

# **Centro de Investigación Científica y de Educación Superior de Ensenada**



**Diseño de una Arquitectura Distribuida Multiservidor para la  
Administración de Sistemas de Workflow**

**TESIS  
MAESTRIA EN CIENCIAS**

**DAVID ABDEL MEJIA MEDINA**

**ENSENADA BAJA CFA, MEXICO AGOSTO DE 2005**



TESIS DEFENDIDA POR  
**David Abdel Mejía Medina**  
Y APROBADA POR EL SIGUIENTE COMITÉ



---

Dra. Ana Isabel Martínez García  
*Director del Comité*



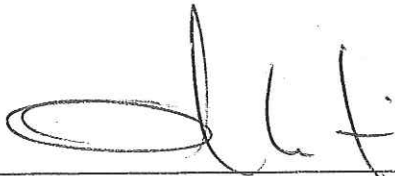
---

Dr. José Antonio García Macías  
*Miembro del Comité*



---

Dr. Fernando Rojas Iñiguez  
*Miembro del Comité*



---

Dr. Jesús Arturo Martín Barajas  
*Miembro del Comité*



---

Dr. Gilberto López Mariscal  
*Coordinador del programa de  
posgrado en Ciencias de la  
Computación*



---

Dr. Raúl Ramón Castro Escamilla  
*Director de Estudios de Posgrado*



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



---

**PROGRAMA DE POSGRADO EN CIENCIAS  
EN CIENCIAS DE LA COMPUTACIÓN**

---

**DISEÑO DE UNA ARQUITECTURA DISTRIBUIDA MULTISERVIDOR PARA LA  
ADMINISTRACIÓN DE SISTEMAS DE WORKFLOW**

TESIS

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

Presenta:

DAVID ABDEL MEJÍA MEDINA

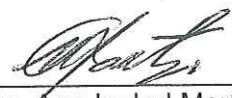
Ensenada, Baja California, México, Agosto del 2005.



**Resumen** de la tesis de **David Abdel Mejía Medina**, presentada como requisito parcial para la obtención del grado de **MAESTRO EN CIENCIAS** en **CIENCIAS DE LA COMPUTACIÓN**. Ensenada, Baja California. Agosto del 2005.

## **DISEÑO DE UNA ARQUITECTURA DISTRIBUIDA MULTISERVIDOR PARA LA ADMINISTRACIÓN DE SISTEMAS DE WORKFLOW.**

Resumen aprobado por:

  
Dra. Ana Isabel Martínez García  
Directora de Tesis

En años recientes ha habido un creciente interés en el desarrollo de aplicaciones de workflow distribuido. Este interés ha surgido de la necesidad de proveer soporte al *enactment* de grandes procesos, donde los sistemas tradicionales de workflow basados en una arquitectura cliente-servidor presentan algunos problemas, como sobrecarga de trabajo en el servidor, poca escalabilidad, excesivo flujo de datos sobre la red, y poca tolerancia a fallos, entre otros.

En este trabajo se presenta una arquitectura distribuida de workflow, la cual esta basada en roles y en la implementación de nodos distribuidos independientes. Cada nodo contiene todas las actividades de los roles y la información que un actor requiere para desarrollarlas. Los nodos coordinan las tareas de los procesos de manera independiente. Esto ayuda a evitar el congestionamiento de la red y a aprovechar la capacidad de procesamiento de cada nodo independiente.

Se considera un nodo central para controlar el proceso de inicialización del sistema de workflow y para la recuperación de errores con un modelo de transacciones avanzadas. Se ilustra el uso de la arquitectura a través de la implementación de un prototipo.

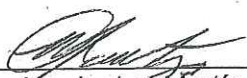
**Palabras clave:** workflow, sistemas de workflow, arquitectura de software, sistemas distribuidos.



**Abstract** of the Thesis of **David Abdel Mejía Medina**, presented as partial requirement to obtain the **MASTER in SCIENCES** degree in **COMPUTER SCIENCES**. Ensenada, Baja California. August of 2005.

**DESIGN OF A MULTISERVER DISTRIBUTED ARCHITECTURE FOR THE  
MANAGEMENT OF WORKFLOW SYSTEMS.**

Abstract approved by:

  
\_\_\_\_\_  
Dra. Ana Isabel Martínez García  
Directora de Tesis

In recent years there has been an increasing interest in the development of distributed workflow applications. This interest has emerged from the need of providing support to the enactment of large processes where the traditional client-server architecture presents some problems, such as server work overload, low scalability, excessive data flow over the network, and low failure tolerance, among others.

In this work we present a workflow distributed architecture, which is based on roles and the implementation of distributed independent nodes. Each node contains all the role's activities and information that an actor requires to perform. The nodes coordinate the execution of the process tasks independently. This helps to avoid network congestion and to take advantage of the processing capabilities of each independent node.

We consider a central node for process control to initialize a workflow system and for failure recovery with an advance transactional model. We illustrate the use of the architecture through the development of a prototype.

**Keywords:** workflow, workflow systems, software architecture, distributed systems.



## Dedicatoria

A mis padres, Silvia y David,  
quienes me dieron la vida y me enseñaron a vivir en ella.

A mi madrina Alicia,  
quien ha sido un gran pilar en mi vida.

A mi hermana Syndái,  
quien ha sido mi mejor amiga desde que nació.

A Yenny,  
quien siempre supo hacerme sentir su presencia, a pesar de la distancia.

## Agradecimientos

Un especial agradecimiento a mi directora de tesis, Dra. Ana Isabel Martínez García, por su invaluable apoyo en la realización de este trabajo, porque sin sus regaños y sus preocupaciones nada de esto hubiera sido posible.

A los miembros del comité de tesis, Dr. José Antonio García Macías, Dr. Fernando Rojas Iñiguez y Dr. Jesús Arturo Martín Barajas, por el tiempo dedicado y las valiosas contribuciones que hicieron a este trabajo.

A mi familia, porque la unión y el cariño que tenemos me ha dado tranquilidad y fuerzas para sortear los momentos difíciles de estos dos años: a mis padres Silvia y David, porque siempre han respetado y compartido mis sueños, y los han impulsado igual o más que los suyos propios. A mi madrina Alicia, porque sin importar la meta que me proponga, siempre he recibido su apoyo y su cariño. A mi hermana Syndai, porque durante el tiempo que no he estado en casa ha sabido apoyar a la familia en los momentos difíciles, y a mi cuñado Daniel, porque nos ha permitido ser parte de su familia y ha aceptado ser parte de la nuestra. ¡Los quiero y los admiro mucho familia!.

A Yenny, porque desde que nos conocimos ha sido una compañera y amiga excepcional, quien poco a poco se fue convirtiendo en parte de mi vida y hoy es un elemento indispensable tanto de este sueño que termina, como de aquellos que aún están por cumplirse.



A mis compañeros de la generación 2003, especialmente a Lizbeth, Elisa, Irma, Mónica, Isabel, Diana, Cynthia, Pedro, Milton, Luís y Leonardo, porque con la amistad que me brindaron me hicieron más llevadero el tiempo en que transcurrió la maestría.

A Caro, Lidia y Jorge, por el gran apoyo que me brindaron cada vez que lo necesité.

A todos los investigadores de CICESE que fueron parte de mi formación durante el tiempo que duró la maestría.

Al Consejo Nacional de Ciencia y Tecnología.

## CONTENIDO

|                                                                                                                       | Página        |
|-----------------------------------------------------------------------------------------------------------------------|---------------|
| <b>Capítulo I. Introducción.</b>                                                                                      | <b>1</b>      |
| I.1 Antecedentes.....                                                                                                 | 2             |
| I.2 Planteamiento del problema.....                                                                                   | 11            |
| I.3 Objetivo general.....                                                                                             | 17            |
| I.4 Objetivos específicos.....                                                                                        | 17            |
| I.5 Metodología.....                                                                                                  | 17            |
| I.6 Contenido de la tesis.....                                                                                        | 19            |
| <br><b>Capítulo II. Arquitecturas de Workflow Distribuido.</b>                                                        | <br><b>20</b> |
| II.1 Introducción.....                                                                                                | 20            |
| II.2 Arquitecturas de Workflow Distribuidas Existentes: Análisis comparativo...                                       | 21            |
| II.2.1 INCA's.....                                                                                                    | 21            |
| II.2.2 EXOTICA.....                                                                                                   | 23            |
| II.2.3 MENTOR.....                                                                                                    | 25            |
| II.2.4 WONDER.....                                                                                                    | 26            |
| II.2.5 RainMaker.....                                                                                                 | 27            |
| II.2.5 Tabla Comparativa.....                                                                                         | 29            |
| II.4 Resumen.....                                                                                                     | 33            |
| <br><b>Capítulo III. WODIS: Una Arquitectura Distribuida Multiservidor para la Ejecución de Sistemas de Workflow.</b> | <br><b>35</b> |
| III.1 Introducción.....                                                                                               | 35            |
| III.2 Requerimientos de la arquitectura.....                                                                          | 35            |
| III.3 Análisis y diseño de la arquitectura WODIS.....                                                                 | 37            |
| III.3.1. Descripción de la arquitectura WODIS.....                                                                    | 38            |
| III.4 Análisis de la arquitectura.....                                                                                | 48            |
| III.4.1 Casos de uso.....                                                                                             | 48            |
| III.3.2 Diagramas de secuencia.....                                                                                   | 54            |
| III.5 Enfoque transaccional en la ejecución y fallas del sistema.....                                                 | 60            |
| III.5.1 Modelo de transacciones ACID.....                                                                             | 61            |
| III.5.2 Modelo de transacciones avanzadas.....                                                                        | 62            |
| III.5.3 Ejemplificación del funcionamiento de un sistema de workflow basado en transacciones.....                     | 67            |
| III.6 Escenarios de manejos de errores.....                                                                           | 79            |
| III.7 Guía breve para el desarrollo de un sistema de workflow distribuido basado en la arquitectura WODIS.....        | 86            |
| III Resumen.....                                                                                                      | 88            |



## CONTENIDO (continuación)

|                                                                                                                                                      | Página     |
|------------------------------------------------------------------------------------------------------------------------------------------------------|------------|
| <b>Capítulo IV. Implementación de prototipos.</b>                                                                                                    | <b>90</b>  |
| IV.1 Introducción.....                                                                                                                               | 90         |
| IV.2 Tecnología de desarrollo y modelado de procesos utilizados en el desarrollo de los prototipos.....                                              | 91         |
| IV.3 Prototipo del proceso del juego de la cerveza.....                                                                                              | 100        |
| IV.3.1 Análisis del prototipo del proceso del juego de la cerveza. ....                                                                              | 104        |
| IV.3.2. Implementación del prototipo del proceso del juego de la cerveza.....                                                                        | 108        |
| IV.3.3 Implementación del mecanismo de recuperación de errores del prototipo del proceso del juego de la cerveza.....                                | 115        |
| IV.3 Prototipo del proceso de reclutamiento, selección y contratación de personal.....                                                               | 117        |
| IV.3.1 Análisis del prototipo del proceso de reclutamiento, selección y contratación de personal.....                                                | 120        |
| IV.3.2. Implementación del prototipo del proceso de reclutamiento, selección y contratación de personal.....                                         | 122        |
| IV.3.3 Implementación del mecanismo de recuperación de errores del prototipo del proceso de reclutamiento, selección y contratación de personal..... | 130        |
| IV.5 Pruebas a los prototipos.....                                                                                                                   | 131        |
| IV.5.1 Pruebas y resultados.....                                                                                                                     | 133        |
| IV.4.2 Resumen de las pruebas.....                                                                                                                   | 143        |
| IV.5 Resumen.....                                                                                                                                    | 144        |
| <br><b>Capítulo V. Conclusiones</b>                                                                                                                  | <b>146</b> |
| V.1 Conclusiones.....                                                                                                                                | 146        |
| V.2 Aportaciones.....                                                                                                                                | 148        |
| V.3 Trabajo futuro.....                                                                                                                              | 149        |
| <br><b>Referencias.....</b>                                                                                                                          | <b>153</b> |
| <br><b>Apéndice A. (Análisis de la arquitectura).....</b>                                                                                            | <b>158</b> |
| <br><b>Apéndice B. (Aspectos técnicos de la implementación de los prototipos basados en la arquitectura WODIS).....</b>                              | <b>187</b> |
| <br><b>Apéndice C (Casos de prueba).....</b>                                                                                                         | <b>198</b> |

## LISTA DE FIGURAS

| Figura |                                                                     | Página |
|--------|---------------------------------------------------------------------|--------|
| 1      | Ilustración de <i>proceso de venta de seguros</i> .....             | 3      |
| 2      | Diagrama de estados del rol <i>Agente de seguros</i> .....          | 5      |
| 3      | Diagrama de estados del rol <i>Ejecutivo de seguros</i> .....       | 6      |
| 4      | Diagrama de estados del rol <i>Gerente de seguros</i> .....         | 7      |
| 5      | Sistema de workflow en un esquema cliente-servidor.....             | 10     |
| 6a     | Sistema de workflow con esquema centralizado.....                   | 13     |
| 6b     | Forma natural de trabajo con esquema centralizado.....              | 13     |
| 7a     | Sistema de workflow con esquema distribuido.....                    | 14     |
| 7b     | Forma natural de trabajo.....                                       | 14     |
| 8      | Esquema de la Arquitectura MENTOR.....                              | 25     |
| 9      | Arquitectura global WODIS.....                                      | 38     |
| 10     | Comunicación entre nodos WODIS.....                                 | 39     |
| 11     | Componentes del Nodo Distribuidor.....                              | 40     |
| 12     | Componentes del Nodo Distribuido.....                               | 46     |
| 13     | Diagrama de casos de uso de la arquitectura propuesta.....          | 50     |
| 14     | Diagrama de secuencia del caso de uso Validar actor.....            | 55     |
| 15     | Diagrama de secuencia del caso de uso Obtener workflow.....         | 57     |
| 16     | Diagrama de secuencia del caso de uso <i>Ejecutar proceso</i> ..... | 59     |
| 17     | Elementos de un <i>Diagrama Rol Actividad</i> .....                 | 68     |



## LISTA DE FIGURAS (continuación)

| Figura |                                                                                                     | Página |
|--------|-----------------------------------------------------------------------------------------------------|--------|
| 18     | Rol-Actividad del Rol de ejemplo.....                                                               | 69     |
| 19     | Documento de flujo de actividades para el rol de ejemplo.....                                       | 70     |
| 20     | Configuración del estado <i>A1</i> del <i>rol de ejemplo</i> .....                                  | 73     |
| 21     | Configuración del estado <i>A2</i> del <i>rol de ejemplo</i> .....                                  | 74     |
| 22     | Escenario de errores debido a comunicación fallida con otro<br>usuario.....                         | 83     |
| 23     | Escenario de errores debido a que el nodo fue cerrado<br>incorrectamente.....                       | 85     |
| 24     | Pasos a seguir para desarrollar sistemas basados en la<br>arquitectura propuesta.....               | 87     |
| 25     | Comunicación entre nodos por medio de CORBA en los<br>prototipos desarrollados en este trabajo..... | 95     |
| 26a    | Diagrama RAD del proceso del <i>juego de la cerveza</i> .....                                       | 102    |
| 26b    | Sub proceso <i>Dar servicio a cliente</i> .....                                                     | 102    |
| 26c    | Sub proceso <i>Ordenar productos</i> .....                                                          | 102    |
| 27     | Diagrama del caso de uso <i>Surtir producto a cliente</i> .....                                     | 105    |
| 28     | Diagrama de secuencia <i>Surtir pedido a cliente</i> .....                                          | 107    |
| 29     | IDL del Nodo Distribuidor.....                                                                      | 108    |
| 30     | Directorio de actores utilizado en el prototipo del juego de la<br>cerveza.....                     | 109    |

## LISTA DE FIGURAS (continuación)

| Figura |                                                                                                                      | Página |
|--------|----------------------------------------------------------------------------------------------------------------------|--------|
| 31     | Workflow utilizado en el prototipo del juego de la cerveza.....                                                      | 110    |
| 32     | IDL de los objetos distribuidos en el prototipo del juego de la<br>cerveza.....                                      | 111    |
| 33     | Datos del estado <i>mn3</i> del workflow del proceso del juego de la<br>cerveza.....                                 | 112    |
| 34     | Datos del estado <i>mn4</i> del workflow del proceso del juego de la<br>cerveza.....                                 | 114    |
| 35     | Datos del estado <i>mn5</i> del workflow del proceso del juego de la<br>cerveza.....                                 | 114    |
| 36     | Diagrama RAD del proceso reclutamiento, selección y<br>contratación de personal.....                                 | 118    |
| 37     | Diagrama del caso de uso <i>Solicitar cobertura de vacante</i> .....                                                 | 121    |
| 38     | Diagrama de secuencia del caso de uso <i>Solicitar cobertura de<br/>vacante</i> .....                                | 122    |
| 39     | IDL del nodo distribuidor en el prototipo del proceso de<br>reclutamiento, selección y contratación de personal..... | 123    |
| 40     | IDL de algunos nodos distribuidos en el prototipo de<br>reclutamiento, selección y contratación de personal.....     | 124    |

## LISTA DE FIGURAS (continuación)

| Figura |                                                                                                                               | Página |
|--------|-------------------------------------------------------------------------------------------------------------------------------|--------|
| 41     | Flujo de actividades del rol <i>responsableArea</i> para el caso de uso<br><i>Solicitar cobertura de vacante</i> .....        | 125    |
| 42     | Flujo de actividades del rol <i>controlPlazas</i> para el caso de uso<br><i>Solicitar cobertura de vacante</i> .....          | 126    |
| 43     | Flujo de actividades del rol <i>directorAdministrativo</i> para el caso<br>de uso <i>Solicitar cobertura de vacante</i> ..... | 128    |
| 44     | Actividades del que corresponden al rol <i>Director Administrativo</i>                                                        | 134    |
| 45     | Pantalla de selección de actividades del rol <i>Director</i><br><i>Administrativo</i> .....                                   | 135    |
| 46     | Pantalla la actividad <i>Evaluar solicitud</i> del rol <i>Director</i><br><i>Administrativo</i> .....                         | 137    |
| 47     | Bitácora de las actividades realizadas por el rol <i>Director</i><br><i>Administrativo</i> .....                              | 138    |
| 48     | Pantallas donde se utiliza la misma entidad de información.....                                                               | 139    |
| 49     | Diagrama de casos de uso de la arquitectura propuesta.....                                                                    | 159    |
| 50     | Diagrama de secuencia del caso de uso Configurar nodos<br>distribuidos.....                                                   | 166    |
| 51     | Diagrama de secuencia del caso de uso Validar actor.....                                                                      | 167    |



## LISTA DE FIGURAS (continuación)

| Figura |                                                                         | Página |
|--------|-------------------------------------------------------------------------|--------|
| 52     | Diagrama de secuencia del caso de uso Obtener workflow.....             | 168    |
| 53     | Diagrama de secuencia del caso de uso Escribir bitácora.....            | 170    |
| 54     | Diagrama de secuencia del caso de uso Ejecutar proceso.....             | 171    |
| 55     | Diagrama de secuencia del caso de uso Salir.....                        | 173    |
| 56     | Diagrama de secuencia del caso de uso Obtener tareas<br>pendientes..... | 174    |
| 57     | Diagrama de secuencia del caso de uso Enviar mensaje.....               | 175    |
| 58     | Diagrama del caso de uso <i>Hacer pedido de producto</i> .....          | 176    |
| 59     | Diagrama del caso de uso <i>Monitorear respuesta</i> .....              | 177    |
| 60     | Diagrama del caso de uso <i>Obtener respuesta a pedido</i> .....        | 177    |
| 61     | Diagrama del caso de uso <i>Surtir producto a cliente</i> .....         | 178    |
| 62     | Diagrama del caso de uso <i>Inicializar inventario</i> .....            | 179    |
| 63     | Diagrama del caso de uso <i>Actualizar inventario</i> .....             | 179    |
| 64     | Diagrama del caso de uso <i>Revisar inventario</i> .....                | 180    |
| 65     | Diagrama del caso de uso <i>Surtir producto a minorista</i> .....       | 181    |
| 66     | Diagrama de secuencia <i>Hacer pedido de producto</i> .....             | 181    |
| 67     | Diagrama de secuencia <i>Surtir pedido a cliente</i> .....              | 183    |
| 68     | Diagrama de secuencia <i>Surtir producto a minorista</i> .....          | 184    |

## LISTA DE FIGURAS (continuación)

| Figura |                                                                                                                            | Página |
|--------|----------------------------------------------------------------------------------------------------------------------------|--------|
| 69     | Diagrama de secuencia Monitorear Respuesta.....                                                                            | 185    |
| 70     | Diagrama de secuencia Obtener respuesta a pedido.....                                                                      | 186    |
| 71     | Diagrama de secuencia Actualizar inventario.....                                                                           | 186    |
| 72     | IDL del objeto Nodo Distribuidor.....                                                                                      | 188    |
| 73     | Clase NodoDistribuidorServant del Prototipo del proceso de reclutamiento, selección y contratación de personal en CICESE.. | 190    |
| 74     | Clase NodoDistribuidorServer del Prototipo del proceso de reclutamiento, selección y contratación de personal en CICESE..  | 191    |
| 75     | Clase responsableAreaClient del Prototipo del proceso de reclutamiento, selección y contratación de personal en CICESE..   | 192    |
| 76     | Directorio de actores del prototipo de reclutamiento, selección y contratación de personal.....                            | 194    |
| 77     | Porción de código del mecanismo Emisor de mensajes.....                                                                    | 195    |
| 78     | DTD del documento de definición de flujo de trabajo (workflow.xml).....                                                    | 196    |
| 79     | DTD del documento directorio de actores (directorioActores.xml).....                                                       | 196    |
| 80     | DTD del documento manejador de instancias (instancias.xml)....                                                             | 197    |
| 81     | DTD del documento manejador bitácora (bitácora.xml).....                                                                   | 197    |

## LISTA DE TABLAS

| Tabla |                                                                                                      | Página |
|-------|------------------------------------------------------------------------------------------------------|--------|
| I     | Análisis comparativo de las arquitecturas de workflow existentes.....                                | 32     |
| II    | Tabla que ejemplifica el arreglo de subtransacciones activas (ASA).....                              | 71     |
| III   | Tabla que ejemplifica el manejador de estados.....                                                   | 72     |
| IV    | Tabla que ejemplifica un arreglo de subtransacciones.....                                            | 73     |
| V     | Tabla que ejemplifica la bitácora después de la ejecución del estado A1.....                         | 74     |
| VI    | Tabla que ejemplifica el manejador de estados después de la ejecución del estado A1.....             | 74     |
| VII   | Tabla que ejemplifica la bitácora durante la ejecución del estado A2.....                            | 74     |
| VIII  | Tabla que ejemplifica el manejador de estados durante la ejecución del estado A2.....                | 74     |
| IX    | Tabla que ejemplifica el arreglo de subtransacciones activas durante la ejecución del estado A2..... | 75     |
| X     | Tabla que ejemplifica la bitácora después la ejecución del estado A3.....                            | 75     |



## LISTA DE TABLAS (continuación)

| Tabla                                                                                                           | Página |
|-----------------------------------------------------------------------------------------------------------------|--------|
| XI      Tabla que ejemplifica el manejador de estados después de la ejecución del estado A3.....                | 75     |
| XII     Tabla que ejemplifica el arreglo de subtransacciones activas después de la ejecución del estado A3..... | 75     |
| XIII    Tabla que ejemplifica la bitácora después la ejecución del estado A5.....                               | 76     |
| XIV    Tabla que ejemplifica el manejador de estados después de la ejecución del estado A5.....                 | 76     |
| XV     Tabla que ejemplifica el arreglo de subtransacciones activas después de la ejecución del estado A5.....  | 76     |
| XVI    Tabla que ejemplifica la bitácora después la ejecución del estado A4.....                                | 77     |
| XVII   Tabla que ejemplifica el manejador de estados después de la ejecución del estado A4.....                 | 77     |
| XVIII   Tabla que ejemplifica el arreglo de subtransacciones activas después de la ejecución del estado A5..... | 77     |
| XIX    Tabla que ejemplifica la bitácora después la ejecución del estado A2.....                                | 78     |
| XX     Tabla que ejemplifica la bitácora después la ejecución del estado A6.....                                | 78     |

## LISTA DE TABLAS (continuación)

| Tabla                                                                                                                                                                       | Página |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------|
| XXI      Tabla que ejemplifica la bitácora después de la ejecución del estado mn3.....                                                                                      | 113    |
| XXII     Tabla que ejemplifica la bitácora después de la ejecución del estado mn4.....                                                                                      | 113    |
| XXIII    Tabla que ejemplifica la bitácora después de la ejecución del estado mn5.....                                                                                      | 114    |
| XXIV    Tabla que ejemplifica la bitácora después de la ejecución del estado finalizado .....                                                                               | 115    |
| XXV     Bitácora después de la ejecución de una instancia del caso de uso Surtir producto a cliente .....                                                                   | 116    |
| XXVI    Tabla      con      entrada      de      transacción      más avanzada.....                                                                                         | 117    |
| XXVII   Bitácora del rol responsableArea en el nodo distribuido, después de la ejecución de una instancia del caso de uso Solicitar cobertura de vacante.....               | 126    |
| XXVIII   Bitácora del rol <i>controlPlazas</i> en el nodo distribuido, después de la ejecución de una instancia del caso de uso <i>Solicitar cobertura de vacante</i> ..... | 127    |

## LISTA DE TABLAS (continuación)

| Tabla  |                                                                                                                                                                             | Página |
|--------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------|
| XXIX   | Bitácora del rol <i>directorAdministrativo</i> en el nodo distribuido, después de la ejecución de una instancia del caso de uso <i>Solicitar cobertura de vacante</i> ..... | 128    |
| XXX    | Bitácora global en el nodo Distribuidor, después de la ejecución de una instancia del caso de uso <i>Solicitar cobertura de vacante</i> .....                               | 129    |
| XXXI   | Caso de prueba de requerimiento 1.....                                                                                                                                      | 134    |
| XXXII  | Resumen de pruebas al prototipo .....                                                                                                                                       | 144    |
| XXXIII | Caso de prueba de requerimiento 1.....                                                                                                                                      | 198    |
| XXXIV  | Caso de prueba de requerimiento 2.....                                                                                                                                      | 199    |
| XXXV   | Caso de prueba de requerimiento 3.....                                                                                                                                      | 200    |
| XXXVI  | Caso de prueba de requerimiento 4.....                                                                                                                                      | 201    |
| XXXVII | Caso de prueba de requerimiento 5.....                                                                                                                                      | 202    |



## Capítulo I. Introducción.

---

Día con día, las organizaciones buscan la manera de mejorar la ejecución de sus procesos con la finalidad de ser o mantenerse competitivos. Por esto, es común ver o escuchar sobre cambios que las organizaciones hacen a su forma de trabajar a través del uso de la reingeniería de procesos organizacionales y de negocios (BPR por sus siglas en inglés).

La BPR tiene entre sus funciones el análisis del proceso y en la mayoría de los casos la reconfiguración y rediseño de las actividades del mismo, lo que usualmente conlleva a la implementación de tecnologías de información (TI) o, en caso de que exista previamente en la organización el mejoramiento de ésta [Warboys et al., 1998].

En los últimos años se ha podido constatar no solo el desarrollo de herramientas que ayudan en el cumplimiento de una tarea, sino también que permitan administrar y dar seguimiento a la forma en como se lleva a cabo un proceso en su totalidad [WfMC 1998]. Dentro de estas herramientas se encuentran los administradores de sistemas de workflow (WFMS por sus siglas en inglés). De hecho, se ha reportado que desde la década de los 90's, las aplicaciones para la automatización del flujo de trabajo o workflow, han sido frecuentemente utilizadas como parte de un ejercicio de soporte a los BPR para automatizar los procesos organizacionales mejorados [Prior, 2003].

La mayoría de los WFMSs desarrollados actualmente están basados en el esquema cliente-servidor. Estos sistemas generalmente presentan problemas al momento de dar soporte al

*enactment* de procesos grandes, debido principalmente a que toda la carga de trabajo es realizada en el servidor. Por *enactment* se entiende la implementación de un proceso en un sistema informático, con la finalidad de darle soporte a dicho proceso en la forma como este ocurre en la vida real [Rojas y Martínez-García, 1998].

Actualmente se está tratando de solucionar los problemas mencionados mediante la distribución de la ejecución de las actividades entre los nodos (PC's en las que los actores llevan a cabo las actividades) que participan en el proceso. En este sentido se propone una arquitectura de workflow que sirva de base al desarrollo de sistemas de flujo de trabajo (workflow) en el que cada una de las actividades del proceso sea ejecutada en el nodo en que trabaja el actor al que le corresponde llevar a cabo esas actividades.

### ***1.1 Antecedentes.***

En estos días, un WfMS es una de las principales tecnologías para mejorar la efectividad y eficiencia de los procesos dentro de una organización [Lavagno y Reisig, 1998], debido a sus características de apoyo a la ejecución de los procesos, como por ejemplo el rastrear el estado que guarda el proceso en un determinado momento, la asignación y delimitación de las responsabilidades dentro de la organización, reducción del tiempo de búsqueda de documentos y coordinación de actividades entre otras [Arroyo-Sandoval et al., 2003].

En el contexto de este trabajo de tesis, un proceso es considerado como una agrupación de actividades dentro de una organización, donde las personas trabajan conjuntamente para lograr un resultado deseado [Ould, 1995]. Un proceso está compuesto por roles y actores.

Un *rol* es un conjunto de actividades que son ejecutados por un actor o un conjunto de ellos dentro de la organización. Un *actor* es aquella persona encargada de llevar a cabo las actividades que corresponden a un rol. Con la finalidad de ilustrar la forma en que se realizan los procesos, la figura 1 ilustra la forma en como se lleva a cabo en una organización un *proceso de venta de seguros*.

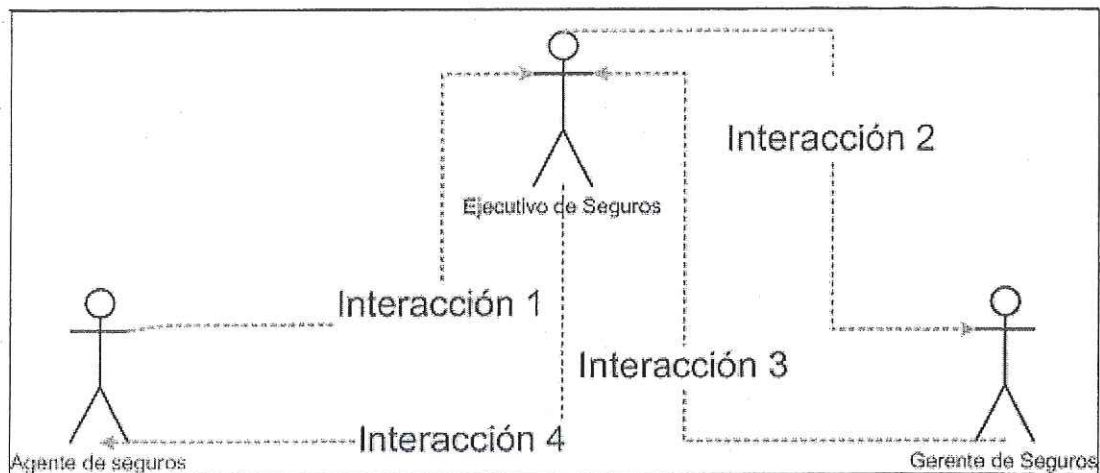


Figura 1. Ilustración de *proceso de venta de seguros*.

El proceso está constituido por los roles *Agente de seguros (AS)*, *Ejecutivo de seguros (ES)* y *Gerente de seguros (GS)*. La finalidad del proceso es asegurar a todos aquellos clientes que cumplan con un conjunto de requisitos establecidos por la organización. El *AS* es el encargado de ofrecer estos seguros a los posibles clientes, y en caso de que alguno de ellos se interesara en adquirirlo, le entrega al *AS* la documentación requerida para que su



solicitud sea evaluada. El *AS* le envía esta documentación al *ES* (interacción 1), y éste debe evaluar si la solicitud cumple con los requisitos y plasmar los resultados de la evaluación en un documento que debe enviar al *GS* (interacción 2). El *GS* recibe la evaluación y en base a ella decide si el seguro será aprobado o rechazado. Esta respuesta es enviada por escrito, junto con la evaluación, al *ES*, quien archiva el resultado y envía el resultado de la evaluación al *AS* para que éste a su vez le haga llegar la información al cliente.

En el proceso anterior se pueden apreciar algunas de las características en la forma natural de llevar a cabo los procesos. Primeramente, cada actor conoce cuales actividades forman parte de las responsabilidades de su rol desde antes de que el proceso comience a ejecutarse. Además, las interacciones entre actores se da de manera directa, es decir, sin que haya algún intermediario que se encargue de establecer y coordinar la comunicación entre ellos, de tal manera que en caso de que algún recurso o documento necesite ser enviado a algún actor, tales como solicitudes y evaluaciones en el caso del proceso anterior, se establece la comunicación de uno a uno y se envía. Finalmente se puede apreciar que los actores tienen disponibles todos los recursos que requieren para llevar a cabo las actividades que les corresponde ejecutar.

Una manera de mejorar la forma en como los procesos son llevados a cabo, es por medio de sistemas de coordinación de procesos o sistemas de workflow, los cuales son definidos por la *workflow Management Coalition* (WfMC) como la automatización y coordinación de un proceso organizacional, ya sea completa o parcialmente, durante la cual documentos,

información y/o tareas son pasados de un participante a otro para la realización de una acción, de acuerdo a un conjunto de reglas procedurales previamente establecidas.

Un sistema de workflow funciona en base a cambios de estado. El concepto de estado se refiere al comportamiento que tiene el proceso en un momento de tiempo determinado, y puede cambiar como reacción a algún evento que suceda. Estos eventos por lo general son el cumplimiento de alguna actividad. Para ilustrar mejor el funcionamiento de un sistema de workflow, las figuras 2, 3 y 4 muestran los estados por los que pasan los roles del proceso de *venta de seguros*.

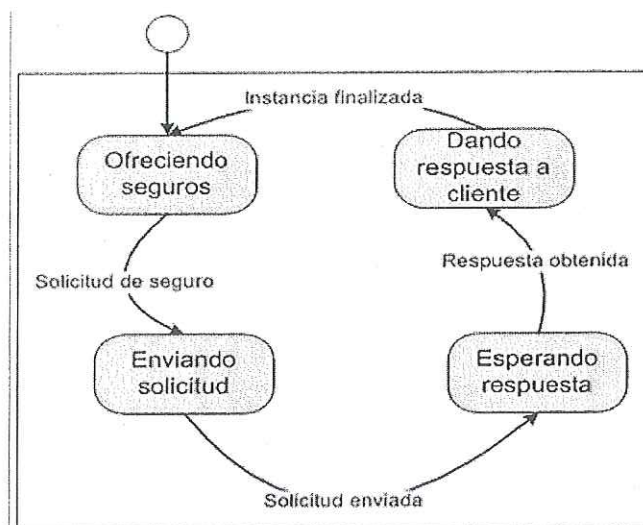


Figura 2. Diagrama de estados del rol *Agente de seguros*.

