mensaje del total de fuentes encontradas, se activa la ventana de la interfaz de usuario del mismo agente, lo que hace que se sobreponga a la ventana con los datos del reporte.

Posible solución: es necesario buscar un mecanismo que no interfiera tanto con el trabajo del usuario, pero a la vez que informe al usuario del total de fuentes encontradas en el momento en que el agente reciba este dato. Una alternativa podrían ser los mecanismos que utilizan algunos sistemas de mensajería para indicar que alguien acaba de ingresar y está disponible para ser contactado.

3.- El tamaño de los campos de descripción de reporte de error y tipo de error, en la ventana de datos del reporte de error, no es suficiente.

Causa: se ha establecido un campo de tamaño fijo para desplegar estas descripciones.

Posible solución: cambiar los campos para que puedan ser de tamaño variable, por ejemplo, agregando una barra de desplazamiento en caso de que la descripción sobrepase el espacio de despliegue.

4.- Los productos y proyectos se ordenan según el orden en que el agente de persona recibe los mensajes enviados por los agentes de proyecto.

Causa: los proyectos se agregan conforme se reciben los mensajes enviados por los agentes correspondientes.

Posible solución: definir un criterio con base en el cual ordenar tanto los productos, como los proyectos, y utilizar este criterio para insertar el proyecto en el lugar que le corresponda.

5.- Se usa el código del proyecto en la lista de proyectos.

Causa: se usa el código del proyecto como identificador del mismo dentro de la lista.

Posible solución: utilizar la descripción del proyecto en vez del código.

6.- Se usa el código del cliente al desplegar las fuentes de conocimiento encontradas.

Causa: similar al anterior.

Posible solución: utilizar el nombre del cliente en vez del código.

7.- No se muestra la localización de los reportes de error

Causa: al mostrar la localización de las fuentes de conocimiento, no se están tomando en cuenta las localizaciones de tipo base de datos, el cual es el tipo de localización de los reportes de error.

Posible solución: agregar este tipo de localización al momento de desplegar los datos de las fuentes de conocimiento.

8.- Cuando se muestran algunas de las ventanas del prototipo, no es posible pasar a otras sin cerrarlas.

Causa: algunas de las ventanas se han declarado como de tipo “modal”, es decir, que no se puede pasar a otra ventana hasta que esta esté cerrada.

Posible solución: quitar la declaración de tipo modal de estas ventanas.

VI.5 Resumen

En este capítulo se presentaron las pruebas realizadas para validar la funcionalidad del prototipo desarrollado. Estas pruebas se definieron siguiendo un escenario similar al que dio paso al diseño del prototipo. Fue posible observar que el prototipo permite dar seguimiento a escenarios similares al definido para su diseño, que era lo que se buscaba validar. Si bien se identificaron algunas fallas durante la realización de las pruebas, estas fallas no son severas, y no comprometen en gran medida la funcionalidad esperada del prototipo.

Como se menciona en este capítulo, las pruebas fueron realizadas tomando en consideración escenarios y datos reales obtenidos de uno de los grupos donde se realizaron los casos de estudio. Esto nos lleva a considerar que un sistema con funciones similares a las del prototipo puede ser útil en escenarios reales, sin embargo, todavía hace falta un análisis más riguroso para validar que tan útil puede ser en un ambiente real de trabajo. Aún

así, una primera vista de los resultados obtenidos durante estas pruebas, muestran que un sistema similar al desarrollado podría permitir dar apoyo en algunas de las necesidades que dieron paso a los requerimientos definidos para una herramienta de administración del conocimiento en el mantenimiento del software, por ejemplo, en la identificación de archivos a modificar mediante la captura y recuperación de experiencias previas, o en la identificación de personas expertas en ciertos temas.

En el capítulo siguiente se describen las conclusiones obtenidas con la realización del presente trabajo de tesis, así como el trabajo futuro que puede realizarse para dar soporte a la solución de los problemas de pérdida y desaprovechamiento del conocimiento en el mantenimiento del software. En particular se presentan algunas líneas de trabajo que podrían complementar los resultados obtenidos en esta tesis.

Capítulo VII. Conclusiones, Aportaciones y Trabajo Futuro

VII.1 Conclusiones

En este trabajo establecimos la importancia de dar soporte a los problemas que se presentan en el mantenimiento del software, ya que es la etapa que consume la mayor parte del tiempo y recursos en las organizaciones de desarrollo de software. Aun cuando los problemas del mantenimiento del software han sido estudiados desde muchas perspectivas, se observó la necesidad del desarrollo de herramientas orientadas a disminuir los problemas de pérdida y desaprovechamiento del conocimiento existente en este proceso.

Lo anterior dio la pauta para aplicar la administración del conocimiento como soporte a los problemas antes mencionados. Se observó que los estudios sobre administración del conocimiento que se han realizado dentro de la ingeniería del software se han orientado a la reutilización de experiencias para mejorar el rendimiento de los desarrolladores, a la vez que se aumenta la calidad de los productos y se reducen los costos. Se estableció cómo algunos de estos estudios se han orientado en la utilización de agentes de software para reducir la carga de trabajo de los usuarios.

Realizamos una propuesta tecnológica basada en agentes de software que permita dar soporte a los problemas relacionados con el conocimiento en el mantenimiento del software. Los requerimientos para el desarrollo de esta propuesta se obtuvieron de dos casos de estudio que se realizaron en dos grupos dedicados al mantenimiento de software. De éstos se extrajeron diversos escenarios que permitieron observar la manera en que la administración del conocimiento puede ayudar a reducir algunos de los problemas que se les presentan a los encargados del mantenimiento del software durante su trabajo cotidiano.

Con base en los requerimientos definidos, la propuesta tecnológica derivó en el diseño de una arquitectura basada en agentes cuyo objetivo es servir como la base para el desarrollo de sistemas de soporte a la administración del conocimiento en el mantenimiento del software. Para validar la viabilidad del desarrollo de sistemas basados en dicha arquitectura, se desarrolló un prototipo, el cual se diseñó tomando como base un escenario de uso, similar a los escenarios que dieron paso a los requerimientos definidos para una herramienta de administración del conocimiento en el mantenimiento del software, y posteriormente al diseño de la arquitectura.

Para validar que el prototipo permite dar soporte al tipo de escenarios definidos para la obtención de los requerimientos de diseño del mismo, se realizó un conjunto de pruebas basadas en un escenario similar, en las cuales se utilizaron datos reales, obtenidos de uno de los grupos donde se realizaron los casos de estudio. Durante la evaluación del prototipo, se observó que efectivamente permite dar seguimiento al tipo de escenarios que dieron paso a la obtención de los requerimientos para su diseño.

Al analizar los resultados de la evaluación del prototipo desarrollado, vemos que es posible que la arquitectura de agentes propuesta sirva como base para el desarrollo de sistemas de soporte a la administración del conocimiento en el mantenimiento del software. Además, analizando el tipo de escenarios que pueden ser seguidos mediante el prototipo, vemos que también es posible que una herramienta de este tipo permita dar soporte a algunas de las necesidades identificadas en los escenarios definidos durante los casos de estudio. Sin embargo, es necesario hacer estudios más exhaustivos para poder determinar que tan útil podría ser una herramienta de este tipo dentro de un ambiente real de trabajo.

Tomando en cuenta todo lo anterior, consideramos que los objetivos que guiaron la realización de este trabajo fueron cubiertos, ya que fue posible dar respuesta a las preguntas establecidas al inicio de esta tesis. Primeramente, fue posible identificar distintos escenarios que muestran diversas maneras en que la administración del conocimiento puede ayudar en la solución de algunos de los problemas que se les presentan a los grupos del

mantenimiento de software. Finalmente, la arquitectura de agentes de software propuesta, así como el prototipo desarrollado con base en la misma, hacen ver que es posible el desarrollo de herramientas que permitan dar soporte al tipo de escenarios arriba descritos.

Durante la realización de este trabajo de tesis se obtuvieron algunas aportaciones importantes, pero también se han identificado necesidades que deben ser tomadas en cuenta en estudios posteriores. A continuación se describen las aportaciones obtenidas de este trabajo, seguidas de una lista de algunos logros adicionales, para finalizar con algunos aspectos considerados para trabajos futuros.

VII.2 Aportaciones

El objetivo principal de la realización de esta tesis, fue el analizar cómo la administración del conocimiento puede ayudar a dar soporte a algunos de los problemas presentes en el mantenimiento del software. Es en este sentido que la principal aportación de este trabajo fue el establecimiento de elementos, teóricos y tecnológicos, que pueden ser utilizados para el desarrollo de herramientas para dar soporte a la administración del conocimiento dentro de los grupos de mantenimiento de software, con el fin de reducir los problemas de pérdida y desaprovechamiento del conocimiento que se da dentro de éstos. En base al establecimiento de los elementos antes mencionados, es que se realizaron las siguientes aportaciones relevantes.

Se desarrolló un modelo para catalogar los tipos de conocimiento, así como las fuentes del mismo, existentes en los grupos de mantenimiento del software. Este modelo facilitó la obtención de los datos durante los casos de estudio realizados, a la par, fue posible constatar que el modelo se asemeja a la realidad existente en ambos grupos estudiados. Este modelo también sirvió para el diseño de la estructura de la base de conocimientos que se desarrolló para la implementación del prototipo.

Se realizaron dos casos de estudio en dos grupos de mantenimiento del software, durante los cuales se identificaron aspectos que deben ser tomados en cuenta para dar soporte a los problemas de pérdida y desaprovechamiento del conocimiento existentes dentro de los grupos de mantenimiento de software. Estos casos de estudio permitieron identificar algunos escenarios que muestran la manera en que la administración del conocimiento puede ayudar a facilitar el trabajo de los encargados del mantenimiento. De estos escenarios se obtuvo una serie de requerimientos que deben ser cubiertos por sistemas que busquen dar soporte a la administración del conocimiento dentro del mantenimiento del software.

Se diseñó una arquitectura basada en agentes de software que tiene como fin el ser la base para el desarrollo de sistemas de administración del conocimiento en el mantenimiento del software. Esta arquitectura fue validada mediante la implementación de un prototipo basado en la misma. Este prototipo puede ser de ayuda para la obtención de requerimientos más detallados que permitan refinar la arquitectura, así como desarrollar un sistema que de soporte a la administración del conocimiento en el mantenimiento del software de una manera más completa y robusta.

VII.3 Logros adicionales

- Reporte técnico de uno de los casos de estudio.
Rodríguez Elias, Oscar Mario y Ana Isabel Martínez García. 2003. “Caso de estudio: Mantenimiento del Software en el Departamento de Informática del CICESE”. CICESE. Reporte Técnico, en revisión.
- Artículo aceptado en el taller de Ingeniería del Software en el 4to. Encuentro internacional de ciencias de la computación.
Rodríguez, Oscar M., Ana I. Martínez, Jesús Favela y Aurora Vizcaino. 2003. “Administración de Conocimiento como Soporte al Mantenimiento de Software”. 4to. Encuentro Internacional de Ciencias de la Computación (ENC’2003).

VII.4 Trabajo futuro

El presente trabajo de tesis permitió dar cuenta de la gran cantidad de aristas que pueden ser tomadas cuando se trata de dar soporte a los problemas del mantenimiento del software, así como también, cuando se trata de aplicar la administración del conocimiento dentro de alguna organización, sea cual fuere su giro. De esta diversidad de posibilidades, hemos tomado en cuenta algunas propuestas que nos parecen importantes para continuar con su estudio. Los puntos que aquí se consideran son de diversa índole, algunos con orientación técnica, mientras que otros están más orientados a trabajo de investigación. A continuación se listan las propuestas para trabajo futuro.

- El prototipo implementado sólo tuvo el objetivo de validar la posibilidad del desarrollo de sistemas con base en la arquitectura de agentes diseñada, es por esto que no se consideró la posibilidad de utilizarlo en un ambiente de trabajo real. Por lo tanto, proponemos que el prototipo sea rediseñado, o que se desarrolle un sistema más completo y robusto, con el fin de que sea implementado en un ambiente de trabajo real, y de esta manera, poder medir la utilidad y beneficios que pudiera aportar la administración del conocimiento a los grupos de mantenimiento del software.
- En la implementación del prototipo no se consideraron mecanismos para la creación de conocimiento, pensamos que es importante hacer un estudio para identificar los mejores mecanismos para crear conocimiento durante las actividades realizadas por los ingenieros de mantenimiento.
- El mecanismo de búsqueda y recuperación de conocimiento implementado por el prototipo es muy simple. Consideramos importante desarrollar un mecanismo que permita asignar pesos o grados de relevancia a las distintas fuentes de conocimiento, a fin de que se recuperen las que sean más significativas para la tarea a realizar.
- Otro aspecto que debe ser considerado dentro del prototipo, son los mecanismos de comunicación entre los distintos miembros del personal, es importante que un sistema de administración del conocimiento no solo encuentre personas con ciertos conocimientos, sino que también facilite los medios para consultarla.

- El diseño de la ontología mediante la cual se desarrollo la base de conocimientos, se basó en el modelo de tipos y fuentes de conocimiento propuesto en esta tesis, sin embargo, consideramos que esta ontología puede extenderse todavía más, por lo que proponemos un estudio más extenso para identificar los elementos del mantenimiento del software que deban ser tomados en cuenta dentro de la base de conocimientos.
- Desarrollar un mecanismo de control de acceso y manejo de permisos para la utilización de recursos, que considere a las personas como tales, así como el tipo de actor del que se trate y los roles que juegan dentro del proceso de mantenimiento del software.
- Estudiar el efecto que pueden tener en la arquitectura de agentes propuesta, los distintos roles que pueden jugar los miembros del personal de mantenimiento. Esto con el fin de identificar los aspectos que deben ser tomados en cuenta para poder dar el soporte necesario al personal de mantenimiento, independientemente del rol que juegue cada uno de sus miembros.
- Uno de los aspectos más relevantes durante el mantenimiento del software, es el control de cambios y versiones. Es por ello que proponemos que se estudie la manera de considerar estos elementos dentro de la arquitectura de agentes propuesta.
- Las herramientas de reingeniería e ingeniería inversa son mecanismos que les permiten a los ingenieros de mantenimiento obtener un mejor entendimiento de los sistemas con los que deben trabajar. El código fuente es una de las principales fuentes de conocimiento con las que éstos pueden contar. Es por ello que consideramos importante tomar en cuenta mecanismos de este tipo, con el fin de proveer a los ingenieros de software de la mayor cantidad de conocimiento que sea posible para facilitar sus tareas.

Literatura Citada

Abdullah, Mohd Syazwan, Ian Benest, Andy Evans, Chris Kimle. 2001. *“Knowledge Modelling Techniques for Developing Knowledge Management Systems”*. 3rd. European Conference on Knowledge Management. Dublin, Ireland. 15-25 p.

Abecker, Andreas, Ansgar Bernardi y Ludger van Elst. 2003. *“Agent Technology for Distributed Organizational Memories: The Frodo Project”*. Proceedings of the 5th International Conference on Enterprise Information Systems. Angers, France. Vol. 2. 3-10 p.

Abecker, Andreas, Ansgar Bernardi, Knut Hinkelmann, Otto Kühn y Michael Sintek. 1998. *“Toward a Technology for Organizational Memories”*. IEEE Intelligent Systems. 13(3): 40-48 p.

Ackerman, Marck S. 1998. *“Augmenting the Organizational Memory: A Field Study of Answer Garden”*. ACM Transactions on Information Systems. 16(3): 203-224 p.

Althoff, Klaus-Dieter, Adnreas Birk y Carsten Tautz. 1997. *“The Experience Factory Approach: Realizing Learning from Experience in Software Development Organizations”*. Proceedings of the Tenth German Workshop on Machine Learning.

Althoff, Klaus-Dieter, Andreas Birk, Susanne Hartkopf, Wolfgang Müller, Markus Nick, Dagmar Surmann y Carsten Tautz. 1999. *“Managing Software Engineering Experience for Comprehensive Reuse”*. Proceedings of the Eleventh International Conference on Software Engineering and Knowledge Engineering. Springer. 10-19 p.

Althoff, Klaus-Dieter, Ulrike Becker-Kornstaedt, Björn Decker, Andreas Klotz, Edda Leopold, Jörg Rech y Angi Voß. 2002. *“Enhancing Experience Management and Process Learning with Moderated Discourses: the indiGo approach”*. Proceedings of the European Conference on Artificial Intelligence (ECAI'02), Workshop on Knowledge Management and Organizational Memory.

- Andersson, Thorbjörn e Inger Eriksson. 1991. "Steering the maintenance costs". Proceedings of the 14th IRIS. 11-25 p.
- Banker, Rajiv D., Srikant M. Datar, Chris F. Kemerer y Dani Zweig. 1993. "Software Complexity and Maintenance Costs". Communications of the ACM. 36(11): 81-94 p.
- Barquin, Ramon C. 2001. "What is Knowledge Management?". Knowledge and Innovation: Journal of the KMCI. 1(2): 127-143 p.
- Barry, Evelyn, Sandra Slaughter y Chris F. Kemerer. 1999. "An Empirical Analysis of Software Evolution Profiles and Outcomes". Proceeding of the 20th International Conference on Information Systems, Charlotte, North Carolina, Estados Unidos. 453-458 p.
- Basili, Victor R. y Carolyn Seaman. 2002. "The Experience Factory Organization". IEEE Software. 19(3): 30-31 p.
- Basili, Victor R. y Gianluigi Caldiera. 1995. "Improve Software Quality by Reusing Knowledge and Experience". Sloan Management Review. MIT Press. 37(1): 55-64 p.
- Basili, Victor R. y Harland D. Mills. 1982. "Understanding and Documenting Programs". IEEE Transactions on Software Engineering. 8(3): 270-283 p.
- Basili, Victor R. y Salwa K. Abd-El-Hafiz. 1996. "A Method for Documenting Code Components". Journal of Systems and Software. 34(2): 89-104 p.
- Basili, Victor R., Patricia Costa, Mikael Lindvall, Manoel Mendonca, Carolyn Seaman, Roseanne Tesoriero y Marvin Zelkowitz. 2001. "An Experience Management System for a Software Engineering Research Organization". 26th Annual NASA Goddard Software Engineering Workshop. 26-35 p.
- Bellifemine, Fabio, Agostino Poggi, Giovanni Rimassa. 1999. "JADE – A FIPA-compliant agent framework". Proceedings of PAAM'99. 97-108 p.

Benjamins, V. Richard, Dieter Fensel y Asunción Gómez Pérez. 1998. “*Knowledge Management through Ontologies*”. Practical Aspects of Knowledge Management. 5.1-5.12 p.

Bennett, Keith y Vaclav Rajlich. 2000. “*Software Maintenance and Evolution: A Roadmap*”. ICSE: The Future of SE Track. 73-87 p.

Bergenti, Federico, Agostino Poggi y Giovanni Rimassa. 2000. “*Agent Architecture and Interaction Protocols for Corporate Memory Management Systems*”. ECAI'2000, Workshop on Knowledge Management and Organizational Memories.

Berger, Michael, Bernhard Bauer y Michael Watzke. 2001. “*A Scalable Agent Infrastructure*”. Autonomous Agent Conference, Workshop on Infrastructure for Agents, MAS, and Scalable MAS.

Bigus, Joseph P. y Jennifer Bigus. 2001. “*Constructing Intelligent Agents Using Java*”. Segunda Edición. Wiley Computer Publishing. New York. 408 pp.

Birk, Andreas, Torgeir Dingsoyr y Tor Stålhane. 2002. “*Postmortem: Never Leave a Project without it*”. IEEE Software. 19(3): 43-45 p.

Booch, Grady, James Rumbaugh e Ivar Jacobson. 1999. “*El Lenguaje Unificado de Modelado*”. Addison Wesley. Madrid. 432 pp.

Bradshaw, Jeffrey M. 1997. “*An Introduction to Software Agents*”. En: Software Agents. AAAI Press/ The MIT Press. 3-46 p.

Briand, Lionel, Yong-Mi Kim, Walcélio Melo, Carolyn Seaman y Victor Basili. 1998. “*Q-MOPP: Qualitative Evaluation of Maintenance Organizations, Processes, and Products*”. Journal of Software Maintenance. 10(4): 249-278 p.

Buckley, Fletcher J. 1989. “*Some Standards for Software Maintenance*”. Computer, IEEE. 22(11): 69-70 p.

Burg, Bernard, Jonathan Dale y Steven Willmott. 2001. "*Open Standards and Open Source for Agent-Based Systems*". AgentLink News, No. 6.

Carrol, John M. y Mary Beth Rosson. 1992. "*Getting Around the Task Artifact Cycle: How to Make Claims and Design by Scenario*". ACM Transactions on Information Systems. 10(2): 181-212 p.

Cheah Yu-N y Syed Sibte Raza Abidi. 2000. "*A Scenarios Mediated Approach for Tacit Knowledge Acquisition and Crystallisation: Towards Higher Return-On-Knowledge and Experience*". Proceedings of the Third International Conference on Practical Aspects of Knowledge Management (PAKM2000). 5.1-5.12 p.

Checkland, Peter y Jim Scholes. 1999. "*Soft System Methodology in Action*". John Wiley and Sons. 418 pp.

Chikofsky, Elliot J., James H. Cross II. 1990 "*Reverse Engineering and Design Recovery: A Taxonomy*", IEEE Software. 7(1): 13-17 p.

Choi, Song C. y Walt Scacchi. 1990. "*Extracting and Restructuring the Design of Large Systems*". IEEE Software. 7(1): 66-71 p.

Choo, Chun Wei. 1999. "*La organización Inteligente: el Empleo de la Información para dar Significado, Crear Conocimiento y Tomar Decisiones*". Oxford University Press. 346 pp.