El proceso inicia (figura 2) con el actor *Agente de seguros (AS)* en el estado *Ofreciendo seguros*, en el cual se encuentra recaudando clientes. Cuando el *AS* obtiene una *solicitud de seguro* por parte del cliente, significa que la actividad que realizaba el *AS* en el estado *Ofreciendo seguros* ha finalizado, y que el estado del proceso en ese momento (figura 2),

para el rol *AS*, es *Enviando solicitud*. La actividad en este estado consiste en enviar la solicitud hecha por el cliente al actor *Ejecutivo de seguros (ES)*, y una vez que la actividad concluye, pasa al estado *Esperando respuesta*.

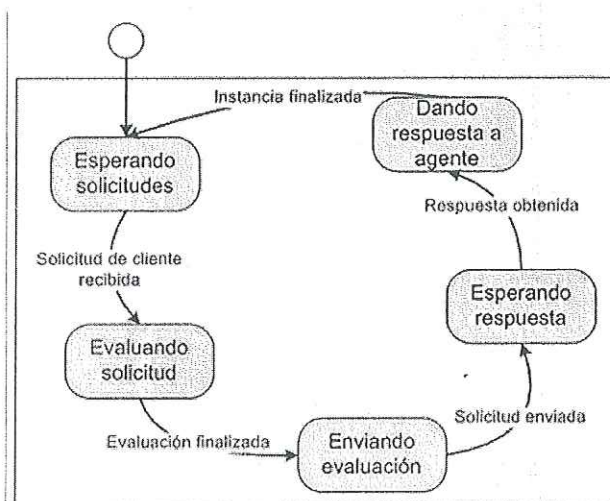


Figura 3. Diagrama de estados del rol *Ejecutivo de seguros*.

El *ES* por su parte inicialmente se encuentra en el estado *Esperando solicitudes* (figura 3), el cual cambia al estado *Evaluando solicitud* cuando recibe la solicitud enviada por el *AS*. En ese estado, el *ES* evalúa la solicitud recibida, y al finalizar, el estado del proceso para este actor cambia a *Enviando solicitud* al *GE* y cuando esta actividad concluye, el *ES* permanece *Esperando respuesta*.

Asimismo, el *GE* que inicialmente se encuentra en el estado *Esperando solicitudes* (figura 4), cambia al estado *Decidiendo aceptación* al momento de recibir la evaluación que envió el *ES*. Ya que el *GE* decide si acepta o no la solicitud de seguro, y lo plasma por escrito,



pasa al estado *Enviando evaluación*, en el cual le hace llegar al *ES* la respuesta a dicha solicitud y finalmente vuelve al estado inicial *Esperando solicitudes*.

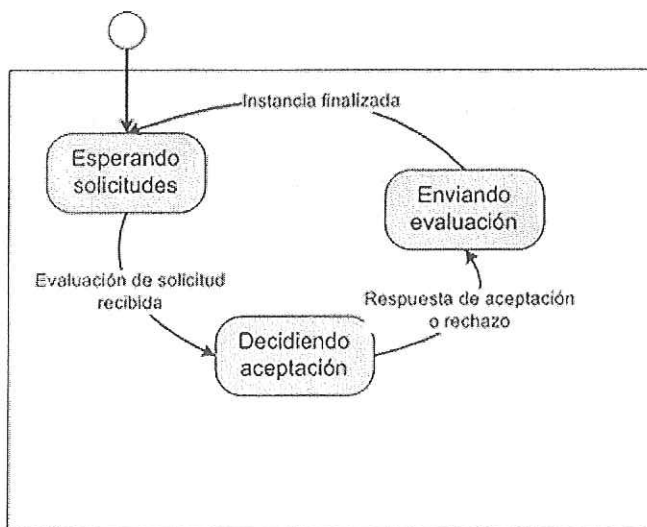


Figura 4. Diagrama de estados del rol *Gerente de seguros*.

Cuando el *ES* recibe la respuesta a la solicitud, abandona el estado *Esperando respuesta* en el que se encontraba y cambia al estado *Dando respuesta al cliente*, en el cual envía al *AS* la respuesta a la solicitud que éste envió. Finalmente, el *AS* recibe la respuesta por parte del *ES* y pasa al estado *Dando respuesta al cliente* y le informa al cliente si la solicitud hecha fue aceptada o rechazada, y una vez que esta actividad finaliza, esta instancia del proceso también termina.

En base a la descripción anterior, se puede ver que cada vez que un evento sucede, el proceso sufre un cambio de estado. Es aquí donde entra en funciones el sistema de workflow, porque tiene la responsabilidad de notificar al actor que el proceso ha cambiado de estado y debe indicarle también las actividades que debe realizar en ese nuevo estado. Además se puede observar que al producirse un cambio de estado en las actividades o rol

de un actor, puede verse afectado el estado de los roles de algún o algunos otros actores, por lo que el sistema de workflow debe notificar a todos los actores afectados con cada cambio de estado, cumpliendo así con la función de automatizar la coordinación del proceso.

Los sistemas de workflow comprenden un área de investigación y desarrollo de mucha relevancia práctica que está creciendo rápidamente. Sin embargo, por su naturaleza distribuida, su evolución está íntimamente ligada al mejoramiento de los modelos de sistemas distribuidos existentes. Un sistema distribuido es aquel en el que los componentes localizados en computadoras, conectados en red, comunican y coordinan sus acciones únicamente mediante el paso de mensajes [Coulouris et al, 2001].

Según Coulouris [Coulouris et al., 2001], los sistemas distribuidos presentan seis características principales:

- Compartir recursos.- Un sistema distribuido permite compartir los recursos de hardware y software como los discos, impresoras, archivos, compiladores, etcétera, asociados a diversas computadoras de una red.
- Heterogeneidad.- La heterogeneidad se refiere al grado en el cual el sistema puede ser extendido agregándole nuevos recursos de distintos proveedores, entre los cuales están las redes, hardware, sistemas operativos, lenguajes de programación y software, sin que las diferencias entre estos sean notorias para el usuario.

- Concurrencia.- En un sistema distribuido, varios procesos operan al mismo tiempo en diferentes nodos de la red. Estos procesos pueden comunicarse con otros durante su operación normal.
- Escalabilidad.-Al menos en principio, los sistemas distribuidos son escalables en el sentido de que las capacidades del sistema se pueden incrementar agregando nuevos recursos para cubrir las nuevas demandas de dicho sistema. En la práctica, la escalabilidad puede ser limitada por la red que vincula a las computadoras individuales en el sistema. Si se agregan muchas computadoras nuevas, entonces la capacidad de la red puede ser inadecuada.
- Tolerancia a fallas.- Los sistemas computacionales son susceptibles a fallas. Cuando aparecen fallas en el hardware o en el software, los sistemas pueden producir resultados incorrectos o pueden finalizar antes de completar la operación solicitada. Los fallos en un sistema distribuido son parciales, es decir, algunos de los componentes fallan mientras otros continúan funcionando adecuadamente.
- Transparencia.- La transparencia consiste en esconder al usuario la naturaleza distribuida del sistema. Una meta del diseño del sistema es que los usuarios tengan acceso transparente a los recursos y no tengan necesidad de conocer algo sobre la distribución del sistema. Sin embargo, en muchos casos, dar a los usuarios algo de conocimiento sobre la organización del sistema les permite una mejor utilización de los recursos.

Existen dos esquemas genéricos de arquitecturas de sistemas distribuidos, clasificados en base a la agrupación y funcionalidad de sus componentes. El primero de ellos es el esquema cliente-servidor, en la cual se tienen dos tipos de nodos, de los cuales el cliente se encarga de proporcionar la interfaz de interacción entre el usuario y el sistema, y el servidor se encarga de proveer la funcionalidad del sistema. El segundo de ellos es un esquema distribuido, en el cual todos los nodos desempeñan tareas semejantes y donde no hay una distinción entre clientes y servidores [Sommerville, 2002].

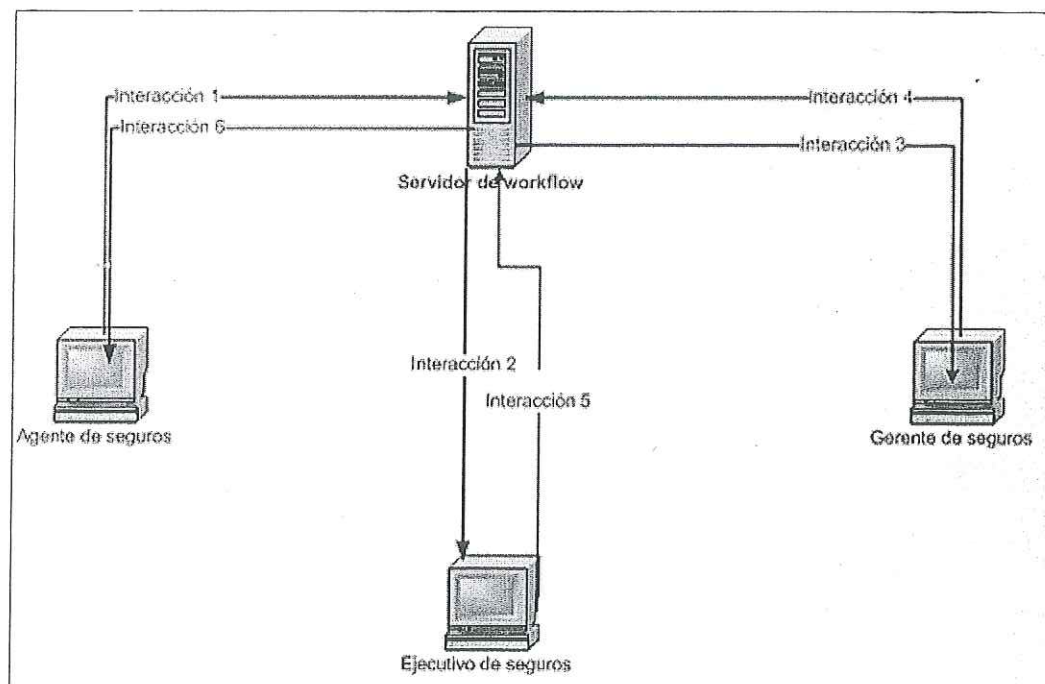


Figura 5. Sistema de workflow en un esquema cliente-servidor.

Como ya se mencionó, la mayoría de los sistemas de workflow actuales están desarrollados en el esquema cliente-servidor. La figura 5 muestra la forma en como se lleva a cabo la comunicación en un esquema cliente-servidor en el proceso de *venta de seguros*. En este esquema, cada vez que alguno de los roles cambia de estado, se establece una



comunicación con el servidor, para que sea éste quien se encargue de determinar el nuevo estado en el cual estarán todos los roles. Una vez que el servidor determina cuales de los roles requieren cambiar de estado, este se encarga de hacérselos saber.

En la siguiente sección se presenta la problemática que presentan los sistemas de workflow basados en un esquema centralizado.

## ***1.2 Planteamiento del problema***

Como ya se mencionó anteriormente, los WfMSs comprenden un área de investigación y desarrollo de mucha relevancia práctica que está creciendo rápidamente. Sin embargo, a pesar de la gran proliferación de sistemas de workflow comerciales, la tecnología de workflow es relativamente inmadura para tratar la complejidad de las aplicaciones en el mundo real [Worah y Sheth, 1997].

Uno de los problemas a los que se enfrentan los sistemas de workflow basados en un esquema cliente-servidor, es el brindar un soporte adecuado al *enactment* de procesos grandes, pues estos sistemas presentan los problemas propios de la [Kwang-Hoon y Dong-Soo, 2001], como son:

- Sobrecarga en el servidor.- Se presenta debido a que existe una comunicación cliente-servidor (o viceversa) cada vez que se asigna una tarea a un actor, durante la ejecución de la actividad, durante la gestión y manejo de información necesaria para la actividad y para notificar el cumplimiento de una actividad, lo que deriva en una

gran carga de trabajo para el servidor. Si a todo esto se le agrega que se puede estar coordinando un proceso de gran tamaño (alrededor de 100,000 actividades y 10,000 usuarios o más grandes) [Muth et. al 1998], es muy probable que el servidor sufra una sobrecarga de trabajo.

- Poca tolerancia a fallos.- En una arquitectura centralizada, toda la coordinación del proceso se lleva solamente en el lado del nodo servidor. En caso de que este nodo falle, el sistema en su totalidad colapsará, ya que los nodos clientes no tendrán acceso a la información necesaria para ejecutar la actividad que están realizando, ni serán informados de las nuevas tareas que deben realizar.
- Poca escalabilidad.- Cuando la capacidad de procesamiento del servidor es superada por la ejecución de un proceso grande, es muy probable que éste sufra de una sobrecarga de trabajo, lo cual se refleja en un bajo rendimiento del servidor y, en algunos casos, en el colapso del mismo. Es por esto que la escalabilidad del proceso en lo que respecta al número de actividades y actores (nodos) que se van a administrar en el flujo de trabajo están limitados por la capacidad del servidor.

También es importante hacer notar que los modelos de WfMS actuales diseñados bajo arquitectura cliente servidor no reflejan la forma natural en que se lleva a cabo un proceso. Se define como forma natural de trabajo a las características de la manera en que los procesos organizacionales son llevados a cabo en la vida real sin la ayuda de un sistema de coordinación o un sistema de workflow.

Con el fin de analizar y comparar el esquema cliente-servidor con respecto a la forma natural de trabajo en las organizaciones, se muestra la comunicación entre nodos en un sistema de workflow en un esquema cliente-servidor (figura 6(a)).

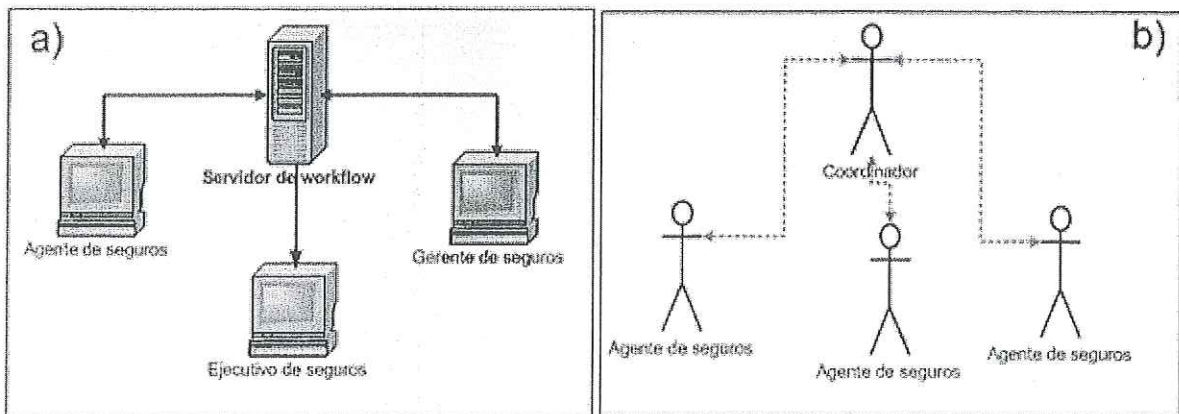


Figura 6. a) Sistema de workflow con esquema centralizado. b) Forma natural de trabajo con esquema centralizado.

Este esquema es trasladado a los actores de un proceso, mostrando como sería la comunicación entre ellos (figura 6(b)). Haciendo un análisis de las figura 6(b) se puede observar que, basados en el esquema cliente-servidor, se requiere un actor coordinador para que haga las funciones de intermediario entre los actores, lo cual no concuerda con la forma natural en como se llevan a cabo los procesos, en donde la comunicación, en la mayoría de los casos, se da de manera directa, como se muestra en la figura 7(b).

Otro punto que no refleja la forma natural de trabajo, es que los actores normalmente conocen las actividades que deben realizar y conocen también de donde obtener los documentos y la información necesaria para llevar a cabo sus actividades, así como el destino de estos una vez terminada la actividades, ya sea que se almacenen, o bien, se envíen a otro actor que los necesite para llevar a cabo alguna actividad. Esto no sucede con

la arquitectura cliente-servidor, ya que la definición del flujo de trabajo, la descripción de las actividades a realizar, así como los recursos necesarios para llevar a cabo las actividades solo son conocidas por los nodos clientes en el momento justo en que deben llevar a cabo una actividad, ya que hasta este momento toda la información al respecto se encuentra en el servidor central.

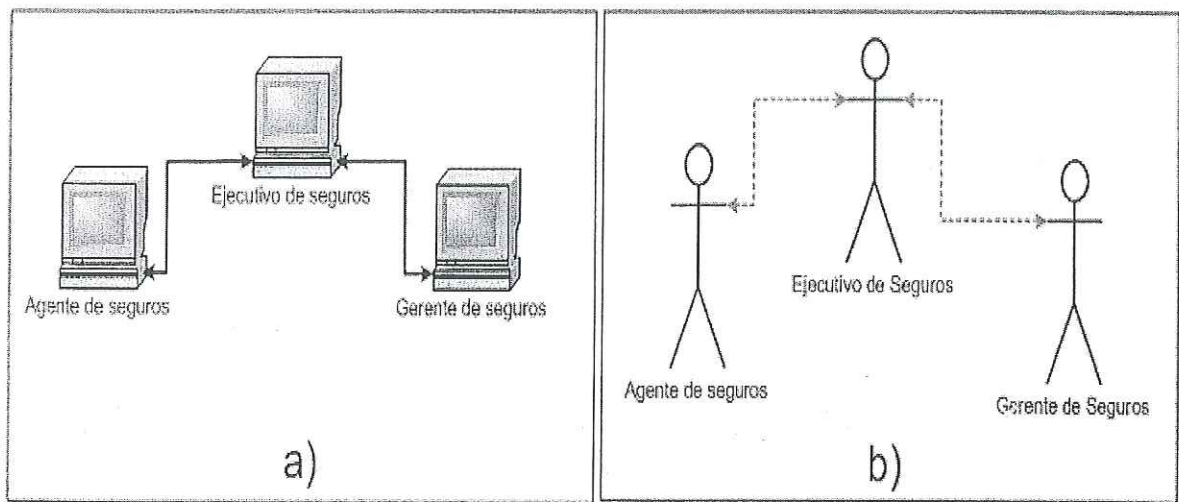


Figura 7. a) Sistema de workflow con esquema distribuido. b) Forma natural de trabajo.

Un enfoque que se está utilizando actualmente para dar solución a los problemas mencionados es por medio de sistemas de workflow basados el concepto de la forma natural de trabajo (figura 7(a)). La idea principal es que cada nodo tenga conocimiento de las actividades que deben ejecutarse en él, de manera que pueda hacerse responsable de la coordinación de dichas actividades y funcione de manera independiente del resto de los nodos. Un esquema de este tipo es factible, gracias a que actualmente la mayoría de los nodos en los que trabajan los actores de un proceso tienen gran capacidad de procesamiento.



En este sentido, en este trabajo se propone el diseño de una arquitectura distribuida que sirva de base a la construcción de sistemas de workflow que puedan dar soporte al enacment de sistemas de gran tamaño. Esta arquitectura cumple con los requerimientos de un sistema de workflow distribuido basado en la forma natural de trabajo, además de cumplir con los requerimientos propios de los sistemas de workflow, los cuales describimos a continuación [Prior, 2002]:

- Escalabilidad.- Los WfMSs no deben sufrir una disminución de su rendimiento al momento de que se presente un incremento de instancias de procesos en ejecución, actividades del proceso, volumen de datos de la aplicación y número de actores que desarrollan las actividades.
- Recuperación de fallas.- Los WfMSs deben poder manejar la mayor cantidad de fallas de hardware y software con la menor participación humana posible.
- Disponibilidad.- Los WfMSs no deben estar fuera de servicio por largos períodos de tiempo, especialmente si están dando soporte a procesos críticos.
- Monitoreo.- Los WfMSs deben conocer en cualquier momento el estado en que se encuentran las instancias de los procesos que están ejecutando.
- Historial.- Los WfMSs deben tener conocimiento de todas acciones que han sido apoyadas, como por ejemplo actividades e instancias en ejecución, así como también las que ya han sido ejecutadas.
- Soporte al uso de aplicaciones externas.- La ejecución de una actividad particular quizás requiera la utilización de herramientas externas, tales como procesadores de

palabras, hojas de cálculo, sistemas CAD, sistemas expertos y otros. Los WFMS deben poder iniciar estas aplicaciones y tener control de cuándo finalizan, así como de los datos que estas aplicaciones leen y modifican.

En años recientes se han desarrollado algunas arquitecturas de Workflow que sirven de base para el desarrollo de sistemas de workflow distribuido. Entre estos trabajos se encuentra INCAs [Barbara et al., 1996], EXOTICA/FMQM [Alonso et al., 1995], MENTOR [Wodtke 1996] y WONDER [Silva et al., 1999]. Estas arquitecturas están diseñadas bajo un enfoque distribuido descentralizado, donde la idea principal es que la carga de procesamiento requerida para la ejecución de las actividades se lleva a cabo en varios nodos de la red. Sin embargo, la coordinación de las actividades está especificada en forma centralizada, de tal manera que cada vez que se requiere iniciar la ejecución de una nueva instancia del proceso es el servidor central quien se encarga de determinar el flujo que seguirán las actividades que se van a llevar a cabo.

Asimismo, la información es manejada de forma centralizada, por lo que sigue existiendo un gran flujo de datos en la red cada vez que alguna actividad requiere parte de esta información. Además, solo algunas de las arquitecturas mencionadas anteriormente proponen alguna estrategia de recuperación de errores, y solo la arquitectura WONDER presenta la implementación de esta estrategia. También se propone la implementación de un prototipo que permite probar la factibilidad de construir aplicaciones de workflow basadas en la arquitectura propuesta.

### ***1.3 Objetivo general.***

El objetivo principal de este trabajo es diseñar una arquitectura distribuida multiservidor para la administración de sistemas de workflow, que sirva como base para el desarrollo de sistemas de workflow.

### ***1.4 Objetivos específicos.***

- Entender la problemática que presentan los sistemas de workflow desarrollados en base a la arquitectura cliente-servidor.
- Obtener los requerimientos para el diseño de una arquitectura de workflow distribuida.
- Desarrollar un prototipo de sistema de workflow distribuido que utilice las características de la arquitectura distribuida en base a un proceso organizacional adecuado a las características de la arquitectura.
- Probar la arquitectura por medio de un prototipo.

### ***1.5 Metodología.***

Para la realización de este trabajo primeramente se hizo una revisión de la literatura relevante sobre sistemas de workflow y sistemas distribuidos con la finalidad de comprender la relación entre estas dos áreas, analizar la problemática de los sistemas de workflow tradicionales y conocer las soluciones que se proponen a esta problemática. Enseguida, se realizó el análisis de los requerimientos que debe cumplir una arquitectura de

workflow distribuida, para posteriormente definir los componentes y la estructura de éstos dentro de la arquitectura con el fin de satisfacer los requerimientos obtenidos.

Como se puede observar en los objetivos de este trabajo, se estableció la necesidad de desarrollar prototipos para probar la viabilidad tecnológica de desarrollar sistemas de workflow basados en la arquitectura propuesta en este trabajo, así como también probar que estos sistemas cumplen con la funcionalidad establecida en los requerimientos de la arquitectura.

Primeramente se desarrolló el prototipo de un sistema de workflow que permitió probar la existencia de tecnología adecuada para construir sistemas de workflow distribuidos basados en nuestra arquitectura. Este prototipo fue diseñado en base a un escenario distribuido ideal en el cual no hubiera interacción de los nodos distribuidos (en los que se lleva a cabo la ejecución del proceso) con un nodo Distribuidor, y sirvió para probar que el diseño de la arquitectura permite llevar a cabo la coordinación y ejecución de un proceso organizacional de forma distribuida. Sin embargo, con este prototipo se pudo apreciar que aspectos importantes de la forma natural en como se lleva a cabo el trabajo, por ejemplo permitir que los actores puedan trabajar en distintos nodos de forma transparente o manejar la ausencia de actores en los nodos (nodos no disponibles), no son abordados en este prototipo, por lo que fue necesario hacer un rediseño de la arquitectura y desarrollar un nuevo prototipo.

El segundo prototipo realizado, nuevamente permitió mostrar la viabilidad tecnológica de desarrollar sistemas de workflow basados en la arquitectura, aún con las modificaciones



que se hicieron a la arquitectura después del desarrollo del primer prototipo. Así mismo, con el desarrollo de este prototipo también se pudo probar que la arquitectura propuesta en este trabajo contempla los componentes necesarios para cumplir con los requerimientos funcionales de un sistema de workflow distribuido.

## ***1.6 Contenido de la tesis***

El contenido de la tesis está dividido en seis capítulos y dos apéndices. En el capítulo II se presenta un análisis comparativo que presentan algunas de las arquitecturas de workflow distribuidas que han sido desarrollados. En el capítulo III se describen el diseño de la arquitectura propuesta, los componentes de la misma, así como la funcionalidad de estos componentes, así como la forma en como los errores son manejados en la arquitectura propuesta. En el capítulo IV se presentan los aspectos de implementación del prototipo, como son análisis de los procesos utilizados y de la tecnología utilizada, además de explicar también las pruebas de funcionalidad aplicada a la arquitectura. Finalmente, en el capítulo V se presentan las conclusiones de este trabajo, así como también el trabajo futuro.

También se presentan tres apéndices. El apéndice A muestra el análisis de la funcionalidad de la arquitectura, en el apéndice B se presentan aspectos técnicos de la implementación del prototipo y en el apéndice C se presentan los casos de prueba aplicados a los prototipos desarrollados en este trabajo.

## **Capítulo II. Arquitecturas de Workflow Distribuidas.**

---

### ***II.1 Introducción***

En los últimos años se ha visto un creciente interés en el desarrollo de proyectos que contribuyan a mejorar la funcionalidad y confiabilidad de los sistemas de workflow. Sin embargo, sólo una pequeña parte de los esfuerzos realizados han sido enfocados ha solucionar la problemática que presentan los sistemas de workflow tradicionales desarrollados bajo un esquema centralizado (cliente-servidor).

En un esquema centralizado, el sistema se visualiza como un conjunto de servicios que se suministran a los clientes que los utilizan. Los servidores y los clientes son tratados de forma diferente en estos sistemas, ya que los servidores se encargan de proveer los servicios y los clientes son quienes utilizan estos servicios [Steflick y Edwards, 1998].

El enfoque que se está utilizando actualmente para solucionar la problemática de los sistemas de workflow tradicionales es por medio de un esquema distribuido, en los cuales no existe diferencia entre los servidores y los clientes, y el sistema se visualiza como un conjunto de objetos que interactúan y su ubicación es irrelevante. En otras palabras, cada nodo cumple de manera simultanea con funciones de cliente, como por ejemplo proveer la interfaz que permite al cliente llevar control de la ejecución de las actividades, así como también funciones de servidor, como la coordinación de las actividades, la recuperación de errores, proveer entidades de información, comunicación entre nodos, entre otras.

En este capítulo se presenta el análisis de algunos de los trabajos más relevantes en el área de sistemas de workflow distribuidos. Además, se definen las características deseables en todos los sistemas de workflow distribuido, y finalmente se muestra una tabla comparativa de estas características en los trabajos analizados.

## ***II.2 Arquitecturas distribuidas de workflow existentes: Análisis comparativo.***

### **II.2.1 INCA's.**

En el proyecto INCA's [Barbara et. al. 1996], se propone un sistema administrador de aplicaciones de workflow basado en el paso de mensajes, con el propósito de que los nodos en la red (estaciones de trabajo) sean independientes y la comunicación entre ellos sea únicamente para el paso de mensajes. La idea es que un mensaje denominado *information carrier* (INCA) contenga la información necesaria para el control y flujo de los datos, y que los nodos tengan la capacidad de entender y modificar la información del INCA's. De esta manera los nodos pueden ejecutar las actividades correspondientes definidas previamente en ese nodo, y posteriormente modificar la información y enviar el INCA al nodo correspondiente y seguir así el flujo de trabajo.

La arquitectura INCA's tiene como ventaja principal que permite aprovechar las capacidades de procesamiento de los nodos, debido a que la ejecución de las actividades se

lleva a cabo de forma descentralizada. De esta manera, la carga de trabajo es distribuida a través de los distintos nodos que forman parte del proceso. Sin embargo, la coordinación de las actividades se hace de manera centralizada en el servidor. Dicho de otra forma, cada vez que un actor requiere iniciar una instancia de un proceso, debe informar al nodo servidor para que éste se encargue de comenzar la nueva instancia. Para comenzar la instancia, el servidor debe determinar el flujo que seguirán las actividades, empaquetar esta información en el INCA y enviarla al nodo que iniciará la instancia del proceso. Aunado a lo anterior, las entidades de información, tales como documentos y datos son manejadas de manera centralizada. Estas entidades de información son alojadas en el servidor y cada vez que algún actor requiere utilizar alguna de ellas, debe obtenerlas del servidor, lo que ocasiona un flujo de datos excesivo sobre la red.

Una limitante de esta arquitectura, los autores no especifican una estrategia de manejo de errores. En base al análisis de la arquitectura INCAs, se puede apreciar que los errores pueden suceder de acuerdo a las siguientes circunstancias:

- El INCA no puede ser enviado a alguno de los nodos debido a problemas de la red.
- El INCA no puede ser enviado a algunos de los nodos debido a que el nodo destino no se encuentra disponible (está apagado el nodo o no se está corriendo la aplicación).
- Los nodos son apagados (de forma planeada o accidental) antes de que termine la ejecución del proceso.



Finalmente, los autores tampoco hacen mención sobre la existencia de alguna aplicación o prototipo que implemente esta arquitectura, por lo que se dificulta conocer la viabilidad de desarrollar sistemas de workflow basados en esta arquitectura.

### **II.2.2 EXOTICA.**

Uno de los proyectos más significativos en el área es Exotica/FMQM (FlowMark on Message Queue Manager) [Alonso et. al 1995], donde se proponen los requerimientos de un sistema de workflow distribuido, en el cual un conjunto de nodos autónomos cooperan para completar la ejecución de un proceso. Cada nodo funciona de una manera independiente al resto del sistema, siendo la única interacción entre nodos a través de mensajes persistentes que informan que la ejecución de una actividad del proceso finalizó.

Los autores mencionan que la funcionalidad de la arquitectura se lleva a cabo de la siguiente manera:

Se define el flujo que sigue el proceso y se define también la forma en que serán llevadas a cabo las actividades en el nodo. En cada nodo que participará en el proceso se almacena la información de control pertinente en tablas, por ejemplo, qué entidades de información requiere, qué requisitos deben ser cumplidos para que la actividad comience a ejecutarse, a quien hay que notificar una vez que la actividad ha sido completada, y cuales actividades se ejecutarán en ese nodo.

Cuando se requiere iniciar una nueva instancia de un proceso, el servidor envía un mensaje a los nodos que contienen la actividad inicial para iniciar la ejecución de esa actividad. El mensaje es recibido en los nodos y almacenado en una cola de mensajes. En cada nodo se recibe el mensaje y se comienza a ejecutar la actividad. Cuando la actividad finaliza, se envía la notificación correspondiente al nodo que debe continuar la ejecución del proceso. El nodo que ejecuta la última actividad del proceso debe indicar al servidor que el proceso ha finalizado.

Sin embargo, a pesar de las ventajas que presenta este trabajo, también presenta algunos inconvenientes, como lo es el que la información de control sea estática en cada nodo, ya que esto limita la movilidad de los usuarios. Esto significa que los actores solo pueden trabajar en un nodo en donde esté almacenada la información de control de los roles que le corresponden.

También se presenta el inconveniente de que las entidades de información son manejadas de forma centralizada (datos y documentos son almacenados en un servidor central), lo que genera un flujo de datos muy grande en la red al momento en que los actores requieren alguna de estas entidades.

### II.2.3 MENTOR.

En el proyecto MENTOR [Wodtke et al., 1996] se propone una arquitectura multiservidor, donde en cada servidor se tiene un motor de workflow. Las entidades de información son manejadas de forma distribuida entre los servidores, y las actividades son ejecutadas de manera independiente en cada servidor. Sin embargo, este trabajo no funciona bajo el esquema de nodos independientes debido a que cada servidor se dedica a atender las peticiones de un segmento de nodos, lo que lleva a un esquema de múltiple cliente-servidor (ver Figura 8).

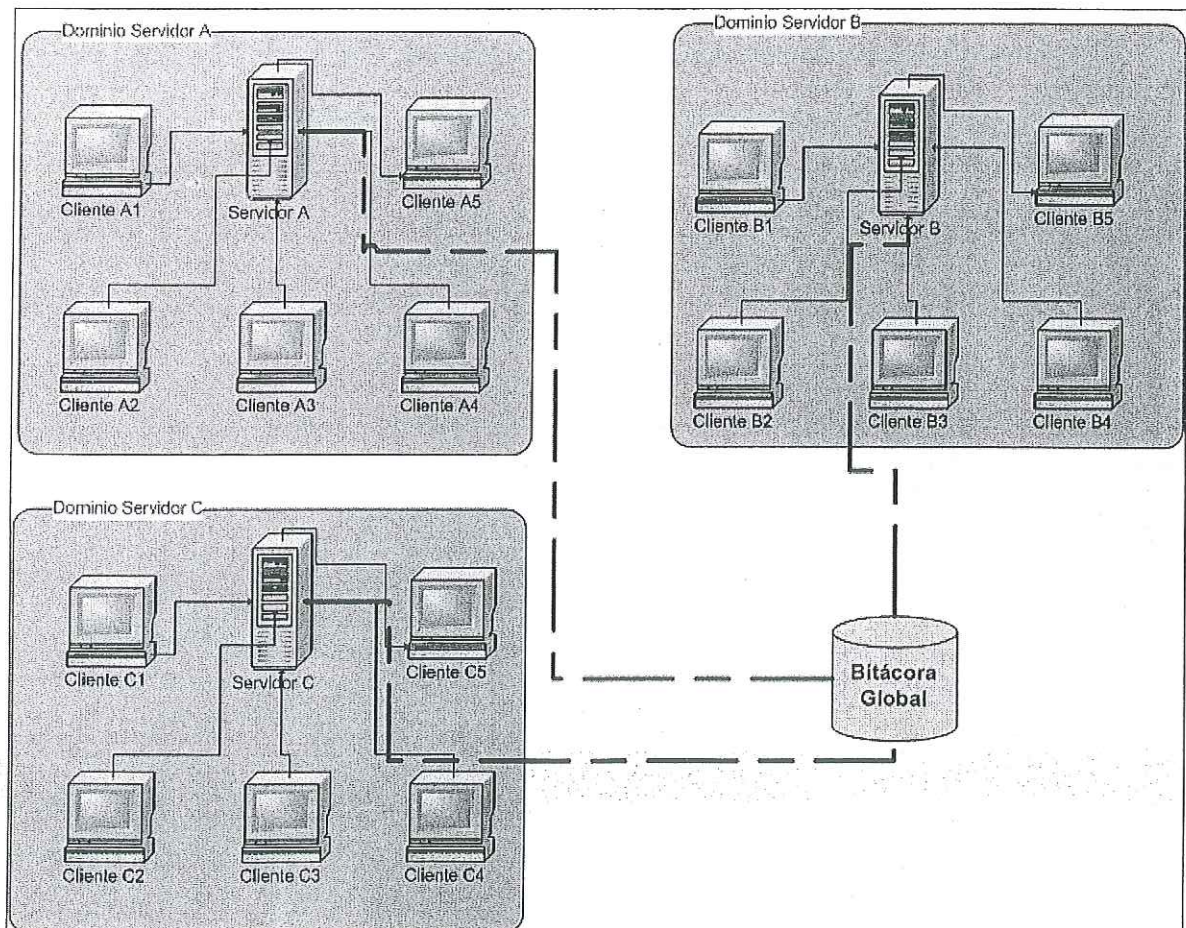


Figura 8. Esquema de la Arquitectura MENTOR.



Una de las características favorables proyecto de MENTOR es que toma en cuenta aspectos de manejo de errores. Para este fin, los autores proponen el uso de un servidor central que sirva como bitácora del proceso, en el cual los servidores restantes registran las actividades que se han ejecutado y que se encuentran en ejecución. En caso de que un error, se obtiene de la bitácora el estado en que se encontraba el proceso y pueda ser restaurado en cualquiera de los servidores en donde se haya presentado el problema.

Si bien es cierto que el proyecto MENTOR aborda la problemática de manejo de errores, cada servidor continúa teniendo (aunque en menor medida) los mismos problemas que sufren los servidores en un esquema cliente-servidor, los cuales fueron discutidos en la sección I.2.

#### **II.2.4 WONDER.**

WONDER [Silva et al., 1999] presenta una arquitectura distribuida de workflow basada en el enfoque de nodos independientes. Las actividades son definidas y almacenadas en los diferentes nodos en los que se lleva a cabo la ejecución del proceso, permitiendo con esto la ejecución descentralizada de las actividades, por lo que se aprovecha la capacidad de procesamiento de los nodos que participan en el proceso.

En este trabajo se propone un mecanismo de recuperación de errores que consiste en que cada vez que una actividad es ejecutada en un nodo, el estado es almacenado de forma



local, de tal forma que si la actividad falla se revisa el estado almacenado en el nodo anterior para recuperar el estado en que se encontraba la instancia que se está desarrollando. Sin embargo, ese enfoque de recuperación de errores presenta los siguientes inconvenientes:

- Si falla el nodo en que se está ejecutando una actividad y el nodo en que se ejecutó la actividad anterior, el estado en que se encontraba la instancia no puede ser recuperado y las actividades realizadas en esos nodos deberán repetirse.
- Un nodo puede ejecutar varias actividades de forma secuencial, por lo que el estado se guardará localmente varias veces. Si ese nodo falla, al recuperar el estado del nodo anterior, no se podrá recuperar el estado en que se encontraba la instancia que se estaba ejecutando.

Por otra parte, esta arquitectura tampoco contempla los aspectos de la forma natural en como los procesos reales son llevados a cabo.

Finalmente, en WONDER tampoco se especifica como son manejadas las entidades de información (documentos y datos) y se limita únicamente a los aspectos de coordinación de actividades.

### **II.2.5 RainMaker**

En el proyecto RainMaker [Paul et al, 1998], se muestra una infraestructura de workflow interoperable y heterogénea. En este enfoque cada nodo tiene capacidad de dar soporte a la

ejecución de uno o varios tipos de actividades (llamados *servicios* por los autores del proyecto) y propone también el uso de un nodo que tenga la función de coordinar la ejecución de las actividades y determinar en cual nodo se debe llevar a cabo cada actividad. Cuando un nodo le indica al servidor que desea iniciar la instancia de un proceso, éste determina los servicios (actividades) que requiere dicha instancia, y en base a esos servicios determina en cuáles nodos se van a ejecutar las actividades requeridas.

La arquitectura de RainMaker cuenta con un mecanismo global en el nodo servidor, que se encarga de coordinar la ejecución de cada instancia del proceso, así como también un directorio de servicios que proveen los nodos participantes, de manera tal que el nodo servidor pueda determinar cuáles actividades se deben ejecutar y cuales nodos están capacitados para hacerlo.

Uno de los inconvenientes de esta arquitectura es que no especifica un mecanismo de manejo de errores, ni tampoco se mencionan aspectos con respecto al manejo de entidades de información. Además, tampoco se refleja la forma natural en la cual los actores llevan a cabo el proceso, ya que no permite que los actores puedan trabajar en nodos distintos al no especificar una manera de recuperar el estado en que se encuentra la parte del proceso que lleva a cabo algún actor en específico.

### ***II.3 Tabla Comparativa.***

En esta sección se presenta una tabla en donde se hace un análisis de las arquitecturas presentadas anteriormente, en base a los aspectos requeridos en una arquitectura distribuida para aplicaciones de sistemas de workflow, que son los siguientes:

- **Coordinación descentralizada de actividades.-** Cada nodo debe ser capaz de determinar el orden en que se debe llevar a cabo cada actividad, de manera que se distribuya la carga que trabajo que se genera en el servidor al llevar un control sobre el estado en que se encuentra cada una de las instancias del proceso y al determinar el nuevo estado en que se encuentra cada instancia después de que cada actividad se ejecuta. Además, se minimiza el flujo de datos sobre la red, ya que como la coordinación de las actividades se lleva a cabo de manera local, no es necesario estar enviando datos sobre la red entre cliente y servidor cada vez que alguna instancia cambia de estado. Finalmente, también se reduce el impacto que tendría una falla en el nodo que se encarga de la coordinación de las actividades en un esquema centralizado, donde el proceso entero podría verse afectado, contrario a lo que sucedería en un esquema distribuido, donde solo afectaría a las actividades que se llevan a cabo en ese nodo.
- **Ejecución descentralizada de actividades.-** Para aligerar la carga de trabajo que se genera en el servidor debido a la ejecución de todas las actividades del proceso, la capacidad de procesamiento de cada nodo debe ser aprovechada, de tal manera que cada uno de ellos pueda dar soporte a la ejecución de una o más actividades. Además de minimizar la posibilidad de una sobrecarga de trabajo en el servidor, se

reduce el que tráfico en la red cuando una actividad requiere el manejo de datos, ya que éstos son enviados del servidor al cliente para que sean vistos por el actor, y son enviados también del cliente al servidor para que éste último realice con ellos alguna operación requerida por la actividad. Finalmente, al igual que en el punto anterior, se reduce el impacto del daño que se presentaría en caso de que sufriera una falla el nodo que da soporte a la ejecución de las actividades en un esquema centralizado.

- Manejo de errores.- Poder recuperar a un sistema de workflow y llevarlo al estado en que se encontraba antes de que la falla ocurriera es de suma importancia, puesto que estos sistemas por lo general llevan el control de procesos cuya información es producto de la ejecución de una o varias actividades en un período de tiempo considerable, por lo que comenzar de nuevo a generar esta información (llevar a cabo las actividades desde el principio) en la mayoría de los casos no es una opción viable, ya que puede generar costos importantes a la organización.
- Entidades de información distribuida.- Los sistemas de workflow con un esquema centralizado generan un gran flujo de datos sobre la red, debido a que los datos se encuentran almacenados en un servidor central, por lo que cada vez que éstos son requeridos por algún actor, son enviados al nodo en que se encuentra, y una vez que han dejado de utilizarse son regresados al servidor. Al almacenarse los datos en el nodo en el cual se ejecutará la actividad que los requiera, se reduce el flujo de datos sobre la red, ya que el acceso se lleva a cabo de manera local.



- Reflejar forma natural de trabajo.- La función principal de un sistema de workflow es dar soporte a la forma en como se lleva a cabo un proceso en una organización. Por ello, el funcionamiento de estos sistemas debe reflejar la forma en como los procesos son llevados a cabo de forma natural por los actores del proceso y contemplar las distintas situaciones que se pueden presentar en la ejecución del proceso, como por ejemplo: ausencia de los actores, los actores cambian de nodo para trabajar, o bien, que los nodos se encuentren apagados o no disponibles en determinados momentos.
- Implementación.- Es importante conocer si las arquitecturas que se presentan han sido implementadas en algún prototipo, porque esto permite conocer la factibilidad para construir sistemas reales de workflow basados en estos esquemas.

Como se puede apreciar en la tabla I, de los trabajos analizados ninguno de ellos indica que la coordinación de las actividades se lleva a cabo de manera distribuida, caso contrario a lo que sucede en la forma natural en como se lleva a cabo el trabajo, en donde cada actor conoce cuáles actividades debe realizar y el orden en que éstas deben ser ejecutadas. Sin embargo, todas las arquitecturas propuestas proponen características que permiten que las actividades sean ejecutadas de manera distribuida.

Tabla I. Análisis comparativo de las arquitecturas de workflow existentes.

|                                             | INCAs | EXOTICA/FMQM | MENTOR | WONDER | RainMaker |
|---------------------------------------------|-------|--------------|--------|--------|-----------|
| Coordinación descentralizada de actividades | No    | No           | No     | No     | No        |
| Ejecución descentralizada de actividades    | Si    | Si           | Si     | Si     | Si        |
| Manejo de errores                           | No    | No           | Si     | Si     | No        |
| Entidades de información distribuidas       | No    | No           | Si     | No     | No        |
| Reflejar forma natural de trabajo           | No    | No           | No     | No     | No        |
| Implementación de la arquitectura           | No    | Si           | No     | Si     | No        |

Solo algunos de los trabajos analizados toma en cuenta características de manejo de errores (tabla I). Además, solo uno de los sistemas (RainMaker) contempla que las entidades de información estén distribuidas entre varios nodos.

Finalmente, la tabla I muestra que ninguno de los trabajos analizados en este capítulo refleja la forma natural en que los procesos son llevados a cabo, y también muestra que solo para algunos de ellos se ha reportado el desarrollo de prototipos que implementen alguna de las arquitecturas mencionadas.

## ***II.4 Resumen.***

En este capítulo se presentaron algunos de los trabajos que se han realizado con la finalidad de aminorar o solucionar la problemática que presentan los sistemas de workflow basados en una arquitectura cliente-servidor.

Las arquitecturas presentadas fueron analizadas en base a 6 aspectos importantes que las arquitecturas distribuidas de workflow deben presentar como son coordinación de actividades de forma descentralizada, ejecución de actividades de forma descentralizada, contar con un mecanismo de manejo de errores, manejar las entidades de información de forma distribuida, reflejar la forma natural en como los procesos son llevados a cabo e implementación de algún prototipo basado en esa arquitectura.

La mayoría de los trabajos revisados presenta la ejecución de las actividades del proceso de forma distribuida. Sin embargo, la coordinación de las actividades es iniciada de forma centralizada. Asimismo, mencionan poco o nada acerca de la forma en como el estado del proceso será recuperado en caso de que alguna falla ocurra durante la ejecución del proceso, además de que la distribución se refiere únicamente a aspectos de coordinación de actividades y no de manejo de las entidades de información en la mayoría de los casos.

En base al análisis realizado se pudo observar que, hasta el momento, no se ha reportado en la literatura ningún trabajo que cubra con las características deseadas en una arquitectura distribuida.

En el siguiente capítulo se presenta un análisis de los requerimientos específicos obtenidos para el desarrollo de una arquitectura distribuida de workflow, así como también la descripción de la arquitectura propuesta y los componentes que la conforman, así como las características, ventajas y desventajas que presenta en comparación con las arquitecturas presentadas en este capítulo.



## **Capítulo III. WODIS: Arquitectura Distribuida Multiservidor para la Ejecución de Sistemas de Workflow.**

---

### ***III.1 Introducción***

En este capítulo se presentan los aspectos relacionados con el análisis y diseño de la arquitectura distribuida. Primeramente se muestran los requerimientos que ésta debe cumplir, los cuales han servido de base al momento de establecer los componentes que integran la misma, así como la funcionalidad que cada uno de ellos debe tener. Posteriormente se muestra el diseño de la arquitectura propuesta, así como la descripción de cada uno de los componentes que la conforman. Por último, se muestra el análisis a través de diagramas UML, en específico mediante diagramas de casos de uso y diagramas de secuencia.

### ***III.2 Requerimientos de la arquitectura***

El término requerimientos hace referencia a la descripción de los servicios y restricciones que presenta un sistema [Sommerville, 2002]. En este trabajo, los requerimientos fueron obtenidos mediante la revisión del trabajo previo en el área y del análisis de la forma natural en la cual los procesos son llevados a cabo. Los requerimientos obtenidos se presentan a continuación:

- *La arquitectura debe cumplir con los requerimientos básicos de un sistema de Workflow* [Prior, 2002] como son escalabilidad, recuperación de fallas, disponibilidad, monitoreo, historial y soporte al uso de aplicaciones externas.
- *Las actividades del proceso deben ejecutarse de forma descentralizada.* Para este fin, en este trabajo se propone que las actividades que le corresponde ejecutar a cada actor sean ejecutadas en el nodo que le corresponde a dicho actor. Con este requerimiento se disminuye considerablemente la carga de trabajo que se genera en el servidor de un sistema de Workflow basado en la arquitectura cliente-servidor.
- *La coordinación de las actividades del proceso debe hacerse de forma descentralizada.* De esta forma, en cada nodo estará disponible la información correspondiente al flujo que siguen las actividades que le corresponden al actor que trabaja en ese nodo, evitándose con esto el flujo de datos en la red que es generado por las notificaciones que el servidor debe hacer en un esquema cliente-servidor para notificar a los clientes la actividad que deben realizar. Además, en caso de que el servidor o la red no funcionen adecuadamente, el flujo de trabajo en un esquema distribuido no será interrumpido.
- *La arquitectura debe incluir un mecanismo que permita recuperar al sistema en caso de que ocurran errores.* Éste es parte de los requerimientos básicos de un sistema de Workflow. Sin embargo, la mayoría de los trabajos previos desarrollados en el área no contempla un mecanismo de recuperación de fallas y se centran en proponer componentes de ejecución descentralizada.

### III.3.1 Descripción de la arquitectura WODIS

La arquitectura WODIS consta de dos tipos de nodos (figura 9): Nodo Distribuidor (DrN), que se encarga de coordinar la distribución de la información necesaria para la coordinación de las actividades y de prestar soporte a mantener la consistencia del proceso, y por otra parte el Nodo Distribuido (DoN) que se encarga de coordinar las actividades que cada actor debe llevar a cabo, mantener la consistencia de las actividades que se desarrollan en ese nodo, administrar la información que requieran dichas actividades y de administrar las interacciones que se requieran con otros nodos.

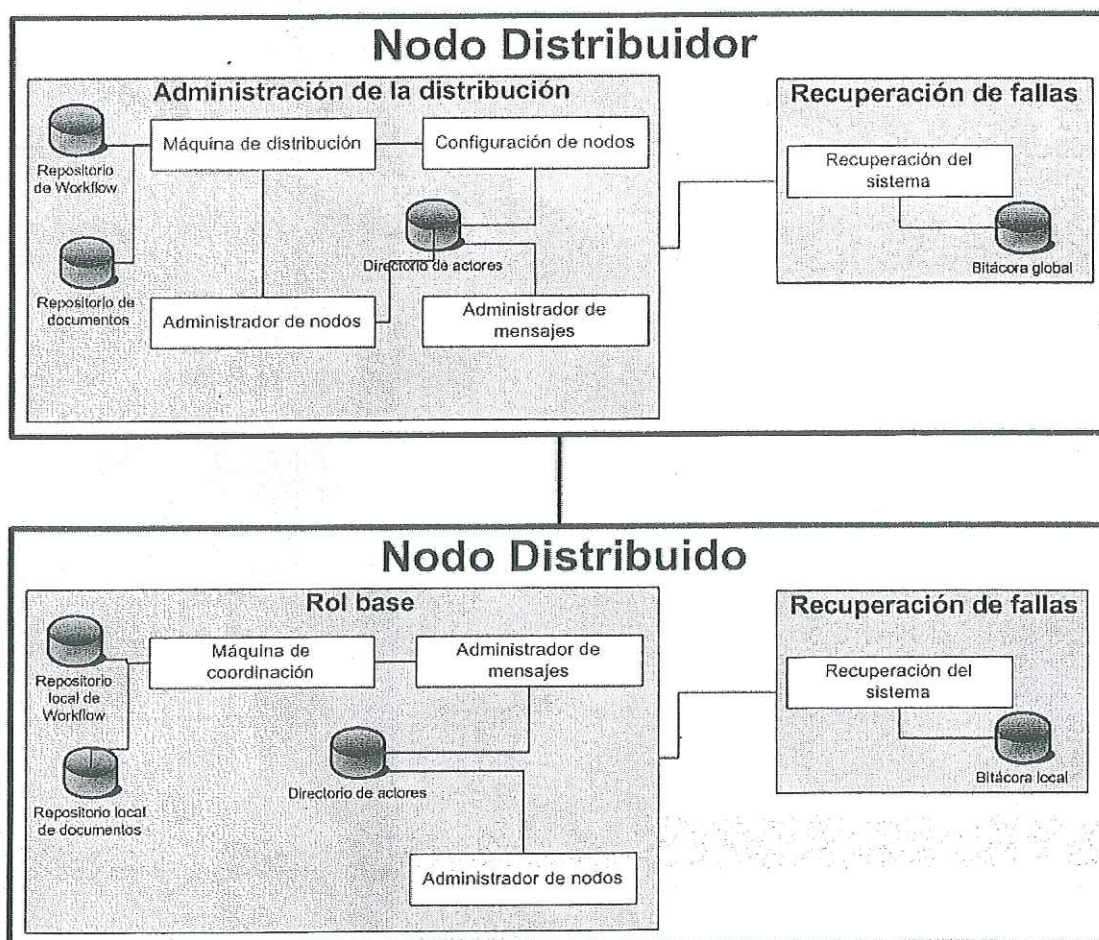


Figura 9. Arquitectura global WODIS.



Al momento de diseñar y desarrollar un sistema de workflow basado en WODIS, se tendrá solo un DrN y tantos DoN's como actores haya en el sistema (figura 10), y estos se comunicarán entre si a través del mecanismo *emisor de mensajes*. Es importante resaltar que las pantallas que muestran los *DoN's* mostrarán información de acuerdo al estado de las instancias de los procesos del actor que se encuentra trabajando en ellas.

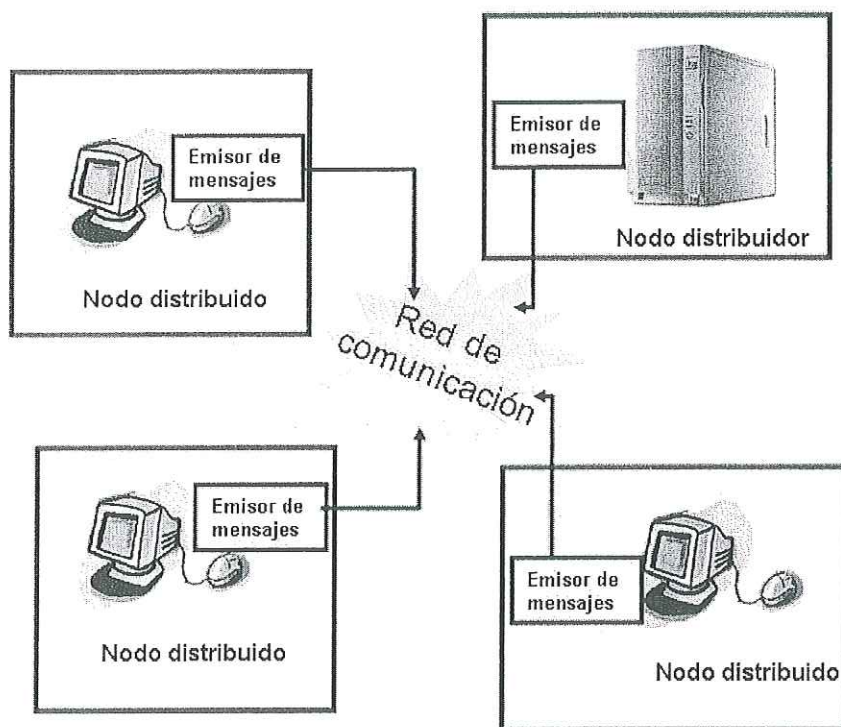


Figura 10. Comunicación entre nodos WODIS.

A continuación se describe más a detalle la funcionalidad de cada uno de los componentes ya mencionados.



### II.3.1.1 Nodo Distribuidor (DrN)

El *DrN* se encarga de entregar a los nodos distribuidos todos los elementos necesarios para que puedan funcionar. Esto incluye la aplicación que utilizarán los actores, soporte a recuperación de errores, actividades que le corresponde realizar a cada actor, así como los datos y documentos requeridos para poder ejecutar estas actividades.

La figura 11 muestra los componentes del *DrN*, en la cual se pueden apreciar dos componentes principales: el administrador de la distribución y recuperación a fallas.

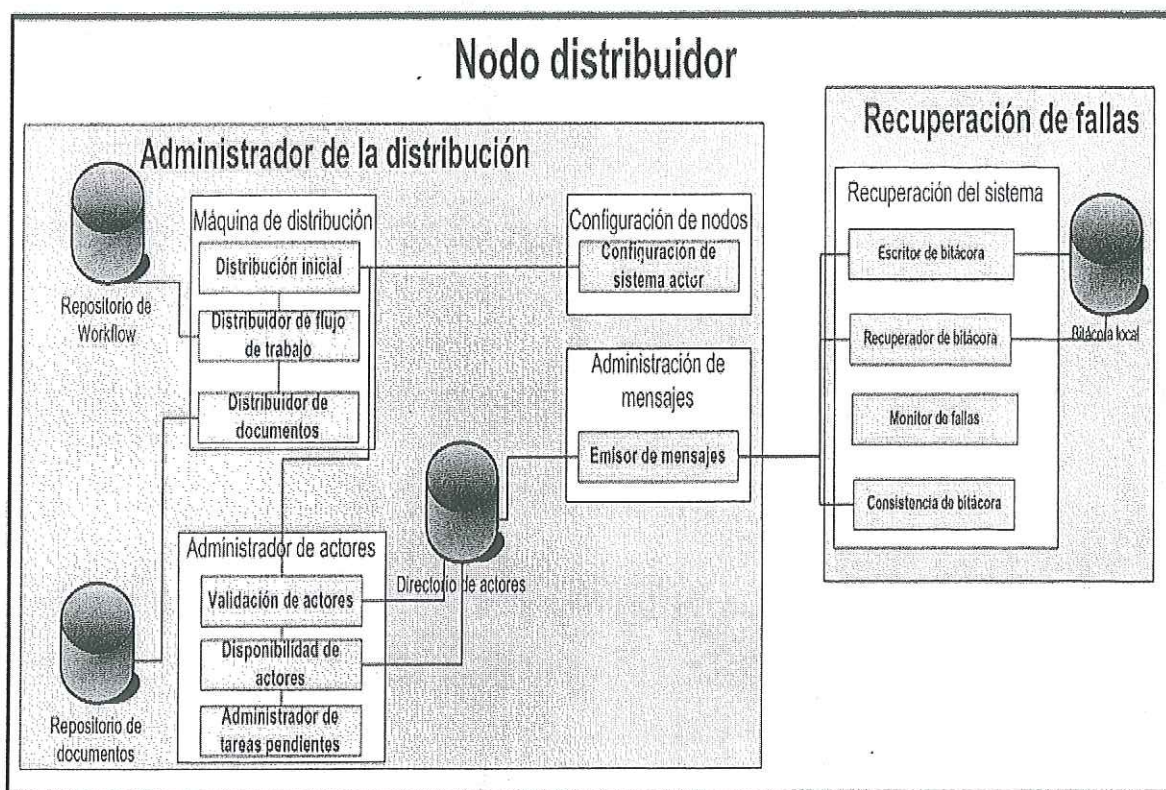


Figura 11. Componentes del Nodo Distribuidor. Administrador de la distribución

El *administrador de la distribución* se encarga de proporcionar a los actores la información necesaria para llevar a cabo las actividades que le corresponden. Este a su vez tiene los siguientes elementos:

- *Máquina de distribución.*- Este componente es el responsable de asignar a cada actor las actividades que debe realizar de acuerdo a los roles que le corresponden, así como los datos que requiere. Para lograr lo anterior, hace uso de los siguientes mecanismos:
  - *Repositorio global de workflow.*- Es un repositorio que almacena el flujo que siguen las actividades que conforman el proceso. Las actividades son agrupadas por roles.
  - *Repositorio global de documentos.*- Es un repositorio que almacena todos los documentos que se utilizan a lo largo de la ejecución del proceso.
  - *Distribución inicial.*- Este mecanismo se encarga de iniciar la distribución del sistema. Su función específica es iniciar a los mecanismos *Distribuidor de flujo de trabajo*, *Distribuidor de actividades* y *Distribuidor de documentos*.
  - *Distribuidor de flujo de trabajo.*- Este mecanismo se encarga de obtener el flujo que seguirán las actividades que debe llevar a cabo el actor. Para esto obtiene los roles que debe realizar cada actor y para cada uno de ellos obtiene del *Repositorio global de Workflow* el flujo que siguen las actividades que corresponden a cada rol.
  - *Distribuidor de documentos.*- Este mecanismo se encarga de obtener los documentos que requerirá el actor para llevar a cabo las actividades que le corresponden, así como de almacenarlos una vez que el actor ya no los necesite.

- *Administrador de actores.*- Se encarga de llevar el control del estado que guardan los actores en el sistema.
  - *Directorio de actores.*- Este repositorio almacena los datos de los actores relevantes al proceso, tales como nombre de cuenta, roles que desempeña, contraseña y otros.
  - *Validación de actores.*- Se encarga de comprobar que los usuarios estén autorizados a utilizar el sistema. Para esto, debe comparar los datos que envía el usuario con los datos almacenados en el *Directorio de actores*.
  - *Disponibilidad de actores.*- Este mecanismo se encarga de controlar la disponibilidad de los actores en el sistema, es decir, lleva un registro de cuales actores se encuentran disponibles, o “en línea”, en el sistema. Además, este mecanismo se encarga de obtener los roles que desempeña cada uno de los actores.
  - *Administrador de tareas pendientes.*- Tiene la responsabilidad de administrar las actividades que involucren interacción entre actores y que no han podido terminar de ser ejecutadas. Estas actividades son enviadas del nodo distribuido al nodo Distribuidor cuando el actor que trabaja en el nodo distribuido desea salir del sistema y no se ha podido establecer comunicación con el nodo con el cual debe comunicarse para concluir la tarea.
- *Administrador de mensajes.*- Tiene como función establecer la comunicación con el resto de los actores. Está constituido por el siguiente mecanismo:



- *Emisor de mensajes.*- Se responsabiliza de establecer la comunicación con los actores disponibles en el sistema.
- *Configuración de nodos.*- Se activa solamente cuando el nodo distribuido va ponerse en marcha por primera vez. Su responsabilidad es instalar la aplicación distribuida en los nodos (PC's) en los que el sistema estará funcionando.

### **Recuperación de fallas.**

El mecanismo de recuperación de fallas se encarga de mantener los nodos en un estado consistente. Este mecanismo administra el historial de actividades que ha llevado a cabo cada usuario. Para lo anterior hace uso de los siguientes mecanismos:

- *Recuperación del sistema.*- Se activa cuando se detecta que alguno de los nodos tiene problemas de inconsistencia. Esta inconsistencia puede ser causada porque algún nodo no fue cerrado correctamente, o bien por cuestiones de comunicación entre los nodos. Está constituido de los siguientes elementos:
  - *Escritor de bitácora.*- Almacena en bitácora todas las actividades que han realizado los actores. Cada vez que se comienza o finaliza la ejecución de una actividad, el nodo distribuido manda los datos de la transacción al nodo distribuidor para que sean almacenados en la bitácora.
  - *Recuperador de bitácora.*- Recupera datos específicos almacenados en la bitácora, tales como historial de actividades que ha ejecutado algún actor en un lapso de tiempo, estado en que se encuentra algún proceso, o el estado en que se encuentran los procesos de algún rol en particular entre otros. Este mecanismo es activado cada vez que se detecta que un nodo ha sido cerrado



incorrectamente o cuando el actor se da de alta en un nodo distinto al que se encontraba trabajando anteriormente.

- *Monitor de fallos.*- Este mecanismo se encarga de estar monitoreando si el sistema presenta algunas fallas, como que el nodo ha sido cerrado de forma no adecuada. Este mecanismo se activa cada vez que un actor se da de alta en el sistema.
- *Consistencia de bitácora.*- Es responsable de solicitar a los nodos distribuidos la bitácora donde han registrado las actividades que han sido realizadas o están en ejecución para compararlas y establecer en ambos nodos (distribuidor y distribuido) la bitácora que tenga registrado un mayor progreso en la ejecución de las actividades. Se activa cuando un actor se da cuando un nodo sufre una falla y el actor que se encontraba trabajando en él se da de alta ahí mismo.

### **III.3.1.2 Nodo Distribuido (DoN)**

El *DoN* está diseñado para dar soporte a la coordinación de las actividades que cada actor debe llevar a cabo. Dentro de este soporte está la coordinación del flujo de trabajo, administración de las actividades, gestión de datos, gestión de pantallas, gestión de aplicaciones, comunicación con otros actores y recuperación del sistema.

En la figura 12 se muestra el *DoN*, donde se pueden apreciar dos componentes principales: *rol base* y *recuperación de fallas*, los cuales son descritos a continuación.

## Rol base

El componente *Rol base* es responsable coordinar la ejecución de las actividades que el actor que trabaja en ese nodo lleva a cabo y de administrar la interacción con otros nodos.

Tiene los siguientes componentes:

- *Máquina de coordinación.*- El componente se encarga del *enactment* del proceso, es decir, se encarga de coordinar las actividades del proceso y gestionar los recursos necesarios para llevarlas a cabo.
  - *Repositorio local de workflow.*- Este componente almacena el flujo que de actividades que corresponde a cada uno de los roles del actor. La información que contiene este repositorio es obtenida del nodo distribuidor por medio del mecanismo *Distribuidor de flujo de actividades*.
  - *Repositorio de documentos.*- Este componente almacena todos los documentos que se utilizan a lo largo de la ejecución de las actividades que le corresponden al actor. La información que contiene este repositorio es obtenida del nodo Distribuidor por medio del mecanismo *Distribuidor de flujo de documentos*.
  - *Administrador de estados.*- Se encarga de coordinar el flujo que siguen las actividades en base a lo establecido en el *repositorio de workflow*. Su función básica es establecer el estado en que se deben de encontrar las instancias de los procesos durante el tiempo en que permanecen activas.

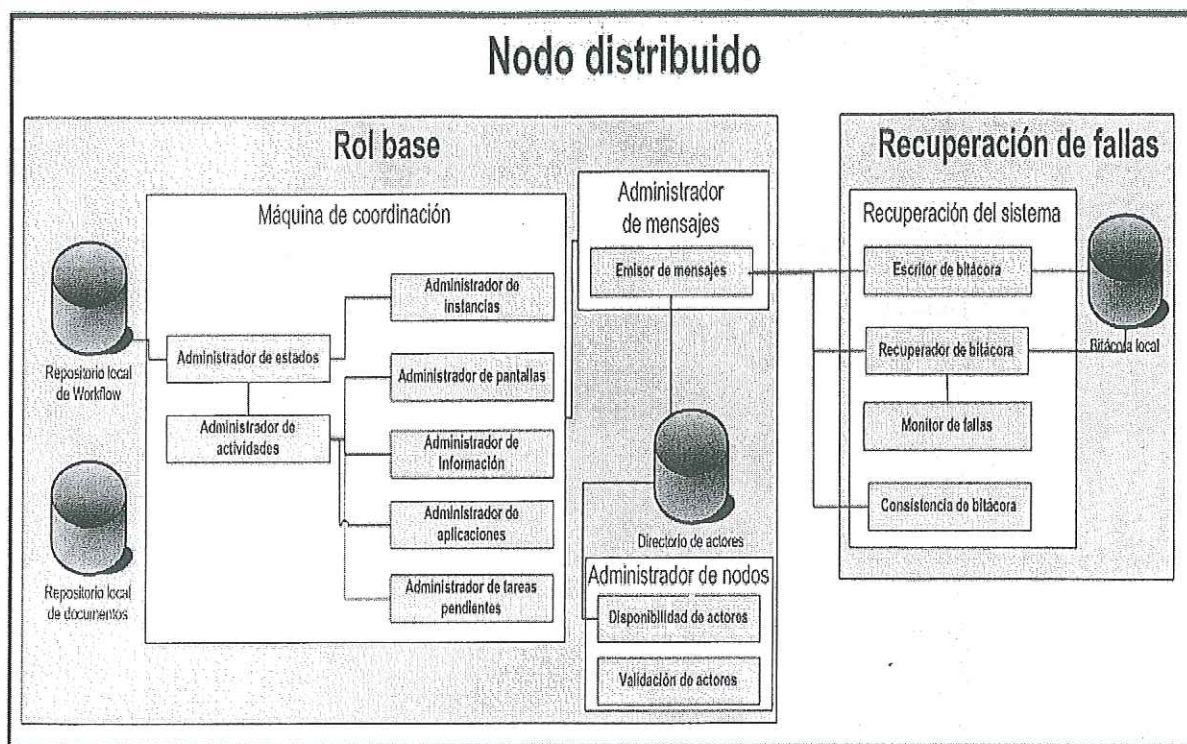


Figura 12. Componentes del Nodo Distribuido.

- *Administrador de actividades.*- Se encarga de administrar la forma en que se llevan a cabo las actividades. Es activado por el mecanismo *Administrador de estados* y su función consiste en determinar y obtener los elementos necesarios para la ejecución de la actividad indicada, y una vez obtenidos éstos, llevar a cabo la actividad.
- *Administrador de instancias.*- Tiene como función administrar el estado en que se encuentra cada una de las instancias del proceso que el actor ha ejecutado. Esta información es utilizada para mostrarle al actor las posibles actividades a realizar en ese momento, de tal forma que el actor pueda elegir la actividad que desee realizar.

- *Administrador de aplicaciones.*- Se encarga de gestionar el uso de aplicaciones externas en caso de que alguna actividad así lo requiera.
- *Administrador de pantallas.*- Muestra la pantalla adecuada para cada una de las actividades. La funcionalidad de este componente es mostrar la pantalla que el mecanismo *Administrador de actividades* requiere.
- *Administrador de información.*- Este mecanismo administra las entidades de información que se encuentran en el nodo. La responsabilidad de este mecanismo consiste en entregarle al mecanismo *Administrador de actividades* las entidades de información que éste le solicite.
- *Administrador de nodos.*- Este componente tiene una funcionalidad similar a la descrita en el mecanismo *administración de actores* del *DrN*, la cual es básicamente administrar la información referente a los actores del proceso, así como llevar un control sobre cuáles actores se encuentran disponibles en el sistema. Además de la funcionalidad ya descrita, también tiene también el siguiente mecanismo, adicional a los presentados por este componente en el nodo distribuidor:
  - *Salida.*- Este mecanismo se encarga de enviar al *DrN* las entidades de información almacenadas en el *DoN*, con la finalidad de que el actor pueda utilizar este sistema de forma consistente en cualquier otro nodo.
- *Administrador de mensajes.*- Este componente tiene la misma funcionalidad descrita en el mecanismo *Administración de mensajes* del nodo distribuidor, consistente en establecer la comunicación con los demás nodos del sistema.



## Recuperación de fallas

El mecanismo de recuperación de fallas se encarga de mantener el nodo en un estado consistente. Tiene una funcionalidad similar a la descrita en el componente *Recuperación de fallas* del *nodo distribuidor*, solo que en el *nodo distribuido*, el mecanismo de *Consistencia de bitácoras* compara la bitácora local únicamente con la bitácora almacenada en el *nodo distribuidor*, a diferencia de este mismo mecanismo en el nodo distribuidor, que realiza esta comparación con todos los nodos del sistema.