Curtis, Bill, Herb Krasner y Neil Iscoe. 1988. "*A Field Study of the Software Design Process for Large Systems*". Communications of the ACM. 31(11): 1268-1287 p.

Curtis, Bill, Marc I. Kellner y Jim Over. 1992. "*Process Modeling*". Communications of the ACM. 35(4): 75-90 p.

Dale, J. y E. Mamdani. 2001. "*Open Standards for Interoperating Agent-Based Systems*". Wiley. Software Focus, 1(2).

Dart, Susan, Alan M. Christie y Alan W. Brown. 1993. "*A Case Study in Software Maintenance*". Software Engineering Institute, Carnegie Mellon University, Pittsburgh. Technical Report CMU/SEI-93-TR-8, ESC-TR-93-185.

Davenport, Thomas H. y Laurence Prusak. 2000. "*Working Knowledge: How Organizations Manage What they Know*". Harvard Business School Press. Boston, Massachusetts. 199 pp.

Ellis, C. A., S. J. Gibbs, y G. L. Rehn. 1991. "*Groupware: some issues and experiences*". Communications of the ACM. 34(1): 39-58 p.

Estublier, Jacky. 2000. "*Software Configuration Management: A Roadmap*". ICSE: The Future of SE Track. 279-289 p.

Fensel, Dieter. 2001. "*Ontologies: A Silver Bullet for Knowledge Management and Electronic Commerce*". Springer. Berlin. 138 pp.

Firestone, Joseph M. 2001. "*Key Issues in Knowledge Management*". Knowledge and Innovation: Journal of the KMCI. 1(3): 8-38 p.

Frank, Ulrich. 2001. "*Knowledge Management Systems: Essential requirements and generic Design Patterns*". En: Smari, W. W., N. Melab y K. Yetongnon (eds.): Proceedings of the International Symposium on Information Systems and Engineering, ISE'2001. Las Vegas: CSREA Press. 114-121 p.

Froufe Quintanas, Agustin. 2000. "*Java 2: Manual de usuario y tutorial*". 2da. edición. Alfaomega, Ra-Ma. México. 682 pp.

Gandon, Fabien. 2002. "*A Multi-Agent Architecture for Distributed Corporate Memories*". AT2AI-3, Third International Symposium "From Agent Theory to Agent Implementation", en el European Meeting on Cybernetics and Systems Research. 623-628 p.

Girgensohn, Andreas, Alison Lee, y Kevin Schlueter. 1996. "*Experience in Developing Collaborative Applications Using the World Wide Web 'Shell'*". Seventh ACM Conference on Hypertext. 246-255 p.

Griswold, William G. y David Notkin. 1993. "*Automated Assistance for Program Restructuring*". ACM Transactions on Software Engineering and Methodology. 2(8): 228-269 p.

Gruber, Thomas R. 1993. "*Toward Principles for the Design of Ontologies Used for Knowledge Sharing*". En: Formal Ontology in Conceptual Analysis and Knowledge Representation. N. Guarino y R. Poli (eds.). Kluwer Academic Publishers.

Han, Jefferson y Brian Smith. 1996. "*CU-SeeMe VR: Immersive Desktop Teleconferencing*". ACM Multimedia 96. 199-207 p.

Harandi, Mehdi T. y Jim Q. Ning. 1990. "*Knowledge-Based Program Analysis*". IEEE Software. 7(1): 74-81 p.

Hartmann, Jean y David J. Robson. 1990. "*Techniques for Selective Revalidation*". IEEE Software 7(1): 31-36 p.

Hausler, Philip A., Mark G. Pleszkoch, Richard C. Linger, Alan R. Hevner. 1990. "*Using Function Abstraction to understand Program Behaviour*". IEEE Software. 7(1): 55-63 p.

Hindus, Debby y Chris Schumandt. 1992. "*Ubiquitous Audio: Capturing Spontaneous Collaboration*". Proceedings of the ACM Conference CSCW'92. 210-217 p.

Huff, Cliff, Dennis Smith, Ed Morris y Paul Zarrella. 1992. "*Proceedings of the CASE Management Workshop*". Software Engineering Institute, Carnegie Mellon University, Pittsburgh. Technical Report CMU/SEI-92-TR-6, ESC-TR-92-006.

Hwang, John D. 2002. "*Information Resources Management: New Era, New Rules*". IT Professional. IEEE Computer Society. 9-19 p.

IEEE Std. 1044.1-1995. 1996. IEEE Guide to Classification for Software Anomalies. IEEE Computer Society. New York.

IEEE Std. 1219-1998. 1998. IEEE Standard for Software Maintenance. IEEE Computer Society. New York.

IEEE Std. 610.12-1990. 1990. IEEE Standard Glossary of Software Engineering Terminology. IEEE Computer Society. New York.

Isaacs, Ellen A., Trevor Morris y Thomas K. Rodríguez. *“A Form for Supporting Interactive Presentations to Distributed Audiences”*. Proceedings of the ACM Conference CSCW’94. 405-416 p.

Kalfoglou, Yannis. 2000. *“On the convergence of core technologies for knowledge management and organizational memories: ontologies and experience factories”*. Proceedings of the ECAI2000 Workshop on Knowledge Management and Organizational Memories. 48-55 p.

Kolp, Manuel. 2002. *“Agent-based IT support for Knowledge Management”*. Working paper 29/02. Institut d’administration et de gestion, Université catholique de Loivain.

Kucza, Timo, Minna Nättinen, Päivi Parviainen. 2001. *“Improving Knowledge Management in Software Reuse Process”*. En: Bomarius, Frank y Seija Komi-Sirviö (eds.): Product Software Process Improvement: Proceedings of the 3rd. International Conference: PROFES 2001. 141-152 p.

Kucza , Timo y Seija Komi-Sirviö. 2001. *“Utilising Knowledge Management in Software Process Improvement – The Creation of a Knowledge Management Process Model”*. Proceedings of the 7th International Conference on Concurrent Enterprising. ICE’01.

- Lacher, Martin S. y Michael Koch. 2000. "*An Agent-based Knowledge Management Framework*". Proceedings of the AAAI, Workshop on Bringing Knowledge to Business Processes.
- Leon, Alexis. 2000. "*A Guide to Software Configuration Management*". Artech House, Boston. 382 pp.
- Lewis, T. G. 1990. "*CASE: Computer-Aided Software Engineering*". Van Nostrand Reinhold. 593 pp.
- Liao, Minghong, Andreas Abecker, Ansgar Bernardi, Kunt Hinkelman y Michael Sintek 1999. "*Ontologies for Knowledge Retrieval in Organizational Memories*". Proceedings of the Learning Software Organizations(LSO'99) workshop, 19-26 p.
- Lientz, B. P., E. B. Swanson y G. E. Tompkins. 1978. "*Characteristics of Application Software Maintenance*". Communications of the ACM. 21(6): 466-471 p.
- Lientz, Bennet P. y Burton Swanson. 1981. "*Problems in Application Software Maintenance*". Communications of the ACM. 24(11): 763-769 p.
- Lientz, Bennet P. 1983. "*Issues in Software Maintenance*". Computing Surveys. 15(3): 271-278 p.
- Liotta, Matt. 2003. "*Apache's Xindice Organizes XML Data Without a Schema*". DevX. <http://www.devx.com/xml/article/9796>.
- Marwick, A. D. 2001. "*Knowledge management technology*". IBM Systems Journal. 40(4): 814-830 p.
- Mayrhauser, Anneliese von y A. Marie Vans. 1995. "*Program Comprehension During Software Maintenance and Evolution*". IEEE, Computer. 28(8): 44-55 p.

McDonald, David W. y Mark S. Ackerman. 2000. "*Expertise Recommender: A Flexible Recommendation System and Architecture*". Proceeding of the ACM 2000 Conference on Computer Supported Cooperative Work. 231-240 p.

McElroy, Mark W. 2000. "*The New Knowledge Management*". Knowledge and Innovation: Journal of the KMCI. 1(1): 43-67 p.

Mendonca Neto, Manoel Gomes de, Carolyn B. Seaman, Victor R. Basili y Yong-Mi Kim. 2001. "*A Prototype Experience Management System for a Software Consulting Organization*". Proceedings of the SEKE 2001 Conference.

Mercer, Sarah y Sue Greenwod. 2002. "*A Multi-Agent Architecture for Knowledge Sharing*". AT2AI-3, Third International Symposium "From Agent Theory to Agent Implementation", en el 16th European Meeting on Cybernetics and Systems Research.

Monk, Andrew y Steve Howard. 1998. "*The Rich Picture: A Tool for Reasoning About Work Context*". Interactions. 5(2): 21-30 p.

Moran, Thomas P., William van Melle y Patrick Chiu. 1998. "*Tailorable Domain Objects as Meeting tools for an Electronic Whiteboard*". Proceedings of the ACM Conference CSCW'98. 295-304 p.

Moreau, Luc, Nick Gibbins, David DeRoure, Samhaa El-Beltagy, Wendy Hall, Gareth Hughes, Dan Joyce, Sanghee Kim, Danius Michaelides, Dave Millard, Sigi Reich, Robert Tansley y Mark Weal. 2000. "*SoFAR with DIM Agents: An Agent Framework for Distributed Information Management*". Proceedings of the 5th International Conference on the Practical Applications of Intelligent Agents and Multi-Agent Technology. 369-388 p.

Munson, John C. 1998. "*Software Lives Too Long*". IEEE Software 15(4): 18-20 p.

Nonaka, Ikujiro y Noboru Konno. 1998. "*The Concept of "Ba": Building a Foundation for Knowledge Creation*". California Management Review. 40(3): 40-54 p.

- Nwana, Hyacinth S. 1996. “*Software Agents: An Overview*”. Knowledge Engineering Review. 11(2): 205-244 p.
- Oman, Paul W. y Curtis R. Cook. 1990. “*The Book paradigm for Improved Maintenance*”. IEEE Software. 7(1): 39-45 p.
- Osborne, Wilma M., Elliot J. Chikofsky. 1990. “*Fitting Pieces to the Maintenance Puzzle*”. IEEE Software. 7(1): 11-12 p.
- Parikh, Girish. 1985. “*Handbook of Software Maintenance*”. Wiley- Interscience. 421 pp.
- Pedersen, Mogens Kühn y Michael Holm Larsen. 2000. “*Distributed Knowledge Management Based on Product State Models – The Case of Decision Support in Health Care Administration*”. Working Papers 2000-12. Copenhagen Business School, Department of Informatics.
- Pigoski, T. M. 1997. “*Practical Software Maintenance*”. Best Practices for Managing Your Investment. Ed. John Wiley & Sons, USA.
- Poggi, Agostino, Giovanni Rimassa y Paola Turci. 2002. “*An Intranet Based Multi-Agent System for Corporate Memory Management*”. AT2AI-3, Third International Symposium “From Agent Theory to Agent Implementation” en el 16th European Meeting on Cybernetics and Systems Research.
- Polo, M., M. Piattini, y F. Ruiz. 2002. “*Using a Qualitative Research Method for Building A Software Maintenance Methodology*”. Software Practice & Experience. John Wiley and Sons. 32(13): 1239-1260 p.
- Polo, Macario, Mario Piattini, Francisco Ruiz y Coral Calero. 1999. “*Roles In The Maintenance Process*”. ACM SIGSOFT, Software Engineering Notes. 24(4).
- Pressman, Roger S. 2002. “*Ingeniería del Software: un enfoque práctico*”. McGraw-Hill. 5ta. Edición. 601 pp.

Ramage, Magnus y Keith Bennett. 1998. “*Maintaining Maintainability*”. International Conference on Software Maintenance. (ICSM’98).

Rich, Charles y Linda M. Wills. 1990. “*Recognizing a Program’s Design: A Graph-Parsing Approach*”. IEEE Software. 7(1): 82-89.

Riecken, Doug. 1994. “*Agents that Reduce Work and Information Overload*”. Communications of the ACM. 37(7): 31-41 p.

Robertson, George G., Stuart K. Card y Jock D. Mackinlay. 1993. “*Information Visualization Using 3D Interactive Animation*”. Communications of the ACM. 36(4): 57-71 p.

Robillard, Pierre N. 1999. “*The Role of Knowledge in Software Development*”. Communications of the ACM. 42(1): 87-92 p.

Rus, Ioana y Mikael Lindvall. 2002. “*Knowledge Management in Software Engineering*”. IEEE Software. 19(3): 26-38 p.

Schneider, Kurt, Jan-Peter von Hunnius y Victor R. Basili. 2002. “*Experience in Implementing a Learning Software Organization*”. IEEE Software. 19(3): 46-49 p.

Seaman, Carolyn B. 2002. “*The Information Gathering Strategies of Software Maintainers*”. International Conference on Software Maintenance.

Singer, Janice. 1998. “*Practices of Software Maintenance*”. Proceedings of the International Conference on Software Maintenance. 139-145 p.

Singer, Janice, Timothy C. Lethbridge. 1996. “*Methods for Studing Maintenance Activities*”. Proceedings of the Workshop on Empirical Studies of Software Maintenance.

Singh, Narinder, Micheal Genesereth y Mustafa Syed. 1995. "*A Distributed and Anonymous Knowledge Sharing Approach to Software Interoperation*". International Journal of Cooperative Information Systems. 4(4): 339-368 p.

Staab, Steffen y Alexander Meadche. 2001. "*Knowledge Portals: Ontologies at Work*". The AI Magazine. 22(2): 63-75 p.

Sure, York. 2002. "*A Tool-supported Methodology for Ontology-based Knowledge Management*". ISMIS 2002, Methodologies for Intelligent Systems.

Tacla. Cesar y Jean-Paul Bartès. 2002. "*A Multi-agent Architecture for Knowledge Management Systems*". Second International Symposium on Advanced Distributed Computing Systems (ISADS 2002).

Tamma, Valentina y Trevor Bench-Capon. 2001. "*A conceptual model to facilitate knowledge sharing in multi-agent systems*". Proceedings of the Autonomous Agents, Workshop on Ontologies in Agent Systems. 69-76 p.

Tautz, Carsten y Klaus-Dieter Althoff. 1998. "*Operationalizing Comprehensive Software Knowledge Reuse Based on CBR Methods*". Proceedings of the 6th German Workshop on Case-Based Reasoning (GWCBR-98). Technical Report, University of Rostock, IMIB series. vol. 7. 89-98 p.

Thomsett, Rob. 1998. "*The year 2000 Bug: A Forgotten Lesson*". IEEE Software. 15(4): 91-95 p.

Tiwana, Amrit. 2000. "*The Knowledge Management Toolkit: Practical Techniques for Building a Knowledge Management System*". Prentice Hall. USA. 608 pp.

Vessey, Iris y Ron Weber. 1983. "*Some factors Affecting Program Repair Maintenance: An Empirical Study*". Communications of the ACM. 26(2): 128-134 p.

Vitaglione, G., F. Quarta y E. Cortese. 2002. "Scalability and Performance of JADE Message Transport System". AAMAS Workshop on AgentCities.

Walsh, J. y G. Ungson. 1991. "Organizational memory". Academy of Management Review. 16(1): 57-91 p.

Walz, Diane B., Joyce J. Elam y Bill Curtis. 1993. "Inside a Software Design Team: Knowledge Acquisition, Sharing, and Integration". Communications of the ACM. 36(10): 63-77 p.

Wang, Alf Inge, Reidar Conradi y Chunnian Liu. 1999. "A Multi-Agent Architecture for Cooperative Software Engineering". Proceedings of the Eleventh International Conference on Software Engineering and Knowledge Engineering.

Wastell, David G., P White y P. Kawalek. 1994. "A Methodology for Business Process Redesign: Experiences and Issues". Journal of Strategic Information Systems. 3(1): 23-40 p.

Wooldrige, Michael y Paolo Ciancarini. 2001. "Agent-Oriented Software Engineering: The State of the Art". En: Agent-Oriented Software Engineering. P. Ciancarini y M. Wooldrige (eds.). Springer-Verlang. Lecture Notes in AI. Vol. 1957.

Young, Michael J. 2001. "Aprenda XML Ya". Mc-Graw-Hill. Madrid. 344 pp.

Ligas de Internet citadas

CUTTER. 2003. Portal de conocimientos sobre negocios. URL: <http://www.cutter.com/index.shtml>

LinuxKP. 2003. Portal de conocimiento sobre linux. URL: <http://www.linux-knowledge-portal.org/en/index.php>

KA2. 2003. Portal KA2 sobre administración del conocimiento. URL:
<http://ka2portal.aifb.uni-karlsruhe.de/>

TIME2RESEARCH. 2003. Portal de conocimientos para negocios. URL:
http://www.time2research.de/portal/portal_en.htm

DI_CICESE. 2003. Departamento de informática del CICESE. URL:
<http://telematica.cicese.mx/informatica/>

INTERSEL. 2003. Sitio en Internet de INTERSEL. URL: <http://www.intersel.com.mx/>

FIPA. 2003. Sitio oficial del estándar FIPA. URL: <http://www.fipa.org/>

JADE. 2003. Sitio oficial de JADE. URL: <http://sharon.cselt.it/projects/jade/>

XINDICE. 2003. Sitio oficial de Xindice. URL: <http://xml.apache.org/xindice/>

APACHE. 2003. Sitio oficial de la Fundación de Software, Apache. URL:
<http://www.apache.org/>

XPATH. 2003. Especificación del lenguaje de consultas XPath. URL:
<http://www.w3c.org/TR/xpath>

Apéndice A. Casos de Estudio

En este apéndice se presentan los detalles de los dos casos de estudio realizados. Principalmente se describe el ambiente de cada uno de los grupos estudiados, y posteriormente se describen los procesos que fueron capturados en cada uno de ellos.

A.1 Caso de Estudio: Departamento de informática del CICESE

El Departamento de Informática (DI) del Centro de Investigación Científica y de Educación Superior de Ensenada (CICESE), tiene como objetivo apoyar a las áreas administrativas del Centro. Las áreas principales a las que atiende son:

- Subdirección de Administración Presupuestal;
- Subdirección de Recursos Humanos;
- Subdirección de Recursos Materiales;
- Subdirección de Recursos Financieros;
- Dirección de Estudios de Posgrado;

así como a las distintas divisiones y departamentos académicos del Centro.

Con el fin de dar soporte a todas estas áreas del CICESE, el DI trabaja en el mantenimiento de los siguientes sistemas:

- Sistema de Información de la Subdirección de Recursos Materiales y de Servicios (SIRMA). En este sistema se maneja lo referente a los procesos de Fondo Fijo, Ordenes de Trabajo, Pedidos, Manejo de Almacén. Así como la preparación para la generación de documentación en los procesos de: entrega de material al solicitante, cartas y seguimiento a proveedores, importaciones y control patrimonial.
- Presupuestos (integrado en los sistemas administrativos). Es un sistema de apoyo para la Subdirección de Presupuestos en sus funciones como: distribución de presupuestos de proyectos nuevos, alta y vigencia de proyectos nuevos, asignaciones presupuestales, seguimiento de proyectos, generación de informes institucionales, elaboración de presupuestos anuales.

- Sistema de Recursos Humanos (SRH). Es un sistema de apoyo para la Subdirección de Recursos Humanos en cuanto a la administración de prestaciones, generación de la nómina y mantenimiento de la información del personal del Centro.
- Sistema de Información de la Subdirección de Recursos Financieros (SIREFI). Es un sistema de apoyo para la Subdirección de Recursos Financieros con la automatización de la mayoría de sus procesos internos.
- Sistema de la Dirección de Estudios de Posgrado (SIDEPE). Es un sistema de apoyo para el Departamento de Servicios Escolares y seguimiento de egresados en cuanto al mantenimiento de la información de todo lo referente a los postrados que ofrece el Centro así como estudiantes y maestros.
- Productividad Académica. Es un sistema de apoyo para la Dirección de Estudios de Posgrado en el manejo de todo lo referente al registro de la productividad académica del Centro.
- Fondo de Ahorro y de Reserva para los trabajadores del CICESE (FONAHR). Es un sistema de apoyo para la administración del Fondo de Ahorro y de Reserva de los trabajadores del CICESE.
- Los sistemas administrativos (para el personal de apoyo administrativo en las Divisiones Académicas). Sistema desarrollado como herramienta integral de comunicación con el Sistema de Información Administrativa.

Además de estos sistemas de apoyo a las tareas administrativas del CICESE, el DI se encuentra actualmente desarrollando otro conjunto de proyectos, como lo son:

- SIRECH. Este proyecto abarca la automatización y mejora de procesos de los departamentos: Control de Plazas, Prestaciones y Personal, de la Subdirección de Recursos Humanos, así como requerimientos específicos de la propia subdirección enfocados a la toma de decisiones.
- SIPPPE. Proyecto para la Subdirección de Programación, Presupuestación y Estadística, que abarca la modernización y mejora de algunos de los procesos que realiza esta subdirección de acuerdo a las demandas del entorno.
- SAT. Es un Sistema para Atención Técnica.

- Portal de Informática. Tiene como objetivo proveer un medio de información abierto e independiente de la plataforma, que permita al personal académico, administrativo y estudiantes acceder a la información administrativa de su interés.

En el caso de los proyectos SIRECH y SIPPRE, podría considerarse que son mejoras o ampliaciones a sistemas existentes.

Los sistemas administrativos del CICESE se encuentran en plataforma Cliente-Servidor. Los servidores son máquinas Sun con sistema operativo Solaris, mientras que las máquinas clientes son principalmente PC's con sistema operativo Windows 9x. Estos sistemas han sido desarrollados principalmente con las herramientas proporcionadas por Oracle (Developer 2000, Developer Forms, Developer Reports), y en menor medida Visual C++ con Pro C de Oracle. Los desarrollos que se hacen para dar soporte a través del Web se realizan principalmente con PHP, aunque existen sistemas desarrollados con tecnología Java (Servlets y JSP).

Actualmente la distribución geográfica de los sistemas a los que da mantenimiento el DI comprende siete servidores de aplicaciones y un servidor de base de datos. Cinco de los servidores se encuentran distribuidos en las distintas áreas de CICESE, uno se encuentra en la ciudad de La Paz, Baja California Sur, y otro en la guardería del CICESE. El servidor del área administrativa contiene la totalidad de los sistemas, el resto solo aquellos que son necesarios para los departamentos correspondientes. Toda la información a la que acceden las distintas aplicaciones se encuentra en un servidor de base de datos central. La figura 60 muestra esta distribución.

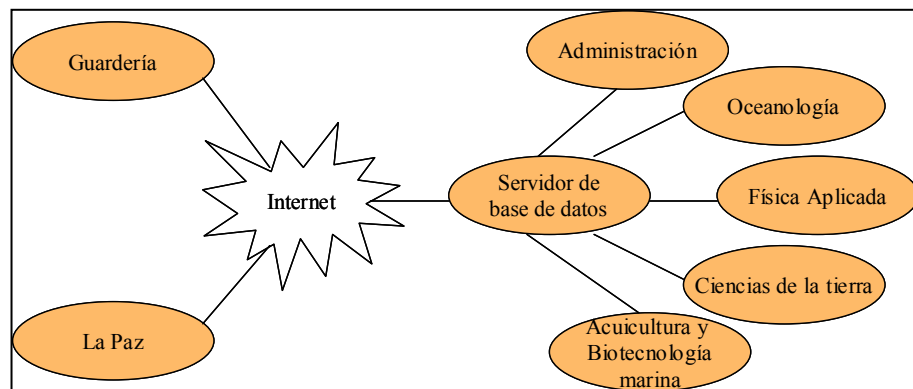


Figura 60. Distribución geográfica de los sistemas a los que da mantenimiento el DI

En el momento de la realización del caso de estudio, el DI se encontraba constituido de 14 personas, entre ellas, el Jefe del departamento, una secretaria, cinco técnicos y 7 becarios o personal externo. A continuación se describen los procesos de atención a usuarios, atención de solicitudes de cambios y reportes de error, y control de versiones; llevados en el DI.

A.1.1 Descripción de los procesos del DI

La figura 61 muestra una aproximación al proceso de mantenimiento que es llevado en el DI. Como se puede observar, el proceso, aun cuando se presenta de una manera muy general, resulta complejo. Con el fin de facilitar un poco el entendimiento de este proceso, hemos decidido dividir este proceso en tres procesos más pequeños: 1) la solicitud que el usuario realiza al personal de mantenimiento. 2) La atención a esta solicitud por parte del personal de mantenimiento. Este proceso se ha dividido en dos: a) la solución de errores y b) la realización de adaptaciones, mejoras, etc. Y por último, 3) la manera en que es llevado a cabo el control de los cambios, que incluye el flujo de los archivos fuentes, desde que se inicia el mantenimiento hasta la liberación del sistema modificado, así como el proceso de liberación de las nuevas versiones, y el control de la base de datos.

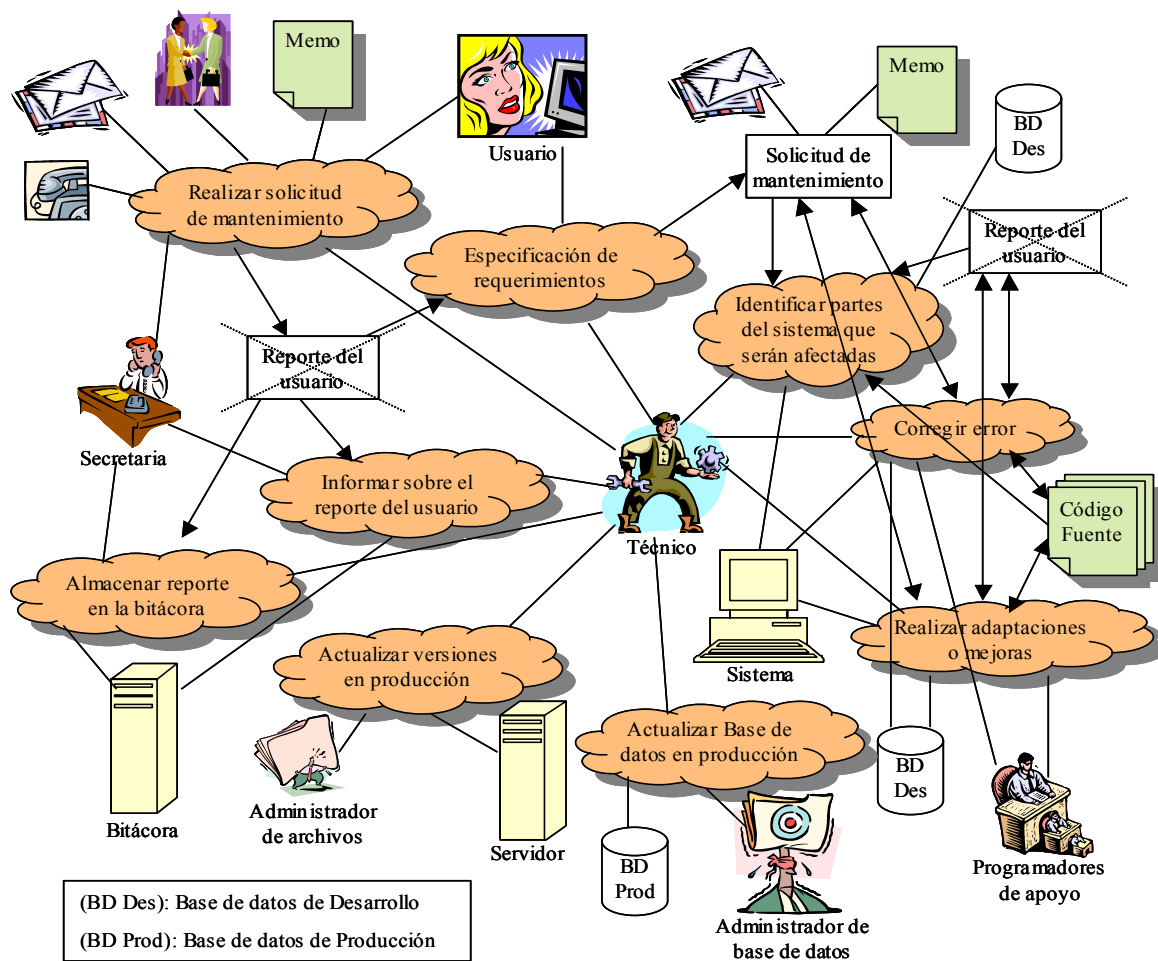


Figura 61. Proceso de mantenimiento en el DI del CICESE

En el DI, la distribución del trabajo se hace tomando como base las áreas dentro del CICESE a las que les da soporte el departamento. De esta manera, cada uno de los técnicos está encargado de un área, y es el responsable directo de la atención a los usuarios de esa área, así como de resolver las solicitudes que estos realizan. La mayor parte de las actividades del DI corresponden a atender a los usuarios. La atención se da por vía telefónica, correo electrónico, mediante reuniones o entrevistas, y en menor medida por medio de memorandos.

Las solicitudes realizadas por los usuarios pueden dividirse en dos grandes grupos:

Solicitudes de consulta. Este tipo de solicitud es el que se da con mayor frecuencia. No requiere cambios a los sistemas, por lo general se refieren a errores del usuarios en la captura de la información, o dudas con respecto al funcionamiento de los sistemas.

Solicitudes de mantenimiento. Son solicitudes que sí requieren cambios en los sistemas. Pueden derivarse, principalmente, de errores del sistema; fallas en los datos de la base de datos; o por modificaciones en la manera de trabajar de los usuarios, como por ejemplo modificaciones en las políticas, reglamentos, procedimientos o procesos.

En este caso nos enfocaremos principalmente a las solicitudes de mantenimiento, pero sin dejar de considerar las solicitudes de consulta, ya que implican un tiempo considerable en el trabajo de los técnicos del DI.

Proceso de realización de solicitudes de mantenimiento

Una solicitud de mantenimiento inicia principalmente como una petición del usuario, aunque hay casos en los que se derivan de necesidades o sugerencias de los técnicos encargados de mantener el sistema. Estas peticiones pueden deberse a problemas detectados en el sistema por parte del usuario, así como a cambios en la manera de trabajar. Muchos de los cambios en la manera de trabajar están relacionados con modificaciones en las políticas, reglamentos, procesos o procedimientos que siguen los usuarios, las cuales pueden estar guiadas por modificaciones federales (como las leyes fiscales) o cambios en las políticas de las jefaturas o direcciones de las distintas áreas, mismas que pueden deberse a la integración de nuevas personas en estos puestos.

Cuando el usuario realiza una solicitud, normalmente hace una llamada telefónica o envía un correo electrónico al técnico encargado de su área, o a la secretaria del DI. Otra opción puede ser que las solicitudes se hagan, de manera personal, durante reuniones entre el personal del DI y el usuario. En ambos casos el personal del DI y el usuario se ponen de acuerdo en los requerimientos de los cambios ya sea por teléfono o mediante entrevistas.

Una vez que se ha determinado qué es lo que se quiere hacer, así como los compromisos entre el DI y el usuario, se le solicita al usuario enviar por escrito su solicitud, cosa que por lo general se realiza mediante un correo electrónico. En raras ocasiones es el mismo personal del DI quien realiza el documento por escrito.

En este documento, si el problema que está reportando es un error del sistema, se indica el tipo de error y las condiciones en que se generó. Si la solicitud es una modificación debido, por ejemplo, a cambios de políticas y procedimientos o sugerencias del usuario o el mismo técnico, se deben establecer los nuevos requerimientos que debe cubrir el sistema, así como las opciones que puedan haber quedado obsoletas con las nuevas modificaciones. Sin embargo, el usuario al final de cuentas es el que decide qué escribir, ya que no existe un criterio definido para reportar errores, modificaciones o sugerencias de mejora a los sistemas.

Cuando los usuarios reportan errores por medio de la secretaria, ésta trata de dar solución por su cuenta al problema (en caso de que resulte solo en una solicitud de consulta), si no puede resolverlo, o detecta que la solución requiere modificaciones al sistema, lo turna al técnico encargado del área a la que pertenece el usuario. La secretaria captura el reporte del usuario en una bitácora con el fin de que quede un registro de lo que se hace. Entre los datos que se capturan en la bitácora están el sistema del cual se originó el error, el usuario que lo reporta, el tipo de error, así como las condiciones en que se originó (datos capturados, etc.). Si fue la secretaria la que resolvió el problema, también captura en la bitácora cuál fue la solución. Por lo general es ella la que resuelve directamente los problemas que le reportan, ya que muchos se deben a errores o dudas del usuario y no requieren cambios a los sistemas.

El usuario también puede hacer sus reportes o solicitudes directamente con los técnicos encargados de su área. En este caso, al igual que cuando el reporte es hecho a través de la secretaria, muchos de ellos no requieren cambios en los sistemas, por lo que son resueltos en el momento; sin embargo, en este caso, a diferencia de con la secretaria, los técnicos rara

vez dejan asentado estos reportes en la bitácora. Incluso varios de los técnicos no la usan en absoluto. Algunos llevan su bitácora personal, ya sea de manera electrónica o en algún cuaderno, otros no llevan registro de lo que se les solicita.

Las solicitudes de cambio más comunes se deben por lo general a cambios en las políticas, procedimientos o reglamentos; en estos casos, el usuario es el que informa al DI de estas modificaciones. En ocasiones este tipo de cambios puede conocerse con anticipación, ya sea porque se dan de manera periódica, o porque se había informado con anterioridad. En cualquier caso, es necesario que el técnico encargado y el usuario platiquen para definir qué es lo que se quiere. Una vez de acuerdo, se le pide al usuario que envíe por escrito qué fue lo que se acordó. En caso de que la solicitud por escrito no sea lo suficientemente clara, el técnico se pone en contacto con el usuario, ya sea de manera personal o vía telefónica, para aclarar las dudas que puedan existir. Cabe destacar que hay ocasiones en que los técnicos creen haber identificado la totalidad de los requerimientos, o que los usuarios dan por hecho que los técnicos conocen o sabrán ciertas cosas (por ejemplo como realizar un determinado cálculo), cuando en realidad no es así. Estas situaciones provocan que deba volverse a modificar el sistema una vez que se le entrega al usuario, para adaptarlo a estos cambios no previstos en un principio.

En la figura 62 se muestra una representación del proceso que se sigue al recibir solicitudes por parte de los usuarios. Es posible ver que cuando el usuario realiza su solicitud, pueden participar tanto la secretaria como el técnico responsable de su área en la recepción de la misma. Una vez recibida la solicitud, se elabora un reporte de ésta, el cual debe ser almacenado en una bitácora, pero esto no sucede en todos los casos. Posteriormente se obtienen los requerimientos del cambio que se pretende hacer. Estos requerimientos pueden deberse, entre otros casos, a la necesidad de corregir errores en el sistema, o por modificaciones a la manera de trabajar de los usuarios (es necesario indicar que en ocasiones, como ya se ha mencionado, los cambios pueden deberse a sugerencias del técnico encargado del sistema). Una vez establecidos los requerimientos, se le pide al usuario que realice una solicitud de mantenimiento que indique los requerimientos y los

acuerdos a que se llegó. Es preciso añadir que en raras ocasiones es el mismo personal del DI el que documenta esta información.

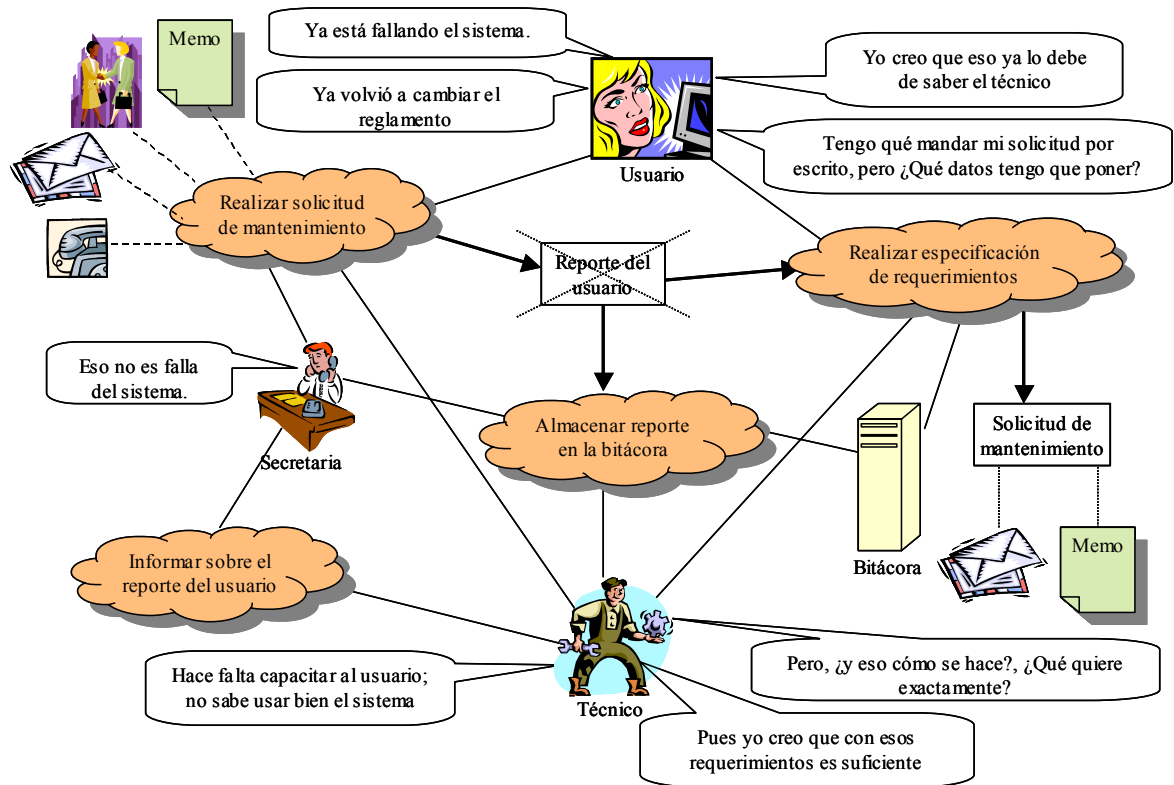


Figura 62. Representación del proceso que se lleva al recibir solicitudes del usuario.

Proceso de atención a la solicitud de mantenimiento

Una vez que el usuario y el técnico responsable se han puesto de acuerdo en los cambios que se realizarán al sistema, el técnico procede a planificar su estrategia de solución. Para esto debe contar con un entendimiento del proceso al que da soporte el sistema. Este entendimiento ayuda a que sea posible identificar cuáles serán las partes del sistema que serán afectadas.

La manera en que los técnicos logran el entendimiento del sistema que requieren (figura 63) es apoyándose, primeramente, en la experiencia que han logrado acumular. Cuando esta experiencia no es suficiente recurren a la documentación del sistema que explica los procedimientos, como puede ser el análisis de requerimientos. Sin embargo, en varios

sistemas la documentación no existe, no se encuentra actualizada, no corresponde con la situación real del sistema o no es suficiente, en estos casos los técnicos se apoyan en los usuarios para que les expliquen el funcionamiento o resuelvan dudas. Para esto último también se apoyan en otros técnicos que hayan trabajado en el sistema con anterioridad. Otra técnica empleada es la de usar el sistema de la misma forma en que lo hacen los usuarios, es decir, jugar el papel de usuarios del sistema.

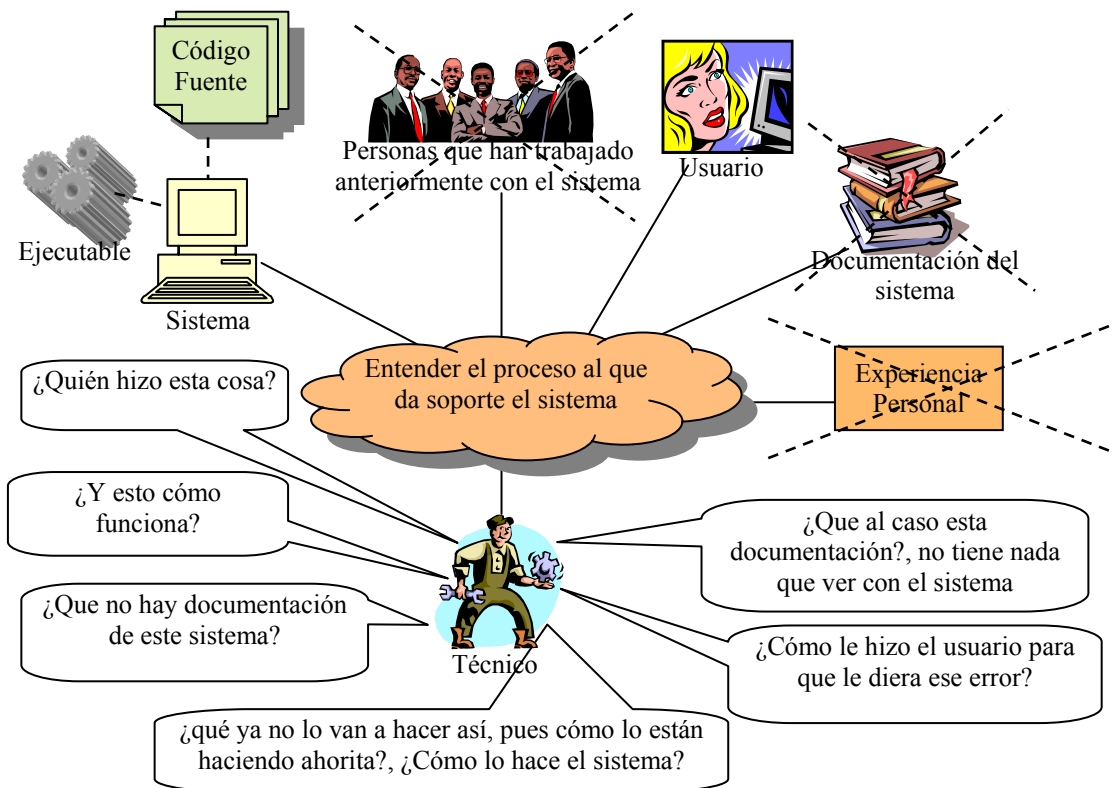


Figura 63. Representación de la manera en que los técnicos logran un entendimiento del proceso al que da soporte el sistema que mantienen.

Cuando el cambio se deriva de un error del sistema (figura 64), uno de los primeros pasos es tratar de reproducir el error para identificar las partes del sistema que deben ser modificadas. El técnico realiza un análisis para determinar en que parte se genera dicho error. Este análisis se basa principalmente en seguir el flujo que siguió el usuario al momento en que se generó el error. El técnico analiza el flujo de los datos, analiza internamente el sistema para ver de que manera está llevando a cabo este flujo. En este

punto el primer paso es verificar que los datos estén correctos, de no ser así se corrigen, en caso contrario, se analiza que parte del código del sistema pueda estar generando el error para corregirlo, por lo que se hace un seguimiento a la ejecución del código del sistema. Debido a lo anterior, con frecuencia esta parte del proceso de corrección de errores consume bastante tiempo, sobre todo si el técnico no cuenta con el suficiente conocimiento de la estructura interna del sistema. Ya que algunos de los sistemas no están documentados, de manera interna o externa, en ocasiones resulta difícil entender el código del mismo.