## III.4 Análisis de la arquitectura

En esta sección se presenta el diseño y funcionalidad de los componentes de la arquitectura propuesta por medio de diagramas UML (diagramas de casos de uso y diagramas de secuencia).

### III.4.1 Casos de uso.

Un caso de uso es en esencia, una interacción típica entre un usuario y un sistema de cómputo. Los casos de uso son llevados a cabo por *actores*, término que es empleado para llamar así al usuario (que puede ser una persona o un sistema externo que necesita interactuar con el sistema actual) cuando desempeña ese papel con respecto al sistema. Un mismo actor puede llevar a cabo varios casos de uso, lo mismo que a la inversa, es decir, un caso de uso puede ser llevado a cabo por varios actores [Fowler y Scott, 1997].

Las interacciones entre un actor y un caso de uso son ilustradas mediante una línea que los une (figura 13). Además de los vínculos entre los actores y los casos de uso, existen otros

tipos de vínculos. Éstos representan las relaciones de *include* (incluye) y *extends* (extiende) entre los casos de uso (figura 13).

Se usa la relación *extends* cuando se tiene un caso de uso que es similar a otro, pero que hace un poco más. Las relaciones *include* ocurren cuando se tiene una porción de comportamiento que es similar en más de un caso de uso.

En la figura 13 se presenta el diagrama de casos de uso global de la arquitectura WODIS y posteriormente se presenta la descripción de los más significativos de ellos.

El **caso de uso** *Configurar nodos distribuidos* es llevado a cabo por el administrador del sistema. Tiene como propósito establecer la aplicación distribuida en los nodos en los que trabajarán los actores.

**Descripción:**

El caso de uso inicia cuando el administrador del sistema desea instalar la aplicación distribuida en los nodos que utilizarán los actores. El administrador del sistema introducirá las direcciones IP y el sistema se encargará de enviar la aplicación a los nodos indicados.

El **caso de uso** *Validar actor* corresponde al usuario y tiene como propósito verificar que los usuarios del sistema estén autorizados a utilizarlo.

### Descripción:

Este caso de uso inicia cuando el usuario se da de alta en el sistema. El usuario introduce su nombre de cuenta y su contraseña y el mecanismo *Validación de actores* en el nodo *Distribuidor* se encarga de revisar si el actor está autorizado a utilizar el sistema.

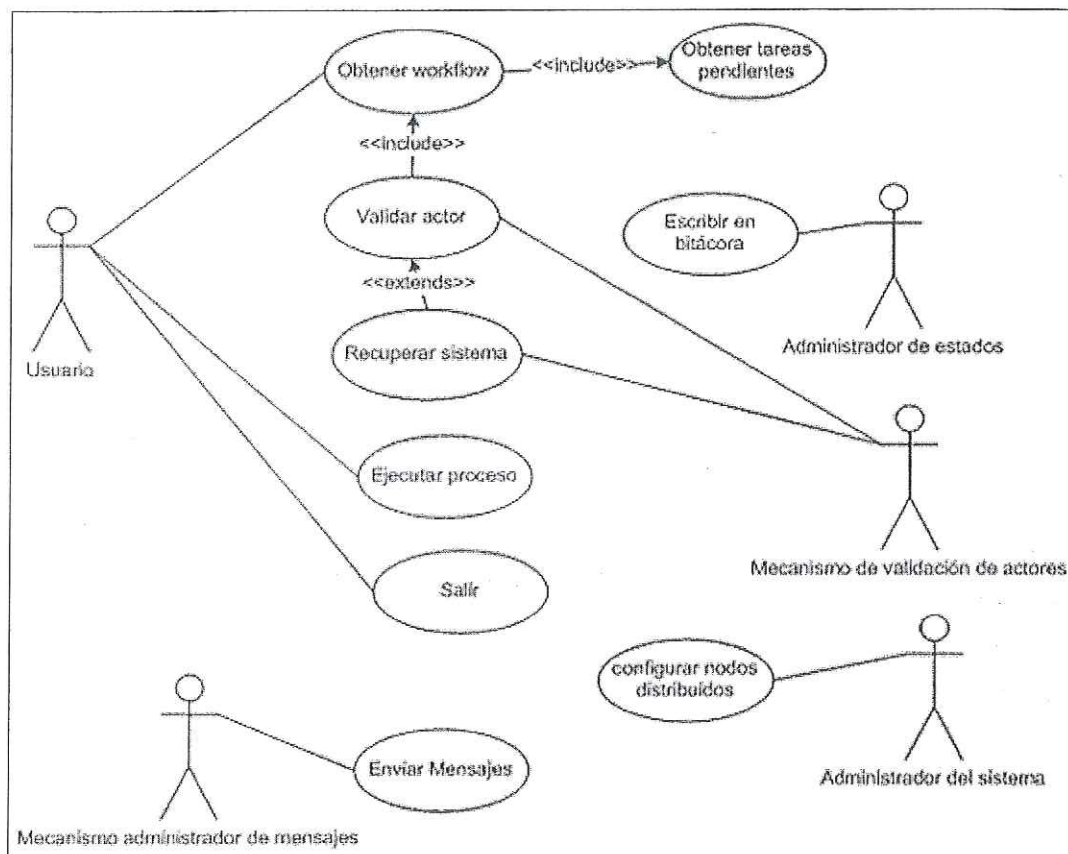


Figura 13. Diagrama de casos de uso de la arquitectura WODIS.

En caso de que así sea, se revisa si el nodo en que trabajará el actor es el mismo en el cual lo hizo la vez anterior. Si el actor trabajó en ese nodo, se envía al nodo distribuido el status que guardan los actores en el proceso y se comienza la ejecución del proceso. En caso de que el nodo no sea el último en el que trabajó el actor, se ejecuta el caso de uso *Obtener*

*workflow* y el caso de uso finaliza. En caso de que el actor no esté autorizado a utilizar el sistema, el caso de uso finaliza.

Por otra parte, el **caso de uso** *Obtener workflow*, es llevado a cabo por el mecanismo de *Validación de actores*, con la finalidad de obtener los elementos que el actor necesita para la ejecución del proceso.

#### **Descripción:**

Este caso de uso inicia una vez que el actor ya ha sido identificado como usuario autorizado. Se obtienen los roles que desempeña el usuario y se obtiene el flujo que siguen las actividades que corresponden a estos roles, el estado que guardan los actores en el sistema y los documentos que se necesitan para la ejecución de las actividades. Además, se revisa en el administrador de tareas pendientes si hay actividades por realizar para alguno o algunos de los roles del actor, y en caso de haberlas, éstas se agregan a la información obtenida. Finalmente se obtiene el estado global de los procesos que lleva a cabo el actor y se envía la información obtenida al nodo distribuido del actor.

El **caso de uso** *Escritura en bitácora* es llevado a cabo por el mecanismo *Administrador de estados* con el propósito de guardar en bitácora los datos que corresponden a los estados que ya han sido ejecutados o están en ejecución.

#### **Descripción:**

Este caso de uso inicia cuando una actividad comienza o termina su ejecución. Si la actividad comienza a ejecutarse, el administrador de estados envía los datos al mecanismo *escritor de bitácora* tanto en el *nodo distribuido* como en el *nodo distribuidor* y les indica



que la actividad se encuentra “en proceso”. Cuando la actividad finaliza, se envían los datos a ambos escritores de bitácora, pero se indica que la actividad se encuentra “finalizada”.

El **caso de uso** *Ejecución del proceso* es llevado a cabo por el *usuario* con el fin de llevar a cabo la ejecución del proceso.

### **Descripción:**

El caso de uso inicia cuando el actor desea ejecutar alguna de las actividades que le corresponden. El *administrador de estados (AE)* verifica en el *administrador de instancias (AI)* los estados actuales de las instancias y le muestra al actor las actividades de las instancias activas que puede realizar, o le pregunta si desea iniciar una nueva instancia. El actor selecciona la actividad que desea realizar y se ejecuta el caso de uso *Escribir en bitácora* estableciéndose la actividad como “en ejecución”. En caso de ser necesario, el *AE* le indica al *AI* que se ha iniciado una nueva instancia. Una vez que el actor ha seleccionado la actividad, se invoca a la actividad por medio del *Administrador de actividades*, quien a su vez se encarga de obtener la pantalla que corresponde a la actividad por medio del *Administrador de pantallas*, los documentos que necesitará del *Administrador de documentos* y en caso de necesitar una aplicación en específico se lo notificará al *Administrador de aplicaciones*. Cuando la actividad ha concluido, se ejecuta nuevamente el caso de uso *Escribir en bitácora* estableciéndose la actividad como “ejecutada” y se vuelve a iniciar este caso de uso. El caso de uso termina cuando el actor decide salir del sistema.

El **caso de uso** *Salir* es iniciado también por el usuario y tiene como propósito enviar al nodo distribuidor las entidades de información almacenadas en el nodo distribuido, con la finalidad de que el actor pueda utilizar el sistema de forma consistente en cualquier nodo.

#### **Descripción:**

Este caso de uso comienza cuando el actor decide salir del sistema. El mecanismo de *Salida* es notificado, y se encarga de enviar al *nodo distribuidor (DrN)* los documentos y datos que se encuentren almacenados en el *nodo distribuido (DoN)*. Además, el mecanismo de *Salida* debe indicarle al mecanismo *Disponibilidad de actores* que se encargue de avisar a los nodos disponibles en el sistema que el actor dejará de estar disponible. Así mismo, también obtiene del mecanismo *Administrador de tareas pendientes* los documentos, datos y notificaciones que no han podido ser entregadas a otros actores y los envía al mecanismo *Administrador de tareas pendientes* del nodo distribuidor. Finalmente, el mecanismo *Salida* indica de forma permanente en el *DoN* que ha sido cerrado de forma correcta.

El **caso de uso** *Obtener tareas pendientes* es ejecutado por el mecanismo *Distribución inicial* con el fin de obtener las tareas que no han podido ser ejecutadas para asignárselas a algún actor.

#### **Descripción:**

Este caso de uso inicia cuando un actor se da de alta en el sistema. El mecanismo *distribución inicial* revisa en el mecanismo *Administración de tareas pendientes (ATPC)* si hay tareas pendientes para uno o más de los roles que desempeña el actor. En caso de haberlas, el *ATPC* obtiene las entidades de información necesarias para que se puedan

llevar a cabo las actividades, y una vez obtenidos estos datos, el *DI* los envía al nodo del actor por medio del mecanismo *Emisor de mensajes*.

Finalmente el **caso de uso** *Enviar mensajes*, el cual es llevado a cabo por el mecanismo *Emisor de mensajes*, tiene como finalidad establecer la comunicación con otros actores del proceso.

#### **Descripción:**

Este caso de uso inicia cuando el alguno de los mecanismos requiere interactuar con otro nodo. El mecanismo *Emisor de mensajes (EM)* obtiene la ubicación del nodo y también el método que va a invocar. Una vez que obtiene estos datos, se establece la comunicación y el caso de uso finaliza.

### **II.4.2 Diagramas de secuencia**

Los diagramas de secuencia son parte de los diagramas de interacción. Los diagramas de interacción son modelos que describen la manera en que colaboran grupos de objetos para cierto comportamiento. [Fowler y Scott, 1997]. Habitualmente, un diagrama de interacción capta el comportamiento de un solo caso de uso.

En un diagrama de secuencia, un objeto se muestra como caja en la parte superior de una línea vertical punteada. Esta línea vertical se llama línea de vida del objeto y representa la vida del objeto durante la interacción. Cada mensaje se representa mediante una flecha entre las línea de vida de dos objetos.

A continuación se presentan algunos de los diagramas de secuencia de los casos de uso más relevante, descritos anteriormente. El resto de ellos es presentado en el Apéndice A.

La figura 14 muestra el **diagrama de secuencia** del caso de uso *Validar actor*, que inicia una vez que el usuario vuelve a darse de alta en el sistema. El usuario introduce sus datos de identificación (nombre de usuario y contraseña) al mecanismo de *validación de actores* del nodo distribuido (VAD) por medio del método *leerActor(login, contraseña)*.

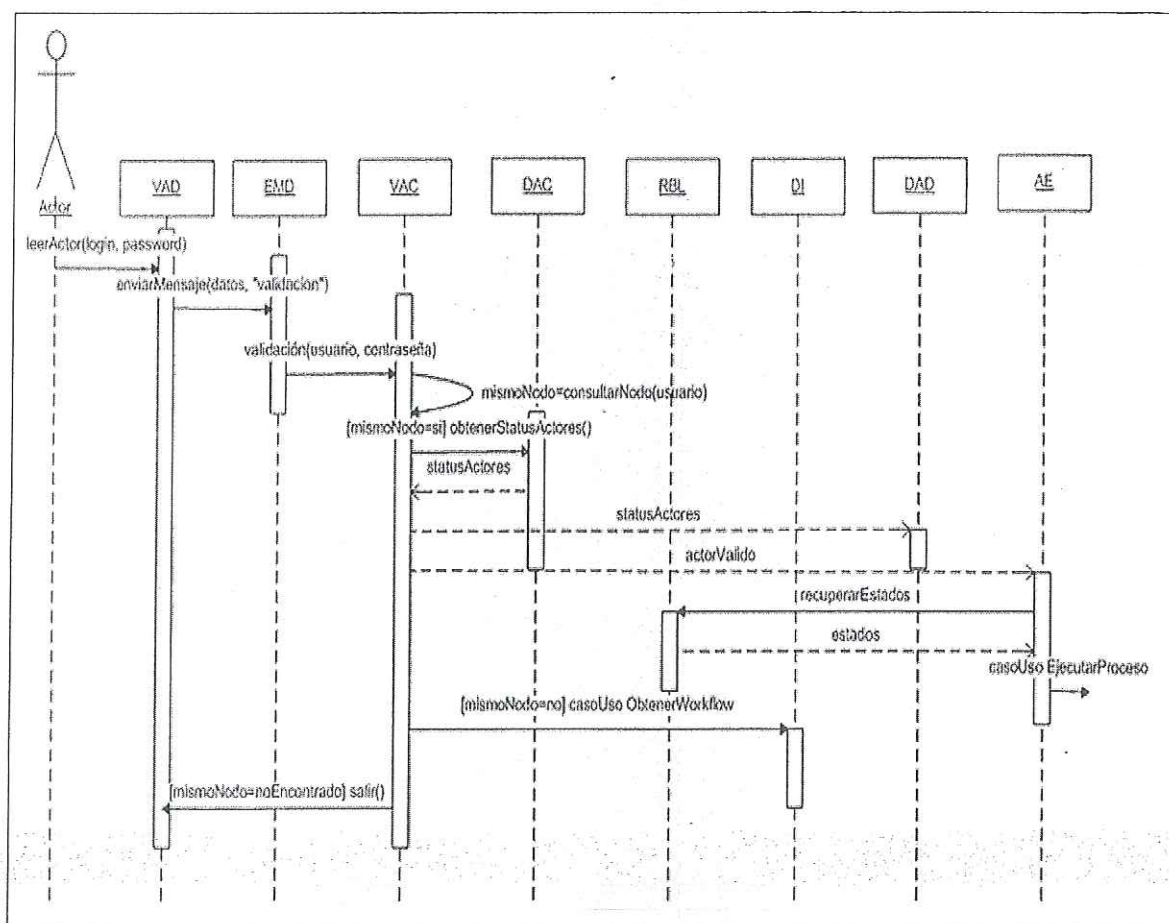


Figura 14. Diagrama de secuencia del caso de uso *Validar actor*.



Este método tiene como finalidad enviar al *nodo distribuidor (DrN)* los datos de validación del actor. En el *DrN* se revisa que el actor esté autorizado a utilizar el sistema con el método *validación(usuario, contraseña)* y si el esto sucede, se revisa, por medio del método *consultarNodo(usuario)*, si el nodo en el que está trabajando el actor es el mismo en el que trabajó la vez anterior que utilizó el sistema. En caso de que así sea, el método *obtenerStatusActores()* obtiene el estado que guardan los actores en el sistema y lo envía al *nodo distribuido (DoN)* y posteriormente en éste se recuperan los estados de las instancias y se comienza la ejecución del proceso.

En caso de que el *DoN* en que se encuentre el actor no sea el mismo en cual trabajó por última vez, se ejecuta el caso de uso *Obtener workflow*. Si el actor no está autorizado a utilizar el sistema, el caso de uso finaliza.

La figura 15 muestra el **diagrama de secuencia** del caso de uso *Obtener workflow*, el cual inicia una vez que el actor ya ha sido identificado como usuario autorizado (*caso de uso Validar actor*). El *mecanismo de distribución inicial (DI)* obtiene los roles que desempeña el actor mediante la invocación al método *obtenerRoles()* en el *mecanismo de validación de actores del nodo distribuidor (VAC)*, los cuales quedaron almacenados ahí al momento de ejecutarse el caso de uso Validar actor. Una vez obtenidos los roles, el *DI* obtiene el flujo que siguen las actividades que le corresponden a cada rol del actor mediante la activación del método *obtenerEstados(roles[ ])* en el *mecanismo de distribución de estados (DE)*. Este método accesa al *repositorio global de workflow* para consultar el flujo de trabajo global del proceso y de ahí obtener solo las actividades que le corresponden a cada rol.

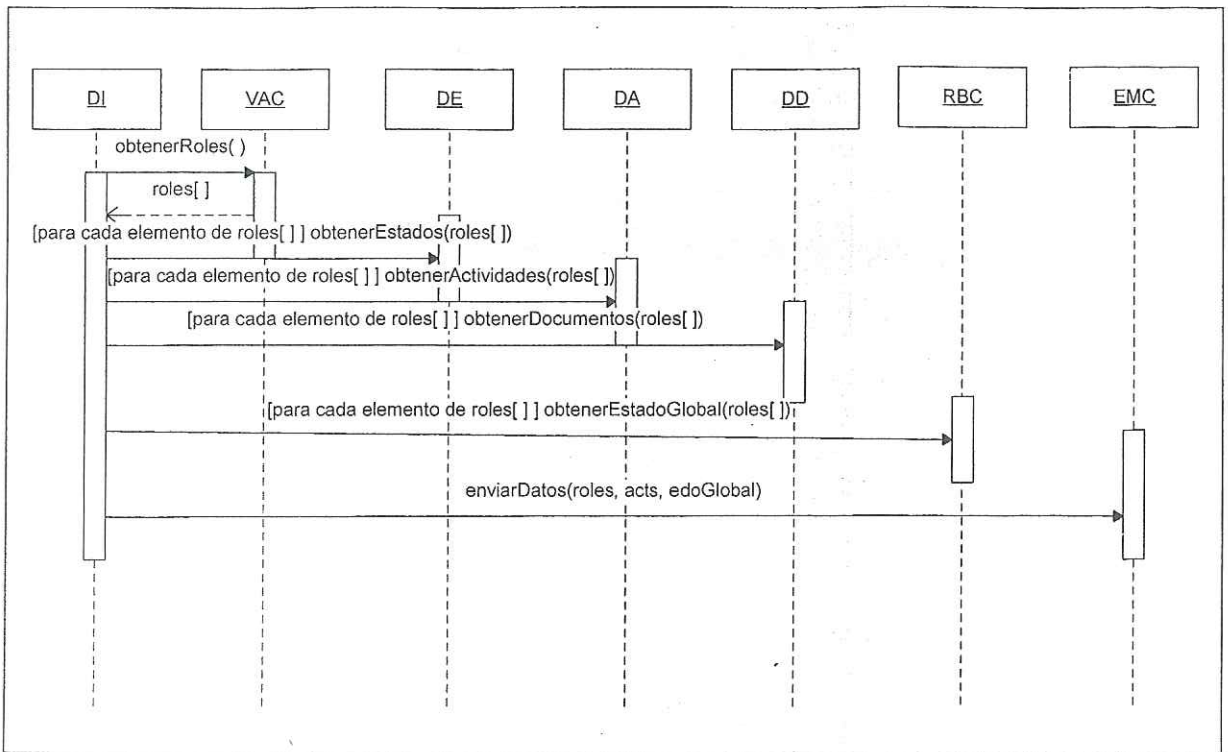


Figura 15. Diagrama de secuencia del caso de uso *Obtener workflow*.

Después de haber obtenido el flujo de las actividades, el *DI* obtiene la definición de las actividades (código) a través del método *obtenerActividades(roles[])* del *mecanismo de distribución de actividades (DA)*, el cual se encarga de serializar los objetos que corresponden a cada rol. Asimismo, ya habiendo obtenido la definición de las actividades, el *DI* obtiene los documentos que el actor necesitará para llevar a cabo las actividades. Para esto, invoca al método *obtenerDocumentos(roles[])* del *mecanismo de distribución de documentos (DD)*. Este método accesa al *repositorio global de documentos* y obtiene todos los documentos que le corresponden a los roles del actor. Una vez obtenidos los documentos, el *DI* obtiene el estado global que guarda cada rol mediante el método *obtenerEstadosGlobales(roles[])*, el cual accede a la *bitácora global* por medio del

*mecanismo recuperador de bitácora (RBC)* y obtiene el estado en que se encuentran las instancias de cada uno de los roles del actor.

Finalmente, el *DI* envía los datos obtenidos a través del *mecanismo emisor de mensajes del nodo distribuidor (EMC)*.

Por otra parte, la figura 16 muestra el **diagrama de secuencia** del caso de uso *Ejecutar proceso*, que inicia cuando el *usuario* desea ejecutar alguna de las actividades que le corresponden. El *mecanismo administrador de estados (AE)* obtiene del *mecanismo administrador de instancias (AI)* los estados correspondientes a las actividades que el actor puede realizar en ese momento a través del método *obtenerActsActuales()*. Una vez obtenidos los estados, el *AE* obtiene el nombre de las actividades que corresponden a los estados y se las muestra al *usuario*. El *usuario* selecciona la actividad y se inicia el método *iniciaActividad(act)* en el *AE*. El método *iniciaActividad(act)* se encarga de iniciar el caso de uso *Escribir en bitácora* para registrar que la actividad ha comenzado en encuentra “en proceso” de ser ejecutada. Después, el *AE* invoca al método *ejecutarActividad(act)* y es este método el encargado de la ejecución de la actividad. Para esto, obtiene del *mecanismo de administración de información (AD)* los documentos que necesita para poder llevar a cabo la actividad mediante la invocación al método *obtenerDocumentos(act)*, le indica al *mecanismo administrador de pantallas (AG)* cual interfaz mostrar mediante el método *despliegaInterfaz(interfaz)*, y invoca a alguna aplicación externa al sistema por medio del método *invocaActividad(aplicación)* del *mecanismo Administrador de Aplicaciones (AP)*.

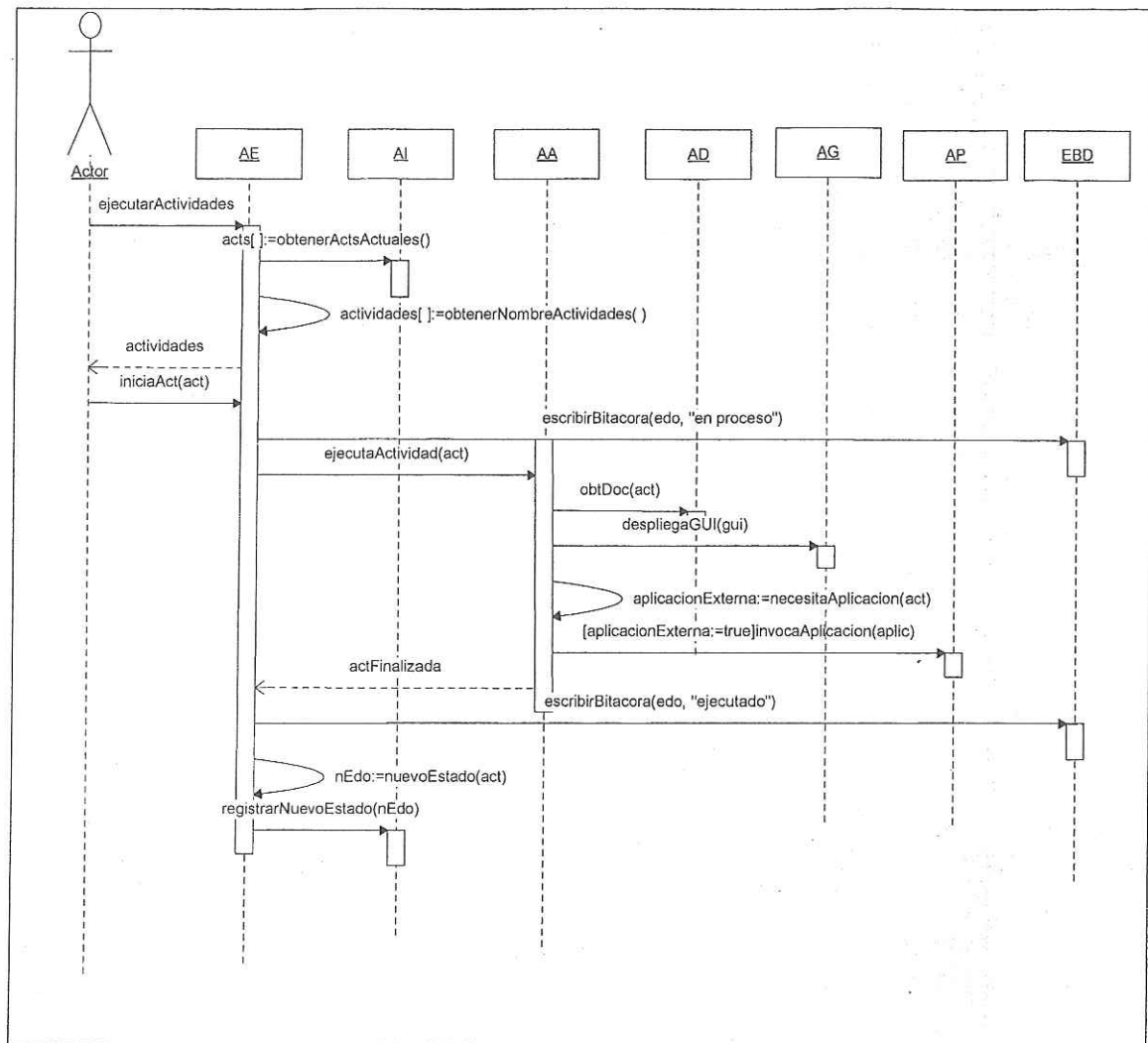


Figura 16. Diagrama de secuencia del caso de uso *Ejecutar proceso*.

Una vez que ya ha obtenido los elementos anteriores (datos, interfaz y aplicación) se ejecuta el *enactment* y al finalizar se le informa al *AE*. El *AE* al recibir la notificación de que la actividad ya ha sido ejecutada, nuevamente inicial el caso de uso *Escribir Bitácora*, para registrar que la actividad ya ha sido “ejecutada”. Finalmente, el *AE* determina el nuevo estado (método *nuevoEstado(act)*) en que se encuentra la instancia a la que pertenece la actividad que se ha ejecutado e indica al *AI* estos cambios por medio del método *registrarNuevoEdo(nEdo)*.



### ***III.5 Enfoque transaccional en la ejecución y recuperación de fallas del sistema.***

Los sistemas de workflow son considerados actualmente como una herramienta eficaz en el mejoramiento de la eficiencia terminal, reducción de costos, mejoramiento de la comunicación y reducción de errores humanos dentro de un proceso organizacional. Sin embargo, uno de sus puntos débiles es que no ofrecen un significativo soporte al manejo y recuperación de errores [Worah y Sheth 1997].

Un enfoque que se está utilizando para atacar este problema es el uso de transacciones. Una transacción es definida como un conjunto de operaciones que transfieren una base de datos o un proceso de un estado confiable a otro [Worah y Sheth, 1997]. Existen varios modelos de procesamiento de transacciones, siendo el modelo ACID (llamado así por las siglas de sus propiedades en inglés, las cuales son atomicidad, coherencia, aislamiento y durabilidad) el pionero y más sencillo de ellos. Sin embargo, a pesar de la gran funcionalidad que proporciona este modelo de transacciones en aplicaciones convencionales, tales como sistemas de información, sistemas de nóminas, etc., presenta varias limitantes al ser implementado en sistemas de workflow.

Uno de los enfoques más modernos en el uso de transacciones en aplicaciones no convencionales son los modelos de transacciones avanzadas, los cuales presentan algunas características que hacen más viable su implementación en aplicaciones no convencionales.

En este capítulo se presentan el análisis que se hizo de los modelos de transacciones más relevantes a este trabajo, así como los escenarios de tolerancia a fallas previstos en éste.

### III.5.1 Modelo de transacciones ACID.

El modelo ACID es el más sencillo de los modelos de transacciones existentes y es muy utilizado en bases de datos. Su finalidad es mantener la consistencia en un ambiente en donde los datos son accedidos por múltiples aplicaciones y múltiples usuarios, y basa su funcionamiento en conservar las siguientes propiedades de una base de datos o proceso:

- *Atomicidad.*- Una transición de estado es atómica si pasa de un estado inicial a un estado objetivo sin que se observe algún estado intermedio, o si éste sucede, la transición debe aparecer como que nunca se ejecutó (principio de todo o nada).
- *Consistencia.*- Una transacción produce solo estados consistentes, o dicho de otro modo, el estado resultante debe cumplir con todas las restricciones que determina la aplicación.
- *Aislamiento.*- Significa que la ejecución de un programa en un ambiente multiusuario bajo el esquema de procesamiento de transacciones debe ser similar a la ejecución de ese mismo programa en un esquema monousuario.
- *Durabilidad.*- Se refiere a que el resultado de una transacción debe ser durable y no debe ser olvidado por el sistema.

No obstante la gran funcionalidad que presenta este modelo de transacciones, presenta varias limitantes para ser implementadas en un sistema de workflow, entre otras, por las siguientes razones:

- Las actividades de un proceso organizacional son por lo general de larga duración, por lo que ejecutar estas actividades bajo el control de transacciones ACID puede retardar la ejecución de algunas otras actividades (debido a los mecanismos que se utilizan para el control de concurrencia tales como bloqueos), por lo que tratar un proceso como una transacción derivaría seguramente en grandes periodos de inactividad de muchas actividades y en grandes cantidades de actividades abortadas.
- Por su naturaleza cooperativa, la ejecución de las actividades de un proceso organizacional por lo regular requiere la interacción con varias entidades tales como otras actividades, varias bases de datos independientes entre sí u otras aplicaciones. Para asegurar la consistencia del proceso se necesita una completa coordinación entre estas entidades.

Una solución a las limitaciones que presenta el modelo ACID, son los modelos de transacciones avanzadas, los cuales se analizan en la siguiente sección.

### **III.5.2 Modelo de transacciones avanzadas.**

Actualmente los modelos de transacciones avanzadas son muy utilizados en el desarrollo de aplicaciones no convencionales con enfoque al manejo de transacciones. Existen varios modelos, los cuales son catalogados en base a varias características que incluyen estructura

de las transacciones, visibilidad, requerimientos de aislamiento, durabilidad y atomicidad.

Entre los más relevantes se encuentran los siguientes:

- *Modelo de transacciones anidadas (Nested Model).*- El modelo de transacciones avanzadas tiene un esquema de árbol. En este modelo las transacciones padres solo pueden hacer *commit* (confirmación de que la transacción ha finalizado de manera exitosa) hasta que las transacciones hijas hacen *commit*.

Si una transacción hija falla, pero existe alguna otra opción por ejecutarse, la transacción padre decide si se continúa con la ejecución o se aborta también la transacción padre.

Finalmente, se tiene que los resultados de las transacciones hijas que ya hicieron *commit* no pueden ser mostrados a alguna otra transacción que requiera conocer utilizar estos resultados, hasta que la transacción padre haga *commit*.

- *Modelo de transacciones anidadas abiertas (Open Nested Model).*- Este modelo es conceptualmente muy similar al modelo de transacciones anidadas. La diferencia radica en que en el modelo de transacciones anidadas abiertas sí se permite que las transacciones hijas puedan mostrar los resultados que han producido al momento de que hicieron *commit* a alguna otra transacción que interactúe con ellas, aún sin que la transacción padre haya hecho *commit*, a diferencia del modelo de transacciones



anidadas, en el cual estos resultados solo pueden ser mostrados hasta que la transacción padre hace *commit*.

- *Modelo de transacciones Saga (Saga Model).*- A diferencia de los dos modelos anteriores, el modelo Saga presenta un conjunto de transacciones definidas a través de un solo nivel. Cada una de estas transacciones tiene definida una actividad de compensación, la cual tiene la finalidad de deshacer los cambios que las transacciones que ya hicieron *commit*. Cuando una actividad *n* aborta, se ejecutan las actividades compensatorias definidas para las transacciones anteriores (desde *n-1* hasta 1) que ya fueron ejecutadas exitosamente.

*Modelo de transacciones multinivel (MultiLevel Model).*- Este modelo es una mezcla de los modelos de transacciones anidadas abiertas y del modelo de transacciones Saga. Es similar al modelo de transacciones anidadas abiertas en el sentido de que hay múltiples niveles de transacciones y las transacciones padre solo pueden hacer *commit* hasta que las transacciones hijas hacen *commit*. Además, el resultado de las transacciones hijas que ya hicieron *commit* se pueden mostrar a otras transacciones que interactúan con ellas. Sin embargo, a diferencia ese modelo, cuando una transacción hija falla no es la transacción padre quien decide si se ejecuta alguna otra opción, sino que se ejecuta un conjunto de acciones compensatorias previamente definidas (al igual que se hace en el modelo de transacciones Saga).

- *Modelo de transacciones flexibles (Flexible Transactions Model).*- El modelo de transacciones flexibles puede ser definido de manera informal como el modelo de

transacciones multinivel complementado con algunas otras características de flexibilidad.

La característica de este modelo es que tiene un esquema de árbol, donde las transacciones padre solo pueden hacer *commit* hasta que todas las transacciones hijas hacen *commit*, y el resultado de las transacciones hijas que hicieron *commit* si puede ser utilizado por algunas otras transacciones que interactúan con estas transacciones hijas.

La diferencia de éste modelo con respecto al modelo de transacciones multinivel, es que mientras el segundo define actividades compensatorias que se ejecutarán en caso de que alguna de las transacciones hija falle, el modelo de transacciones flexible permite que las transacciones sean etiquetadas de acuerdo al comportamiento que puede presentar cuando una falla suceda. Estos comportamientos son los siguientes:

- *Actividades compensatorias (Compensatables).*- Por cada transacción se define una transacción que se ejecutará en caso de que alguna transacción falle, con la finalidad de revertir los cambios que se hayan hecho.
- *Transacciones reintentables (retriabables).*- Cuando una transacción etiquetada como *reintentable* sufre una falla, ésta se intenta nuevamente hasta que puede ser completada.
- *Transacciones terminales.*- Este tipo de transacciones se ejecutan solo una vez. En caso de que alguna de estas transacciones no pueda ser completada,

se cancelan todas las transacciones del nivel en el que se encuentra, sin que se ejecute ninguna actividad de compensación. Además, la transacción padre también se cancela.