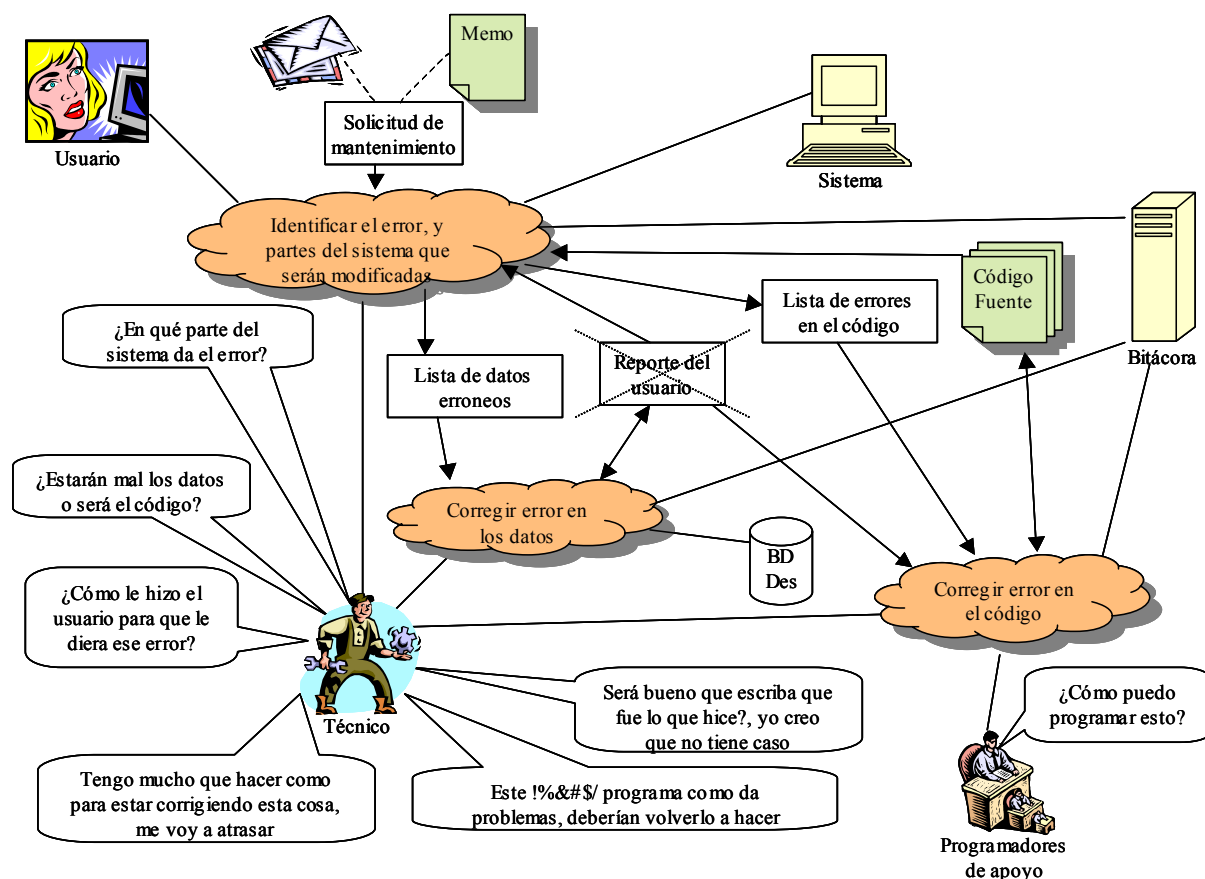


Figura 64. Representación del proceso de Solución de errores del sistema.

El seguimiento que el técnico hace, se basa en lo reportado por el usuario, en caso de que algunos aspectos no queden claros, se consulta con el usuario para aclararlos. Si el técnico no puede realizar personalmente el cambio, ya sea por falta de tiempo o algún otro factor, solicita la ayuda de programadores de apoyo. Cabe señalar que la experiencia juega un papel importante en la detección de errores en el código, ya que, por ejemplo, el conocer el

tipo de fallas más comunes, así como la manera de solucionarlas, ayuda a identificar y solucionar el error de una manera más rápida.

Los cambios que se derivan de modificaciones en los procedimientos, reglamentos, políticas o procesos en las áreas a las que atiende el DI, así como en sugerencias, con frecuencia pueden preverse con anticipación; debido a esto, resultan en cambios que pueden planearse y que frecuentemente requieren de tiempos considerables por ser modificaciones que involucran gran parte del sistema, o cambios muy profundos en su estructura interna. En estos casos, una vez que el técnico cuenta con los requerimientos, el primer paso es identificar las partes del sistema que serán afectadas, para posteriormente elaborar un plan para la realización de estas modificaciones.

La elaboración del plan requiere de hacer una estimación del tiempo que tomará hacer las modificaciones. Esta estimación se basa principalmente en la experiencia de la persona encargada de hacerla. Con regularidad estas estimaciones se alejan bastante con respecto a los resultados reales (en ocasiones pueden ser desvíos de más del 50%). Estas desviaciones pueden deberse a la falta de experiencia en el personal para identificar posibles riesgos, así como también al tiempo que le dedican a estar atendiendo consultas y solicitudes no programadas de los usuarios. Otra causa puede ser el no contar con referencias previas que les permitan hacer estimaciones más acercadas a la realidad.

La figura 65 presenta la manera en que se lleva a cabo el proceso antes mencionado. Primeramente muestra cómo el técnico identifica las partes del sistema que serán modificadas. Para esto requiere de los datos que haya proporcionado el usuario al momento de realizar la solicitud de cambios (reporte, y solicitud). En caso de que al técnico no le resulte claro lo que el usuario solicita, o no exista documentación que contenga esta información, puede consultar con él para aclarar las dudas. En ocasiones los técnicos se apoyan en el sistema ejecutable y en el código fuente para identificar qué módulos serán los afectados; sobre todo cuando no cuentan con la experiencia suficiente o no existe documentación que los ayude en esto. Una vez que se han identificado los módulos que

serán modificados, se realiza un plan a seguir para la realización de las modificaciones, este plan puede estar escrito en un documento; pero no es necesariamente así en todos los casos, puede resultar que este plan se encuentre solo en la mente del técnico (sobre todo cuando los cambios no son muchos o muy complicados). Por último se realizan las modificaciones necesarias, tanto en el código como en la base de datos, según sea el caso. Hay ocasiones en que los cambios no pueden ser realizados por la persona encargada, o requieren de más de una persona para llevarlos a cabo, en estos casos el técnico recibe ayuda de otros programadores.

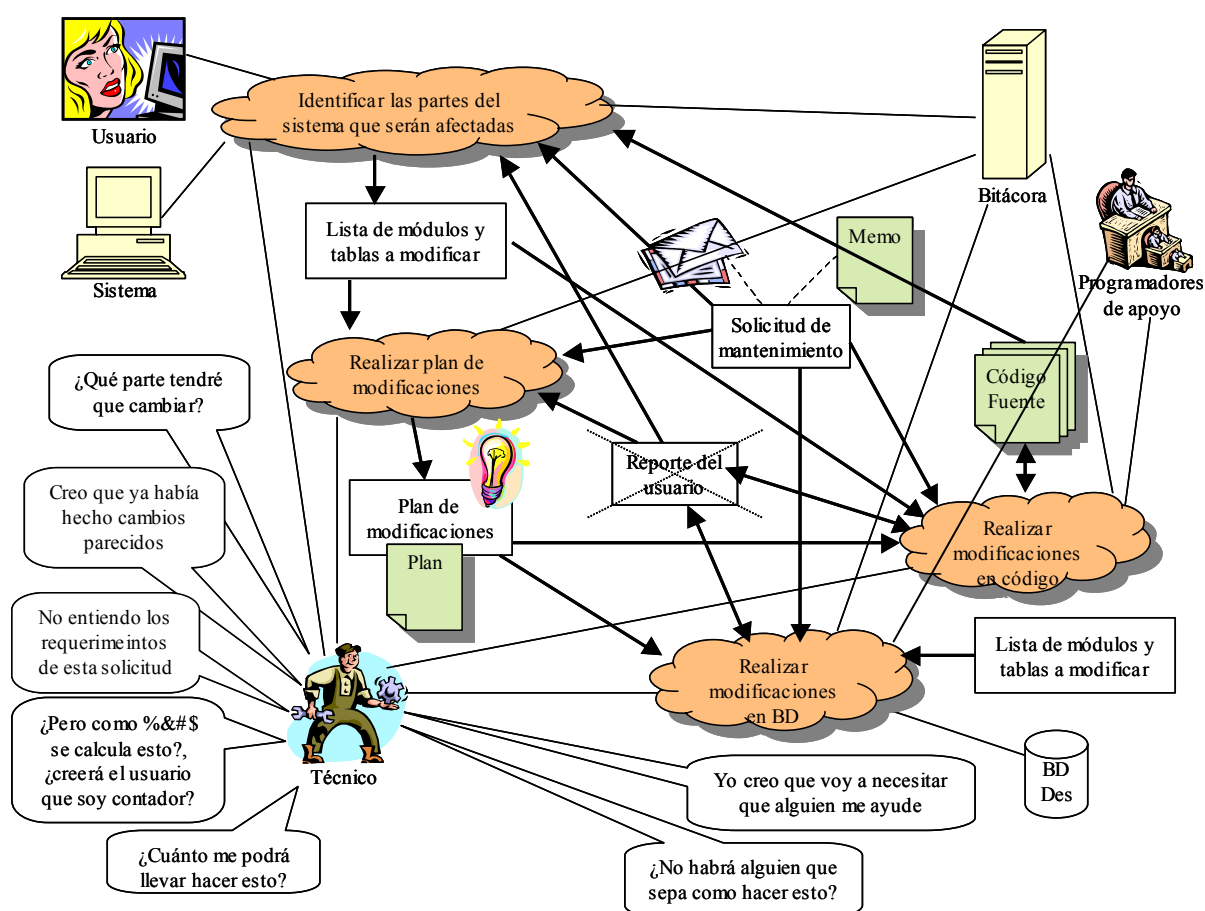


Figura 65. Realización de modificaciones que pueden resultar en cambios grandes. En estos casos por lo general se requiere de establecer un plan a seguir.

Cuando el técnico encargado tiene la suficiente experiencia en el sistema que maneja (y los cambios requeridos no son muy grandes), por lo general se va directamente a modificar el código. En otros casos busca primero entender el proceso, después hace un seguimiento de

éste, para posteriormente identificar qué partes del sistema resultarán afectadas. Al igual que en la identificación de errores, este proceso puede ser tardado, sobre todo si el técnico no tiene experiencia, ya que muchas veces involucra analizar el código del sistema.

Un aspecto importante es el identificar si los cambios que se realizan pueden afectar a algún otro programa o sistema; aun cuando esto no es muy común, en ocasiones sucede. Por lo general los técnicos se basan en su experiencia personal para determinar si un cambio puede afectar o no a algún otro sistema. Cuando se ha determinado que una modificación puede resultar en esta situación, se les indica a las personas encargadas de los sistemas que puedan ser afectados. Esto puede hacerse realizando reuniones donde se indique cuales serán los cambios, con el fin de que cada quien evalúe si los sistemas bajo su cargo no requieren también ser modificados. En otras ocasiones, se realiza el cambio y se solicita al resto del grupo que prueben sus sistemas para ver si estos no resultaron afectados. Otro caso puede ser que el técnico sepa cual sistema puede ser afectado y se lo indique directamente a la persona encargada del mismo. Sin embargo, pueden darse casos en los que se sepa que un sistema resulta afectado solo después de que ya se han liberado las modificaciones.

Los técnicos por lo general son quienes realizan las modificaciones a los sistemas que manejan, pero en ocasiones se requiere de un mayor número de personas; en estos casos es posible proporcionar el apoyo de personal externo, como otros técnicos, becarios o personal por honorarios. Cuando se da esta situación, el técnico es el que indica al resto de los programadores lo que tienen que hacer. El técnico encargado del sistema es el que mantiene la responsabilidad, aun cuando no sea él el que implementa las modificaciones.

Cabe señalar que todos los cambios realizados, tanto en la base de datos como en el código, se hacen a nivel de desarrollo, el cual es un nivel de pruebas, con el fin de no afectar las labores de los usuarios con el sistema (nivel de producción). Los módulos y la base de datos a los que accede el usuario son actualizados hasta que las modificaciones han sido probadas.

Administración de la Base de Datos

Todos los sistemas mantienen su información en una base de datos centralizada, la cual está administrada por uno de los técnicos del DI que juega el rol de Administrador de la Base de Datos (DBA). El DI mantiene una copia de ésta en su área de desarrollo. Con el fin de mantener esta copia actualizada, cada semana se realiza un respaldo de la base de datos de producción, y se copia en desarrollo.

Cuando alguna solicitud de mantenimiento requiere de modificaciones a la base de datos, el técnico encargado las realiza en la base de datos del área de desarrollo, una vez que éstas han sido probadas, y el técnico se ha convencido de que son correctas, le informa al DBA, generalmente por medio de un correo electrónico, que realice dichos cambios en la base de datos del área de producción. Para esto, el técnico debe indicar que tipo de cambios se hicieron (se eliminó o creo una tabla, se agregaron o quitaron campos a una tabla, etc.), así como los datos necesarios para realizar las modificaciones en la base de datos de producción. El DBA mantiene un registro o bitácora de todos los cambios que le son solicitados por los técnicos.

Proceso de liberación de los sistemas modificados

El DI tiene como política manejar las modificaciones de los fuentes en tres capas (figura 66): la primera corresponde al trabajo que cada técnico hace en su propia máquina; la segunda es un área de pruebas (conocida como desarrollo) donde las modificaciones son probadas sin perjuicio de las versiones de los sistemas que se encuentran en funcionamiento; y la tercera es la de liberación (conocida como producción), que es donde se almacenan los sistemas e información para que posteriormente sea accesible para los usuarios.



Figura 66. Modelo de capas en el control de versiones en el DI del CICESE. La parte superior muestra las capas, mientras que la parte inferior el nivel de acceso que tiene cada capa.

Cuando un técnico desea hacer modificaciones al código de algún módulo, forma, reporte, etc., del sistema que maneja, tiene que bajar a su computadora personal la última versión de los archivos que va a modificar. Estos archivos se encuentran en un servidor, y el acceso a los mismos está protegido mediante contraseñas. Sólo los técnicos encargados de un determinado sistema tienen acceso a los archivos fuente del mismo. Cada técnico tiene asignada una carpeta dentro del servidor de archivos, donde se localizan los archivos fuente de los sistemas que están a su cargo.

Una vez que el técnico ha realizado los cambios en los archivos fuente, los sube al área de desarrollo para probar que no existan errores. Cuando se ha comprobado que no existen errores y que las modificaciones hechas están correctas, se copian los archivos fuente y los ejecutables modificados en una carpeta preestablecida en el servidor de archivos. Posteriormente se le indica al administrador de archivos, mediante un correo electrónico, cuales son los archivos que fueron modificados para que éste los copie al área de producción.

En los casos en que varias personas están trabajando con la modificación de algún sistema, el técnico responsable del sistema es el que les proporciona los archivos fuente al resto de los programadores. Una vez que estos han hecho sus modificaciones se los pasan al técnico

para que éste los suba a desarrollo o los envíe al administrador de archivos para que sean enviados a producción, según sea el caso.

Por último, el administrador de archivos pone la nueva versión del sistema en un servidor local, del cual es tomada y distribuida al resto de los servidores de aplicaciones. Esto se hace de una manera automática. Existen solo dos sitios donde la instalación de nuevas versiones de los sistemas debe hacerse de manera manual, por lo que, por lo general, las modificaciones son enviadas por correo electrónico para que sean actualizados los sistemas en estos dos sitios. En la figura 67 se muestra una aproximación al proceso aquí mencionado. Este proceso da una idea de la manera en que se lleva el control de versiones. El mecanismo de liberación de las nuevas versiones queda englobado en la tarea “Actualizar versiones en producción”, la cual es realizada por el administrador de archivos.

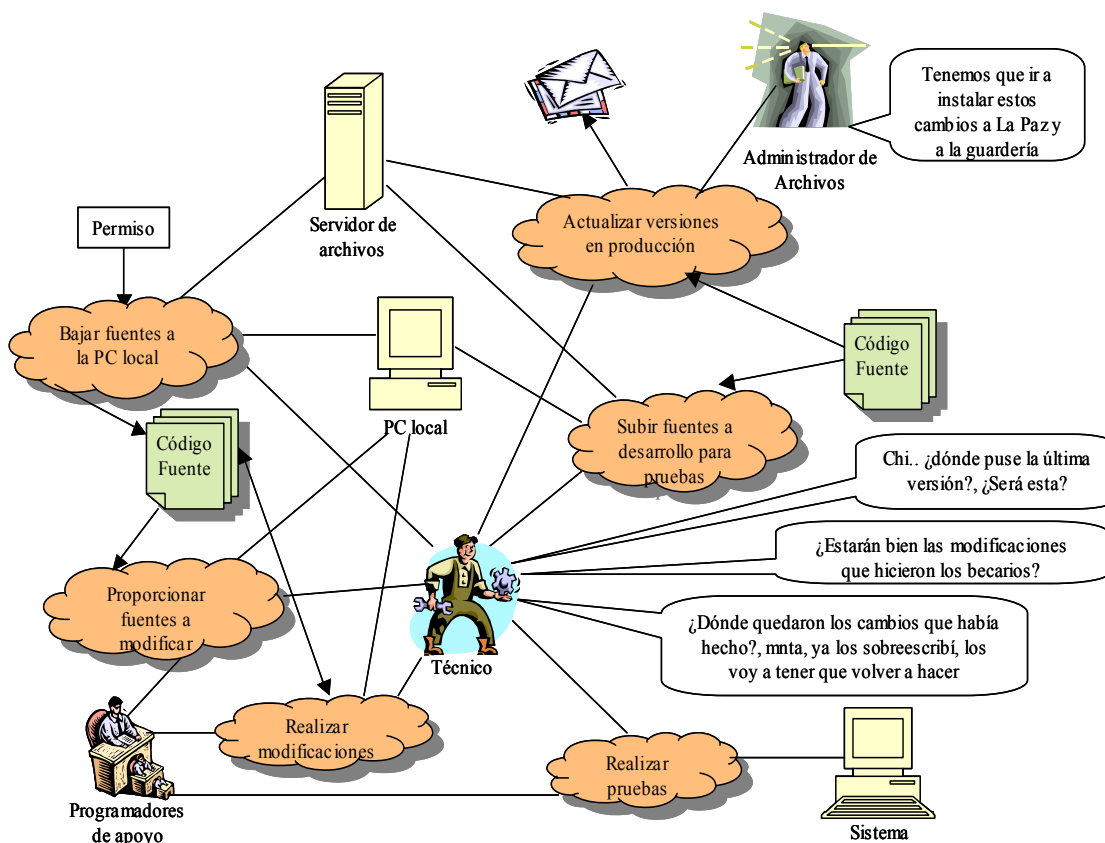


Figura 67. Proceso que muestra la manera en que se lleva el control de las versiones de los archivos fuente.

Es preciso añadir que la responsabilidad de que la versión que se encuentra en producción sea la más reciente, recaerá en el técnico responsable del sistema, ya que es él quien conoce cuáles archivos han sido modificados, y es él quien debe informar al administrador de archivos una vez que las modificaciones hayan sido desarrolladas y probadas. Debido a esto, existe el riesgo de que el técnico responsable olvide subir las nuevas versiones, informar al administrador de archivos, o que se confunda y suba versiones anteriores o incorrectas, ya que no existen en el DI herramientas para controlar esto de manera automática.

En esta sección se ha presentado la descripción de los principales procesos realizados por el DI, identificados durante el caso de estudio. En la siguiente sección se presentan los datos que corresponden con lo que se observó en el caso de estudio en Intersel.

A.2 Caso de estudio: Intersel

Intersel es una empresa dedicada al desarrollo de software de administración telefónica. Actualmente desarrolla y mantiene diversos productos de software, donde el principal es el sistema Intertel, el cual tiene un tiempo de vida de más de 10 años.

Intersel consta de cinco departamentos (figura 68), los cuales se componen del departamento de administración, encargado de las tareas de gerencia, contabilidad, recepción y administración; ventas, que se encarga de la comercialización de los distintos productos manejados por la empresa; soporte técnico, encargado de la atención a los distribuidores, clientes y usuarios del producto; producción, que realiza las copias del software, así como el empaquetado de las mismas; y por último, el departamento de desarrollo, donde se llevan a cabo las tareas de desarrollo y mantenimiento de los sistemas de software.

Departamento de Ventas	Departamento de Soporte Técnico	Departamento de Producción	Departamento de Administración
Departamento de Investigación y Desarrollo			

Figura 68. Organización interna de Intersel

En el área de desarrollo, se encuentran tres sub-áreas: una encargada de las labores propias del desarrollo y mantenimiento de sistemas, otra encargada de la documentación para los usuarios del sistema, y otra más que se encarga de mantener al día las tarifas telefónicas utilizadas por los sistemas de administración telefónica desarrollados por el mismo. En el departamento de desarrollo se cuenta con diversas computadoras tipo PC. El equipo de cómputo principal del departamento se constituye de una PC para cada uno de los integrantes del equipo, un servidor principal, donde se almacenan los archivos fuente de los distintos sistemas a los que se da mantenimiento, así como copias de las versiones más recientes. Y un servidor de Internet dedicado.

Los sistemas manejados por el departamento han sido desarrollados para plataformas Windows. Cuyo desarrollo se ha hecho principalmente en los lenguajes, FoxPro, C y C++, y más recientemente bajo la plataforma de desarrollo Visual Studio de Microsoft (Visual FoxPro, Visual Basic y Visual C++). Actualmente también se encuentran desarrollando aplicaciones para Internet bajo las herramientas que proporciona Microsoft, particularmente bajo el lenguaje de páginas activas (ASP) y Visual Basic. Por otro lado, para las bases de datos de los sistemas manejados por el departamento se utiliza tanto FoxPro, como SQL Server.

Al momento de la realización del caso de estudio, el departamento de desarrollo de Intersel estaba constituido de 8 personas; el jefe del departamento, cinco ingenieros de software, una persona encargada de la documentación de usuarios de los sistemas desarrollados por el departamento, y otra encargada de mantener al día las tarifas telefónicas utilizadas por estos mismos sistemas.

A.1.2 Descripción de los procesos de Intersel

La captura de los procesos llevados a cabo por el personal de Intersel se enfocó en cuatro aspectos principales: 1) la manera en que se da atención a los clientes, 2) cómo se realizan las solicitudes de modificaciones, 3) la forma en que se realiza la implementación de los cambios y el control de las versiones, y 4) el proceso de liberación de nuevas versiones. A continuación se detalla cada uno de estos procesos.

Atención a usuarios y clientes

La atención de los usuarios y clientes en Intersel se lleva a cabo mediante un esquema basado en estratos. Debido a que Intersel distribuye sus productos a una gran variedad de lugares, y organizaciones, maneja una serie distribuidores encargados de conseguir a los clientes, y dar atención directa a estos. A través de estos distribuidores es que se da la comunicación entre los clientes e Intersel. Además de lo anterior, Intersel cuenta con un departamento dedicado exclusivamente a recibir las solicitudes de atención de estos distribuidores.

La figura 69 muestra el flujo de comunicación que se da, de los ingenieros de mantenimiento al usuario y viceversa. Se puede ver que esta comunicación se maneja por estratos, donde en la base están los ingenieros encargados de desarrollar y mantener los sistemas. La siguiente capa la forman los encargados de ventas y soporte técnico. Por un lado, el departamento de ventas se encarga de hacer llegar las nuevas versiones a los usuarios, lo cual hace a través de distribuidores, estos a su vez tratan con alguna persona dentro de la organización que compra el producto o la actualización del mismo, y dentro de la cual existe alguien encargado de utilizar el sistema.

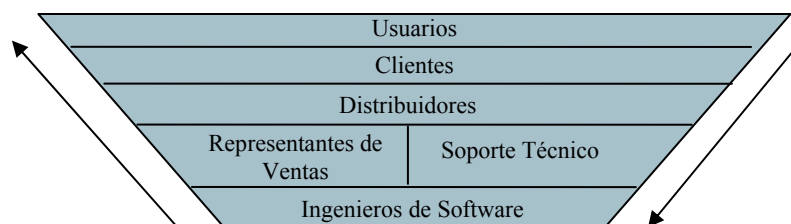


Figura 69. Flujo de comunicación hacia y desde el usuario en Intersel.

Para dejar un poco más claro el proceso aquí mencionado, se presenta un escenario típico de atención al cliente, el cual es mostrado en la figura 70. En este caso hemos agrupado al cliente como al usuario en un solo tipo de actor, denominado cliente. La figura muestra como el cliente hace su solicitud al distribuidor que le vendió el sistema. Si el distribuidor puede solucionar el problema lo hace inmediatamente, en caso de no ser así, se comunica con el departamento de soporte técnico de Intersel para que se le de la solución. Posteriormente, la solución sigue la misma ruta, del departamento de soporte al distribuidor, y de éste al cliente.

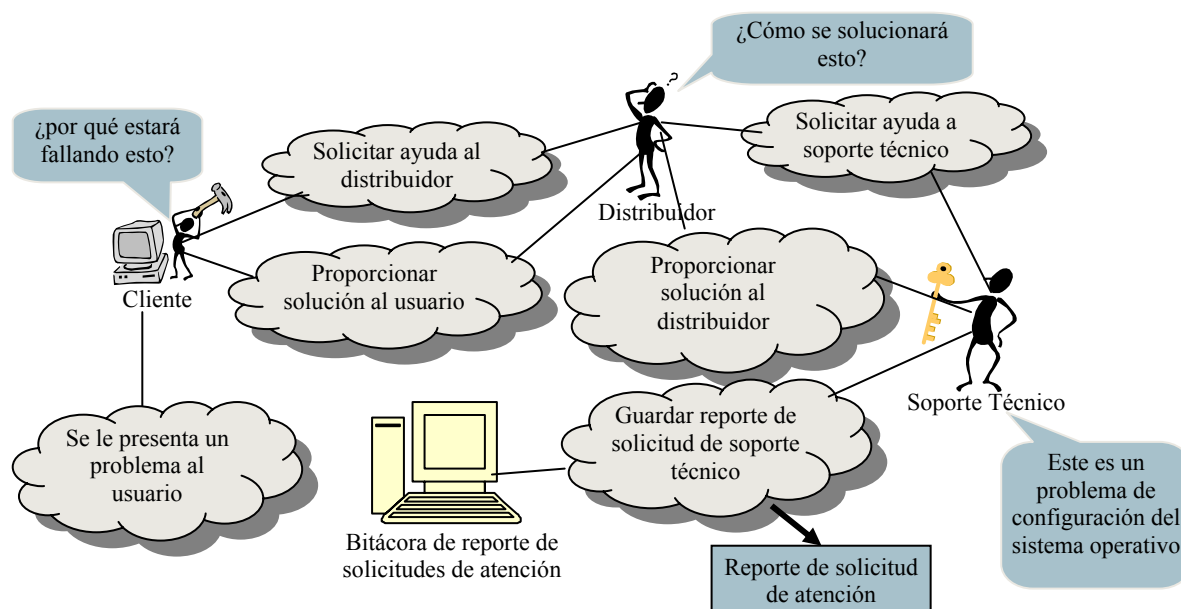


Figura 70. Proceso de atención a usuarios en Intersel

Es importante mencionar que los encargados de soporte técnico mantienen una bitácora de las solicitudes atendidas, por lo que cualquier solicitud de atención, por insignificante que ésta sea, es almacenada. Esta bitácora es utilizada para identificar los casos más comunes, así como las soluciones que se dieron a estos. Con base en estos datos, se desarrollan programas de inducción y capacitación para los nuevos miembros del departamento de soporte, así como también para los distribuidores. Otra de las aportaciones para las que ha servido esta bitácora, es la realización de un sistema de preguntas frecuentes que puede ser accedido, a través de la página de Internet de Intersel, por cualquier usuario o cliente.

Solicitud de modificaciones

Las solicitudes de cambios en Intersel surgen de una serie de ideas que pueden provenir de cualquiera de los miembros de la empresa. Para esto, se ha implementado un mecanismo para compartir ideas, conocimiento e intereses comunes. En este sistema, cada uno de los miembros de la empresa puede suscribirse a una serie de listas de correo y carpetas utilizadas para compartir ideas entre ellos. En una de estas carpetas compartidas es donde los miembros de la empresa pueden aportar ideas para la mejora de los distintos productos que manejan. Estas ideas pueden provenir de comentarios hechos por los distribuidores o clientes, por problemas detectados en los sistemas, por las características de los productos de la competencia, en fin, por una gran cantidad de factores.

Cuando se decide desarrollar una nueva versión del producto, los líderes de las distintas áreas se reúnen para determinar las características que contendrá la misma (como se muestra en la figura 71). Una vez que se ponen de acuerdo, se genera un documento con los requerimientos de la nueva versión, el cual es firmado por todos los involucrados y enviado al departamento de desarrollo. Una vez hecho esto, el jefe del departamento se reúne con la persona que se hará responsable del proyecto y demás ingenieros de software que participarán en las modificaciones. Esto se hace con el fin de determinar las actividades que requerirán ser realizadas para cubrir los requerimientos planteados, así como para determinar los tiempos que estas actividades consumirán.

Posteriormente se crea un proyecto en el cual se plasman estas actividades con sus tiempos requeridos, así como la persona responsable de llevar a cabo cada una de ellas. La asignación de las actividades se hace tomando como criterio la disponibilidad en tiempo, la carga de trabajo, así como la experiencia de cada ingeniero de software con los módulos que serán modificados. El líder del proyecto también realiza un documento donde establece las características del proyecto, sus objetivos, alcances y recursos que serán requeridos. Se identifican los principales riesgos que pudieran presentarse, así como la manera en que

serán abordados en caso de que se presenten. La figura 72 muestra una representación gráfica de este proceso.

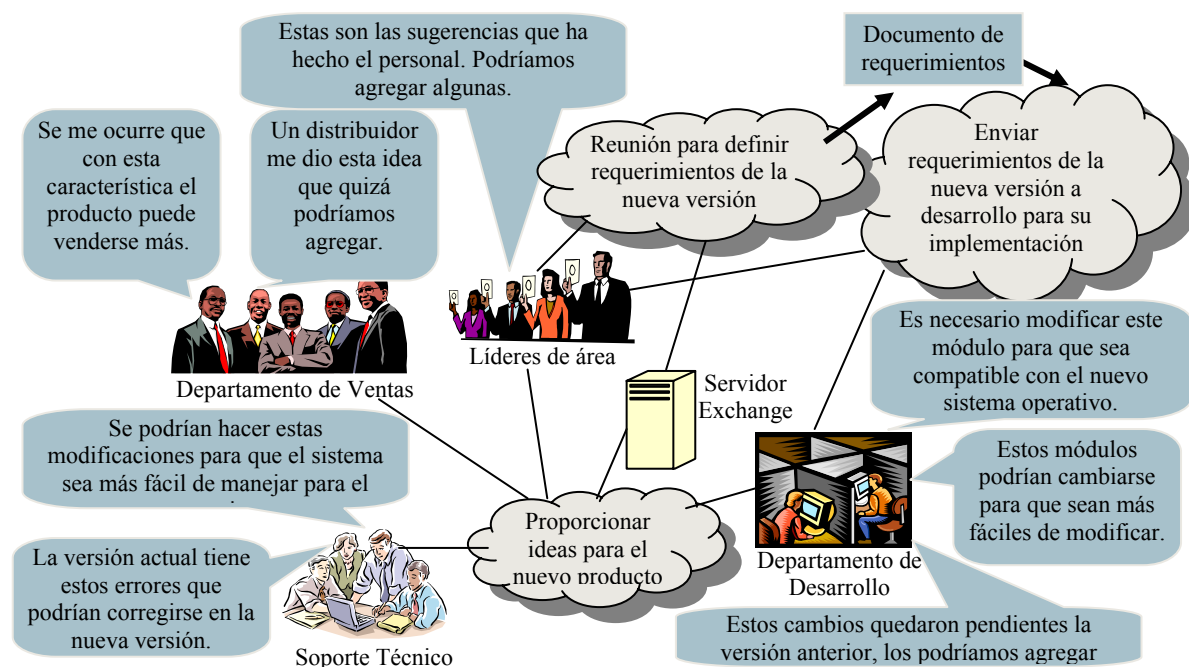


Figura 71. Proceso de realización de una solicitud de modificaciones en Interseal

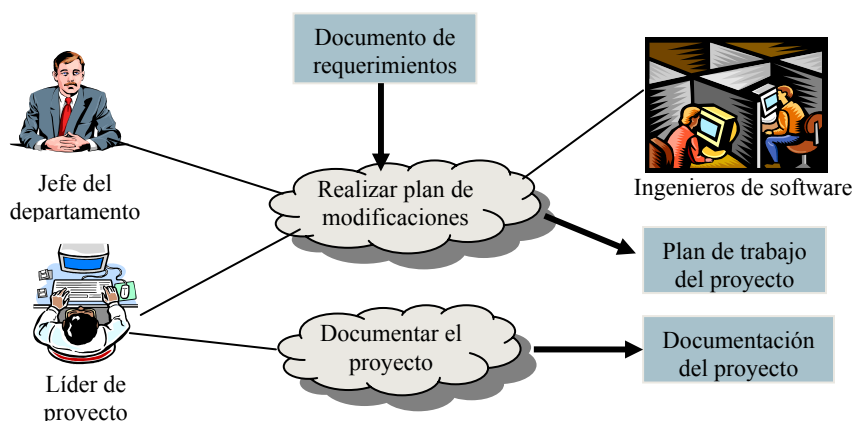


Figura 72. Elaboración del proyecto de modificaciones en Interseal

Cuando ya ha sido realizado el documento, se presenta a los líderes de área para su aprobación. Si el documento es aprobado, se inician las modificaciones, de lo contrario, se negocia con las personas que no estén de acuerdo con las condiciones. Estas negociaciones pueden resultar por la necesidad de terminar el proyecto en un menor tiempo o con una

menor cantidad de recursos, entre otros factores. Con frecuencia, en estos casos, la solución resulta ser la disminución de los requerimientos a cubrir. Si durante la realización del proyecto alguna de las áreas decide agregar o eliminar algún requerimiento, es necesario que esto quede por escrito y que cada uno de los involucrados firme de enterado. Este documento se anexa al documento de requerimientos inicial.

Implementación de cambios y Control de versiones

El departamento de desarrollo de Intersel maneja el programa Visual SourceSafe (VSS) de Microsoft para el control de las versiones de los archivos fuente de los sistemas a los que dan mantenimiento. Todos los archivos fuente se encuentran guardados dentro de un servidor, donde se están organizados por producto y versión. Cada uno de los miembros de grupo tiene un nombre de usuario y clave por medio de la cual puede acceder a estos archivos. El acceso a éstos se realiza mediante el programa cliente del VSS.

Cuando un ingeniero en Intersel requiere modificar algún archivo, solicita al programa VSS que le proporcione permisos de escritura sobre el archivo, en caso de que este archivo esté siendo modificado por alguien más, el programa se lo informa, de manera que es posible evitar, en la mayoría de los casos, que dos personas modifiquen un archivo al mismo tiempo. De esta manera reduce el riesgo de que al subir los archivos modificados se sobre escriban unos a otros. Sin embargo, es posible que se presente esta situación, para lo que el programa mencionado maneja un historial de los cambios realizados a cada archivo, desde la persona que lo hizo, la fecha y las diferencias en el código fuente de una versión a otra.

El proceso de implementación de cambios inicia una vez que se ha definido el plan de proyecto, y se han asignado las tareas. El plan de proyecto se encuentra en un archivo creado en el programa Project de Microsoft, en él se especifican las tareas, el tiempo programado para realizar cada una de ellas, la prioridad, el orden de dependencia entre las tareas, si es que existe, y la persona encargada de llevarlas a cabo. Los ingenieros acceden a este archivo para ver cuales son las tareas que tienen asignadas y eligen la que atenderán. Posteriormente identifican qué partes del sistema requerirán ser modificadas. Cuando

tienen la suficiente experiencia, directamente saben que archivos serán los que posiblemente requerirán cambios, de no ser así, en ocasiones hacen un seguimiento al código fuente, al programa ejecutable, o consultan con alguno de sus compañeros que haya trabajado con anterioridad en el módulo que se requiere modificar. Una vez que se han identificado los archivos a modificar, solicitan al programa VSS que les de permiso de escritura sobre dicho archivo, y proceden a realizar las modificaciones. Cuando las modificaciones han sido llevadas a cabo, y se ha probado que el programa funcione correctamente, se actualizan los archivos fuente en el servidor.

Liberación de nuevas versiones

Cuando se llega la fecha de liberar la nueva versión de un producto, se asigna a un grupo de los miembros del departamento de desarrollo para realizar una etapa de pruebas “Alfa” sobre el producto; a este grupo por lo general también se le asigna un miembro del departamento de soporte técnico. El departamento de desarrollo tiene definido un plan de pruebas para los principales módulos de que se compone el producto a probar, en caso de que se haya agregado un nuevo módulo, también se define un conjunto de pruebas para el mismo. Las pruebas se realizan tomando como guía estos planes de prueba. Si durante las pruebas surgen errores en el sistema, estos se capturan en una carpeta compartida con un formato en el cual se define el producto donde se dio el error, la versión del producto, el módulo, quien realizó la prueba, una descripción del error, la severidad del mismo, así como algunos datos que puedan ayudar a reproducirlo.

Al final de la etapa de pruebas se evalúan los errores que se hayan detectado y se genera un plan de proyecto para solucionarlos. Una vez solucionados los errores se realiza otra etapa de pruebas y así sucesivamente hasta que no se detecten errores graves. Cada período de pruebas por lo general consume de dos a cuatro vueltas.

Una vez finalizada la etapa de pruebas Alfa, se realiza otra etapa de pruebas llamada “Beta”. Esta etapa de pruebas tiene la finalidad de verificar que el sistema funciona correctamente en un ambiente real, por lo que se eligen ciertos clientes de Intersel que estén

dispuestos a participar para probar la nueva versión del sistema. Para esta serie de pruebas, también se asigna un grupo de personas de Intersel, tanto del departamento de desarrollo como de soporte técnico. Este grupo se encarga de instalar el sistema en las organizaciones donde se realizarán las pruebas, y, durante unos días, se dedican a observar la manera en que se comporta el sistema en estos ambientes. Durante estas observaciones, se toma nota de los comentarios, dudas o sugerencias de los usuarios, así como de las fallas que se vayan detectando.

Una vez terminado el período de pruebas “Beta”, se evalúan las fallas detectadas. Finalmente se determina si el sistema está listo para su liberación o si es necesario corregir algunas de las fallas detectadas antes de liberarlo.

Apéndice B. Análisis y Diseño del Prototipo

Este apéndice presenta aspectos del análisis y diseño del prototipo. Específicamente se muestran algunos elementos que no fueron considerados en el capítulo V, como algunos casos de uso y diagramas de secuencia, así como algunos diagramas de clases y el diccionario de clases.

B.1 Diagramas de casos de uso

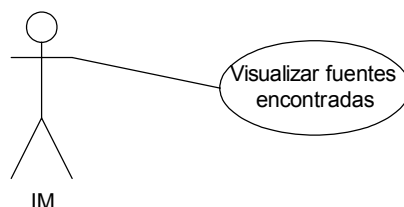


Figura 73. Caso de uso Visualizar fuentes encontradas.

Caso de uso: Visualizar fuentes encontradas.

Actor: Ingeniero de mantenimiento (IM).

Objetivo: Permitir al ingeniero de mantenimiento ver las fuentes de conocimiento que fueron encontradas.

Condición inicial: El agente manejador de conocimiento (AMC) debió haber realizado la búsqueda de fuentes de conocimiento, e informado al agente de personal del total de fuentes de conocimiento encontradas.

Descripción: El caso de uso inicia cuando el IM decide ver las fuentes que fueron encontradas por el sistema. Para esto, el IM presiona el botón “Ver fuentes...”, el agente de personal envía un mensaje al AMC solicitándole que muestre la interfaz gráfica donde presenta las fuentes de conocimiento que fueron encontradas. Al recibir el mensaje, el AMC despliega dicha interfaz gráfica, en la cual se muestran las fuentes de conocimiento encontradas, ordenadas por el tipo de fuente al que pertenecen.

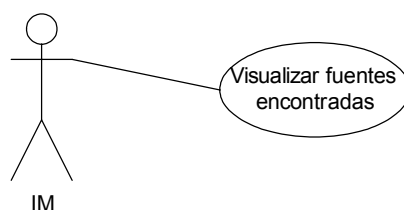


Figura 74. Caso de uso Visualizar datos de fuentes encontradas.

Caso de uso: Visualizar datos de fuentes encontradas.

Actor: Ingeniero de mantenimiento (IM).

Objetivo: Permitir al IM ver los datos de las fuentes de conocimiento que fueron encontradas.

Condición inicial: La interfaz gráfica del AMC debió haberse desplegado.

Descripción: Este caso de uso tiene dos variantes, la primera es cuando el IM visualiza los datos de la referencia a la fuente de conocimiento, los cuales son la localización de la fuente, así como el tipo de conocimiento que tiene; y la segunda variante es cuando el IM visualiza los datos de la fuente de conocimiento, por ejemplo, los datos de un reporte de error. El primer caso se da cuando el IM selecciona una de las fuentes de la lista de fuentes de conocimiento encontradas, en este momento se le despliegan al IM los datos de la referencia a dicha fuente, posteriormente, cuando el IM oprime el botón “Mostrar datos de fuente”, el AMC envía un mensaje al agente manejador de fuentes de conocimiento (AMFC) donde le solicita que muestre los datos de la fuente al IM. Cuando el AMFC recibe dicho mensaje, recupera los datos de la fuente de conocimiento y los despliega al IM mediante una ventana.

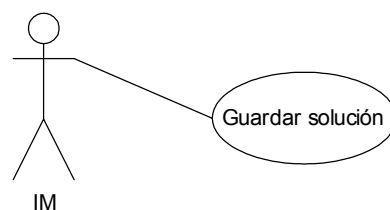


Figura 75. Caso de uso Guardar solución.

Caso de uso: Guardar solución.

Actor: Ingeniero de mantenimiento (IM).

Objetivo: Permitir al IM guardar los datos de las soluciones dadas a los errores reportados.

Condición inicial: El IM debe haber solicitado atender un proyecto, y el sistema debe haber presentado la ventana con los datos del reporte de error que dio origen al proyecto elegido.

Descripción: Este caso de uso inicia cuando el IM decide capturar la solución dada al error reportado. Para hacer esto, el IM presiona el botón “Asignar solución” en la ventana de datos del reporte de error, esta acción provoca que se despliegue una ventana para la captura de los datos de la solución. Una vez que el IM termine de capturar los datos, presiona el botón “Aceptar”, esto cierra la ventana de captura de datos de la solución. Posteriormente el IM presiona el botón de “Aceptar” en la ventana de datos del reporte de error, lo que provoca que se cierre dicha ventana, a la par que se actualizan los datos del reporte de error, agregando los datos de la solución.