Una vez que se hizo el análisis de las características que presentan cada uno de los modelos de transacciones avanzadas más relevantes existentes, se eligió el modelo de transacciones flexible (Flexible Transactions Model) para el manejo de errores en la arquitectura WODIS por los siguientes motivos:

- Todos los modelos propuestos, salvo el modelo Saga tienen un esquema de árbol, lo que permite que haya varios niveles de transacciones. En el caso de un proceso real, tener varios niveles es útil cuando se tienen actividades paralelas, puesto que deben terminar todas las actividades para que la siguiente actividad pueda ser ejecutada. Por esta razón, el modelo Saga queda descartado.
- De los modelos restantes (transacciones anidadas, transacciones anidadas abiertas, transacciones multinivel y transacciones flexibles), sólo el modelo de transacciones anidadas no permite que los resultados de una transacción que ya finalizó puedan ser utilizados por transacciones que interactúan con esa transacciones. En un proceso real, puede ser que una de estas transacciones produzca un resultado que otro actor requiere para continuar con las actividades que le corresponden, y quizás no sea pertinente ni necesario que tenga que esperar a que todas las transacciones del mismo nivel sean completadas con éxito. Por esta razón se descarta el modelo de transacciones anidadas.

- El modelo de transacciones anidadas abiertas y el modelo de transacciones multinivel permiten que las transacciones solo sean tratadas de una misma manera cuando alguna de ellas sufre alguna no puede ser completada correctamente. Sin embargo, en un proceso real, no siempre que ocurre un error se actúa de la misma manera, lo cual descarta a ambos modelos.

En resumen, se elige el modelo de transacciones flexibles, debido a que tiene las características para poderlas adaptar a un sistema de workflow distribuido, a diferencia de los otros modelos, los cuales presentan solo algunas de éstas.

### **III.5.3 Ejemplificación del funcionamiento de un sistema de workflow basado en transacciones**

Como se menciona en la sección anterior, para llevar a cabo la recuperación de un sistema de workflow en caso de que una falla suceda durante su ejecución se propone la utilización de transacciones, basándose en el modelo de transacciones flexible.

Para poder realizar el desarrollo de un sistema de workflow, es necesario primero llevar a cabo un modelado del proceso al que se le va a dar soporte. El modelado de procesos en este trabajo es realizado por medio de diagrama rol-actividad (RAD). Un diagrama RAD muestra los roles de un proceso, las actividades que componen cada rol y sus interacciones, junto con eventos externos y la lógica que determina cuáles actividades son ejecutadas y cuando [Ould, 1995].



La figura 17 muestra los elementos de los diagramas RAD. Los roles son representados por un rectángulo sombreado. Las actividades de cada rol son indicadas por medio de cuadros blancos, los cuales se sitúan dentro del rol al que pertenecen. La transición entre una actividad y otra, sea en el mismo rol o en roles diferentes es representada por una línea que va de actividad a otra.

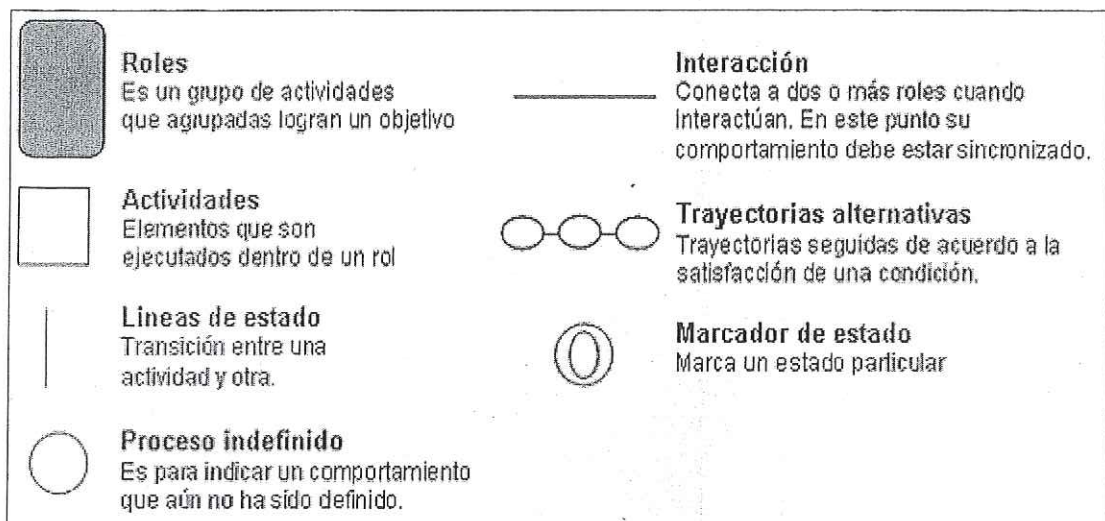


Figura 17. Elementos de un *Diagrama Rol Actividad*.

En el caso de que las actividades estén en roles diferentes, se está denotando que existe una interacción entre roles y la línea que representa esta interacción es remarcada con respecto a la línea que denota transición entre actividades del mismo rol.

Algunos procesos tienen flujos de actividades que serán ejecutados en base al cumplimiento de una actividad, los cuales son modelados en un diagrama RAD por medio de círculos unidos por líneas horizontales, en donde cada círculo representa una opción. También se pueden modelar flujos de actividades paralelas, donde cada trayectoria es representada por

un triángulo. Cuando hay flujos paralelos no importa cual de ellos se ejecute primero, pero si deben ser cumplidos todos para que se pueda realizar la siguiente actividad.

En este trabajo se sugiere que en los sistemas basados en la arquitectura el documento que define el flujo de trabajo sea desarrollado con tecnología XML, que es un metalenguaje que permite manejar la información de manera contextual. Lo utilizamos porque nos permite definir la estructura de la información que estará contenida en estos documentos y nos ayuda a tener una idea más clara de la información que estamos modelando.

La estructura de estos documentos está basada en los diagramas de rol actividad (RAD). Para ilustrar la forma en que funciona el mecanismo de manejo de errores, utilizamos el siguiente rol de ejemplo (figura 18) modelado por medio de un diagrama RAD.

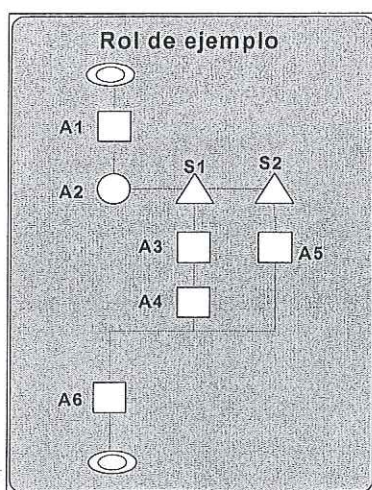


Figura 18. Diagrama Rol-Actividad del Rol de ejemplo.

El documento XML que define al proceso que lleva a cabo el rol de ejemplo se muestra en la figura 18:

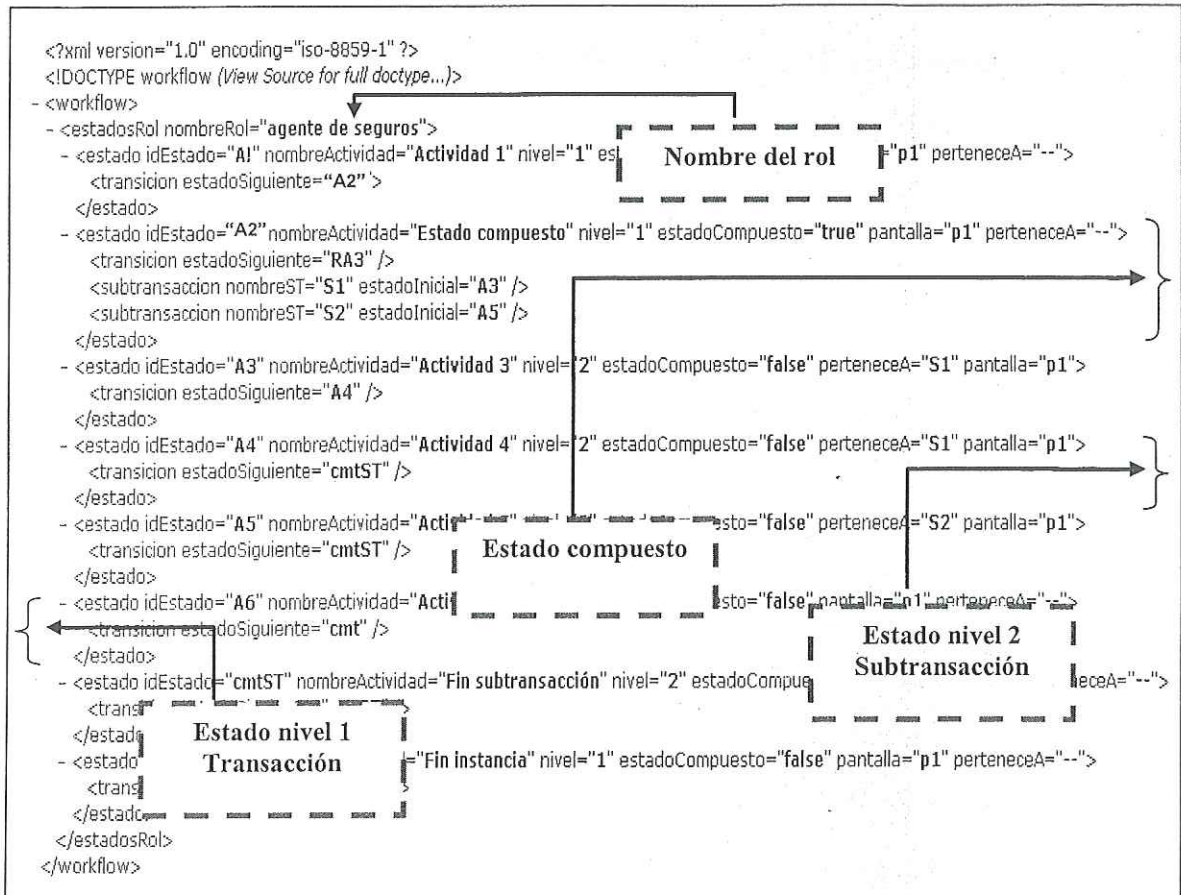


Figura 19 Documento de flujo de actividades para el rol de ejemplo.

Para implementar este modelo transaccional, se requiere una bitácora (es un documento en el cuál se registran las transacciones que se han realizado en el proceso). Se propone que ésta bitácora sea implementada por medio de documentos XML (en el capítulo IV se muestra algunos detalles de la implementación de la bitácora).

La bitácora contiene los siguientes campos:



- *Instancia.*- En este campo se registra el identificador de la instancia del proceso.
- *Estado.*- En este campo se registra el estado en que se encuentra la instancia del proceso.
- *Status.*- En este campo se indica sí la actividad que corresponde al estado se encuentra en ejecución (*procesando*) o ya ha sido ejecutada (*finalizado*).
- *Nivel.*- En este campo se indica sí la transacción es una transacción padre (*nivel 1*) o es una transacción hija <sup>"A2"</sup> (*nivel 2*).
- *Act paralelas.*- En este campo se indica sí el estado está compuesto por varias actividades (*true*) o sí solo le corresponde una actividad (*false*).
- *Subtransacción.*- En este campo se indica la subtransacción a la que pertenecen las actividades de los estados con nivel 2. Sí la actividad tiene nivel uno, se especifica la cadena "--".

Asimismo, tendremos un *manejador de subtransacciones activas (MSA)*, el cual manipula un *arreglo* (un arreglo es una variable que puede contener varios valores ordenados por medio de un índice) en el cual se almacenarán las subtransacciones que se encuentren activas (Tabla II). El ASA solo tendrá un campo *subtransacción* en el que se registrará el nombre de la subtransacción a la que pertenece un estado y tendrá también un campo estado en que se almacenará el estado en que se encuentra dicha transacción.

Tabla II. Tabla que ejemplifica el arreglo de subtransacciones activas (ASA).

| Subtransacción | Estado |
|----------------|--------|
|                |        |



Así mismo, se tiene el control del estado actual y del estado siguiente de las actividades mediante el registro y manipulación de estos datos en un arreglo (tabla III) con los campos estado actual y estado siguiente.

Tabla III. Tabla que ejemplifica el manejador de estados.

| Estado actual | Estado siguiente |
|---------------|------------------|
|               |                  |

Una vez definidos los elementos necesarios para llevar a cabo el manejo de errores, es necesario encontrar la forma de implementarlos. La estrategia de recuperación es la siguiente:

Sí se detecta que el sistema no se cerró de forma correcta (mediante la revisión de una bandera), se revisará en la bitácora cual fue el último estado de nivel 1 que se realizó o se estaba realizando por cada instancia al momento de la falla y esa información será establecida como estado actual en cada una de las instancias activas.

Al momento de recuperar los estados se puede presentar los siguientes casos:

- El estado no tiene actividades paralelas (*act paralelas = false*).
  - Si la actividad tiene status “*procesando*”, se ejecuta nuevamente esta actividad.
  - Si la actividad tiene como status “*finalizado*”, se ejecuta la siguiente actividad en el workflow.
- El estado tiene actividades paralelas (*act paralelas = true*).

Sí el estado tiene a su vez subactividades (actividades que se pueden realizar en forma paralela), éstas se almacenan en un arreglo de subtransacciones (AST), el cual tendrá los siguientes datos:

Tabla IV. Tabla que ejemplifica un arreglo de subtransacciones.

| Instancia | Estado | Subtransacción | Status |
|-----------|--------|----------------|--------|
|           |        |                |        |

- *Instancia.*- En este campo se almacenará el identificador de la instancia del proceso recuperado.
- *Estado.*- En este campo se registra el estado en que se encuentra la instancia del proceso.
- *Subtransacción.*- En este campo se almacena la subtransacción a la que pertenece el estado.
- *Status.*- En este campo se indica si la actividad que corresponde al estado se encuentra en ejecución (*procesando*) o ya ha sido ejecutada (*finalizado*).

Al final de la revisión de la bitácora, en el AST deben estar los datos de la última actividad que finalizó o estaba en ejecución de cada subtransacción en la instancia.

### **Ejemplo de funcionamiento**

Enseguida se muestra brevemente el funcionamiento del mecanismo.

1. Se ejecuta el estado A1 (figura 20), por lo que la bitácora (tabla V) y el manejador de estados (tabla VI) quedan de la siguiente manera:

```
- <state idState="A1" ActivityName="Activity 1" level="1" compoundState="false" st="--">
  <transition nextState="A2" />
</state>
```

Figura 20. Configuración del estado A1 del rol de ejemplo.

Tabla V. Tabla que ejemplifica la bitácora después de la ejecución del estado A1.

| Instancia | Estado | Status     | Nivel | Act paralelas | Subtransaccion |
|-----------|--------|------------|-------|---------------|----------------|
| 001       | A1     | procesando | 1     | false         | no             |
| 001       | A1     | finalizado | 1     | false         | no             |

Tabla VI. Tabla que ejemplifica el manejador de estados después de la ejecución del estado A1.

| Estado actual | Estado siguiente |
|---------------|------------------|
| A1            | A2               |

2. Una vez ejecutado el estado A1, se comienza a procesar el estado A2 (figura 21). Se puede observar en el documento XML (figura 21) que el estado es compuesto, por lo que se inicializa el MSA. La bitácora (tabla VII), el manejador de estados (tabla VIII) y el ASA (tabla IX) quedarían de la siguiente manera:

```
- <state idState="A2" ActivityName="Parallel item" level="1" compoundState="true" st="--">
  <transition nextState="A6" />
  <subtransition stName="S1" initialState="A3" />
  <subtransition stName="S2" initialState="A5" />
</state>
```

Figura 21. Configuración del estado A2 del rol de ejemplo.

Tabla VII. Tabla que ejemplifica la bitácora durante la ejecución del estado A2.

| Instancia | Estado | Status     | Nivel | Act paralelas | Subtransaccion |
|-----------|--------|------------|-------|---------------|----------------|
| 001       | A1     | procesando | 1     | false         | no             |
| 001       | A1     | finalizado | 1     | false         | no             |
| 001       | A2     | procesando | 1     | true          | no             |

Tabla VIII. Tabla que ejemplifica el manejador de estados durante la ejecución del estado A2.

| Estado actual | Estado siguiente |
|---------------|------------------|
| A2            | A6               |



Tabla IX. Tabla que ejemplifica el arreglo de subtransacciones activas durante la ejecución del estado A2.

| Subtransacción | Estado |
|----------------|--------|
| S1             | A3     |
| S2             | A5     |

2.1. En este punto vemos (tabla IX) que se pueden llevar a cabo 2 actividades (A3 y A5). Para ejemplificar, suponemos que el actor eligió realizar A3. La bitácora (tabla X), el manejador de estados (tabla XI) y el ASA (tabla XII) quedarían de la siguiente manera:

Tabla X. Tabla que ejemplifica la bitácora después la ejecución del estado A3.

| Instancia | Estado | Status     | Nivel | Act paralelas | Subtransaccion |
|-----------|--------|------------|-------|---------------|----------------|
| 001       | A1     | procesando | 1     | false         | no             |
| 001       | A1     | finalizado | 1     | false         | no             |
| 001       | A2     | procesando | 1     | true          | no             |
| 001       | A3     | procesando | 2     | false         | s1             |
| 001       | A3     | finalizado | 2     | false         | s1             |

Tabla XI. Tabla que ejemplifica el manejador de estados después de la ejecución del estado A3.

| Estado actual | Estado siguiente |
|---------------|------------------|
| A2            | A6               |

Tabla XII. Tabla que ejemplifica el arreglo de subtransacciones activas después de la ejecución del estado A3.

| Subtransacción | Estado |
|----------------|--------|
| S1             | A4     |
| S2             | A5     |

2.2 Después, establecemos que el actor ejecutó la actividad A5, entonces la bitácora (tabla XIII), el manejador de estados (tabla XIV) y el ASA (tabla 15) quedarían de la siguiente manera:



Tabla XIII. Tabla que ejemplifica la bitácora después la ejecución del estado A5.

| Instancia | Estado | Status     | Nivel | Act paralelas | Subtransaccion |
|-----------|--------|------------|-------|---------------|----------------|
| 001       | A1     | procesando | 1     | false         | no             |
| 001       | A1     | finalizado | 1     | false         | no             |
| 001       | A2     | procesando | 1     | true          | no             |
| 001       | A3     | procesando | 2     | false         | s1             |
| 001       | A3     | finalizado | 2     | false         | s1             |
| 001       | A5     | procesando | 2     | false         | s2             |
| 001       | A5     | finalizado | 2     | false         | s2             |

Se borra S2 del ASA, puesto que ya no quedan actividades de esa subtransacción por realizarse.

Tabla XIV. Tabla que ejemplifica el manejador de estados después de la ejecución del estado A5.

| Estado actual | Estado siguiente |
|---------------|------------------|
| A2            | A4               |

Tabla XV. Tabla que ejemplifica el arreglo de subtransacciones activas después de la ejecución del estado A5.

| Subtransacción | Estado |
|----------------|--------|
| S1             | A4     |

2.3 Se ejecuta A4, entonces la bitácora (tabla XVI), el manejador de estados (tabla XVII) y el ASA (tabla XVIII) quedarían de la siguiente manera:

Tabla XVI. Tabla que ejemplifica la bitácora después la ejecución del estado A4.

| Instancia | Estado | Status     | Nivel | Act paralelas | Subtransaccion |
|-----------|--------|------------|-------|---------------|----------------|
| 001       | A1     | procesando | 1     | false         | no             |
| 001       | A1     | finalizado | 1     | false         | no             |
| 001       | A2     | procesando | 1     | true          | no             |
| 001       | A3     | procesando | 2     | false         | s1             |
| 001       | A3     | finalizado | 2     | false         | s1             |
| 001       | A5     | procesando | 2     | false         | s2             |
| 001       | A5     | finalizado | 2     | false         | s2             |
| 001       | A4     | procesando | 2     | false         | s2             |
| 001       | A4     | finalizado | 2     | false         | s2             |

Se borra S1 del ASA, puesto que ya no quedan actividades de esa subtransacción por realizarse.

Tabla XVII. Tabla que ejemplifica el manejador de estados después de la ejecución del estado A4.

| Estado actual | Estado siguiente |
|---------------|------------------|
| A2            | A4               |

Tabla XVIII. Tabla que ejemplifica el arreglo de subtransacciones activas después de la ejecución del estado A5.

| Subtransacción | Estado |
|----------------|--------|
|                |        |

2.4 Como el ASA está vacío, significa que ya no quedan actividades del estado A2 por realizarse, por lo tanto se marca el estado A2 como finalizado (tabla XIX).

Tabla XIX Tabla que ejemplifica la bitácora después la ejecución del estado A2.

| Instancia | Estado | Status     | Nivel | Act paralelas | Subtransaccion |
|-----------|--------|------------|-------|---------------|----------------|
| 001       | A1     | procesando | 1     | false         | no             |
| 001       | A1     | finalizado | 1     | false         | no             |
| 001       | A2     | procesando | 1     | true          | no             |
| 001       | A3     | procesando | 2     | false         | s1             |
| 001       | A3     | finalizado | 2     | false         | s1             |
| 001       | A5     | procesando | 2     | false         | s2             |
| 001       | A5     | finalizado | 2     | false         | s2             |
| 001       | A4     | procesando | 2     | false         | s2             |
| 001       | A4     | finalizado | 2     | false         | s2             |
| 001       | A2     | finalizado | 1     | true          | no             |

3. Finalmente se ejecuta A6 y el proceso finaliza (tabla XX).

Tabla XX. Tabla que ejemplifica la bitácora después la ejecución del estado A6.

| Instancia | Estado | Status     | Nivel | Act paralelas | Subtransaccion |
|-----------|--------|------------|-------|---------------|----------------|
| 001       | A1     | procesando | 1     | false         | no             |
| 001       | A1     | finalizado | 1     | false         | no             |
| 001       | A2     | procesando | 1     | true          | no             |
| 001       | A3     | procesando | 2     | false         | s1             |
| 001       | A3     | finalizado | 2     | false         | s1             |
| 001       | A5     | procesando | 2     | false         | s2             |
| 001       | A5     | finalizado | 2     | false         | s2             |
| 001       | A4     | procesando | 2     | false         | s2             |
| 001       | A4     | finalizado | 2     | false         | s2             |
| 001       | A2     | finalizado | 1     | true          | no             |

Tabla XX. (Continuación)

|     |    |            |   |       |    |
|-----|----|------------|---|-------|----|
| 001 | A6 | procesando | 1 | false | no |
| 001 | A6 | finalizado | 1 | false | no |

En la siguiente sección se muestran los escenarios en donde se pueden presentar algunos errores al momento de poner a funcionar el sistema.

### ***III.6 Escenarios de manejo de errores.***

El área de manejo de errores en sistemas de workflow es bastante extensa, por lo que el hacer un análisis detallado de todos los posibles escenarios en los cuales se pueden presentar errores y presentar una posible solución a esta problemática requiere un esfuerzo de proporciones similares al realizado en el desarrollo de este trabajo.

Entre las posibles causas que pueden causar un funcionamiento no adecuado en un sistema de workflow se tienen las siguientes:

- **Nodo distribuidor apagado.-** Si el nodo distribuidor se encuentra apagado en el momento en que un actor que está trabajando en un nodo distribuido desea salir del sistema, se puede presentar un problema de inconsistencia en el sistema, ya que si el actor inicia sesión en un nodo diferente, no tendrá disponible la información actualizada del estado en que se encuentran las instancias de los procesos que está realizando. Además, mientras el nodo distribuidor se encuentre apagado, no será posible dar de alta a ningún actor.



- Accesos no autorizados al sistema de workflow.- Es común que los sistemas de cómputo sufran algunos ataques realizados por personas que pretenden acceder al sistema con la finalidad de sacar algún provecho, o simplemente por el reto que representa vulnerar la seguridad de un sistema de cómputo. En este sentido, en la arquitectura WODIS se propone un mecanismo de validación de actores que impida el uso del sistema a usuarios no autorizados.
- Inconsistencia de la información.- El funcionamiento adecuado en un sistema de workflow depende en gran medida de la administración de las entidades de información que conforman a estos sistemas, por lo que si alguna de éstas presenta alguna inconsistencia, el funcionamiento del sistema puede verse afectado. En este trabajo se propone un mecanismo que se encarga de verificar la consistencia de las bitácoras (la existente en el nodo distribuidor y en cada nodo distribuido) con la finalidad de que ambas tengan registrado el mismo estado para cada una de las instancias activas.

Además de las situaciones anteriores, existen algunas otras que pueden causar el funcionamiento inadecuado en el sistema y que se abordan con mayor detalle en este trabajo.

Algo que es muy común que suceda al momento de que se está ejecutando el sistema, es que un actor requiera comunicarse con algún otro, y que la comunicación entre ellos no pueda ser establecida.

Un error de este tipo sucede cuando un nodo necesita interactuar con otro nodo y la comunicación entre ellos no puede ser establecida. Este error se puede presentar debido a algunas de las siguientes causas:

- No hay actores disponibles que desempeñen el rol con el cual se requiere interactuar al momento de intentar establecer la comunicación.
- La comunicación entre los nodos no se puede establecer debido a problemas de la red.

Como se puede ver en la figura 22, el escenario comienza cuando el *administrador de estados* (AE) está coordinando la ejecución de una actividad. Cuando la actividad termina, el AE obtiene un nuevo estado dentro del proceso. Si la actividad del nuevo estado no requiere comunicación con otro actor, se continúa con la ejecución de las actividades. Si por el contrario, este estado requiere interacción con otro actor, se debe enviar esta notificación al actor por medio del *emisor de mensajes* (EM).

Una vez que el *EM* recibe la petición de enviar mensaje, solicita al mecanismo *disponibilidad de nodos* (DN) que verifique si el actor al que se le debe enviar la notificación está disponible. El *DN* busca la información del actor y una vez que la obtiene, la envía al *EM*, quien al recibirla revisa el contenido de ésta. Si el actor no está disponible en el sistema, la notificación se envía al *Administrador de tareas pendientes* (ATP), quien guarda la información y posteriormente se encarga de volver a mandar esta

información hasta que logre su objetivo. Es importante mencionar que para enviar la notificación, el *ATP* nuevamente utiliza al *AE* como una medio de comunicación.

Por otro lado, si el *DA* le comunica al *EM* que el actor está disponible, este envía el mensaje al *rol receptor*, y éste lo recibe y notifica al *EM* que el mensaje ha llegado correctamente. Si el *AE* detecta que el mensaje llegó correctamente, el proceso finaliza. Si el mensaje no pudo ser enviado, se envía nuevamente al *ATP* para que vuelva a intentar la comunicación.

Dentro de las limitaciones que presentan las acciones propuestas para este escenario se encuentra que si la comunicación con el nodo distribuidor no puede establecerse, el nodo distribuido que debe enviar tareas, no debe finalizar su ejecución, ya que las notificaciones podrían perderse si otro actor entra a ese nodo antes de que esas notificaciones sean enviadas.

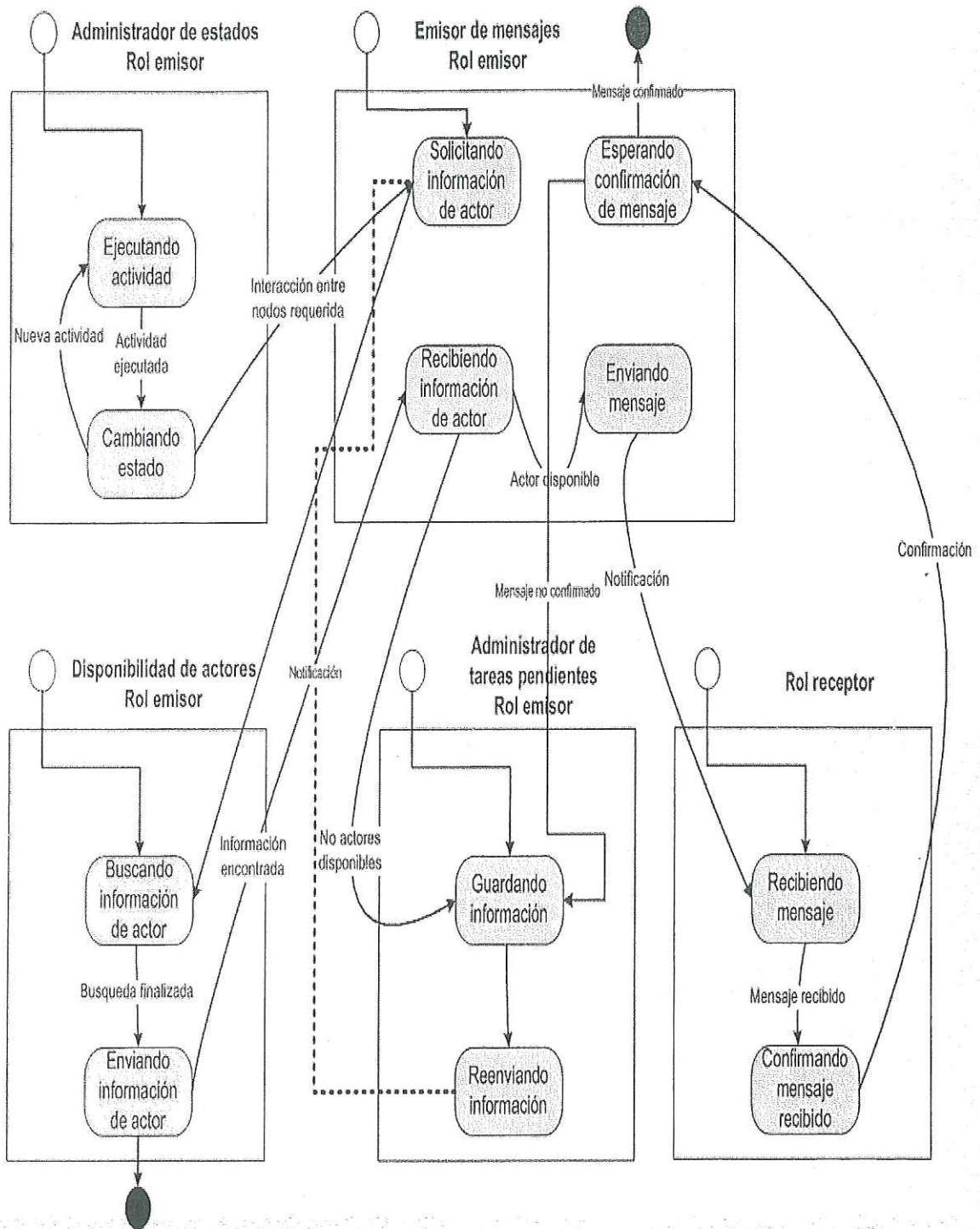


Figura 22. Escenario de errores debido a comunicación fallida con otro usuario.



Otro de los errores que pueden presentarse en un sistema de workflow, es que algún nodo se cierre de manera incorrecta, ya sea por accidente, fallas en la corriente eléctrica, por descuido del usuario o por problemas de la PC entre otras. La figura 23 ilustra este escenario de error y las acciones que se deben tomar cuando esto sucede.

Primeramente, el actor se da de alta nuevamente en un *nodo distribuido (DoN)*, y se manda validar al actor al *nodo distribuidor (DrN)*. En el *DrN*, el mecanismo de *validación* se encarga de revisar que el actor esté autorizado a utilizar el sistema, y en caso de que así sea, revisa si el *DoN* en cual está accedando el actor es el mismo en el que lo hizo la vez anterior que utilizó el sistema. En caso de que no sea el mismo nodo, se envía al *DoN* toda la información necesaria para que pueda iniciar la ejecución del proceso.

En caso contrario, se revisa si el sistema fue cerrado correctamente. Si así fue, se obtiene de la *bitácora local* el estado en que se encuentran las instancias del proceso y se comienza la ejecución del proceso. Si el nodo no fue cerrado correctamente, se obtiene tanto la *bitácora local* como la *bitácora global* y se obtiene de ambas el estado en que se encuentran las instancias del proceso, y finalmente se comienza la ejecución del proceso.

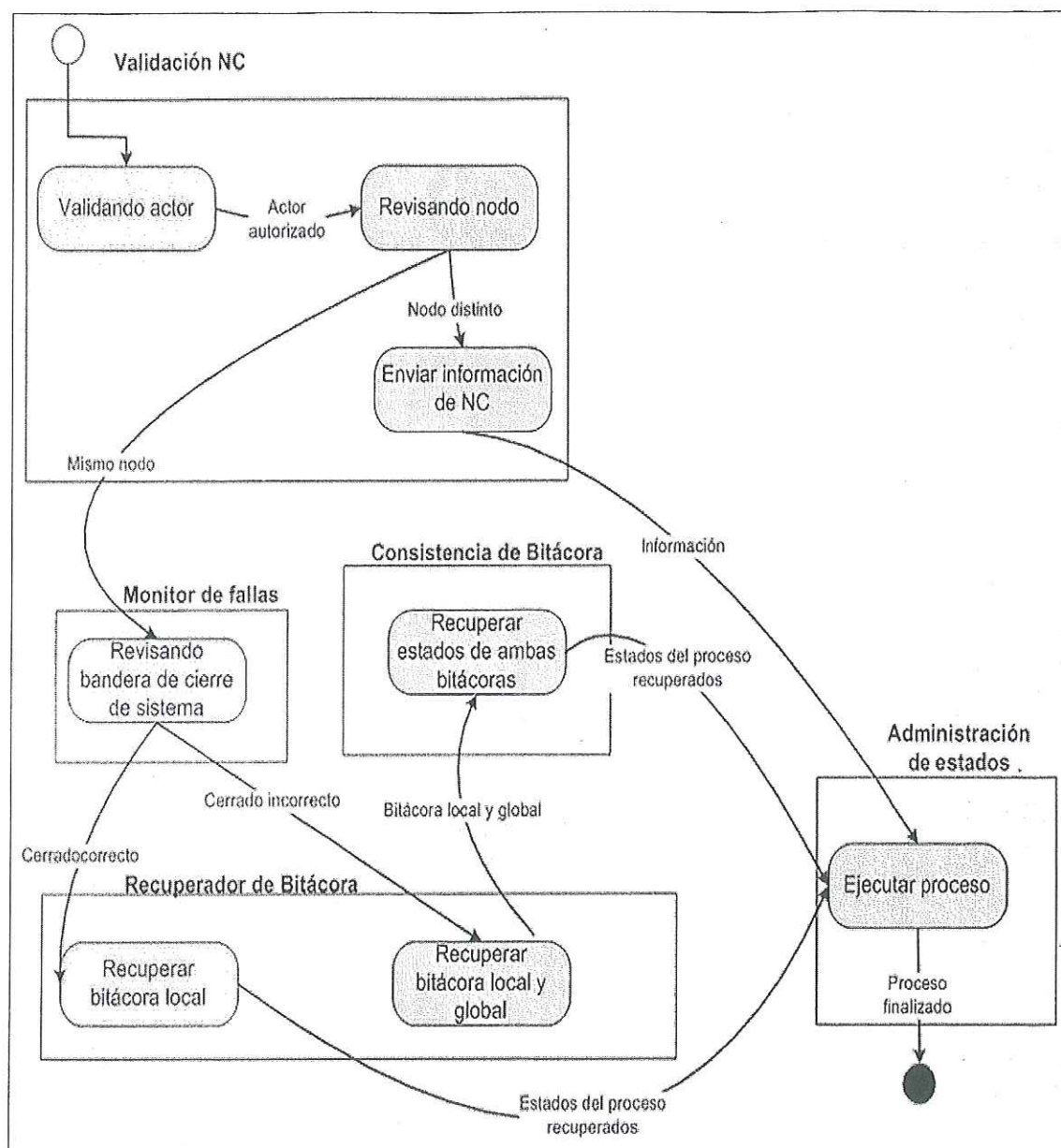


Figura 23. Escenario de errores debido a que el nodo fue cerrado incorrectamente.