A continuación se presentan los diagramas de secuencia de los casos de uso aquí descritos.

B.2 Diagramas de secuencia

En esta sección se presentan los diagramas de secuencia de los casos de uso arriba mostrados. Estos diagramas de secuencia siguen la misma norma utilizada con los presentados en el capítulo V, en los cuales existen tres tipos de comunicación entre los elementos u objetos del diagrama, como se ilustra en la figura 76.

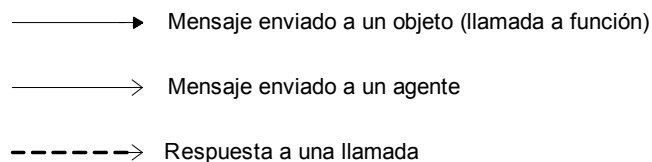


Figura 76. Tipos de comunicaciones usadas en los diagramas de secuencia.

En la figura 77 se ilustra el diagrama de secuencia del caso de uso Visualizar fuentes encontradas. En la figura se observa que el IM Juan solicita ver las fuentes de conocimiento

que fueron encontradas, lo cual hace mediante la interfaz gráfica de usuario del agente de personal (i1), esto genera un evento que es capturado por el agente juan, el cual, al detectar esta solicitud del usuario, envía un mensaje al agente juan_km solicitándole que muestre su interfaz gráfica, en la cual se presentan las fuentes de conocimiento encontradas. Cuando el agente juan_km recibe este mensaje, muestra su interfaz gráfica (i2), la cual le presenta al IM Juan las fuentes de conocimiento encontradas.

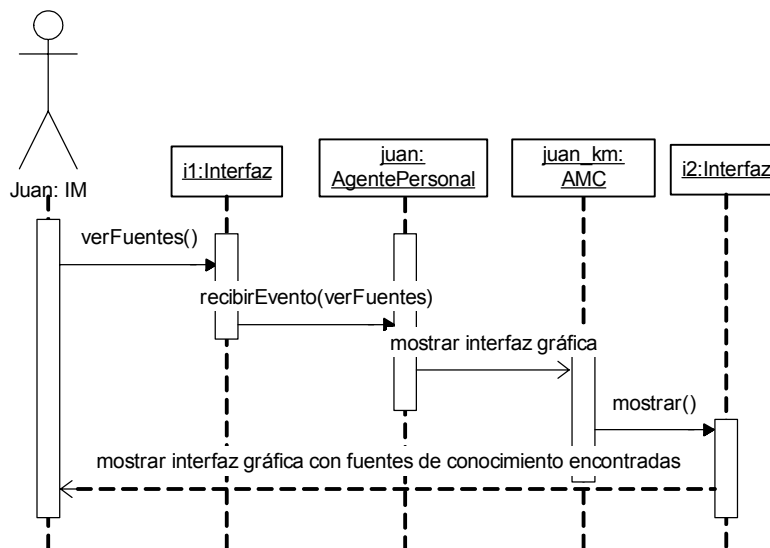


Figura 77. Diagrama de secuencia del caso de uso Visualizar fuentes encontradas.

La figura 78 muestra el diagrama de secuencia del caso de uso Visualizar datos de fuentes encontradas, en el se observa que cuando el IM Juan elige una fuente de la lista, se genera un evento que es capturado por el agente juan_km, el cual, muestra en la interfaz gráfica (i2) los datos de la referencia a la fuente seleccionada (localización de la fuente, y conocimiento que tiene). Posteriormente, el IM Juan solicita que se le muestren los datos de esta fuente, con lo cual, se genera otro evento que también es capturado por el agente juan_km. Al recibir el evento, el agente juan_km envía un mensaje al agente juan_ksm solicitándole que muestre los datos de la fuente. Al recibir este mensaje, el agente juan_ksm busca los datos de la fuente en la base de datos, para finalmente mostrárselos al IM Juan a través de su interfaz gráfica (i3).

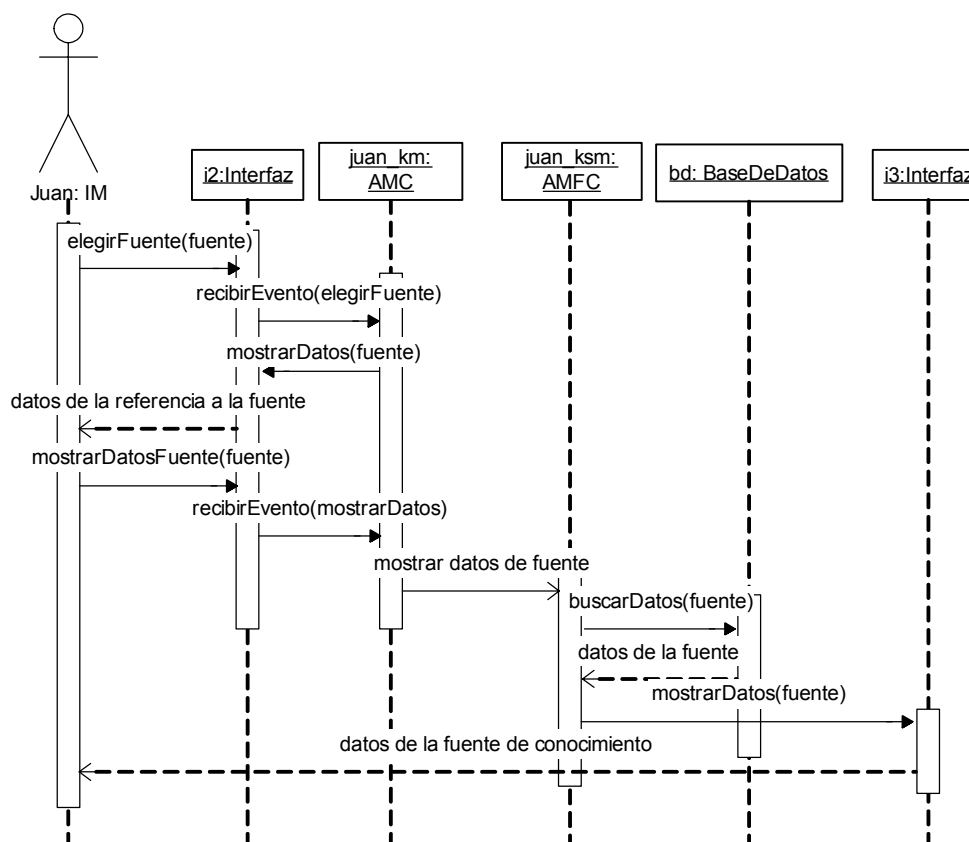


Figura 78. Diagrama de secuencia del caso de uso Visualizar datos de fuentes encontradas.

En la figura 79 se presenta el diagrama de secuencia del caso de uso Guardar solución, en él es posible observar que cuando el IM Juan presiona el botón de “Asignar solución” en la ventana de datos del reporte de error en la interfaz gráfica de usuario, se le muestra una ventana para la captura de los datos de la solución. Una vez que el IM ha capturado los datos de la solución y elige la opción de “Aceptar”, se cierra la ventana de captura de datos de la solución y se muestra la ventana con los datos del reporte de error. Cuando el IM elige la opción “Aceptar” en ésta última ventana, la ventana se cierra y se genera un evento que solicita al agente Juan que actualice los datos del reporte de error. Finalmente, al recibir el evento, el agente Juan actualiza los datos del reporte de error en la base de datos.

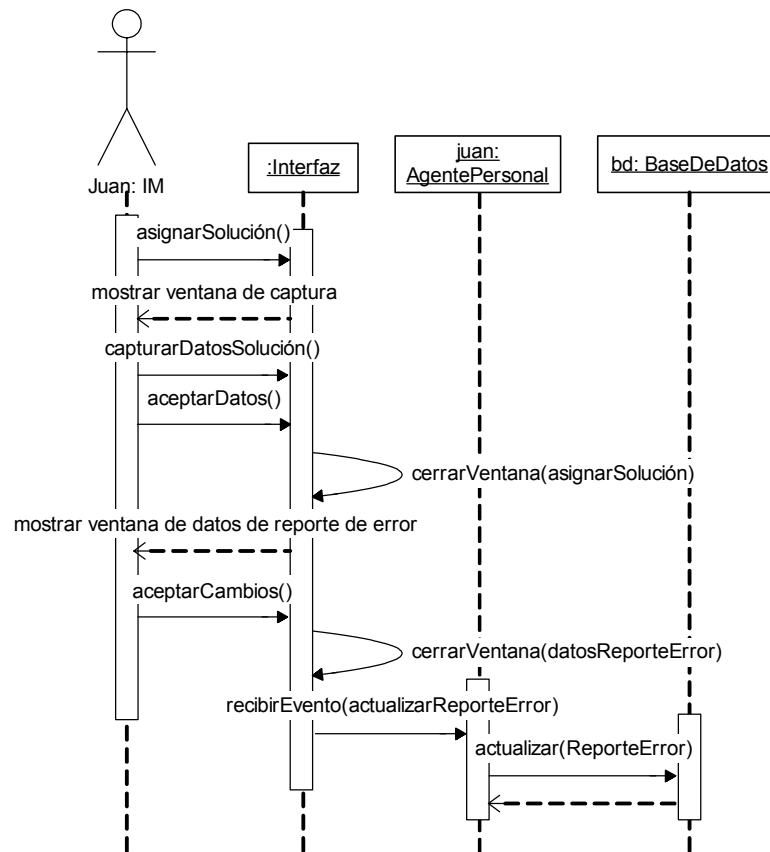


Figura 79. Diagrama de secuencia del caso de uso Guardar solución.

B.3 Diagramas y descripción de clases

En esta sección se presentan los diagramas de clases del prototipo, y una breve descripción de estas clases. Una descripción mas detallada de éstas, así como de sus atributos y funciones, se pueden obtener en la documentación del prototipo, la cual ha sido generada con la herramienta javadoc.

La figura 23 muestra el diagrama de clases del prototipo. Se han agregado dos clases proporcionadas por la plataforma JADE que son parte importante del prototipo, ya que han servido como la base para definir las clases de los agentes y de las actividades que estos realizan. A continuación se describe cada una de las clases mostradas en el diagrama.

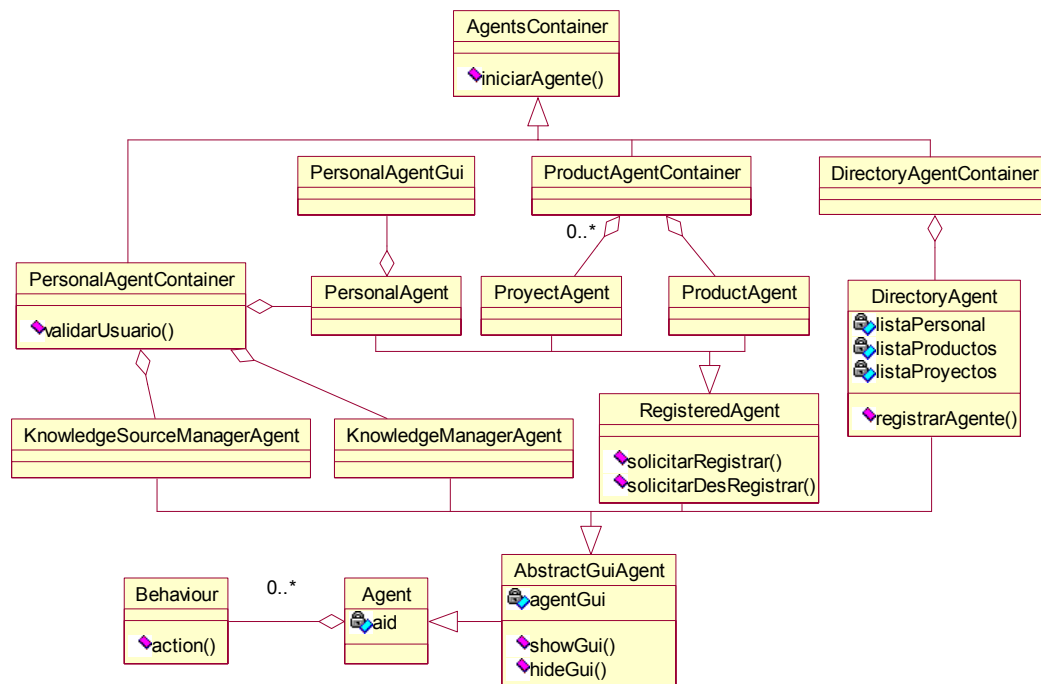


Figura 80. Diagrama de clases del prototipo.

Clase: Agent.

Descripción: esta clase es proporcionada por la API de la plataforma JADE, y es utilizada como la base para crear los agentes utilizados en el prototipo. Su atributo principal es el identificador del agente. Por medio de este identificador es que se hace referencia a los distintos agentes al momento de enviarles mensajes. Sus principales métodos son: send(ACLMessage msg), función utilizada para enviar mensajes a otros agentes; receive() función utilizada para recibir mensajes enviados al agente; y addBehaviour(Behaviour), función utilizada para agregar acciones realizadas por el agente.

Clase: Behaviour.

Descripción: clase proporcionada por la API de la plataforma JADE. Es utilizada como la base para las acciones utilizadas por los agentes del prototipo para enviar y recibir mensajes. Su principal método es action(), función que es llamada para iniciar la ejecución de esta acción.

Clase: RegisteredAgent.

Descripción: esta clase es la base para crear las clases de los agentes que son registrados en el directorio de agentes. Sus principales métodos son: registrar(), que solicita que el agente sea registrado en el directorio de agentes; y desRegistrar() que solicita la eliminación del registro del agente.

Clase: AgentsContainer.

Descripción: Es la clase que sirve como la base para definir los contenedores de agentes. Sus principal función es inicarAgente(), utilizada para inicializar agentes dentro del contenedor.

Clase: PersonalAgentContainer.

Descripción: implementación del contenedor de agente de personal. Se encarga de la validación del nombre y clave de usuario del IM, así como de la inicialización de los agentes que dan soporte al mismo.

Clase: ProductAgentContainer.

Descripción: implementación del contenedor de agente de producto. Cada producto cuenta con un contenedor de este tipo, el cual se encarga de la inicialización del agente de producto, así como de los agentes de los proyectos del producto.

Clase: DirectoryAgentContainer.

Descripción: implementación del contenedor principal. Se encarga de inicializar la plataforma de agentes, así como el agente de directorio.

Clase: PersonalAgent.

Descripción: implementación del agente de personal. Se encarga de monitorear las acciones del IM.

Clase: KnowledgeSourceManagerAgent.

Descripción: implementación del agente manejador de fuentes de conocimiento. Se encarga de recuperar los datos de las fuentes de conocimiento consultadas por el IM.

Clase: KnowledgeManagerAgent

Descripción: implementación del agente manejador de conocimiento. Se encarga de hacer las búsquedas de fuentes de conocimiento.

Clase: ProductAgent.

Descripción: implementación del agente de producto. Lleva el control de los proyectos del producto. Cuando recibe la notificación de que un IM ha ingresado al sistema, revisa si tiene proyectos asignados a este IM, de ser así, notifica a los agentes de estos proyectos que el IM ha sido registrado.

Clase: ProjectAgent.

Descripción: implementación del agente de proyecto. Cuando recibe la notificación de que el IM al que está asignado se ha registrado, envía un mensaje al agente de personal de este IM informándole que este proyecto requiere ser atendido.

Clase: DirectoryAgent.

Descripción: implementación del agente de directorio. Se encarga de recibir solicitudes de registro de los agentes de personal, producto y proyecto. Cuando detecta que el agente que solicita el registro es un agente de personal, envía un mensaje a los agentes de los productos donde el IM al que representa el agente de personal se encuentra trabajando, para notificar que el IM ha sido registrado.

Clase: PrersonalAgentGui.

Descripción: Interfaz gráfica del agente de personal. A través de ella se muestra la lista de proyectos a los que está asignado el IM, así como los datos de los mismos y de los reportes

de error que son atendidos por el IM. También proporciona los medios para capturar los datos de las soluciones dadas a los errores reportados.

En la figura 81 se muestran las clases de las actividades ejecutadas por los agentes, para enviar y recibir mensajes hacia y desde otros agentes respectivamente. Estas clases se describen a continuación.

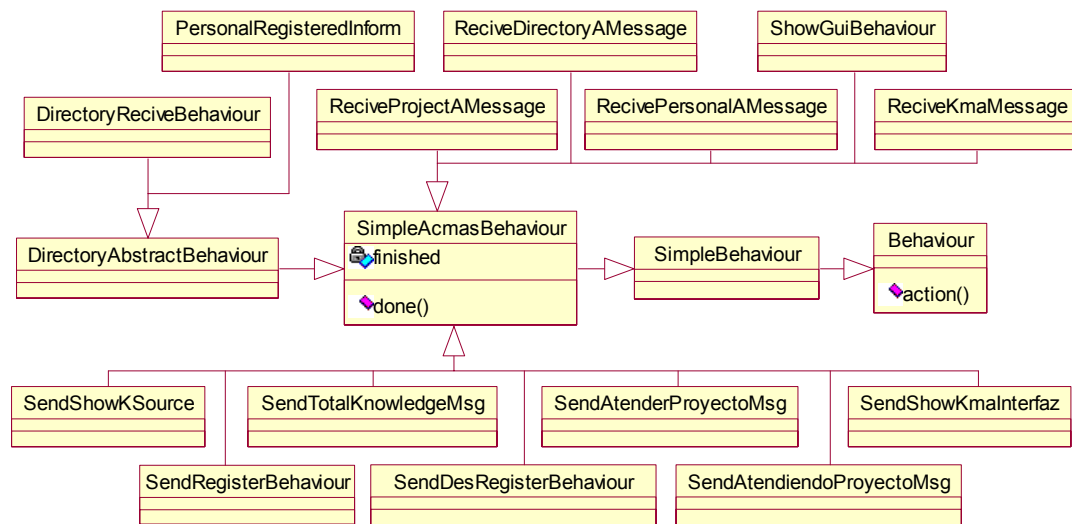


Figura 81. Diagrama de clases de las actividades realizadas por los agentes.

Clase: Behaviour.

Descripción: Esta clase fue descrita anteriormente.

Clase: SimpleBehaviour.

Descripción: Esta clase es proporcionada por la plataforma JADE, y representa un nivel de abstracción un poco más alto que la clase Behaviour. La clase SimpleBehaviour ha sido la base para la implementación del resto de las clases de tipo Behaviour.

Clase: SimpleAcmasBehaviour.

Descripción: Es una clase abstracta que define métodos y atributos comunes a todas las clases tipo Behaviour implementadas en el prototipo.

Clase: DirectoryAbstractBehaviour.

Descripción: Define algunos atributos comunes a todas las clases tipo Behaviour utilizadas por el agente de directorio.

Clase: DirectoryReciveBehaviour.

Descripción: Es la clase utilizada por el agente de directorio para recibir los mensajes que le son enviados.

Clase: PersonalRegisteredInform.

Descripción: Utilizada por el agente de directorio para enviar a otros agentes la notificación de que un agente de personal ha sido registrado.

Clase: ReciveDirectoryAMessage.

Descripción: Esta clase es usada para recibir mensajes enviados por el agente de directorio.

Clase: ShowGuiBehaviour.

Descripción: Usada para recibir mensajes que solicitan el despliegue de la interfaz gráfica de los agentes.

Clase: ReciveProjectAMessage.

Descripción: Utilizada por el agente de personal para recibir los mensajes enviados por los agentes de proyecto.

Clase: RecivePersonalAMessage.

Descripción: Utilizada para recibir mensajes enviados por los agentes de personal.

Clase: ReciveKmaMessage.

Descripción: Utilizada por el agente de personal para recibir los mensajes enviados por el agente manejador de conocimiento.

Clase: SendShowKSource.

Descripción: Utilizada por el agente manejador de conocimiento para solicitar al agente manejador de fuentes de conocimiento que muestre los datos de una determinada fuente.

Clase: SendTotalKnowledgeMsg.

Descripción: Utilizada por el agente manejador de conocimiento para informar el total de fuentes encontradas.

Clase: SendAtenderProyectoMsg.

Descripción: Usada por los agentes de proyecto para informar a los agentes de personal que un proyecto requiere ser atendido.

Clase: SendShowKmaInterfaz.

Descripción: Esta clase la utiliza el agente de personal para solicitar al agente manejador de conocimiento que despliegue su interfaz gráfica.

Clase: SendRegisterBehaviour.

Descripción: Utilizada por los agentes que son registrados en el directorio, para solicitar al agente de directorio que los registre.

Clase: SendDesRegisterBehaviour.

Descripción: Utilizada por los agentes que son registrados en el directorio para solicitar al agente de directorio que elimine el registro de éstos.

Clase: SendAtendiendoProyectoMsg.

Descripción: Utilizada por el agente de personal para informar al agente manejador de conocimiento que el usuario se encuentra atendiendo un proyecto.

define un mecanismo estándar para la definición de atributos, el cual se basa en la clase Atributo.

Clase: Atributo.

Descripción: Esta clase es proporcionada para la definición de los atributos de los conceptos de la ontología. Sus atributos son el nombre del atributo, y el valor del mismo.

Clase: KArea.

Descripción: Utilizada para definir áreas de conocimiento. Los tipos de áreas de conocimiento principales que han sido identificados son: herramientas utilizadas por los ingenieros de mantenimiento (KAHerramienta), lenguajes de programación (KALenguaje), procesos (KAProceso), sistemas operativos (KASistemaOperativo), Bases de datos (KABaseDeDatos), tipos de error (KATipoError), y otros temas (KATema).

Clase: Localizacion.

Descripción: Clase usada para definir la localización de una fuente de conocimiento. Los tipos de localización que se han definido son: física (LocFisica), electrónica (LocElectronica), telefónica (LocTelefono), por medio de un agente (LocAgente), por correo electrónico (LocEmail), o en una base de datos (LocBD).

Clase: Knowledge.

Descripción: Utilizada para definir el tipo y nivel de conocimiento que puede tener una determinada fuente. Sus atributos principales son conocimiento, que define el concepto sobre el cual conoce la fuente; y nivel, que define el nivel de conocimiento que la fuente tiene con respecto al concepto.

Clase: KSource.

Descripción: Utilizada como la base para definir fuentes de conocimiento. Sus atributos principales son knowsAbout, que es una lista de los conceptos sobre los cuales conoce la fuente; y localizacion, que es una lista con las formas o mecanismos por medio de los

cuales se puede localizar esta fuente. Los tipos de fuentes de conocimiento se han dividido en tres clases, los elementos de los productos o sistemas (KSSistema), las personas (KSPersona), y los documentos (KSDocumento).

Clase: KSSistema.

Descripción: Utilizada para definir los elementos que componen a un producto. Esta clase define un atributo que es `neededKnowledge`, que indica los conceptos sobre los cuales se debe conocer para poder trabajar con un elemento del sistema en particular. Los tipos de elementos de sistema se han dividido en: el producto en si (KSSProducto), módulos del producto (KSSModulo), el sistema ejecutable (KSSEjecutable), el código fuente (KSSCodigoFuente), y las bases de datos (KSSBaseDeDatos).

Clase: KSPersona.

Descripción: Utilizada para definir personas que pueden ser fuentes de conocimiento. Se han definido dos tipos de personas: cliente (KSPCliente), y miembros del personal (KSPPersonal).

Clase: KSDocumentacion.

Descripción: Utilizada para definir documentos como fuentes de conocimiento. Los tipos de documentos que han sido definidos son: documentación del sistema (KSDSistema), de usuario (KSDUsuario), técnica (KSDTecnica), reportes de error (KSDReporteError), y solicitudes de mantenimiento (KSDSolicitudMantenimiento).

La figura 83 muestra el diagrama de las clases de la ontología del agente de personal. La clase `Ontology` es proporcionada por la plataforma JADE, así como las interfaces `Concept` y `Predicate`. La descripción del resto de las clases se da a continuación.

Clase: `PersonalAgentOntology`.

Descripción: Clase donde se definen los elementos que componen la ontología.

Clase: AtendiendoReporteError.

Descripción: Sirve para indicar que el agente de personal se encuentra atendiendo un reporte de error. Su principal atributo es el reporte de error que se está atendiendo.

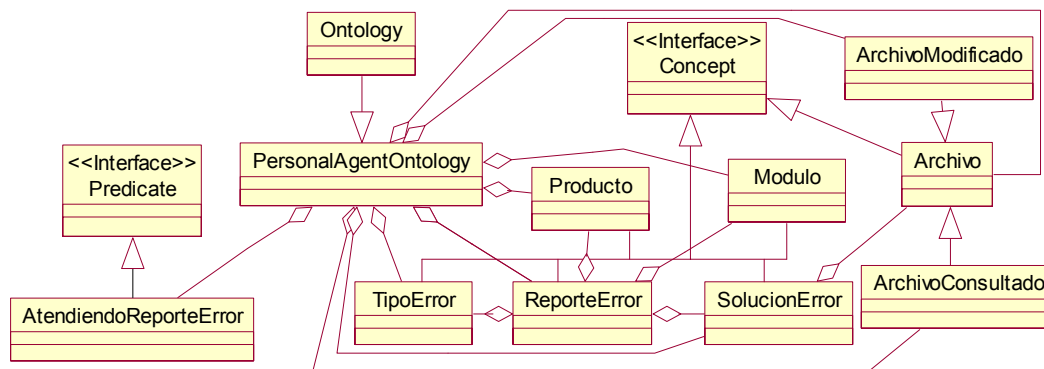


Figura 83. Diagrama de clases de la ontología del agente de personal.

Clase: Archivo.

Descripción: Se utiliza como la base para indicar los archivos que fueron modificados (ArchivoModificado) o consultados (ArchivoConsultado) durante la solución del error.

Clase: TipoError.

Descripción: Utilizada para definir el tipo del error del reporte.

Clase: SolucionError.

Descripción: Contiene los datos de la solución que se le dio al reporte de error.

Clase: ReporteError.

Descripción: Contiene los datos del reporte de error que se está atendiendo.

Clase: Producto.

Descripción: Indica el sistema o producto donde se generó el error reportado.

Clase: Modulo.

Descripción: Indica el módulo donde se generó el error reportado.

La ontología utilizada por el agente manejador de conocimiento fue la definida para los tipos y fuentes de conocimiento, sólo que se agregaron dos clases, las cuales son mostradas en la figura 84. La descripción de estas clases se presenta a continuación.

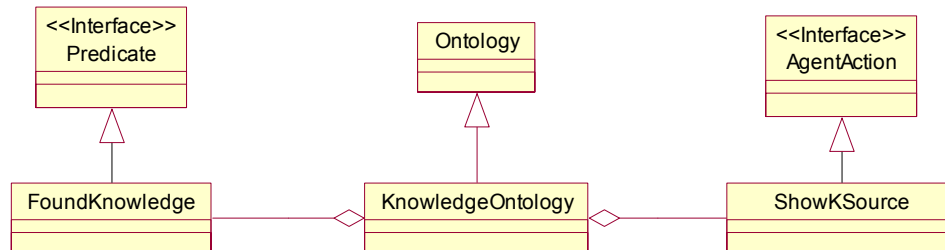


Figura 84. Ontología del agente manejador de conocimiento.

Clase: FoundKnowledge.

Descripción: Se utiliza para informar el total de fuentes encontradas, su principal atributo es total.

Clase: ShowKSource

Descripción: Es utilizada por el agente manejador de conocimiento para solicitarle al agente manejador de fuentes de conocimiento que muestre los datos de una determinada fuente de conocimiento. Su principal atributo es source, el cual es de tipo KSource e identifica la fuente a la que pertenecen los datos que serán mostrados.

Para la ontología manejada por el agente de proyecto sólo se ha definido una actividad mostrada en la figura 85. Las clases se definen a continuación.

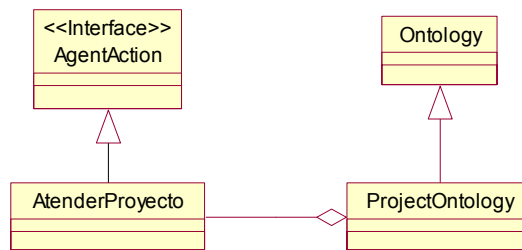


Figura 85. Diagrama de clases de la ontología del agente de proyecto.

Clase: ProjectOntology.

Descripción: Clase que define la ontología del agente de proyecto.

Clase: AtenderProyecto.

Descripción: Es utilizada por los agentes de proyecto para solicitar a los agentes de personal que atiendan un determinado proyecto. Esta clase contiene dos atributos: producto y proyecto, los cuales son usados para definir el proyecto que se solicita sea atendido, así como el producto al que pertenece el proyecto.

Apéndice C. Funcionalidad del Prototipo

En esta sección se presentan los aspectos de la funcionalidad del prototipo que no fueron tratados en el capítulo V. Específicamente se presenta la funcionalidad de las interfaces gráficas de los agentes de directorio y de producto, así como una aplicación que fue desarrollada para apoyar en el manejo de la base de conocimientos.

C.1 Interfaz gráfica del agente de directorio.

La interfaz gráfica del agente de directorio presenta dos funciones básicas, la primera es la visualización de los agentes que han sido registrados, y la segunda es el manejo de los catálogos de miembros del personal, productos, clientes, reportes de error y tipos de error. Para esto, la interfaz está dividida en seis partes, como se muestra en la figura 86, una para la visualización de los agentes registrados, y las otras cinco para cada uno de los catálogos.

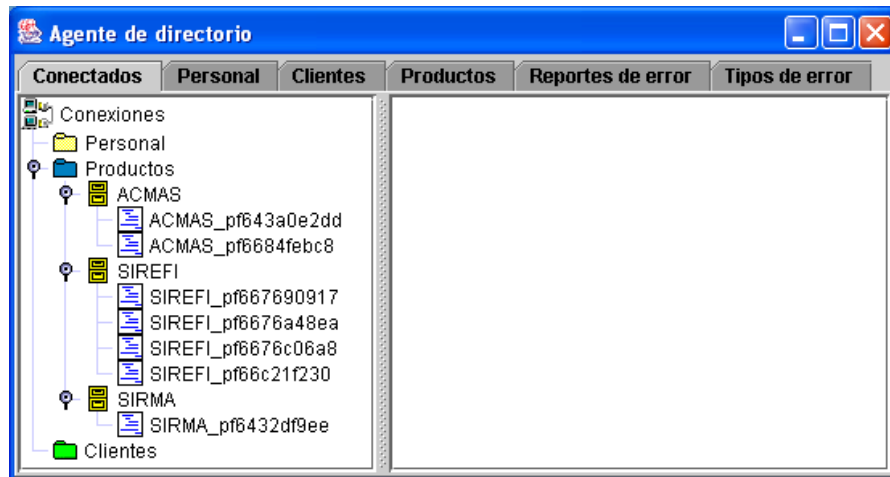


Figura 86. Interfaz gráfica del agente de directorio.

C.1.1 Visualización de los agentes registrados

Los agentes que se encuentran registrados pueden ser visualizados en la pestaña “Conectados”, la cual, como se muestra en la figura 86, es la primera. Esta parte de la interfaz se encuentra dividida en dos grandes áreas, en la parte izquierda de la ventana se

muestra la lista de agentes que están registrados, los cuales se agrupan en Personal para los agentes de personal; Productos para los agentes de producto; y Clientes para los agentes de cliente. Además, los agentes de proyecto se encuentran agrupados dentro del producto al que pertenecen. Por su parte, la parte derecha de la ventana es utilizada para mostrar los datos de los agentes que son seleccionados en la lista, como lo muestra la figura 87.

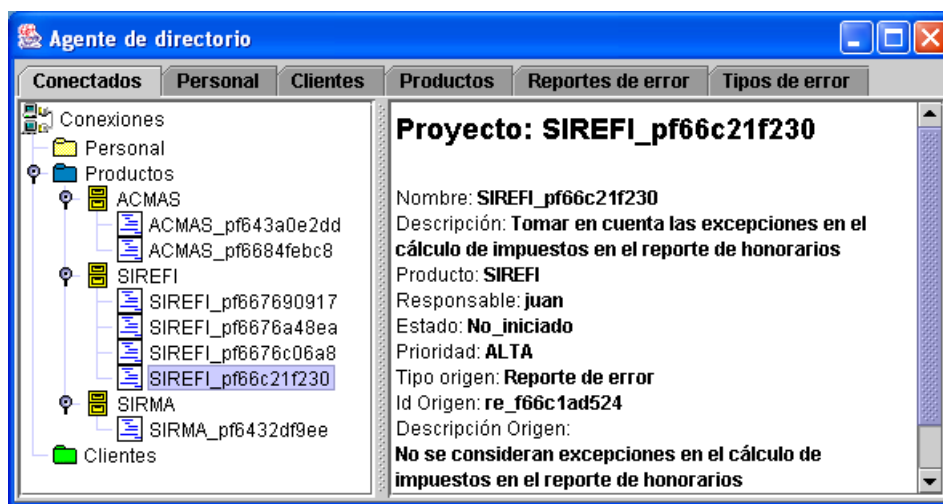


Figura 87. Muestra los datos de los agentes seleccionados.

C.1.2 Manejo de catálogos

Debido a que el manejo de catálogos es muy similar, la funcionalidad básica de esta parte solo se ejemplificará con el catálogo de miembros del personal, el cual se muestra en la figura 88. Como es posible observar en la figura 88, la ventana se compone de tres partes principales, en la parte izquierda se muestra una lista con los miembros del personal que han sido dados de alta en el catálogo; en la derecha se presenta un área donde son desplegados los datos de los miembros del personal que son seleccionados en la lista; y en la parte inferior se presentan tres botones que permiten crear un nuevo elemento, o modificar o borrar uno ya existente.



Figura 88. Muestra el catálogo de miembros del personal.

Al presionar el botón “Nuevo” se abre una ventana para capturar los datos del miembro del personal, la cual es mostrada en la figura 89. Una vez que se han capturado los datos, se presiona “Aceptar” para que estos sean guardados en la base de datos. Posteriormente el nuevo miembro del personal es agregado a la lista de la ventana del catálogo (figura 88).

The image shows a form window titled "Datos de personal". It contains several input fields for data entry:
Login:
Nombre:
Apellido:
E-mail:
Teléfono:
Dirección:
At the bottom, there are two buttons: "Aceptar" and "Cancelar".

Figura 89. Ventana de captura de datos de miembros del personal.

Para la función de “Modificar”, el proceso es similar al anterior, solo que al abrirse la ventana de captura se muestran los datos del miembro del personal que se encuentre seleccionado en la lista.

Por otro lado, al presionar el botón “Borrar”, primeramente aparece un mensaje solicitando que se confirme esta acción (figura 90), si se presiona “Aceptar”, el miembro del personal que esté seleccionado en la lista, será borrado de la base de datos y de la lista del catálogo.

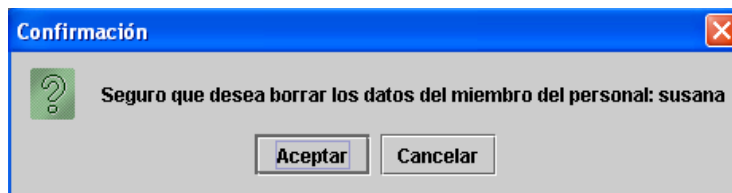


Figura 90. Confirmación de la eliminación de un miembro del personal, del catálogo.

C.2 Interfaz gráfica del agente de producto.

La interfaz gráfica del agente de producto, mostrada en la figura 91, presenta dos funcionalidades básicas, la primera es la captura y modificación de los datos del producto, y la segunda es un catálogo de los proyectos del producto.

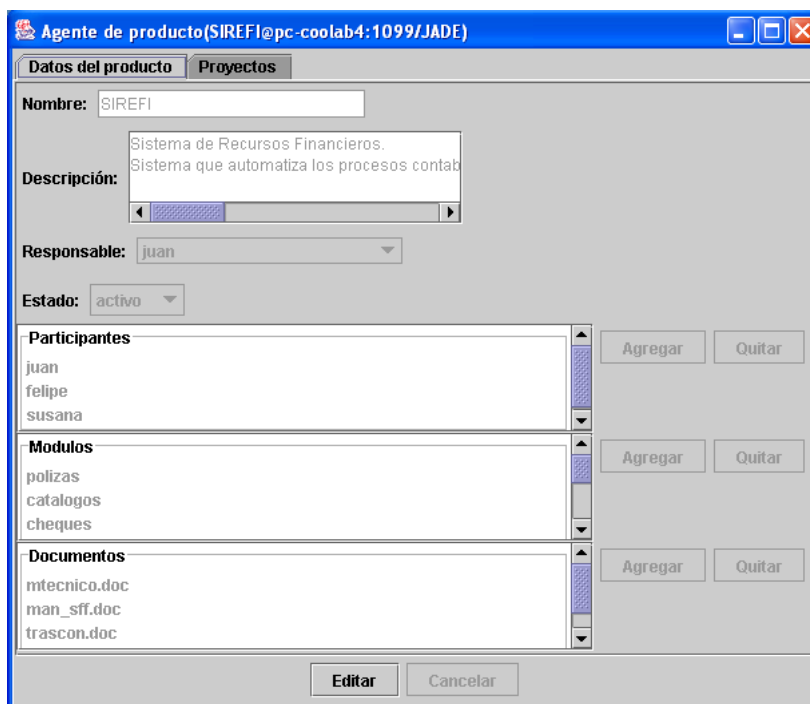


Figura 91. Interfaz gráfica del agente de producto.

Primeramente, para modificar los datos del producto es necesario oprimir el botón “Editar”, el cual hace que los campos de captura permitan la edición, como se muestra en la figura 92. Es posible observar que entre los datos que pueden ser modificados están el nombre, descripción, responsable del producto y el estado. Este último campo indica si el agente del producto será inicializado (activo) o no (inactivo). También se puede observar que es posible agregar, o quitar del producto, a los miembros del personal que participan en él, así como los módulos de que se compone el producto, y documentos que pertenezcan al mismo. Para guardar los datos del producto, una vez modificados, solo es necesario oprimir el botón “Aceptar”. El botón “Cancelar”, sirve para descartar los cambios realizados.

The screenshot shows a software window titled "Agente de producto(SIREFI@pc-coolab4:1099/JADE)". It features two tabs: "Datos del producto" (selected) and "Proyectos". The "Datos del producto" tab contains several input fields and lists:

- Nombre:** SIREFI
- Descripción:** Sistema de Recursos Financieros. Sistema que automatiza los procesos contab...
- Responsable:** juan
- Estado:** activo

Below these fields are three lists, each with "Agregar" and "Quitar" buttons:

- Participantes:** juan, felipe, susana
- Modulos:** polizas, catalogos, cheques
- Documentos:** mtecnico.doc, man_sff.doc, trascon.doc

At the bottom of the window are "Aceptar" and "Cancelar" buttons.

Figura 92. Edición de los datos del producto.

Cuando se agrega un miembro del personal como participante en el producto, se abre la ventana mostrada en la figura 93. Se puede observar que es posible asignar roles a los distintos participantes, sin embargo, estos roles no son tomados en cuenta dentro del funcionamiento del prototipo.

Al agregar módulos al producto, la ventana desplegada para la captura de los datos de éstos es la mostrada en la figura 94, como es posible observar, los datos a capturar son el nombre y una descripción del módulo.

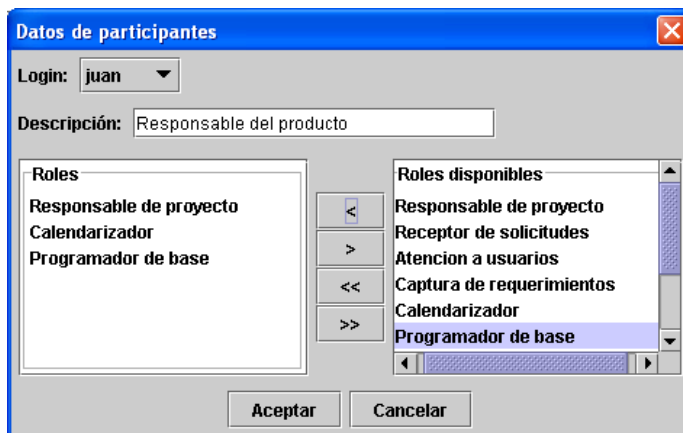


Figura 93. Ventana para la captura de participantes del producto.

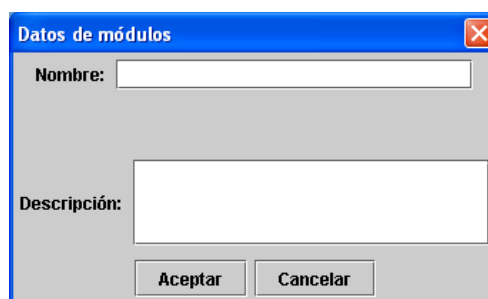


Figura 94. Ventana para la captura de los datos de los módulos del producto.

Por su parte, la ventana para capturar los datos de los documentos del producto es la mostrada en la figura 95. Se puede ver que los datos capturados son la clase y tipo de documento, una definición del formato en el que éste se encuentra, el nombre y una descripción del documento.

Datos de documentos

Clase: Documentación

Tipo: Documentación del sistema

Formato:

Nombre:

Descripción:

Aceptar Cancelar

Figura 95. Ventana para la captura de datos de los documentos del producto.

La segunda funcionalidad de la interfaz gráfica del agente de producto, es el manejo de un catálogo de proyectos. La utilización de este catálogo es similar a la de los catálogos que se encuentran en la interfaz gráfica del agente de directorio, como lo ilustra la figura 96.

Agente de producto(SIREFI@pc-coolab4:1099/JADE)

Datos del producto Proyectos

SIREFI_pf667690917
SIREFI_pf6676a48ea
SIREFI_pf6676c06a8
SIREFI_pf66c21f230

Proyecto: SIREFI_pf66c21f230

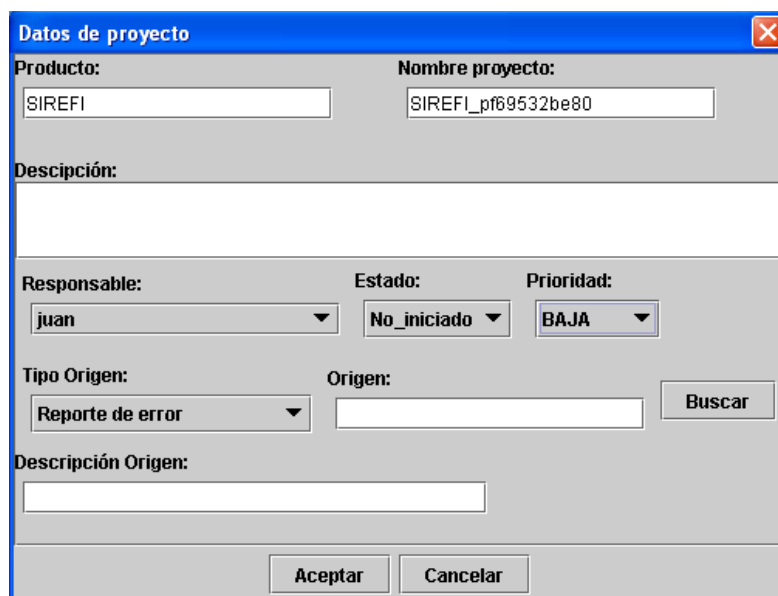
Nombre: SIREFI_pf66c21f230
Descripción: Tomar en cuenta las excepciones en el cálculo de impuestos en el reporte de honorarios
Producto: SIREFI
Responsable: juan
Estado: No_iniciado
Prioridad: ALTA
Tipo origen: Reporte de error
Id Origen: re_f66c1ad524
Descripción Origen:
No se consideran excepciones en el cálculo de impuestos en el reporte de honorarios

Nuevo Modificar Borrar

Figura 96. Muestra el catálogo de proyectos.