En la siguiente sección se presenta una breve guía para desarrollar un sistema de workflow distribuido basado en la arquitectura.

### ***III.7 Guía breve para el desarrollo de un sistema de workflow distribuido basado en la arquitectura WODIS.***

La figura 24 muestra las actividades que debe realizar el usuario para desarrollar un sistema de Workflow basado en la arquitectura propuesta.

Antes de desarrollar el sistema de workflow, es necesario realizar un análisis del proceso a automatizar, con la finalidad de identificar las actividades que se llevan a cabo en el proceso, los actores del proceso, los roles del proceso, las interacciones entre actores y las entidades de información que se requieren en el proceso.

Una vez analizado el proceso, es necesario modelarlo por medio de diagramas Rol-Actividad y en base a este diagrama, generar el modelo base del sistema en XML. También debe de hacerse un directorio de actores en XML en donde se indiquen los datos de los usuarios, tales como nombre de usuario, contraseña, ubicación del nodo en que se encuentra y roles que desempeña. Estas dos actividades deben desarrollarse antes de comenzar con el diseño de los objetos distribuidos, por medio del lenguaje de definición de interfaces de CORBA (IDL), que darán soporte a la ejecución de cada actividad y finalmente se construyen las aplicaciones que fungirán como nodo distribuidor y nodo distribuido respectivamente.

Al momento de desarrollar la aplicación que fungirá como nodo distribuido, se debe tomar en cuenta que ésta debe cumplir con los requisitos básicos de un sistema de workflow,

como son escalabilidad, recuperación de fallas, disponibilidad, monitoreo, historial y soporte al uso de aplicaciones externas. Además, se debe implementar una estrategia que permita a un nodo conocer los cambios que realizan que realizan otros actores en sus nodos respectivos y que afectan el estado en que este nodo se encuentra.

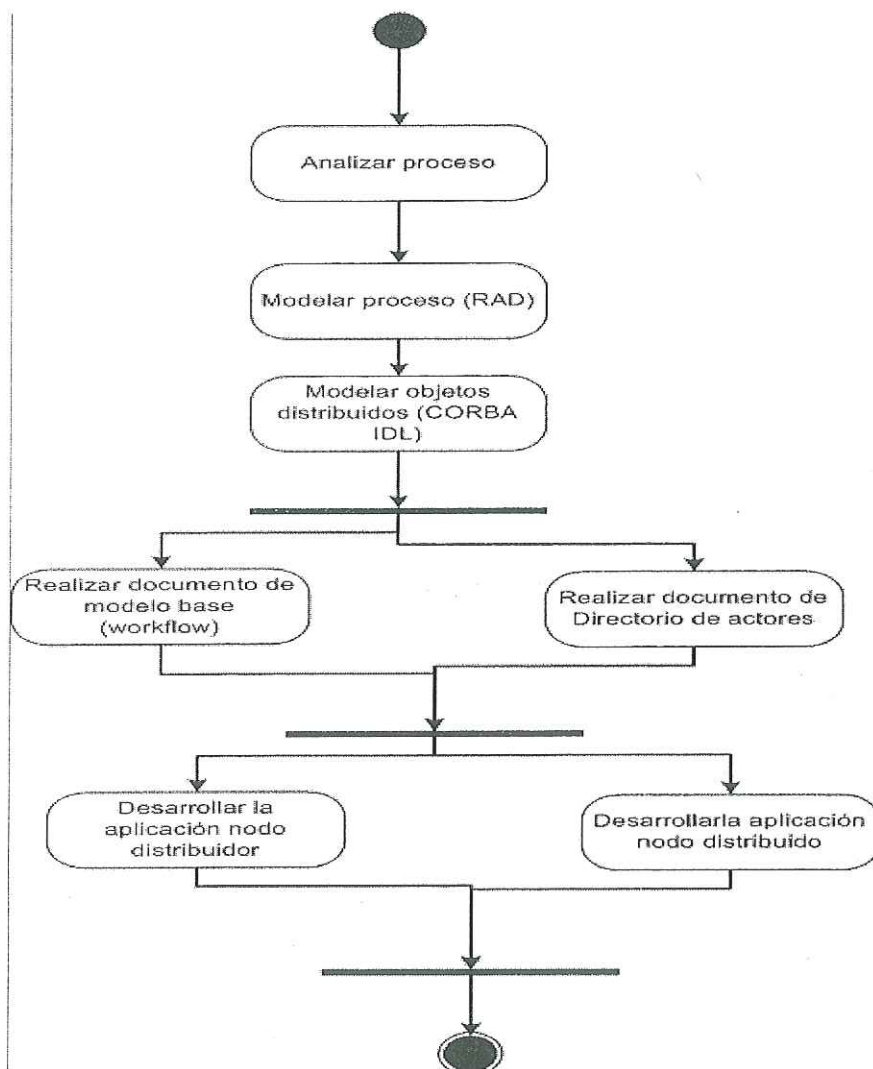


Figura 24. Pasos a seguir para desarrollar sistemas basados en la arquitectura propuesta.

En la siguiente sección se presenta un resumen de lo presentado en este capítulo.



### **III.8 Resumen.**

En este capítulo se ha presentaron los requerimientos mínimos que una arquitectura distribuida de sistemas de Workflow debe satisfacer. Así mismo, se presentó el diseño de la arquitectura, la cual consta de dos diferentes tipos de nodos y se mostró también la descripción de sus componentes. La funcionalidad que brinda la arquitectura es mostrada por medio de diagramas UML, en específico diagramas de casos de uso y diagramas de secuencia.

Además, explicó la importancia y necesidad de contar con un método de recuperación de errores en los sistemas de workflow, y se mencionó también que un enfoque muy útil en el manejo de errores, es por medio de manejo de transacciones.

Sin embargo, de los modelos de transacciones existentes, no todos son adecuados para ser implementados como estrategia de recuperación de errores en sistemas de workflow, como es el caso del modelo ACID, el cuál a pesar de su sencillez y confiabilidad, está diseñado principalmente para ser utilizado en sistemas convencionales de bases de datos, por lo tanto presenta algunas limitantes al ser implementado en sistemas de workflow.

Los modelos de transacciones avanzadas están más orientados a ser utilizados en aplicaciones no convencionales. Existen varios de estos modelos, siendo el modelo de transacciones flexibles el que más se adapta a las características de los sistemas de workflow distribuido, principalmente porque presenta el manejo de transacciones a

múltiples niveles, y permite además que en caso de que alguna transacción falle, las transacciones pueden actuar de distintas maneras.

Finalmente, se muestra una pequeña guía para utilizar la arquitectura propuesta en el desarrollo de sistemas de Workflow. En el siguiente capítulo se presentan los aspectos de implementación de los dos prototipos que se desarrollaron en este trabajo de tesis.

## Capítulo IV. Implementación de un prototipo

---

### IV.1 Introducción

Con la finalidad de probar la viabilidad tecnológica de desarrollar sistemas de workflow basados en la arquitectura WODIS, se construyeron dos prototipos de sistemas de workflow. El primer prototipo se desarrollo bajo un esquema distribuido descentralizado, el cual solo presenta interacción con el *nodo distribuidor (DrN)* al momento de que cada actor comienza a utilizar el sistema por primera vez. La finalidad de la construcción de este prototipo, fue probar la tecnología de desarrollo elegida, así como probar algunos aspectos del diseño de la arquitectura. Entre sus ventajas principales se destaca la coordinación y ejecución de las actividades de manera descentralizada e independiente del *DrN*. Este prototipo fue desarrollado con la finalidad de probar la factibilidad tecnológica y la funcionalidad de los componentes de la arquitectura.

Con el desarrollo del primer prototipo, se realizaron las modificaciones necesarias en la arquitectura WODIS de acuerdo a los problemas encontrados y se desarrolló un segundo prototipo, el cual al igual que el primer prototipo, permite que la coordinación y ejecución de las actividades se lleve a cabo de forma independiente del *DrN*. La finalidad de este prototipo es implementar y probar de forma completa el diseño de la arquitectura.

Para reflejar la forma natural en como los procesos se llevan a cabo en forma natural, fue necesario incluir algunas características centralizadas a este nuevo prototipo. Entre estas nuevas características se tiene que cada vez que un actor sale del sistema, el nodo

distribuido (DoN) en el cual el actor estaba trabajando, establece comunicación con el *DrN* para enviarle las entidades de información, con el fin de que éstas puedan ser utilizadas por algún otro actor, en caso de que sean requeridas durante el tiempo en que el actor que acaba de salir del sistema se encuentre ausente. Otra característica que se añadió al nuevo prototipo es que el *DrN* lleva un control del estado en que se encuentran todas las instancias del proceso, por medio de actualizaciones que hacen todos los *DoN* a una bitácora global ubicada en el *DrN*, las cuales se realizan cada vez que se ejecuta una transacción en el *DoN*; esta característica permite tener un mejor control para recuperar el estado de un proceso en caso de que un error ocurra.

En el resto del capítulo se presenta el proceso de desarrollo de ambos prototipos: selección del proceso, modelado del proceso, tecnología de desarrollo seleccionada y funcionamiento del sistema.

## ***IV. 2 Tecnología de desarrollo utilizada.***

Para implementar los prototipos, fue necesario el contar con tecnología que permitiera el desarrollo en tres aspectos principales: comunicación entre nodos, funcionalidad del prototipo y manejo de la información. En el desarrollo del prototipo se optó por la siguiente tecnología de desarrollo:

- **CORBA.**- Fue utilizado para implementar la comunicación entre nodos. Es un estándar para sistemas distribuidos definido por la OMG (Object Management Group), en el cual son definidas las interfaces de programación de aplicaciones



(API por sus siglas en inglés), el protocolo de comunicación y los mecanismos necesarios para permitir la interoperabilidad entre diferentes aplicaciones, ejecutadas en diferentes plataformas y escritas en diferentes lenguajes [Mowbray y Malveau, 1997]. Para proporcionar esta independencia de lenguaje y plataforma, CORBA proporciona el *lenguaje de definición de interfaces (IDL por sus siglas en inglés)*.

El lenguaje de definición de interfaces (IDL) de CORBA sirve exactamente para lo que su nombre indica: definir interfaces. En el no se especifican detalles de implementación o funcionalidad, sino que esta será dada una vez que el código en IDL ya fue convertido a algún lenguaje que soporta el uso de un ORB (Object Request Broker) de CORBA. Un ORB es el encargado de administrar y establecer la comunicación entre objetos remotos en CORBA.

Antes de la elección de CORBA como tecnología de comunicación entre nodos, se hizo un análisis de otras tecnologías que pudieran cumplir con este fin, entre las cuales se encuentran RMI, .NET, Servlets y Sockets.

En el caso de *RMI* y *.NET* tienen características similares a CORBA, con la excepción de que éstas están asociadas a una tecnología en particular (*RMI* está asociado a *JAVA* y *.NET* está asociado a *Microsoft*), por lo que no fueron una opción adecuada en el desarrollo de los prototipos, ya que se quiso evitar dar la idea de que la arquitectura está ligada a una tecnología en particular.

Por otra parte, los *servlets* son una tecnología que permite la invocación de funciones remotas. Sin embargo, no se puede establecer comunicación de forma directa entre *servlets*, por lo que para establecer la comunicación entre nodos es necesario instalar en cada nodo una aplicación independiente que sirva de enlace con cada uno de los otros nodos, lo que lleva a que sea necesario tener varias aplicaciones en ejecución para poder cumplir con la funcionalidad requerida en un sistema de workflow distribuida.

Finalmente se tienen los *sockets*, los cuales son un protocolo de bajo nivel para la comunicación e intercambio de datos entre dos máquinas. Uno de los inconvenientes que presentan los *sockets* para ser utilizados como tecnología de comunicación entre nodos en un sistema de workflow distribuido, es que la invocación de las actividades de forma remota no se da de manera directa, sino que se tiene que enviar un conjunto de parámetros que ayuden a determinar cual es el método que se va a ejecutar, lo cual lo hace quedar en desventaja ante CORBA, ya que permite la invocación remota de actividades.

CORBA es cada vez mas utilizado en aplicaciones *e-business* (negocio electrónico) y *e-commerce* (comercio electrónico), principalmente porque permite la interacción entre sistemas desarrollados en diferentes lenguajes, de manera que se fomenta la reutilización de código y de sistemas ya existentes, además de que la utilización de

recursos de cómputo para brindar los servicios que ofrecen las organizaciones es distribuida entre varios nodos [Mowbray y Malveau, 1997].

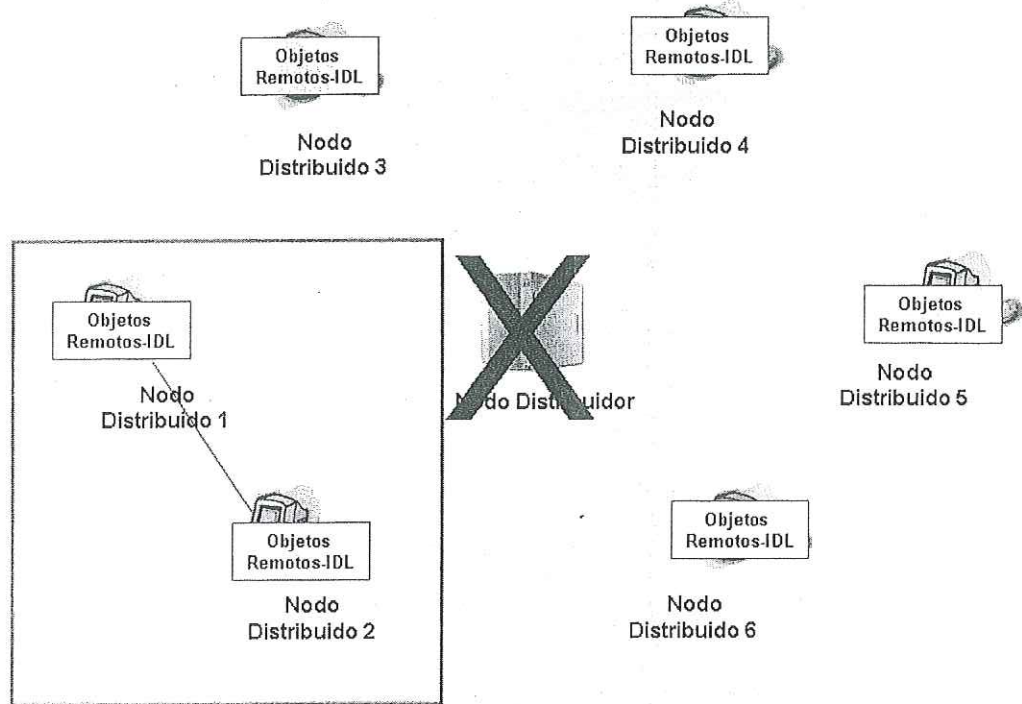


Figura 25. Comunicación entre nodos por medio de CORBA en los prototipos desarrollados en este trabajo.

CORBA se utilizó en el desarrollo de los prototipos de este trabajo, debido a que sus características se adaptan a las requeridas en una arquitectura de workflow distribuida que permita la implementación de la forma natural de trabajo. Estas características son las siguientes y se muestran en la figura 25:

- Por su naturaleza orientada a objetos, CORBA permite que las actividades correspondientes a cada actor sean encapsuladas en un objeto, de manera que este pueda ser alojado en el nodo distribuido en el que éste actor realiza sus actividades, permitiendo con esto que las actividades remotas puedan ser

ejecutadas en diferentes nodos, contribuyendo a que se distribuya la carga de trabajo originada por la ejecución de las actividades en un mismo punto.

- La comunicación entre nodos se lleva a cabo de manera directa entre dos nodos sin necesidad de un servidor central que se encargue de controlar la comunicación entre nodos, lo cual se asemeja a la forma natural en que se llevan a cabo los procesos.

Es importante mencionar que cada uno de los nodos es responsable de identificar el momento en que otro nodo establece comunicación con él, y se establece como parte del flujo de trabajo, es decir, en la definición del proceso se deben tomar en cuenta las tareas internas que el nodo distribuido debe realizar para la interacción con otros nodos.

- **JAVA.**- Es un lenguaje de programación orientado a objetos desarrollado por SUN y tiene como principal característica la independencia de la plataforma. La funcionalidad del prototipo fue desarrollado por medio de JAVA [Orfali et al., 1997].

Para esto, se utilizaron las APIs que provee JAVA. Una API es un conjunto de especificaciones de comunicación entre componentes de software. Uno de los principales propósitos de una API consiste en proporcionar un conjunto de funciones de uso general, por ejemplo, para dibujar ventanas o iconos en la pantalla.



Las principales APIS utilizadas fueron las siguientes:

- **org.omg.CORBA.**- Esta API provee el mapeo de la especificación CORBA que establece la OMG al lenguaje JAVA. Además, provee una clase ORB, la cual es implementada de tal manera que el programador pueda utilizarlo como un ORB totalmente funcional.
- **javax.swing.**- Provee un conjunto de componentes visuales que permite facilidades a los programadores para desarrollar la interfaz gráfica de usuario.
- **org.w3c.dom.**- Es un conjunto de clases que permite la manipulación y creación de documentos XML. Provee las interfaces para la implementación del modelo de objeto de documento (DOM por sus siglas en inglés).
- **org.xml.sax.**- Es una API que proporciona las clases e interfaces que implementan las especificaciones para la API Simple para XML (SAX por sus siglas en inglés).
- **XML.**- Es un metalenguaje que define una sintaxis útil para definir otros lenguajes de etiquetas estructuradas [Gutierrez y Martínez, 2001]. Los documentos están compuestos por elementos, los cuales son considerados como las unidades básicas para la creación de un documento XML. Es similar a HTML, con la diferencia que éste tiene etiquetas predefinidas, a diferencia de XML que permite a los desarrolladores crear las propias. Son utilizados para la representación de la estructura del documento y los datos que éste contiene. Además, pueden contener atributos, los cuales muestran información sobre el contenido del elemento. Están

constituidos por un nombre de atributos y su valor asociado. Los elementos tienen una etiqueta inicial y una etiqueta final (las cuales deben coincidir exactamente).

XML se utilizó ya que permite el almacenamiento de datos y el manejo de éstos se lleva a cabo de forma más ligera que como se haría con una base de datos tradicional, debido a que no es necesario tener el esquema de la base de datos en cada nodo en el cual se requiere cierta información, sino que simplemente se envía entre nodos el documento que contiene la información requerida.

En la implementación de este prototipo se utilizó XML para la definición de los documentos propios del sistema, como el cuerpo del flujo de trabajo (workflow), la bitácora y el directorio de actores, debido principalmente a las características que presenta. Estos documentos fueron definidos de la siguiente manera:

- **Workflow.-** Este documento está basado en los diagramas rol actividad (RAD).

Las actividades son agrupadas en base a roles. Este documento está definido por los siguientes elementos:

- **workflow.-** Este es el elemento principal del documento. Contiene uno o varios elementos *estadosRol*.

- **estadosRol.-** Este elemento agrupa los distintos estados por los cuales pasa cada rol. En el documento se especifica un elemento *estadosRol* por cada rol en el proceso. Contiene uno o varios elementos *estado*. Además, tiene el atributo *nombreRol* en el cual se indica el nombre del rol al cual pertenecen los estados definidos en este elemento.

▪ **estado.-** Define propiamente cada uno de los estados por los que pasa un rol. Puede o no contener uno o más elementos *transición* y mismo caso aplica para el elemento *comunicarseCon*. Sus atributos son *idEstado* en el cual se asocia un identificador único a cada estado del proceso, *nombreActividad* en donde se indica la actividad que se debe llevar a cabo cuando el proceso se encuentre en ese estado, *nivel* el cual se indica si el estado pertenece a un subflujo de ejecución, *estadoCompuesto* se marca verdadero si ese estado es el punto en el cual se inician dos o más subflujos de ejecución y falso en caso contrario, *pantalla* indica la interfaz de usuario que debe mostrarse al usuario cuando el proceso se encuentre en ese estado y finalmente en *perteneceA* se indica el subflujo de ejecución al cual pertenece el estado.

▪ **transición.-** En este elemento se almacena el estado en el que se encontrará el sistema una vez que la actividad que corresponde a este estado es finalizada. Contiene el atributo *estadoSiguiente* en el cual se especifica el identificador (atributo *idEstado*) del estado siguiente.

▪ **comunicarseCon.-** Este elemento se incluye en el elemento *estado* en caso de que la actividad correspondiente a este estado requiera interactuar con algún otro actor.

○ **Bitácora.-** El documento bitácora es utilizado para llevar un registro de las actividades que han ejecutado los actores, y de las que se encuentran en ejecución. Este elemento se construye de forma dinámica conforme los actores van ejecutando las actividades que les corresponde. Los elementos de este documento son:

- **bitácora.**- es el elemento principal del documento. Contiene uno o más elementos *transacción*.

- **transacción.**- Este elemento representa cada actividad que ha sido ejecutada o se encuentra en ejecución. Tiene los atributos *idInstancia* en el cual se especifica la instancia a la que pertenece esa transacción, *idEstado* en donde se indica el estado al que pertenece esa transacción, *status* en donde se establece si la actividad que corresponde al estado de la transacción ya fue ejecutada o se encuentra en ejecución, *paralelas* en donde se especifica si la actividad a la que pertenece la transacción es parte de un subflujo de actividad, *subtransacción* en donde se indica, en los casos que así corresponde, a cual subflujo pertenece la actividad, y finalmente *información* en donde se indica la información asociada a cada transacción.

- **Directorio de actores.**- Este documento es utilizado para la administración de actores. En él se concentran los datos de los actores que participan en el proceso, además de indicarse también el estado que guardan los actores en el proceso, de manera que de él se pueda obtener información sobre cuales actores se encuentran disponibles en el sistema y cuales no. Está formado por los siguientes elementos:

- **actores.**- Es el elemento principal del documento. Contiene uno o varios elementos *actor*.

- **actor.**- Este elemento contiene los datos que identifican a cada actor, los cuales son almacenados en los atributos *nombrePersonal*, *nombreCuenta*,



*contraseña*, *status* el cual contiene verdadero si el actor está disponible en el sistema y falso en caso contrario. Contiene también uno o más elementos *rol*.

- **rol.**- Contiene el nombre del rol que desempeña el actor. Esta información es almacenada en el atributo *nombreRol*.

En las siguientes secciones se explica la forma en como esta tecnología fue utilizada en la implementación de los prototipos.

### ***IV.3 Prototipo del proceso del juego de la cerveza.***

Uno de los objetivos de este trabajo fue el desarrollo de prototipos que nos permitieran probar la viabilidad tecnológica de desarrollar sistemas basados en la arquitectura, así como también probar que los sistemas basados en ésta cumple con la funcionalidad requerida.

El prototipo que implementa el proceso del juego de la cerveza fue desarrollado para probar que existe tecnología adecuada para el desarrollo de sistemas basados en esta arquitectura, así como también implementar la funcionalidad básica de un sistema de workflow distribuido. Por funcionalidad básica se entiende que la coordinación y ejecución del proceso pueda llevarse a cabo de adecuadamente de forma distribuida, lo cual implica que la coordinación del proceso, la comunicación entre nodos y los mecanismos de detección de errores funcionan adecuadamente.

Sin embargo, en este prototipo no se implementan algunos aspectos de la forma natural en que se llevan a cabo los procesos en la vida real, por lo que el proceso seleccionado para ser

coordinado por el prototipo es el proceso del juego de la cerveza, un proceso pequeño que trata sobre un vendedor que debe tener en el almacén la cantidad de producto suficiente para poder cumplir con las demandas de los clientes, pero a la vez, cuidar de no tener demasiado producto en el almacén que le pueda ocasionar un gasto de almacenamiento [García, 2001].

La figura 26 muestra el diagrama rol-actividad (RAD) del proceso del juego de la cerveza, en el cual se puede apreciar el flujo que siguen las actividades del proceso y las interacciones entre ellos. En este diagrama se puede apreciar claramente que participan tres roles en el proceso:

- Cliente.- Las actividades de este rol consisten en hacer pedidos de producto al responsable del expendido de cerveza (minorista).
- Minorista.- Este rol tiene es responsable de atender los pedidos que realizan los clientes y decrementar el nivel del inventario de acuerdo a la cantidad de producto que sea surtido al cliente. Además, debe revisar el inventario con el fin de que la existencia no esté por debajo del mínimo permitido; en caso de que esto suceda, debe hacer un pedido de producto al mayorista.
- Mayorista.- La responsabilidad de este rol es surtir los pedidos que haga el minorista.

En la figura 26(a) se pueden apreciar las actividades que lleva a cabo cada uno de los roles del proceso. El *cliente*, representado por el rectángulo izquierdo, debe escribir una orden de



pedido, y una vez que la termine debe enviarla al minorista para que este procese la orden. Posteriormente, el cliente permanece esperando a que el minorista le de respuesta a la orden solicitada. En caso de que requiera una nueva orden, debe realizar nuevamente estas actividades.

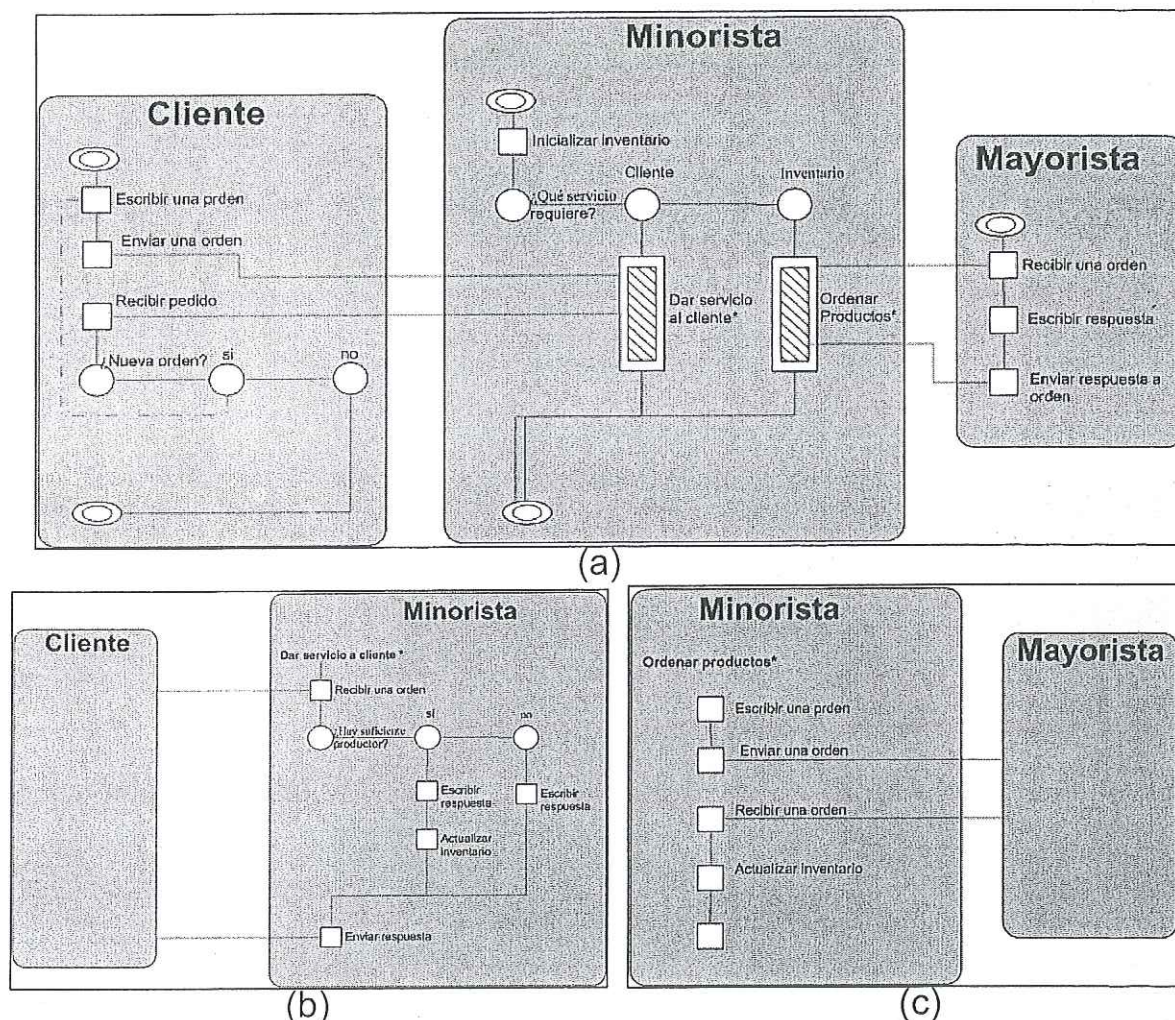


Figura 26. a) Diagrama RAD del proceso del juego de la cerveza. b) Sub proceso Dar servicio a cliente. c) Sub proceso Ordenar productos.

Por su parte, el *minorista*, representado en el rectángulo del centro (figura 26(a)) debe inicializar el inventario solamente la primera vez que entra al sistema. Después debe elegir

si da servicio al cliente o hace un pedido de producto al mayorista. El servicio al cliente se ilustra en la figura 26b. Ahí se puede apreciar que el minorista recibe la orden de pedido que le hizo algún cliente y revisa en el inventario si tiene suficiente producto para cumplir la orden.

En caso de que si haya el producto necesario en el almacén, se escribe la respuesta y se actualiza el inventario.

En caso necesario solo se le informa al cliente que no hay producto suficiente para cubrir el pedido. Además del servicio al cliente, el minorista puede solicitar productos al mayorista (figura 26(b)). El minorista debe escribir una orden de pedido de productos, enviarla al mayorista y esperar la respuesta. Cuando recibe la respuesta del mayorista debe actualizar el inventario.

Finalmente, el *mayorista*, representado en el rectángulo derecho (figura 26(a)), debe revisar el pedido que realice el minorista, escribir la respuesta y enviarla al minorista.

En lo que respecta a la comunicación entre roles, en las figuras 26(a) y 26(b) se puede apreciar las interacciones que se presentan entre el rol cliente y el rol minorista, las cuales se presentan cuando:

- El cliente hace un pedido de producto al minorista.
- El minorista da respuesta al pedido del cliente.



Asimismo, las figuras 26(a) y 26(c) muestran las interacciones entre el rol minorista y el rol mayorista, las cuales se presentan debido a las siguientes actividades:

- El minorista solicita productos al mayorista.
- El mayorista surte el producto del minorista.

De acuerdo a lo anterior se puede ver que el rol minorista interactúa con los otros dos roles, y que rol cliente y el rol mayorista no pueden interactuar entre sí.

En la siguiente sección se presenta el análisis del proceso del juego de la cerveza mediante diagramas del Lenguaje de Modelado Unificado (UML por sus siglas en inglés).

#### **IV.3.1 Análisis del prototipo del juego de la cerveza**

Como se mencionó en el capítulo III, un caso de uso es una interacción típica entre un usuario y un sistema de cómputo. Dicho de otra manera, un diagrama de casos de uso representa la funcionalidad del sistema desde la perspectiva del usuario. En esta sección solo se presenta el caso de uso *Surtir producto a cliente*, con el cual se ilustra la implementación del prototipo. El resto de los casos de uso pueden ser analizados en el Anexo A.

El caso de uso *Surtir producto a cliente* (figura 27) es llevado a cabo por el minorista. La finalidad de este caso de uso es atender las solicitudes de producto que han realizado los clientes.

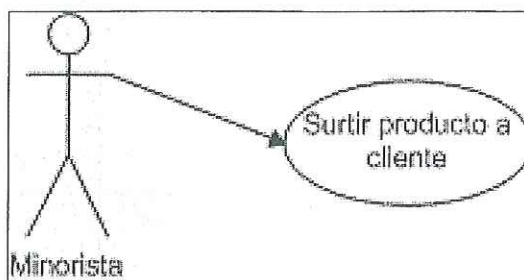


Figura 27. Diagrama del caso de uso *Surtir producto a cliente*.

### Descripción:

El caso de uso inicia cuando el actor minorista desea atender las solicitudes de pedido hechas por el cliente. El minorista revisa la lista de pedidos, y en caso de haber alguno, revisa en el inventario para saber si cuenta con el producto suficiente para cumplir con la demanda de producto indicada en la hoja de pedido. Si el pedido es cumplido, se marca el pedido como atendido y se ejecuta el caso de uso *Actualizar inventario*. Si por el contrario el pedido no puede satisfacerse, se marca el pedido como atendido y se ejecuta el caso de uso *Hacer pedido de producto*.

Una vez mostrado el caso de uso *Surtir producto a cliente*, se muestra también el diagrama de secuencia de este caso de uso. Como se mencionó también en el capítulo III, los diagramas de secuencia son modelos que describen la manera en que colaboran grupos de objetos para cierto comportamiento. Mientras que el diagrama de casos de uso permite el modelado desde punto de vista del usuario, el diagrama de secuencia contiene detalles de implementación del escenario, incluyendo los objetos y clases que se usan para implementar el escenario, y mensajes pasados entre los objetos. A continuación se presentan los diagramas de secuencia del proceso correspondientes a los casos de uso más

importantes del proceso del juego de la cerveza. En esta sección se presentan solo alguno de los diagramas de secuencia más importantes, y el resto son mostrados en el apéndice A.

El diagrama de secuencia que pertenece al caso de uso *Surtir pedido a cliente* (figura 28) comienza cuando el actor minorista decide dar servicio al cliente. Para esto revisa los pedidos existentes por medio del método *revisarPedidos()* del objeto minorista. El objeto minorista retorna una lista con el detalle de los pedidos existentes y el actor minorista elige el pedido que va a atender por medio de la invocación del método *atenderPedido(numPedido)*, y envía como parámetro el número de pedido que desea atender (a través de la variable *numPedido*). Este método a su vez obtiene, por medio del método *obtenerNomCli()*, el nombre del cliente que realizó el pedido. Asimismo, obtiene también la cantidad de producto solicitada, esta vez por medio del método *obtenerNumProd()* y almacena este valor en la variable *cantProd*.

Una vez que obtiene la cantidad de producto solicitada, el método *revisarInventario(cantProd)* verifica si cuenta en el almacén con la existencia de producto suficiente para cumplir con el pedido. En caso de que la existencia de producto sea suficiente, el método *actualizarInventario(cantProd)* se encarga de actualizar el inventario. Enseguida, el método *marcarPedido(numPed)* se encarga de etiquetar el pedido como “atendido” y finalmente se envía la respuesta al cliente mediante el método *enviarRespuesta(nomCli)*.

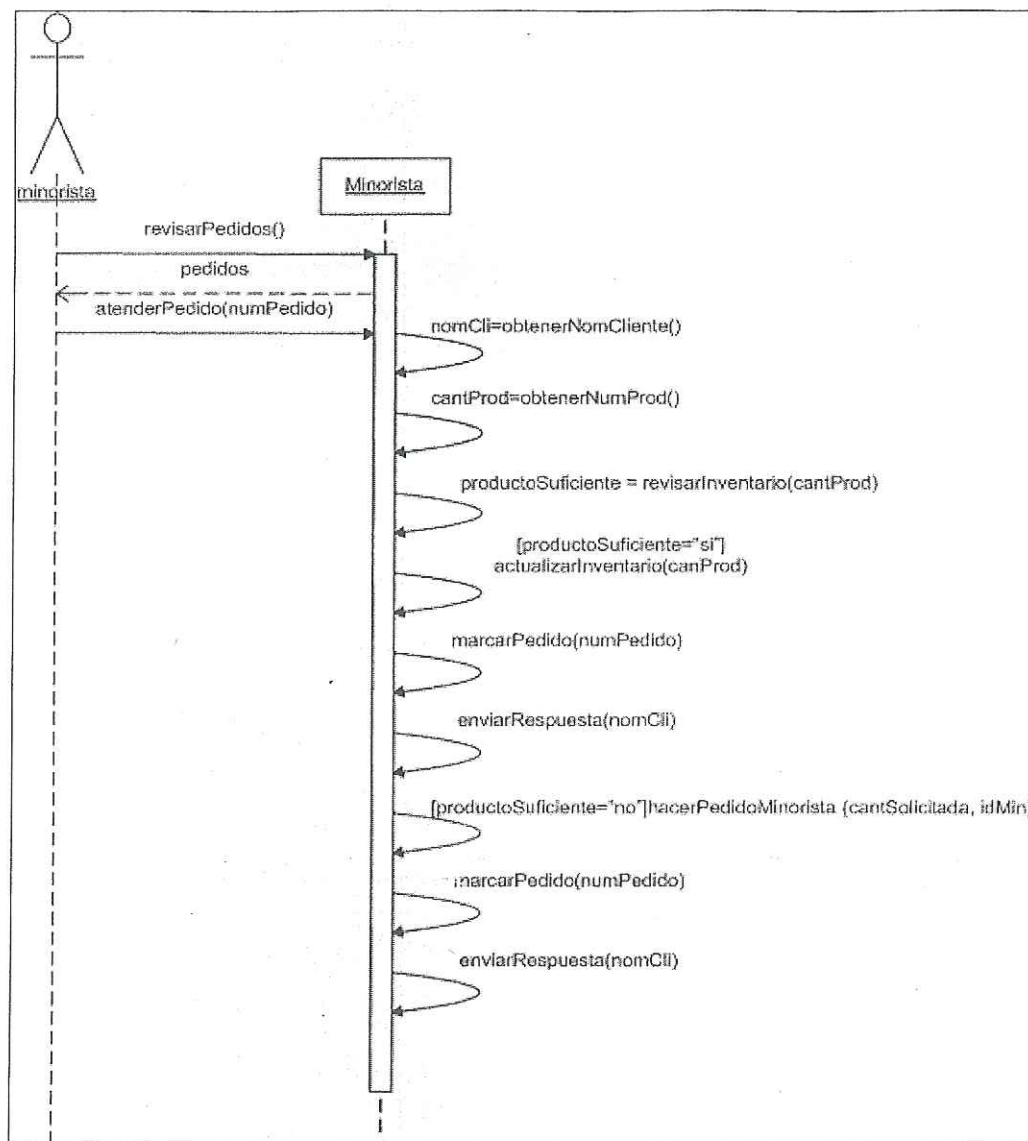


Figura 28. Diagrama de secuencia *Surtir pedido a cliente*.

Por otra parte, si la cantidad de producto existente en el almacén no es suficiente para cubrir la demanda del cliente, se hace un pedido al mayorista por medio del método *hacerPedidoMayorista(cantSolicitada, idMin)*, se etiqueta el pedido como atendido y se envía la respuesta al cliente.



### IV.3.2 Implementación del prototipo del proceso del juego de la cerveza.

De acuerdo a lo mencionado en este capítulo, se utilizó CORBA, JAVA y XML como tecnología de desarrollo. En esta sección se muestra la forma en como fue utilizada esta tecnología en la implementación del prototipo.

```

module BeerGame
{
    interface NodoDistribuidor
    {
        typedef string actividades[50][5];
        typedef string roles[8];
        typedef string datos[20][3];
        typedef string actores[5][6];
        roles validacion(in string usuario, in string contraseña);
        actividades retornaWorkflow(in string rol);
        datos retornaDatos(in string rol);
        actores retornaActores ();
    }
};

```

Figura 29. IDL del Nodo Distribuidor.

En cada nodo del proceso se sitúa uno o más objetos CORBA, donde cada uno de estos objetos representa un rol. El IDL para el objeto *NodoDistribuidor* (figura 29) contiene los siguientes métodos:

- *validación*.- Es el método mediante el cual el *Nodo Distribuidor* determina si el actor puede utilizar el sistema o no está autorizado para ello. El método recibe como parámetros de entrada el nombre de cuenta del usuario y la contraseña, y en caso de que éstas sean correctas, el método retorna los roles que desempeña el usuario.
- *retornaWorkflow*.- Por medio de este método el sistema envía a los nodos distribuidos las actividades que le corresponde realizar. El método recibe como parámetro el nombre del rol del cual se desea obtener el flujo de las actividades.

- *retornaActores*.- La información relacionada a los actores que participan en el proceso es enviada al nodo distribuido cuando este método es invocado. No recibe ningún parámetro de entrada.

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<!DOCTYPE Bitacora (View Source for full doctype...)>
- <actores>
- <actor nombrePersonal="Abdel Mejía" nombreCuenta="mejiam" contraseña="asd" status="false" identificador="1">
  <rol nombreRol="cliente" />
</actor>
- <actor nombrePersonal="Lizbeth Escobedo" nombreCuenta="escobedo" contraseña="asd" status="false" identificador="2">
  <rol nombreRol="minorista" />
</actor>
- <actor nombrePersonal="Ana Martínez" nombreCuenta="martinea" contraseña="asd" status="false" identificador="3">
  <rol nombreRol="mayorista" />
</actor>
</actores>
```

Figura 30. Directorio de actores utilizado en el prototipo del juego de la cerveza.

El proceso que se sigue al momento que un actor se da de alta en el sistema es el siguiente:

El actor envía sus datos al *Nodo Distribuidor*, los cuales son recibidos por el método *validacion(in string usuario, in string contraseña)* nodo distribuidor y éste revisa en el *directorio de actores* (figura 30) si los datos enviados por el actor coinciden con alguna entrada de este documento.

Si los datos coinciden, se obtiene el nombre de los roles que desempeña el actor y se obtienen las actividades que corresponden a cada uno de estos roles. Las actividades son obtenidas del *documento de workflow* (figura 31) y son enviadas al nodo que las solicitó.

Además, se envía la información contenida en el *Directorio de actores* (figura 29). Una vez enviada esta información, el nodo distribuido está listo para comenzar la ejecución de los procesos que le corresponden.

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<!DOCTYPE workflow (View Source for full doctype...)>
- <workflow>
+ <estadosRol nombreRol="Cliente">
- <estadosRol nombreRol="minorista">
- <estado idEstado="mn1" nombreActividad="Iniciando inventario" nivel="1" estadoCompuesto="false" pantalla="p1" perteneceA="">
  <transicion estadoSiguiente="mn2" />
</estado>
- <estado idEstado="mn2" nombreActividad="Eligiendo servicio" nivel="1" estadoCompuesto="false" pantalla="p1" perteneceA="">
  <transicion estadoSiguiente="mn3" />
  <transicion estadoSiguiente="mn7" />
</estado>
- <estado idEstado="mn3" nombreActividad="Revisando orden" nivel="1" estadoCompuesto="false" pantalla="p1" perteneceA="">
  <transicion estadoSiguiente="mn4" />
</estado>
- <estado idEstado="mn4" nombreActividad="Revisando inventario" nivel="1" estadoCompuesto="false" pantalla="p1" perteneceA="">
  <transicion estadoSiguiente="mn5" />
</estado>
- <estado idEstado="mn5" nombreActividad="Actualizando inventario" nivel="1" estadoCompuesto="false" pantalla="p1" perteneceA="">
  <transicion estadoSiguiente="mn6" />
</estado>
- <estado idEstado="mn6" nombreActividad="Escribiendo una respuesta" nivel="1" estadoCompuesto="false" pantalla="p1" perteneceA="">
  <transicion estadoSiguiente="cmt" />
</estado>
- <estado idEstado="mn7" nombreActividad="Escribiendo una respuesta" nivel="1" estadoCompuesto="false" pantalla="p1" perteneceA="">
  <transicion estadoSiguiente="mn8" />
</estado>
- <estado idEstado="mn8" nombreActividad="enviando orden" nivel="1" estadoCompuesto="false" pantalla="p1" perteneceA="">
  <transicion estadoSiguiente="mn9" />
</estado>
- <estado idEstado="mn9" nombreActividad="Esperando una respuesta" nivel="1" estadoCompuesto="false" pantalla="p1" perteneceA="">
  <transicion estadoSiguiente="mn10" />
</estado>
- <estado idEstado="mn10" nombreActividad="Obteniendo una respuesta" nivel="1" estadoCompuesto="false" pantalla="p1" perteneceA="">
  <transicion estadoSiguiente="cmt" />
</estado>
- <estado idEstado="cmt" nombreActividad="Finalizando rol" nivel="1" estadoCompuesto="false" pantalla="p1" perteneceA="">
  <transicion estadoSiguiente="cmt" />
</estado>
</estadosRol>
+ <estadosRol nombreRol="mayorista">
</workflow>
```

Figura 31. Workflow utilizado en el prototipo del juego de la cerveza.

Por otra parte, en los nodos distribuidos se lleva a cabo la ejecución de las actividades del proceso. Para este fin, se definen distintos objetos CORBA, los cuales representan un rol, y

contienen los métodos que serán invocados remotamente por algunos otros actores del proceso. El IDL utilizado para definir estos objetos se muestra en la figura 32:

```
module BeerGame
{
    interface Minorista
    {
        void obtenerSolicitud(in string idCliente, in long numProd);
        boolean monitorearPedido(in long idPedido);
        boolean darRespuesta(in long idPedido);
    };

    interface Mayorista
    {
        void obtenerSolicitud(in string idMinorista, in long numProd);
        boolean monitorearPedido(in long idPedido);
        boolean darRespuesta(in long idPedido);
    };
};
```

Figura 32. IDL de los objetos distribuidos en el prototipo del juego de la cerveza.

En el caso de los métodos de la interfaz Minorista, éstos serán invocados remotamente por una aplicación del rol Cliente, por lo que la funcionalidad será la siguiente:

- *obtenerSolicitud*.- Por medio de este método el Cliente pasará al Minorista los datos de la hoja de pedido (actividad enviar orden).
- *monitorearPedido*.- Este método tiene como responsabilidad permitir al Cliente revisar si el pedido ha sido atendido (actividad esperar respuesta).
- *darRespuesta*.- El Cliente conocerá si el pedido realizado pudo ser surtido o no a través de éste método.



En lo que respecta a los métodos de la interfaz Mayorista, serán invocados remotamente por una aplicación del rol Minorista, y la funcionalidad es igual a la que presentan los métodos de la interfaz Minorista.

Puesto que sólo estos métodos serán accesados remotamente, sólo ellos son incluidos en el IDL del proceso del juego de la cerveza. Los métodos restantes que se utilizaron en el prototipo fueron declarados localmente en la aplicación que les corresponde.

En lo que respecta a la parte transaccional del prototipo, la implementación se hizo en base al modelo de transacciones flexibles y para ello se hizo uso de una bitácora con las características especificadas en la sección 2 de este capítulo y dicha implementación será mostrada de acuerdo al diagrama de secuencia del caso de uso *Surtir producto a cliente*.

Asimismo, acorde al flujo de trabajo de actor *minorista*, al iniciar este caso de uso la instancia del proceso debe encontrarse en el estado *mn3*, lo que indica que la siguiente actividad a realizarse es *Revisando orden* (figura 33).

```
- <estado idEstado="mn3" nombreActividad="Revisando orden" nivel="1" estadoCompuesto="false" pantalla="p1" perteneceA="">
  <transicion estadoSiguiente="mn4" />
</estado>
```

Figura 33. Datos del estado *mn3* del workflow del proceso del juego de la cerveza.

Cuando el actor comienza a ejecutar la actividad, esta transacción es registrada y al finalizar la ejecución de esta actividad, la bitácora tendría registradas las transacciones que se aprecian en la tabla XXI, además de las transacciones previamente realizadas.

Tabla XXI. Tabla que ejemplifica la bitácora después de la ejecución del estado mn3.

| Instancia | Estado | Status     | Nivel | Act paralelas | Subtransaccion |
|-----------|--------|------------|-------|---------------|----------------|
| 001       | mn3    | ejecutando | 1     | false         | no             |
| 001       | mn3    | finalizado | 1     | false         | no             |

Además, al finalizar la actividad, el estado actual para esta instancia será *mn4* en base a lo indicado en los datos del flujo de trabajo que corresponden a estado *mn3* (figura 32), por lo que el minorista puede elegir ejecutar la actividad. Posteriormente, cuando el *minorista* decida ejecutar la actividad *Revisando inventario* (figura 33), la bitácora tendrá registrados los datos mostrados en la tabla XXII.

Tabla XXII. Tabla que ejemplifica la bitácora después de la ejecución del estado mn4.

| Instancia | Estado | Status     | Nivel | Act paralelas | Subtransaccion |
|-----------|--------|------------|-------|---------------|----------------|
| 001       | mn3    | ejecutando | 1     | false         | no             |
| 001       | mn3    | finalizado | 1     | false         | no             |
| 001       | mn4    | ejecutando | 1     | false         | no             |
| 001       | mn4    | finalizado | 1     | false         | no             |

Asimismo, después de la ejecución de esta actividad, el estado actual de la instancia será *mn5* (figura 34).

```
- <estado idEstado="mn4" nombreActividad="Revisando inventario" nivel="1" estadoCompuesto="false" pantalla="p1" perteneceA="">
  <transicion estadoSiguiente="mn5" />
</estado>
```

Figura 34. Datos del estado *mn4* del workflow del proceso del juego de la cerveza.

```
- <estado idEstado="mn5" nombreActividad="Actualizando inventario" nivel="1" estadoCompuesto="false" pantalla="p1" perteneceA="">
  <transicion estadoSiguiente="mn6" />
</estado>
```

Figura 35. Datos del estado *mn5* del workflow del proceso del juego de la cerveza.

Cuando el actor decida ejecutar la actividad *Escribiendo respuesta* (figura 35), la bitácora tendrá almacenados los datos mostrados en la tabla XXIII.

Tabla XXIII. Tabla que ejemplifica la bitácora después de la ejecución del estado *mn5*.

| Instancia | Estado | Status     | Nivel | Act paralelas | Subtransaccion |
|-----------|--------|------------|-------|---------------|----------------|
| 001       | mn3    | ejecutando | 1     | false         | no             |
| 001       | mn3    | finalizado | 1     | false         | no             |
| 001       | mn4    | ejecutando | 1     | false         | no             |
| 001       | mn4    | finalizado | 1     | false         | no             |
| 001       | mn5    | ejecutando | 1     | false         | no             |
| 001       | mn5    | finalizado | 1     | false         | no             |

Finalmente, una vez que se ha escrito la respuesta al cliente, la instancia pasa al estado *commit*, indicando que esa instancia ha finalizado. Al final, la bitácora tendrá registrados los datos mostrados en la tabla XXIV.

Tabla XXIV. Tabla que ejemplifica la bitácora después de la ejecución del estado finalizado.

| Instancia | Estado   | Status     | Nivel | Act paralelas | Subtransaccion |
|-----------|----------|------------|-------|---------------|----------------|
| 001       | mn3      | ejecutando | 1     | false         | no             |
| 001       | mn3      | finalizado | 1     | false         | no             |
| 001       | mn4      | ejecutando | 1     | false         | no             |
| 001       | mn4      | finalizado | 1     | false         | no             |
| 001       | mn5      | ejecutando | 1     | false         | no             |
| 001       | mn5      | finalizado | 1     | false         | no             |
| 001       | finaliza | ejecutando | 1     | false         | no             |
| 001       | finaliza | finalizado | 1     | false         | no             |

En la siguiente sección se muestran los aspectos de implementación del mecanismo de recuperación de errores en el prototipo del juego de la cerveza.

#### **IV.3.3. Implementación del mecanismo de recuperación de errores del prototipo del proceso del juego de la cerveza.**

En el capítulo anterior se presentaron algunos aspectos sobre la importancia de contar con un mecanismo de recuperación de errores en un sistema de workflow, así como también se presentó el modelo de transacciones flexibles, el cual al ser implementado brinda soporte a la recuperación del estado en el cual se encontraba el proceso antes de un error, en caso de que éste sucediera.



La recuperación de errores en este prototipo consiste en revisar las entradas de la bitácora (tabla XXIV) y tomar la que represente mayor avance de cada instancia. Para este fin, se dice que un estado está más avanzado mientras más grande es el número que identifica al estado, en este caso mn5 representa mayor avance que mn3, presentándose como caso especial el estado *finaliza*, el cual indica que la instancia ha finalizado, y por tanto, es el estado que presenta mayor avance .

Tabla XXV. Bitácora después de la ejecución de una instancia del caso de uso *Surtir producto a cliente*.

| Instancia | Estado   | Status     | Nivel | Act paralelas | Subtransaccion |
|-----------|----------|------------|-------|---------------|----------------|
| 001       | mn3      | ejecutando | 1     | false         | no             |
| 001       | mn3      | finalizado | 1     | false         | no             |
| 001       | mn4      | ejecutando | 1     | false         | no             |
| 001       | mn4      | finalizado | 1     | false         | no             |
| 001       | mn5      | ejecutando | 1     | false         | no             |
| 001       | mn5      | finalizado | 1     | false         | no             |
| 001       | finaliza | ejecutando | 1     | false         | no             |
| 001       | finaliza | finalizado | 1     | false         | no             |

En caso de que 2 entradas de la misma instancia coincidan en el estado, se considera como más avanzada el status *finalizado* que el status *ejecutando*. Por lo tanto, si la recuperación de errores se hiciera sobre la bitácora del caso de uso *Surtir producto a cliente* (tabla XXV), el resultado sería que la instancia ha finalizado (tabla XXVI).

Tabla XXVI. Tabla con entrada de transacción más avanzada.

| Instancia | Estado   | Status     | Nivel | Act paralelas | Subtransaccion |
|-----------|----------|------------|-------|---------------|----------------|
| 001       | finaliza | finalizado | 1     | false         | no             |

En la siguiente sección se muestran los aspectos de implementación del prototipo del proceso de reclutamiento, selección y contratación de personal.

#### ***IV.4 Prototipo del proceso de reclutamiento, selección y contratación de personal en CICESE.***

Este proceso al que se le da soporte en la implementación de este prototipo tiene como finalidad analizar las vacantes que se generan en CICESE y contratar al candidato más apto para cubrir dicha vacante. Cada vez que se genera una vacante, se revisa detalladamente la información de los candidatos y se determina qué tipos de evaluaciones serán aplicadas de acuerdo al tipo de vacante que surgió. Una vez que se obtiene la información necesaria para que las personas involucradas puedan determinar cuál de los candidatos es quien cumple con los requisitos necesarios para ocupar el puesto, de acuerdo a las políticas de la institución.

El prototipo tiene como finalidad probar que la funcionalidad requerida en los sistemas de workflow distribuidos si puede ser cumplida. La figura 36 muestra el diagrama RAD del proceso reclutamiento, selección y contratación de personal en CICESE, en el cual se puede apreciar el flujo que siguen las actividades del proceso y las interacciones entre ellos. En este diagrama se puede apreciar claramente que participan seis roles en el proceso:



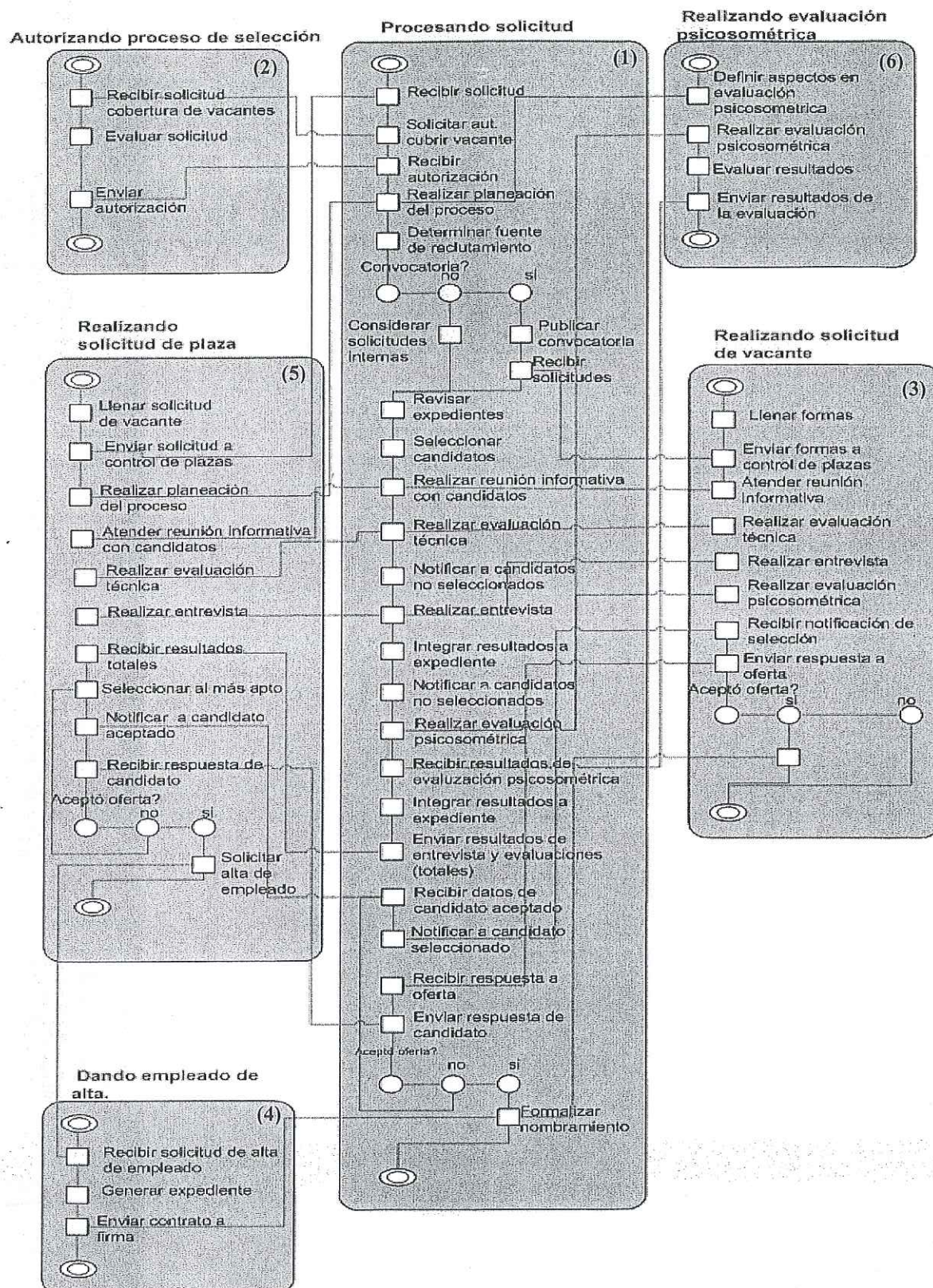


Figura 36. Diagrama RAD del proceso reclutamiento, selección y contratación de personal.

1. *Jefe de departamento de control de plazas.*- Tiene como responsabilidad coordinar el proceso.
2. *Director Administrativo.*- Se encarga de autorizar la cobertura de plazas, así como de supervisar el proceso.
3. *Candidato.*- Es aquella persona que hace llegar sus datos y documentos al departamento de control de plazas, con el fin de ser tomado en cuenta durante el proceso.
4. *Jefe de oficina.*- Se encarga de dar seguimiento al proceso de nombramiento, como enviar el contrato al candidato seleccionado y generar el expediente del nombramiento.
5. *Jefe de área.*- Es quien solicita recursos humanos para cubrir la vacante que se ha generado en su área.
6. *Psicólogo.*- Determina el contenido de la evaluación psicosométrica, aplica esta evaluación a los candidatos y envía los resultados al departamento de control de plazas.

El proceso es comenzado por el *jefe de área (JA)*, quien se encarga de avisar al *departamento de control de plazas (DCP)* que tiene una vacante en su departamento, para que éste a su vez se encargue de pedirle al *director administrativo (DA)* la aprobación para contratar personal que llegue a cubrir dicha vacante.

Una vez que el *DA* ha autorizado que la vacante sea cubierta, el *DCP* se encarga de coordinar el proceso. Para ello debe hacer una planeación del proceso conjuntamente con el



*DA* y con el *Psicólogo*, y posteriormente, el mismo *DCP* en conjunto con el *DA*, deben hacer una serie de evaluaciones a los *candidatos* para determinar cuales candidatos son los que cumplen los requisitos necesarios para cubrir el puesto y enviarle esta información al *DA*, quien se encargará de elegir al candidato que ocupará la vacante.

En lo referente a la interacción entre nodos, la figura 36 muestra también interacciones entre los roles del proceso. Este proceso tiene algunas actividades que requieren interacción entre tres roles, por lo que se debe tener en cuenta que dichas actividades deben esperar a que todos los actores involucrados cumplan las tareas anteriores al punto de interacción con el fin de cumplir con dicha actividad.

En la siguiente sección se presenta el análisis del proceso del juego de reclutamiento, selección y contratación de personal mediante diagramas UML.

#### **IV.4.1 Análisis del prototipo del proceso de reclutamiento, selección y contratación de personal.**

En el análisis del proceso de reclutamiento, selección y contratación de personal se encontraron los siguientes casos de uso:

- Solicitar cobertura de vacante
- Análisis de la vacante
- Reclutar candidatos
- Evaluar y seleccionar candidato
- Dar de alta a empleado

En esta sección se describe la funcionalidad el caso de uso *Solicitar cobertura de vacante* (figura 37), debido a que éste describe de forma completa parte de la funcionalidad del prototipo

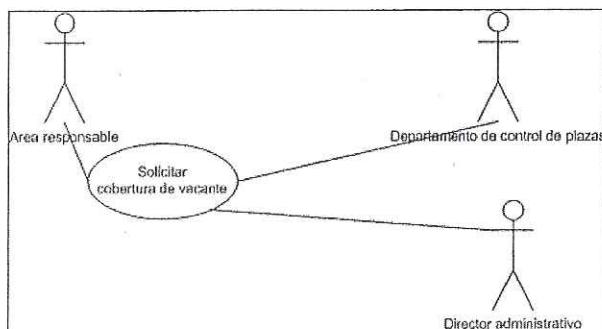


Figura 37. Diagrama del caso de uso *Solicitar cobertura de vacante*.

En el caso de uso *Solicitar cobertura de vacante* intervienen los actores *Responsable de área (RA)*, *Director de departamento de control de plazas (DCP)* y *Director Administrativo (DA)* con la finalidad de que se obtenga la autorización para cubrir la vacante existente.

### Descripción.

El caso de uso inicia cuando el *RA* necesita personal para cubrir una vacante en su área, por lo que tiene que enviar una solicitud al *DCP*, quien a su vez se encarga de pasarla al *DA*. El *DA* revisa la solicitud, la autoriza y el caso de uso termina cuando el *DCP* recibe la autorización de contratar personal para cubrir la vacante.

El diagrama de secuencia del caso de uso es mostrado en la figura 38. En este diagrama, el *Responsable de área* llena la solicitud de vacante, y la envía al *Jefe del departamento de control de plazas*, quien a su vez se encarga de enviarla al *Director administrativo* para que

este evalúe la solicitud de plaza y autorice que el proceso de reclutamiento, selección y contratación de personal sea iniciado.

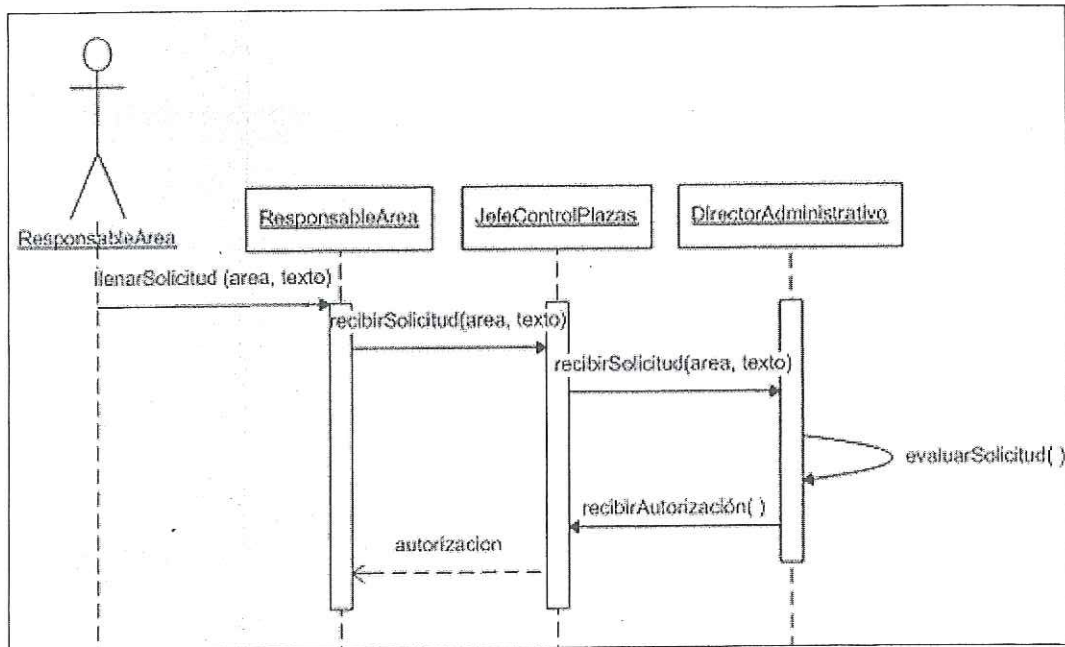


Figura 38. Diagrama de secuencia del caso de uso *Solicitar cobertura de vacante*.

En la siguiente sección se muestra la implementación de este caso de uso.

#### IV.4.2 Implementación del prototipo del proceso de reclutamiento, selección y contratación de personal.

La implementación de este prototipo es muy similar a la que se realizó en el prototipo del proceso del juego de la cerveza, con la diferencia que a este prototipo se le agregaron algunas características que permiten que los procesos sean llevados a cabo de forma similar a como se llevan a cabo usualmente de forma natural.

Dentro de las modificaciones que se le hicieron al *nodo distribuidor* (figura 39) está el hecho de que cuando un usuario finaliza sesión en el nodo distribuido, la información que contiene este nodo es enviada al *nodo distribuidor*, por lo que debe ser enviada nuevamente al nodo distribuido cuando el actor al que pertenece dicha información inicia sesión. Además, cada vez que una transacción se ejecuta en el nodo distribuido debe ser registrada en dicho nodo y a la vez en el *nodo distribuidor*.

```

module Wodisflow
{
    interface NodoDistribuidor
    {
        typedef string actividades[50][5];
        typedef string roles[8];
        typedef string datos[20][3];
        typedef string actores[5][6];
        roles validacion(in string usuario, in string contraseña);
        actividades retornaWorkflow(in string rol);
        datos retornaDatos(in string rol);
        actores retornaActores ();
        void actualizaActores(in string usuario, in string status);
        /**Métodos agregados con respecto al prototipo anterior
        typedef string tareasPendientes[50][5];
        typedef string eBitacora[1000][7];
        typedef string rBitacora[1000][8];
        tareasPendientes retornaTareasPendientes(in string usuario);
        rBitacora retornaBitacora(in string usuario);
        void escribeBitacora(in eBitacora transacción);
        void escribeTareasPendientes(in tareasPendientes tareas);
    };
};

```

Figura 39. IDL del nodo distribuidor en el prototipo del proceso de reclutamiento, selección y contratación de personal.

El método *retornaTareasPendientes(in string usuario)* envía al nodo distribuido las notificaciones pendientes que tiene el usuario y que no ha recibido, el método *retornaBitacora(in string usuario)* retorna las transacciones que ha ejecutado el usuario, el método *escribeBitacora(in eBitacora transacción)* registra en la bitácora global las



transacciones que han realizado los usuarios y el método *escribeTareasPendientes*(in *tareasPendientes tareas*) escribe las tareas que no han podido ser notificadas a un actor.

Por otra parte, el IDL que define los objetos de los *nodos distribuidos* (figura 40) que participan en este caso de uso presenta varios métodos que son utilizados a lo largo de la ejecución de todo el proceso, sin embargo, para este caso de uso solo son útiles algunos de ellos.

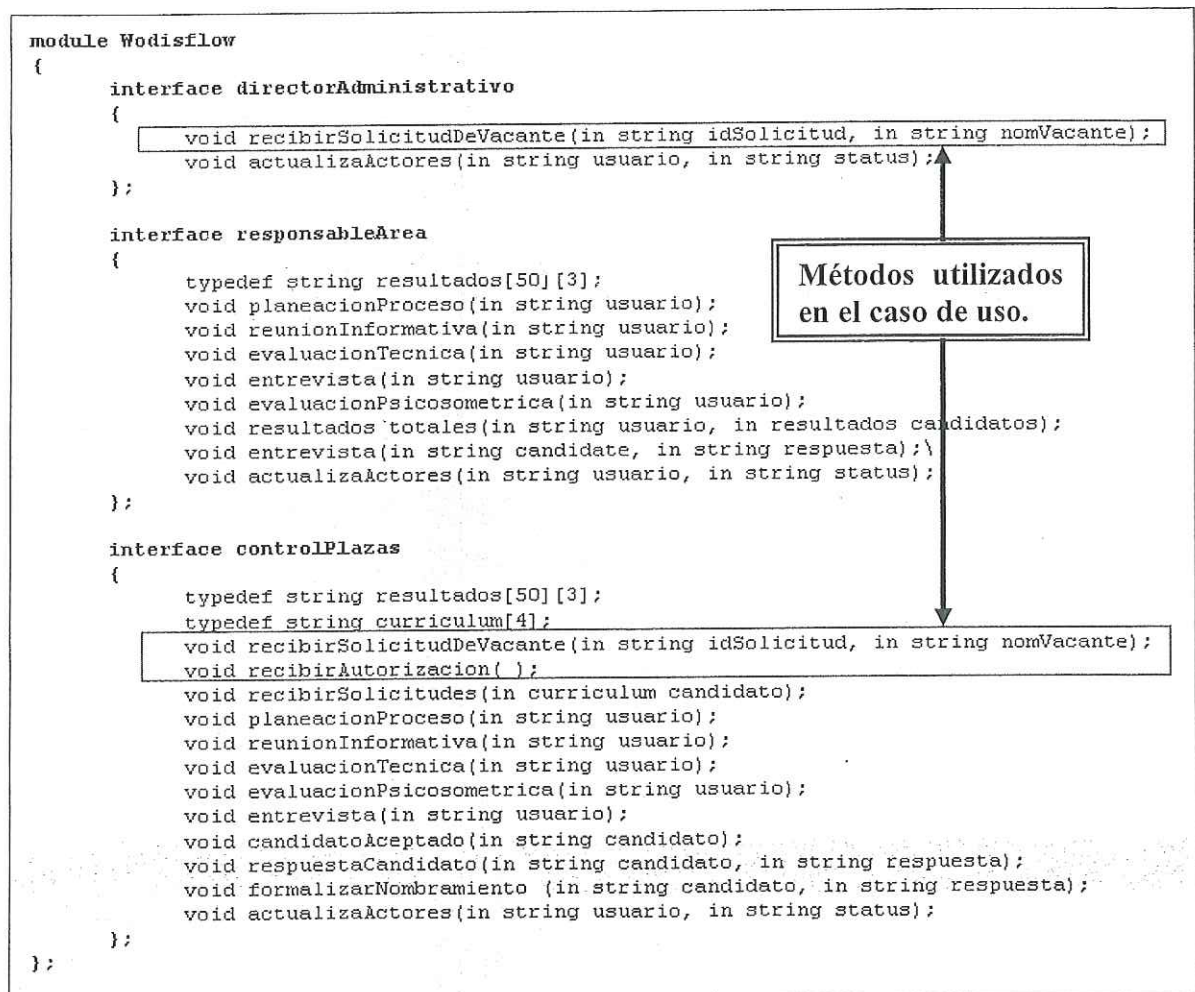


Figura 40. IDL de algunos nodos distribuidos en el prototipo de reclutamiento, selección y contratación de personal.