Al crear un nuevo proyecto, mediante el botón “Nuevo”, se abre la ventana para la captura de los datos de los proyectos, mostrada en la figura 97. Como se puede observar, el nombre del producto y del proyecto son asignados automáticamente, así como la persona responsable del proyecto, la cual es la misma que el responsable del producto, sin embargo, es posible modificar estos datos. Otros datos capturados en esta ventana son la descripción

del proyecto, el estado en el que se encuentra, la prioridad y el origen del proyecto, lo cual se indica mediante el tipo del origen, que puede ser un reporte de error o una solicitud de mantenimiento, así como el identificador y descripción de éste. Como es posible observar, se proporciona un botón “Buscar”, el cual permite hacer una búsqueda en una lista para elegir de ahí la fuente que dio origen al proyecto (actualmente solo funciona para los reportes de error). En la figura 98 se muestra la lista de reportes de error que es presentada al oprimir el botón “Buscar”.



Datos de proyecto

Producto: SIREFI Nombre proyecto: SIREFI_pf69532be80

Descripción:

Responsable: juan Estado: No_iniciado Prioridad: BAJA

Tipo Origen: Reporte de error Origen: Buscar

Descripción Origen:

Aceptar Cancelar

Figura 97. Ventana para la captura de datos de los proyectos.

Para elegir el reporte de error que da origen al proyecto, solo se requiere seleccionarlo de la lista y oprimir el botón “Aceptar”. Como es posible observar en la figura 98, los datos del reporte de error elegido son mostrados en el área de despliegue de datos en la parte derecha de la ventana.

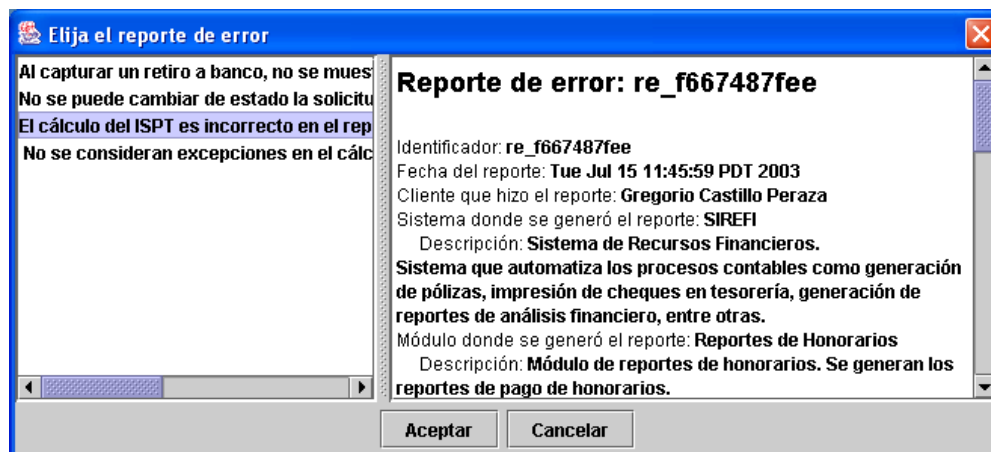


Figura 98. Muestra la lista de reportes de error.

C.3 Aplicación para el manejo de la base de conocimientos

La aplicación que se creó para el manejo de la base de conocimientos, consta de tres elementos principales, el primero es el catálogo de temas de conocimiento, el segundo es un módulo para permitir asignar conocimiento a las fuentes, y el tercero simula el funcionamiento del agente manejador de conocimiento. En esta sección se presentarán los dos primeros, ya que el funcionamiento del tercer módulo es similar al de la interfaz gráfica del agente manejador de conocimiento descrita en el capítulo V.

Primeramente, el módulo para el manejo del catálogo de tipos de conocimiento, mostrado en la figura 99, presenta dos áreas principales, la primera es la lista de temas de conocimiento, y la segunda es un área para el despliegue de los datos de éstos temas. Como se ilustra en la figura. Este módulo presenta dos funciones principales, la primera para agregar temas o áreas de conocimiento (botón “Agregar”), y la segunda para borrarlas (botón “borrar”). Cuando se desea agregar una nueva área de conocimiento se despliega la ventana mostrada en la figura 100, en ella es posible ver que los datos a capturar son el tipo de área, la cual se elige de una lista de tipos predefinidos; así como el nombre y descripción de la misma.

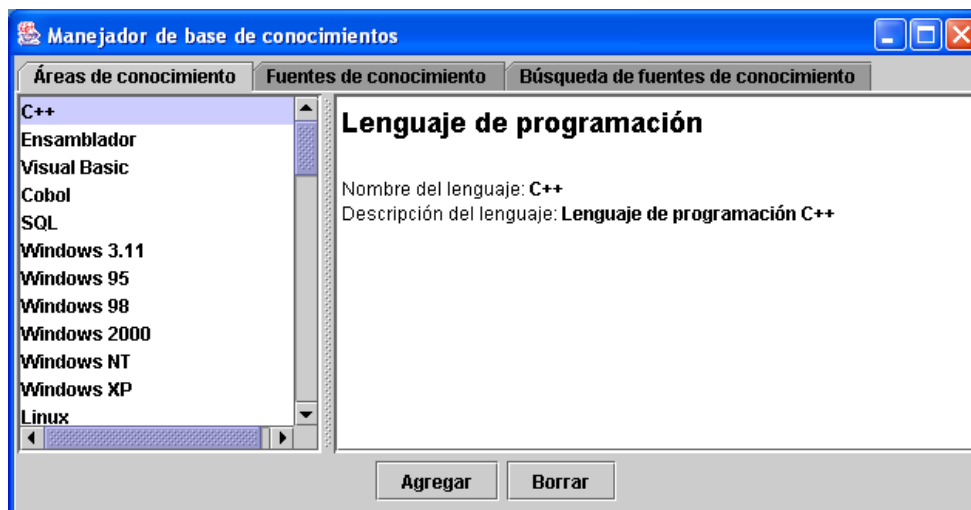


Figura 99. Muestra el catálogo de tipos de conocimiento.

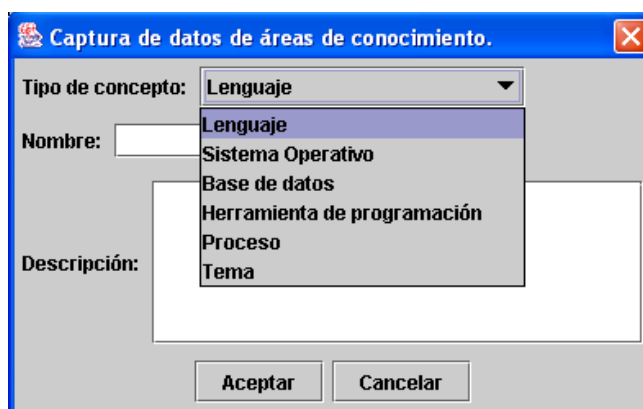


Figura 100. Ventana para captura de datos de temas de conocimiento.

Por otro lado, el módulo para el manejo del conocimiento de las fuentes de conocimiento es mostrado en la figura 101. Se puede observar que en la parte izquierda se presenta la lista de fuentes de conocimiento, y en la parte derecha un área para el despliegue de los datos de las fuentes que son seleccionadas en la lista. Este módulo presenta dos funciones principales, la primera permite agregar conocimiento a las fuentes (botón “Nuevo Conocimiento”), y la segunda es para asignar el conocimiento que se requiere para trabajar con los elementos de un producto. Esta última función sólo se habilita cuando se selecciona una fuente de conocimiento que derive del tipo de fuente Sistema.

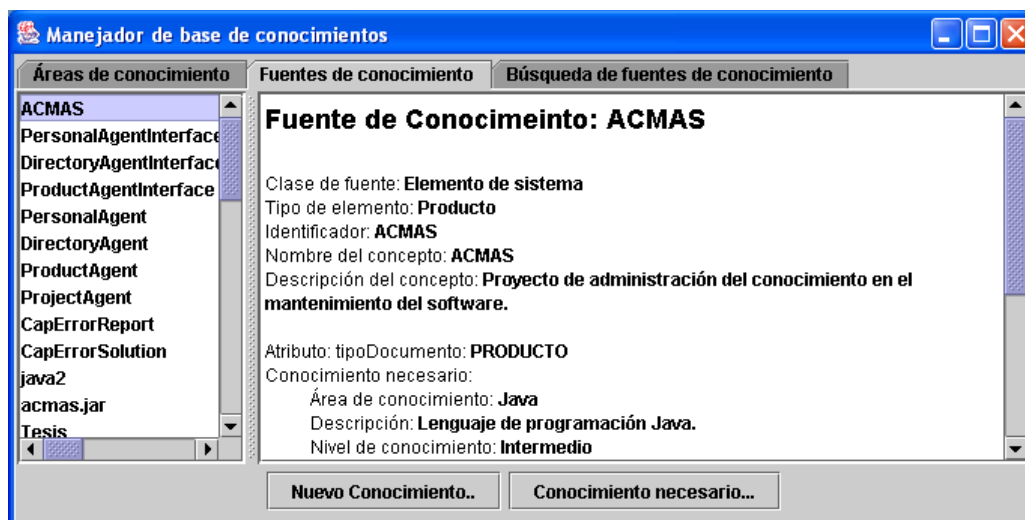


Figura 101. Muestra el módulo para la asignación de conocimiento a las fuentes.

La ventana que permite elegir el tipo de conocimiento que tiene o necesita una fuente, es la mostrada en la figura 102. Es posible observar que la ventana presenta una lista con los temas y fuentes de conocimiento, así como un área para el despliegue de los datos del concepto que se elige en la lista. Además, en la parte inferior es posible elegir el nivel de conocimiento que la fuente tiene sobre el concepto elegido.

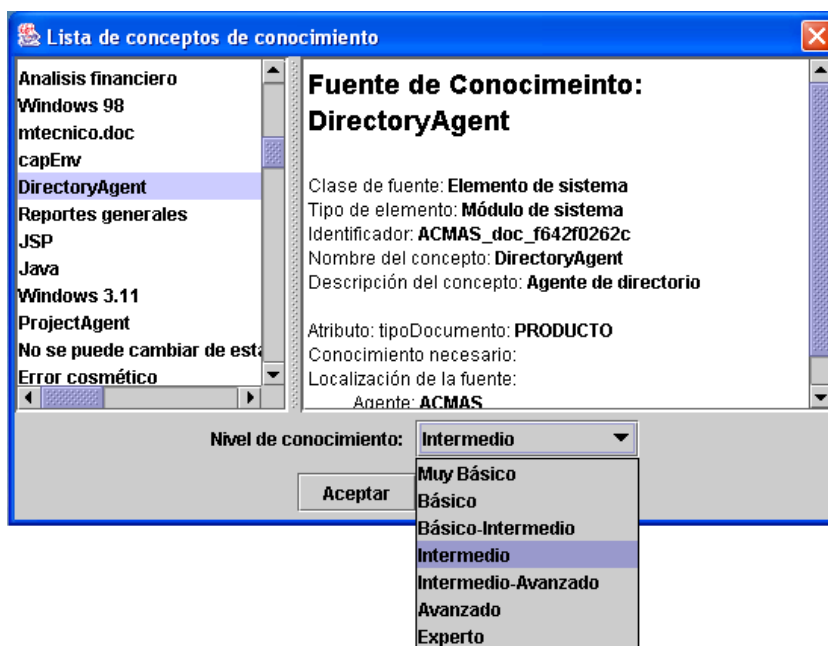


Figura 102. Ventana para elegir el tipo de conocimiento que tiene o necesita una fuente.

Apéndice D. Instalación, Configuración y Ejecución del Prototipo

En este apéndice se describe la forma de instalar, configurar y ejecutar el prototipo para su utilización.

D.1 Procedimiento de instalación.

Para que el prototipo desarrollado pueda funcionar, es necesario que en el sistema se encuentren instalados el RunTime de Java, la plataforma JADE, y el servidor de base de datos Xindice. A continuación se describe el procedimiento de instalación, haciendo mención de los elementos que requieren ser instalados y configurados previamente.

Instalación de Java

El prototipo ha sido desarrollado y probado solamente con la versión 1.4.1_01. Por lo tanto, se recomienda usar esta misma versión o una posterior.

La instalación de Java debe hacerse conforme a las instrucciones definidas en su distribución. Una vez que Java haya sido instalado, se debe configurar la variable de ambiente JAVA_HOME para que haga referencia al directorio donde Java fue instalado.

Ejemplo: set JAVA_HOME=c:\j2sdk1.4.1_01

Instalación de JADE

El prototipo fue desarrollado con la versión 3.01b de JADE, por lo que se recomienda utilizar la misma versión, ya que pueden existir incompatibilidades con otras versiones, sobre todo las anteriores.

Para la instalación de JADE, sólo es necesario seguir las instrucciones definidas en su distribución. Una vez instalado, es recomendable establecer la variable de ambiente JADE_HOME para que haga referencia al directorio donde jade fue instalado.

Ejemplo: `set JADE_HOME=c:\jade`

Además de las librerías que vienen con la distribución normal de JADE, es necesario instalar los agregados para seguridad y para el manejo de lenguajes de contenido XML, `jadesecurity` y `XMLCodec` respectivamente. Para esto siga el procedimiento definido por la distribución de dichos agregados. Una vez instalados, es recomendable copiar los archivos `jadeS.jar` y `XMLCodec.jar`, del directorio `lib` de la instalación de `jadesecurity` y de `XMLCodec` respectivamente, al directorio `lib` de la instalación de JADE (`JADE_HOME\lib`).

Instalación de Xindice

El prototipo fue probado con la versión 1.0 de Xindice, por lo que se recomienda usar la misma versión o una posterior.

Para la instalación de Xindice, siga el procedimiento definido en su distribución. Una vez instalado, se debe establecer la variable de ambiente `XINDICE_HOME` para que haga referencia al directorio donde Xindice fue instalado.

Ejemplo: `set XINDICE_HOME=c:\xml-xindice-1.0`

Instalación del prototipo

Los elementos que constituyen el prototipo se encuentran dentro del archivo `acmas.zip`, para instalarlo solo es necesario descomprimir este archivo en un directorio establecido. Una vez hecho esto, se recomienda establecer la variable de ambiente `ACMAS_HOME`, para que apunte al directorio donde el archivo `acmas.zip` fue descomprimido.

Por ejemplo, si el archivo `acmas.zip` fue descomprimido directamente en `c:\`, la forma de establecer la variable de ambiente sería: “**`set ACMAS_HOME=c:\acmas`**”.

Al descomprimir el archivo `acmas.zip`, se crea un directorio `acmas` con cuatro subdirectorios: `bin`, `lib`, `doc` y `src`. En el directorio `bin` se encuentran tres archivos de

procesamiento por lotes: runDirectory para ejecutar el directorio de agentes, runPersonal para ejecutar el módulo del agente de personal, y runkbManager para ejecutar la aplicación para el manejo de fuentes de conocimiento. En el directorio lib se encuentra el archivo acmas.jar, el cual contiene los archivos ejecutables, es necesario agregar este archivo al CLASSPATH como se verá más adelante. En el directorio doc se encuentra la documentación del prototipo. Finalmente, en el directorio src se encuentra el código fuente del prototipo.

D.2 Configuración

Establecer la variable CLASSPATH

Para que el prototipo pueda funcionar correctamente, es necesario agregar algunos archivos a la variable de ambiente CLASSPATH, estos archivos se listan a continuación:

JADE_HOME\lib\Base64.jar
 JADE_HOME\lib\iiop.jar
 JADE_HOME\lib\jade.jar
 JADE_HOME\lib\jadeTools.jar
 JADE_HOME\lib\jadeS.jar
 JADE_HOME\lib\XMLCodec.jar
 XINDICE_HOME\java\lib\xindice.jar
 XINDICE_HOME\java\lib\xmlldb.jar
 XINDICE_HOME\java\lib\xmlldb-xupdate.jar
 XINDICE_HOME\java\lib\openorb-1.2.0.jar
 ACMAS_HOME\lib\acmas.jar

Configuración de xalan y xerces

El prototipo utiliza algunas funciones para la manipulación de documentos XML, estas funciones son proporcionadas por las librerías desarrolladas por Apache, específicamente xalan y xerces. Debido a que, tanto Java, como JADE y Xindice utilizan versiones distintas de estas librerías, es necesario forzar la utilización de una sola versión. La versión con la

que el prototipo funcionó correctamente es la que se proporciona con Xindice, es por ello que esta es la versión que debe ser configurada. El procedimiento es el siguiente:

En ambientes Windows se debe hacer lo siguiente:

- En el directorio: “c:\Archivos de programas\Java\j2reX.X.X\lib” donde X.X.X es la versión de la máquina virtual de java que se encuentra instalada, se debe crear un subdirectorio llamado endorsed.
- En este directorio copiar la versión 2.0.1 de xalan, y la 1.4.3 de xerces. Estas versiones se encuentran en el directorio XINDICE_HOME\java\lib

En ambientes Unix la diferencia es el directorio donde se encuentra el RunTime de Java. Específicamente se localiza en el directorio donde se instala Java, bajo el subdirectorio jre. El procedimiento es similar al anterior.

Archivo de nombres de usuario y claves de acceso

Al iniciar por primera vez el directorio de agentes, se crea un archivo de nombre acmas.passwd en el directorio user.home\acmas, donde user.home es el directorio de inicio del usuario (Ejemplo: c:\Documents and Settings\usuario\acmas). El archivo acmas.passwd es el utilizado para definir los nombres de usuario y claves de acceso, inicialmente se crea el usuario acmas, el cual es el utilizado por el agente de directorio. Sin embargo, para poder asignar nombres de usuario a los miembros del personal, será necesario agregar más nombres de usuario con sus claves.

Para las claves de acceso, JADE utiliza el mismo tipo de encriptación que es manejado por las claves de acceso de los sistemas Unix, por lo que pueden ser utilizados los nombres de usuario y claves definidos para un sistema Unix, por ejemplo los del archivo “/etc/shadow”.

Otro modo de crear nombres de usuario y claves de acceso, es mediante la aplicación htpasswd del servidor web Apache. La manera de hacer esto sería parecida a la siguiente:

htpasswd acmas.passwd juan

esta instrucción indica que se agregue un usuario de nombre juan al archivo `acmas.passwd`, posteriormente el sistema preguntará por la clave de usuario o password, y solicitará una confirmación de esta clave, una vez que se valida la confirmación, se agrega el nombre de usuario juan y su clave de acceso al archivo `acmas.passwd`.

Posteriormente es necesario agregar estos usuarios y claves creadas al archivo `acmas.passwd` que se localiza en el directorio `.acmas` en `user.home`.

Conexiones con el servidor de base de datos y con JADE

Para que los agentes del módulo de agente de personal puedan encontrar la plataforma JADE, así como la base de datos Xindice, es necesario realizar lo siguiente:

En el directorio `user.home` de la máquina del miembro del personal, se debe crear un directorio `.acmas`. En este directorio se debe crear el archivo `.acmas.properties`, en el cual se deben establecer las siguientes propiedades:

```
host=pc-coolab4.cicese.mx
port=1099
DBHost=xmldb:xindice://pc-coolab4.cicese.mx
DBPort=4080
collection.personal=/db/acmas/personal
collection.products=/db/acmas/products
collection.kbase=/db/acmas/kbase
collection.rerror=/db/acmas/reportes
```

las primeras dos propiedades definen la manera de conectarse con el servidor donde se encuentra la plataforma JADE, se debe establecer el nombre del servidor y el puerto; las siguientes dos propiedades hacen lo mismo pero para el servidor de la base de datos, las otras cuatro propiedades establecen las colecciones donde se localiza la información dentro de la base de datos. La primer colección es para los datos de los miembros del personal, la segunda para los de los productos, la tercera es la base de conocimientos, y la última para los reportes de error.

Colecciones en la base de datos

Como se menciona en la descripción de los detalles de la implementación, el prototipo maneja una estructura de colecciones dentro de la base de datos, por lo que es necesario que esta estructura sea creada antes de iniciar la ejecución del prototipo.

Primeramente se debe crear una colección de nombre “acmas”, dentro de esta se deben crear la colección “personal”, “products”, “clients”, “kbase” y “reportes”.

D.3 Ejecución del prototipo

Para la ejecución del prototipo se han proporcionado tres archivos de procesamiento por lotes, los cuales se encuentran en el directorio *bin* de la instalación.

Para ejecutar el módulo del agente de directorio se debe ejecutar el archivo *runDirectory.bat*, o la instrucción *java acmas.Directory*.

Ejemplo: `c:>%ACMAS_HOME%\bin\runDirectory`

Para ejecutar el módulo del agente de personal, se debe ejecutar el archivo *runPersonal.bat*, o la instrucción *java acmas.Personal*.

Ejemplo: `c:>%ACMAS_HOME%\bin\runPersonal`

Finalmente, para ejecutar la aplicación para el manejo de la base de conocimientos, se debe ejecutar el archivo *runkbManager.bat*, o la instrucción *java acmas.guis.kbas.KBaseManager*.

Ejemplo: `c:>%ACMAS_HOME%\bin\runkbManager`