En el caso del actor que desempeña el rol *responsableArea*, para este caso de uso solo hace invocaciones remotas a los métodos del rol *controlPlazas*, en específico el método *recibirSolicitudDeVacante(in string idSolicitud, in string nomVacante)*, mediante el cual el actor que desempeña este rol obtiene la solicitud, y la envía al *directorAdministrativo*, quien evalúa la solicitud y envía la autorización al rol *controlPlazas* mediante el método *recibirAutorización()* y con esto el caso de uso finaliza.

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<!DOCTYPE workflow (View Source for full doctype...)>
- <workflow>
- <estadosRol nombreRol="responsableArea">
- <estado idEstado="Ini" nombreActividad="Estado Inicial" nivel="1" estadoCompuesto="false" pantalla="p1" perteneceA="--">
  <transicion estadoSiguiente="RA1" />
</estado>
- <estado idEstado="RA1" nombreActividad="Llenar solicitud de vacante" nivel="1" estadoCompuesto="false" pantalla="p1" perteneceA="--">
  <transicion estadoSiguiente="RA2" />
</estado>
- <estado idEstado="RA2" nombreActividad="Enviar solicitud a control de plazas" nivel="1" estadoCompuesto="false" pantalla="p1" perteneceA="--">
  <transicion estadoSiguiente="RA3" />
</estado>
- <estado idEstado="RA3" nombreActividad="Enviando solicitud a control de plazas" nivel="1" estadoCompuesto="false" pantalla="p1" perteneceA="--">
  <transicion estadoSiguiente="RA4" />
  <comunicarseCon rol="Procesando solicitud" />
</estado>
</estadosRol>
</workflow>
```

Figura 41. Flujo de actividades del rol *responsableArea* para el caso de uso *Solicitar cobertura de vacante*.

La implementación en forma transaccional de este prototipo es igual a la presentada en el prototipo del proceso del juego de la cerveza, solo que en este prototipo se agregó la característica de que las transacciones también se guardarían en una *bitácora global* en el *nodo distribuidor*.

Al finalizar la ejecución de las actividades rol *responsableArea* (figura 41), para este caso de uso, la bitácora quedará de acuerdo a la tabla XXVII.

Tabla XXVII. Bitácora del rol *responsableArea* en el nodo distribuido, después de la ejecución de una instancia del caso de uso *Solicitar cobertura de vacante*.

| Instancia | Estado | Status     | Nivel | Act paralelas | Subtransaccion |
|-----------|--------|------------|-------|---------------|----------------|
| 001       | RA1    | ejecutando | 1     | false         | no             |
| 001       | RA1    | finalizado | 1     | false         | no             |
| 001       | RA2    | ejecutando | 1     | false         | no             |
| 001       | RA2    | finalizado | 1     | false         | no             |
| 001       | RA3    | ejecutando | 1     | false         | no             |
| 001       | RA3    | finalizado | 1     | false         | no             |

```

<?xml version="1.0" encoding="iso-8859-1" ?>
<!DOCTYPE workflow (View Source for full doctype...)>
- <workflow>
- <estadosRol nombreRol="controlPlazas">
- <estado idEstado="Ini" nombreActividad="Estado Inicial" nivel="1" estadoCompuesto="false" pantalla="p1" perteneceA="--">
  <transicion estadoSiguiente="PS1" />
</estado>
- <estado idEstado="PS1" nombreActividad="Esperando recibir solicitud" nivel="1" estadoCompuesto="false" pantalla="p1" perteneceA="--">
  <transicion estadoSiguiente="PS2" />
</estado>
- <estado idEstado="PS2" nombreActividad="Recibir solicitud" nivel="1" estadoCompuesto="false" pantalla="p1" perteneceA="--">
  <transicion estadoSiguiente="PS3" />
</estado>
- <estado idEstado="PS3" nombreActividad="Solicitar autorización para cubrir vacante" nivel="1" estadoCompuesto="false" pantalla="p1" perteneceA="--">
  <transicion estadoSiguiente="PS4" />
  <comunicarseCon rol="Autorizando proceso de selección" />
</estado>
- <estado idEstado="PS4" nombreActividad="Esperando recibir autorización" nivel="1" estadoCompuesto="false" pantalla="p1" perteneceA="--">
  <transicion estadoSiguiente="PS5" />
</estado>
- <estado idEstado="PS5" nombreActividad="Recibir autorización" nivel="1" estadoCompuesto="false" pantalla="p1" perteneceA="--">
  <transicion estadoSiguiente="PS6" />
</estado>
</estadosRol>
</workflow>

```

Figura 42. Flujo de actividades del rol *controlPlazas* para el caso de uso *Solicitar cobertura de vacante*.



Asimismo, cuando el rol *controlPlazas* (figura 42) finalice la ejecución de las actividades para este caso de uso, en la bitácora tendrá almacenados los datos mostrados en la tabla XVIII.

Tabla XVIII. Bitácora del rol *controlPlazas* en el nodo distribuido, después de la ejecución de una instancia del caso de uso *Solicitar cobertura de vacante*.

| Instancia | Estado | Status     | Nivel | Act paralelas | Subtransaccion |
|-----------|--------|------------|-------|---------------|----------------|
| 001       | PS1    | ejecutando | 1     | false         | no             |
| 001       | PS1    | finalizado | 1     | false         | no             |
| 001       | PS2    | ejecutando | 1     | false         | no             |
| 001       | PS2    | finalizado | 1     | false         | no             |
| 001       | PS3    | ejecutando | 1     | false         | no             |
| 001       | PS3    | finalizado | 1     | false         | no             |
| 001       | PS4    | ejecutando | 1     | false         | no             |
| 001       | PS4    | finalizado | 1     | false         | no             |
| 001       | PS5    | ejecutando | 1     | false         | no             |
| 001       | PS5    | finalizado | 1     | false         | no             |

En lo que respecta al rol *directorAdministrativo* (figura 43), la bitácora habrá registrado los datos que se muestran en la tabla XXIX, después de que éste haya concluido las actividades que corresponden a este caso de uso.



```

<?xml version="1.0" encoding="iso-8859-1" ?>
<!DOCTYPE workflow (View Source for full doctype...)>
- <workflow>
- <estadosRol nombreRol="directorAdministrativo">
- <estado idEstado="Ini" nombreActividad="Estado Inicial" nivel="1" estadoCompuesto="false" pantalla="p1" perteneceA="--">
  <transicion estadoSiguiente="AS1" />
</estado>
- <estado idEstado="AS1" nombreActividad="Esperando recibir solicitud de cobertura de vacantes" nivel="1" estadoCompuesto="false" pantalla="p1" perteneceA="--">
  <transicion estadoSiguiente="AS2" />
</estado>
- <estado idEstado="AS2" nombreActividad="Recibir solicitud de cobertura de vacantes" nivel="1" estadoCompuesto="false" pantalla="p1" perteneceA="--">
  <transicion estadoSiguiente="AS2" />
</estado>
- <estado idEstado="AS3" nombreActividad="Evaluar solicitud" nivel="1" estadoCompuesto="false" pantalla="p1" perteneceA="--">
  <transicion estadoSiguiente="AS4" />
</estado>
- <estado idEstado="AS4" nombreActividad="Enviar autorización" nivel="1" estadoCompuesto="false" pantalla="p1" perteneceA="--">
  <transicion estadoSiguiente="cmt" />
  <comunicarseCon rol="Procesando solicitud" />
</estado>
- <estado idEstado="cmt" nombreActividad="Fin del proceso" nivel="1" estadoCompuesto="false" pantalla="p1" perteneceA="--">
  <transicion estadoSiguiente="--" />
</estado>
</estadosRol>
</workflow>

```

Figura 43. Flujo de actividades del rol *directorAdministrativo* para el caso de uso *Solicitar cobertura de vacante*.

Tabla XXIX. Bitácora del rol *directorAdministrativo* en el nodo distribuido, después de la ejecución de una instancia del caso de uso *Solicitar cobertura de vacante*.

| Instancia | Estado | Status     | Nivel | Act paralelas | Subtransaccion |
|-----------|--------|------------|-------|---------------|----------------|
| 001       | AS1    | ejecutando | 1     | false         | no             |
| 001       | AS1    | finalizado | 1     | false         | no             |
| 001       | AS2    | ejecutando | 1     | false         | no             |
| 001       | AS2    | finalizado | 1     | false         | no             |
| 001       | AS3    | ejecutando | 1     | false         | no             |
| 001       | AS3    | finalizado | 1     | false         | no             |
| 001       | AS4    | ejecutando | 1     | false         | no             |
| 001       | AS4    | finalizado | 1     | false         | no             |

Tabla XXX. Bitácora global en el nodo Distribuidor, después de la ejecución de una instancia del caso de uso *Solicitar cobertura de vacante*.

| Instancia | Estado | Status     | Nivel | Act paralelas | Subtransaccion |
|-----------|--------|------------|-------|---------------|----------------|
| 001       | RA1    | ejecutando | 1     | false         | no             |
| 001       | RA1    | finalizado | 1     | false         | no             |
| 001       | RA2    | ejecutando | 1     | false         | no             |
| 001       | RA2    | finalizado | 1     | false         | no             |
| 001       | RA3    | ejecutando | 1     | false         | no             |
| 001       | RA3    | finalizado | 1     | false         | no             |
| 001       | PS1    | ejecutando | 1     | false         | no             |
| 001       | PS1    | finalizado | 1     | false         | no             |
| 001       | PS2    | ejecutando | 1     | false         | no             |
| 001       | PS2    | finalizado | 1     | false         | no             |
| 001       | PS3    | ejecutando | 1     | false         | no             |
| 001       | PS3    | finalizado | 1     | false         | no             |
| 001       | PS4    | ejecutando | 1     | false         | no             |
| 001       | AS1    | ejecutando | 1     | false         | no             |
| 001       | AS1    | finalizado | 1     | false         | no             |
| 001       | AS2    | ejecutando | 1     | false         | no             |
| 001       | AS2    | finalizado | 1     | false         | no             |
| 001       | AS3    | ejecutando | 1     | false         | no             |
| 001       | AS3    | finalizado | 1     | false         | no             |
| 001       | AS4    | ejecutando | 1     | false         | no             |
| 001       | AS4    | finalizado | 1     | false         | no             |
| 001       | PS4    | finalizado | 1     | false         | no             |
| 001       | PS5    | ejecutando | 1     | false         | no             |
| 001       | PS5    | finalizado | 1     | false         | no             |

Finalmente, se muestran los datos que tendrá almacenada la *bitácora global* (tabla XXX) después de que la ejecución del caso de uso finalice. Como se puede apreciar en la bitácora global (tabla XXX), las transacciones de algunos roles no están almacenadas en forma continua, y esto es debido a que las transacciones se van almacenando de acuerdo a como se van realizando.

#### **IV.4.3. Implementación del mecanismo de recuperación de errores del prototipo del proceso reclutamiento, selección y contratación de personal.**

El mecanismo de recuperación de fallas se implementó de manera similar a como se hizo en el prototipo del juego de la cerveza, solo que en este caso se debe de tomar en cuenta los datos almacenados en la *bitácora global*. Lo que cambia con respecto a dicho prototipo, es que en este prototipo se busca el estado actual de cada instancia tanto en la *bitácora local* como en la *bitácora global*. Esta búsqueda produce dos tablas con estados de las instancias, por lo que se hace una sola tabla con los registros de ambas tablas y se vuelve a aplicar el mismo criterio de búsqueda de estado actual de cada instancia.

En la siguiente sección se presentan las pruebas que se hicieron a ambos prototipos.

### ***IV.5 Pruebas al prototipo.***

La verificación y validación (V & V) es el nombre que se le da a los procesos de comprobación y análisis que aseguran que el software esté acorde con su especificación y cumpla con los requerimientos establecidos. La validación consiste en saber si se está construyendo el sistema correcto, es decir, si los requerimientos obtenidos realmente coinciden con las necesidades de los usuarios, y la verificación consiste en saber si el sistema se ha construido de forma correcta, o dicho de otra manera, verificar que los requerimientos obtenidos estén siendo reflejados en el sistema desarrollado. [Sommerville, 2002]

Existe gran diversidad de pruebas que pueden ser aplicadas a los sistemas de software. Las pruebas de defectos tienen como propósito exponer los defectos latentes antes de entregar el sistema, las pruebas de integración tienen como finalidad detectar problemas resultantes de la interacción de los componentes y las pruebas de estructura son pruebas de caja blanca, donde se conoce el código y la estructura del sistema, y se utilizan para probar los componentes por separado.

A los componentes de la arquitectura se les aplicaron pruebas de estructura, pero se utilizaron como parte del proceso de desarrollo del prototipo para garantizar el funcionamiento adecuado, por lo que los resultados de las pruebas no son relevantes para las conclusiones de este trabajo.



Las pruebas aplicadas fueron de *tipo funcional*, las cuales tienen como fin encontrar las diferencias entre los requerimientos funcionales y el sistema, tienen como finalidad probar la factibilidad tecnológica de construir sistemas de workflow distribuidos basados en la arquitectura propuesta en este trabajo. Las pruebas funcionales son una técnica de caja negra. El objetivo es identificar aquellas situaciones que son susceptibles a fallos.

De acuerdo con la metodología de trabajo utilizada, el prototipo del proceso del juego de la cerveza solo sirvió de base para encontrar errores en la primera iteración, los cuales fueron resueltos en la segunda iteración, por lo que solo se muestran las pruebas del prototipo del proceso de reclutamiento, selección y contratación de personal, el cual implementa el proceso de reclutamiento, selección y contratación de personal.

En esta sección se muestran las pruebas sobre si el sistema ha sido construido de acuerdo a los requerimientos propuestos. Estos requerimientos se explicaron en el capítulo 3 y aquí se mencionan brevemente:

1. *La coordinación de las actividades del proceso debe hacerse de forma descentralizada.*
2. *Las actividades del proceso deben ejecutarse de forma descentralizada.*
3. *La arquitectura debe incluir un mecanismo que permita recuperar al sistema en caso de que ocurran errores.*
4. *Las entidades de información, tales como documentos o datos, deben ser manejados de forma distribuida.*

5. *La arquitectura debe incluir componentes que permitan que los sistemas desarrollados reflejen la forma natural en la cual los procesos son llevados a cabo.*
6. *La arquitectura debe cumplir con los requerimientos básicos de un sistema de Workflow* [Prior, 2002] *como son escalabilidad, recuperación de fallas, disponibilidad, monitoreo, historial y soporte al uso de aplicaciones externas.*

#### **IV.5.1 Pruebas y resultados**

En esta sección se presentan las actividades que se hicieron para probar el cumplimiento de los requerimientos establecidos inicialmente. Dichas actividades consistieron en llevar a cabo el proceso con ayuda del sistema, pero se restringió el número de actores en algunos casos para probar que el sistema funcionaba de manera independiente. Los casos de prueba se muestran en el Apéndice B, por lo que en esta sección solo se muestran el caso de prueba para el primer requisito.

##### **IV.5.1.1 Prueba de requerimiento 1**

En el caso del requisito que especifica que *la coordinación de las actividades del proceso debe hacerse de forma descentralizada*, la actividad consistió en dar de alta a todos los usuarios del sistema con el fin de que el nodo Distribuidor enviara a cada nodo distribuido los elementos necesarios para poder llevar a cabo la coordinación de las actividades. La primera prueba, fue verificar de forma manual que todos los datos necesarios estuvieran en los documentos XML que deberían de estar.

Tabla XXXI. Caso de prueba de requerimiento 1.

| <b>Requerimiento 1 – Coordinación Descentralizada</b> |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|-------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Condición inicial</b>                              | Solamente el <i>Nodo Distribuido</i> está funcionando                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>Flujo de eventos</b>                               | <ol style="list-style-type: none"> <li>1. Se da de alta un actor en cada nodo.</li> <li>2. Se comienza la ejecución de una instancia del proceso.</li> <li>3. Se da de baja el <i>Nodo Distribuido</i>.</li> <li>4. Se continúa con la ejecución de la instancia.</li> <li>5. Se da de baja el nodo <i>Director administrativo</i>.</li> <li>6. Se da de baja el nodo <i>Psicólogo</i>.</li> <li>7. Se da de baja el nodo <i>Jefe de área</i>.</li> <li>8. Se da de baja el nodo <i>Candidato</i>.</li> <li>9. Se da de baja el nodo <i>Jefe de oficina</i>.</li> </ol> |
| <b>Condición final</b>                                | <ul style="list-style-type: none"> <li>• Cada nodo debe tener el flujo de actividades que le corresponden al actor que trabajó en ese nodo</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>Resultado</b>                                      | Correcto                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |

El caso de prueba se puede ver en la tabla XXXI. En esta actividad, el resultado fue que los documentos se encontraban en el directorio en el cual debían estar y con los datos que se esperaba, lo que dejó como conclusión que el nodo distribuido tuvo los elementos necesarios para coordinar la ejecución del proceso. La figura 44 muestra el documento donde se especifican las actividades que corresponden al rol *Director Administrativo*, el cual fue generado al momento de que el actor se dio de alta en el sistema.

```

<?xml version="1.0" encoding="iso-8859-1" ?>
<!DOCTYPE workflow (View Source for full doctype...)>
- <workflow>
- <estadosRol nombreRol="autorizando proceso de seleccion">
- <estado idEstado="Ini" nombreActividad="Estado Inicial" nivel="1" estadoCompuesto="false" pantalla="p1" perteneceA="--">
  <transicion estadoSiguiente="AS1" />
</estado>
- <estado idEstado="AS1" nombreActividad="Esperando recibir solicitud de cobertura de vacantes" nivel="1" estadoCompuesto="false" pantalla="p1" perteneceA="--">
  <transicion estadoSiguiente="AS2" />
</estado>
- <estado idEstado="AS2" nombreActividad="Recibir solicitud de cobertura de vacantes" nivel="1" estadoCompuesto="false" pantalla="p1" perteneceA="--">
  <transicion estadoSiguiente="AS2" />
</estado>
- <estado idEstado="AS3" nombreActividad="Evaluar solicitud" nivel="1" estadoCompuesto="false" pantalla="p1" perteneceA="--">
  <transicion estadoSiguiente="AS4" />
</estado>
- <estado idEstado="AS4" nombreActividad="Enviar autorización" nivel="1" estadoCompuesto="false" pantalla="p1" perteneceA="--">
  <transicion estadoSiguiente="cmt" />
  <comunicarseCon rol="Procesando solicitud" />
</estado>
- <estado idEstado="cmt" nombreActividad="Fin del proceso" nivel="1" estadoCompuesto="false" pantalla="p1" perteneceA="--">
  <transicion estadoSiguiente="--" />
</estado>
</estadosRol>
</workflow>

```

Figura 44. Actividades del que corresponden al rol *Director Administrativo*.

Otra de las actividades consistió en ejecutar el proceso seleccionado para este prototipo en el sistema, y reportar que las actividades se habían cumplido. Para esta actividad, se tuvo a la mano el diagrama RAD del proceso implementado, para comprobar que la coordinación fue hecha de forma adecuada. Con esta actividad, también se buscó probar que la ejecución de las actividades se lleva a cabo de forma descentralizada.

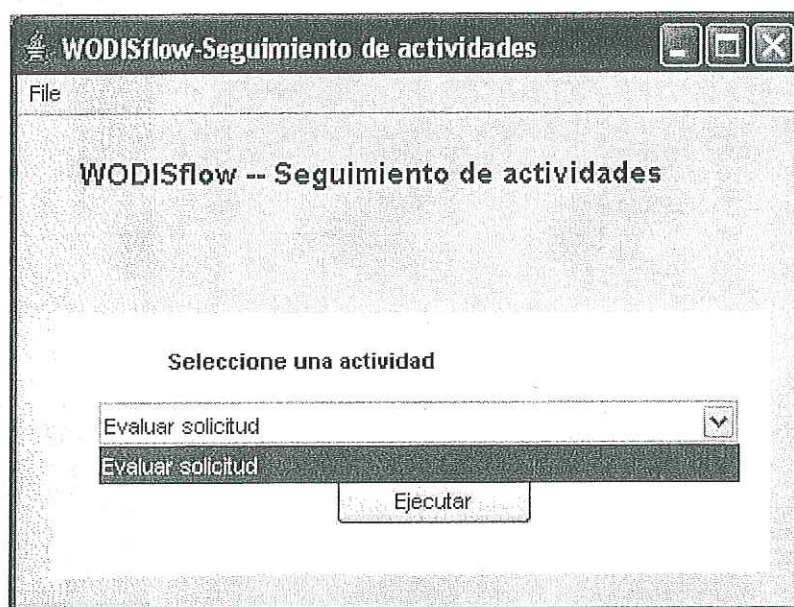


Figura 45. Pantalla de selección de actividades del rol *Director Administrativo*.

La figura 45 muestra la pantalla de selección de actividades, la cual fue cambiando de acuerdo a lo especificado en el diagrama RAD. Se apagó el nodo distribuidor y se ejecutó nuevamente el proceso, con la finalidad de probar que la coordinación podía hacerse sin que el nodo distribuidor estuviera funcionando, y finalmente se desconectaron algunos nodos al azar, para ver si realmente el sistema seguía funcionando de manera independiente, hasta dejar un solo nodo funcionando.



En todos los casos, una instancia del proceso pudo ser finalizada. Esto implicó que no es necesario que haya funcionando algún nodo en específico para que cada actor pueda cumplir con sus actividades en el orden en que deben ser ejecutadas. De hecho cuando estuvo un solo nodo funcionando, la instancia también pudo ser completada, lo que permitió concluir que el requerimiento 1 fue cubierto.

#### **IV.5.1.2 Prueba de requerimiento 2**

La actividad anterior, que consistió en ejecutar el proceso con el soporte del sistema de workflow, no solamente sirvió para probar la coordinación descentralizada de las actividades, sino que también para probar el requisito que establece que *Las actividades del proceso deben ejecutarse de forma descentralizada*. Como ya se mencionó, primeramente se ejecutó el proceso de manera completa, después se dio de baja al nodo distribuidor y posteriormente se desconectaron algunos nodos al azar, hasta dejar un solo nodo distribuido funcionando.

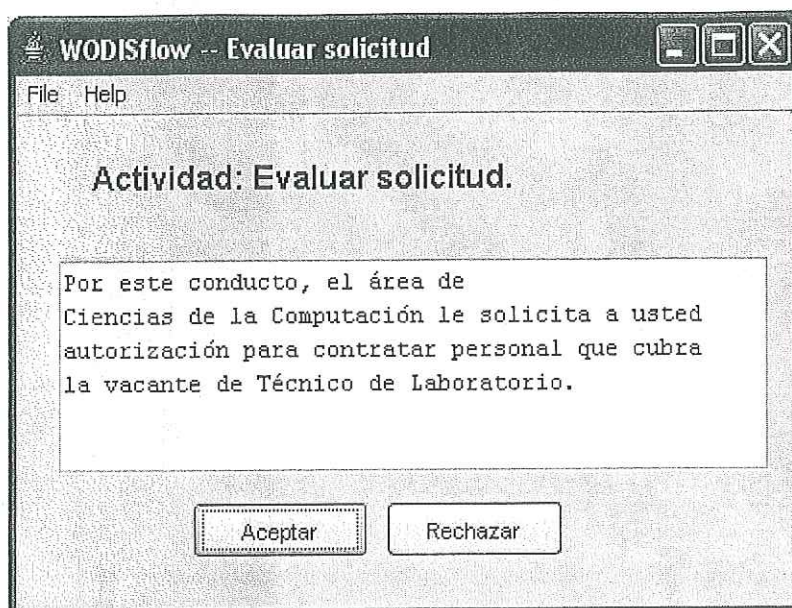


Figura 46. Pantalla la actividad *Evaluar solicitud* del rol *Director Administrativo*.

Como ya se mencionó en la prueba anterior, la actividad de ejecutar el proceso con todos los nodos disponibles fue cubierta satisfactoriamente. Posteriormente, cuando se fueron desconectando los nodos distribuidos, el proceso siguió ejecutándose de manera correcta, aunque es importante mencionar que en los casos en que requerían para continuar una notificación de un nodo que ya había sido apagado, la instancia quedaba detenida en ese estado. Al final se pudo ver que cuando quedó un solo nodo disponible, el sistema siguió funcionando de forma normal (la figura 46 muestra la pantalla de ejecución de la actividad *Evaluar solicitud*), y la instancia que estaba ejecutándose en ese nodo llegó hasta el final, lo que permitió concluir que el requerimiento 2 fue cubierto.

#### IV.5.1.3 Prueba de requerimiento 3

Dentro de los requerimientos, se estableció que la *arquitectura debe incluir un mecanismo que permita recuperar al sistema en caso de que ocurran errores*. Este mecanismo está

implementado con una bitácora que registra cada transacción que ocurre en el proceso. Las actividades de esta prueba fueron ejecutar el proceso completamente y revisar manualmente tanto la bitácora de los nodos distribuidos como la bitácora global en el nodo distribuidor.

El resultado de la revisión manual de las bitácoras permitió constatar que todas las transacciones que se llevaron a cabo quedaron registradas en la bitácora correspondiente, lo que indica que la escritura de las transacciones en las bitácoras se cumplió de acuerdo a lo especificado. En la figura 47 se muestra la bitácora que se generó después de que el actor que desempeñar el rol *Director Administrativo* ejecutó las actividades que le corresponden.

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<!DOCTYPE bitacora (View Source for full doctype...)>
- <bitacora>
  <transaccion idInstancia="A01" idEstado="AS1" status="en ejecución" nivel="1" paralelas="false" subtransaccion="--" información="--" />
  <transaccion idInstancia="A01" idEstado="AS1" status="ejecutada" nivel="1" paralelas="false" subtransaccion="--" información="--" />
  <transaccion idInstancia="A01" idEstado="AS2" status="en ejecución" nivel="1" paralelas="false" subtransaccion="--" información="DA1" />
  <transaccion idInstancia="A01" idEstado="AS2" status="ejecutada" nivel="1" paralelas="false" subtransaccion="--" información="DA1" />
  <transaccion idInstancia="A01" idEstado="AS3" status="en ejecución" nivel="1" paralelas="false" subtransaccion="--" información="DA1" />
  <transaccion idInstancia="A01" idEstado="AS3" status="ejecutada" nivel="1" paralelas="false" subtransaccion="--" información="DA1" />
  <transaccion idInstancia="A01" idEstado="AS4" status="en ejecución" nivel="1" paralelas="false" subtransaccion="--" información="DA2" />
  <transaccion idInstancia="A01" idEstado="AS4" status="ejecutada" nivel="1" paralelas="false" subtransaccion="--" información="DA2" />
  <transaccion idInstancia="A01" idEstado="cmt" status="en ejecución" nivel="1" paralelas="false" subtransaccion="--" información="--" />
  <transaccion idInstancia="A01" idEstado="cmt" status="ejecutada" nivel="1" paralelas="false" subtransaccion="--" información="--" />
</bitacora>
```

Figura 47. Bitácora de las actividades realizadas por el rol *Director Administrativo*.

Posteriormente, se ejecutó nuevamente el proceso, y se interrumpió la ejecución del sistema sin dar tiempo a que pudiera cerrar correctamente, simulando con esto que el sistema sufrió una falla de manera imprevista. El siguiente paso de esta actividad fue reiniciar el nodo que había sido interrumpido y comprobar si el sistema podía determinar el



estado en que se encontraba el proceso antes de que ocurriera la interrupción. Sin embargo, cuando el actor volvió a iniciar sesión en el sistema se pudo ver que tenía disponibles las mismas actividades que cuando ocurrió la falla, por lo que con el resultado de esta actividad de prueba se puede concluir que el requisito 3 fue cubierto satisfactoriamente.

#### IV.5.1.4 Prueba de requerimiento 4

Otro de los requisitos establecidos fue que *las entidades de información, tales como documentos o datos, deben ser manejados de forma distribuida*, por lo que para probar que este requerimiento fuera cumplido, primeramente se observó si ocurría algún funcionamiento incorrecto durante el tiempo en que el proceso fue ejecutado en el sistema.

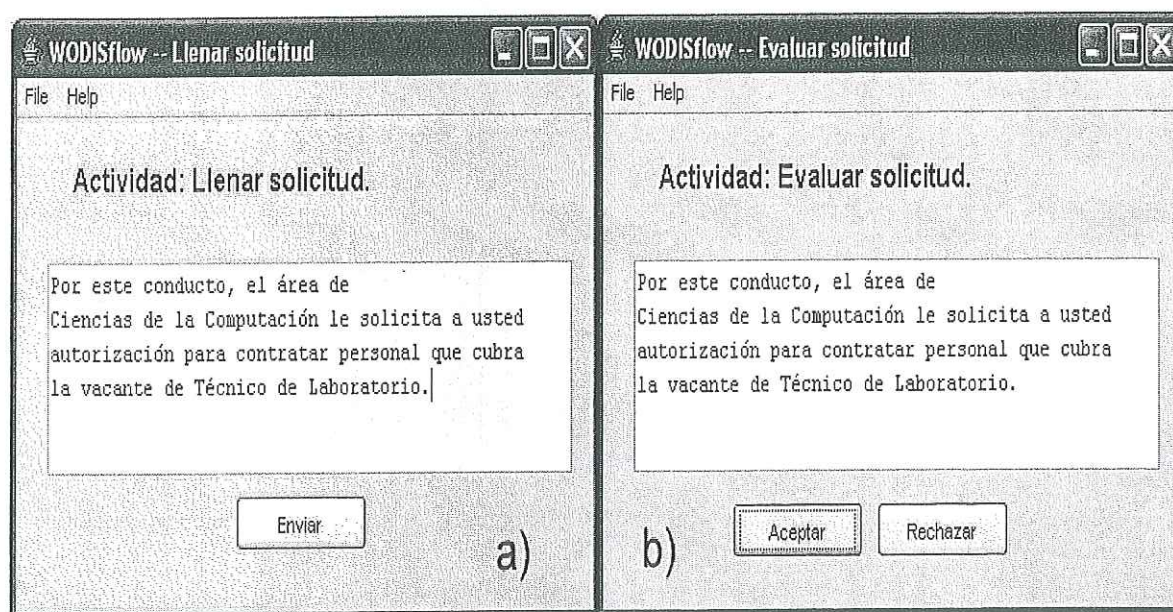


Figura 48. Pantallas donde se utiliza la misma entidad de información.

Después de que al actor *Jefe de oficina* ejecutó la actividad *Enviar solicitud a control de plazas*, se dio de baja al nodo *Jefe de control de plazas* y posteriormente se dio de alta en



otro nodo, y se revisó que la información que le aparecía en el nodo fuera la misma información que tenía antes de que la falla ocurriera.

En la figura 48 se pueden ver una pantalla donde se crea la entidad de información (*actividad llenar solicitud*) para enviarla al *Director administrativo* para que éste autorice la contratación de personal (*actividad evaluar solicitud*).

#### IV.5.1.5 Prueba de requerimiento 5

Un requerimiento que surgió después de la primera iteración fue que *la arquitectura incluyera componentes que permitieran que los sistemas desarrollados reflejen la forma natural en la cual los procesos son llevados a cabo*. Una de las características importantes es que los actores puedan cambiarse a trabajar en varios nodos y trabajar en ellos de manera indistinta, sin que ello afecte el estado en que se encuentran las instancias del proceso que dicho actor ha iniciado. Por ello, se decidió ejecutar el proceso con soporte del sistema de workflow, y se cerraban algunos nodos distribuidos y posteriormente se iniciaba sesión con el mismo actor, pero en otro nodo, y se revisaba que los actores tuvieran los mismos estados en las instancias que tenían activas antes de salir del sistema.

Al cumplirse satisfactoriamente con estas pruebas, se puede asegurar que la arquitectura contempla ausencias de los actores en un determinado momento y no afecta la forma en como proceso es llevado a cabo.

Otra de las características importantes de la forma en como se llevan a cabo los procesos de forma natural, es que puede suceder que en algunos momentos algunos actores no estén disponibles para llevar a cabo las actividades que les corresponde ejecutar, por lo que es importante que haya un lugar en donde se les pueda dejar la notificación de la actividad que puede realizar, para evitar que el actor que envía la notificación permanezca sin avanzar en el cumplimiento de las actividades por estar esperando a que ese actor aparezca.

En el caso del proceso de reclutamiento, selección y contratación de personal, las interacciones se dan entre los siguientes actores:

- Responsable de área – Director de departamento de control de plazas – Psicólogo.
- Responsable de área – Director de departamento de control de plazas – Candidato.
- Candidato – Director de departamento de control de plazas - Psicólogo
- Responsable de área – Director de departamento de control de plazas
- Candidato – Director de departamento de control de plazas
- Jefe de oficina – Director de departamento de control de plazas

Habiendo identificado las interacciones que se presentan en el proceso, la actividad de prueba consistió en que el actor al cual se le enviaría una notificación cerrara su sesión, y ejecutar en el nodo emisor la actividad que en la que se debe notificar al otro actor. Una vez que se envió la notificación, el actor que debía recibir la notificación iniciaba sesión y se revisaba si el actor podía darse cuenta que podría realizar una nueva actividad. Cuando el actor que recibiría la notificación inició sesión, se pudo ver que la actividad que estaba

pendiente por realizar le apareció disponible, lo que permite probar que el nodo emisor se queda monitoreando a que el actor al cual debe enviarle la notificación se da de alta en el sistema.

Otra actividad consistió igualmente en enviarle una notificación a un actor que no estuviera disponible en el sistema, pero que se cerrara la sesión en el nodo emisor una vez que la notificación fuera enviada. Esta actividad tuvo el propósito de que la tarea pendiente se almacenara en el nodo distribuidor y posteriormente cuando el actor que debe recibir la notificación accese de nuevo al sistema, probar si el nodo Distribuidor le informa de la nueva actividad que puede realizar.

En esta actividad también se pudo probar que cuando el actor que recibiría la notificación se dio de alta en el sistema pudo ver la nueva actividad disponible, lo que permite concluir, en conjunto con los resultados de la actividad anterior, se puede concluir que el requerimiento 5 fue cubierto satisfactoriamente.

#### **IV.5.1.6 Prueba de requerimiento 6**

Un último requisito establece que la *arquitectura debe cumplir con los requerimientos básicos de un sistema de Workflow* [Prior, 2002] como son escalabilidad, recuperación de fallas, disponibilidad, monitoreo, historial y soporte al uso de aplicaciones externas. Sin embargo, la funcionalidad que brindan algunos de estos requerimientos como tal no fue provista en el prototipo desarrollado. Sin embargo, algunos de ellos están implícitos en los otros requerimientos probados en esta sección.

Ambos prototipos tuvieron la finalidad de probar la viabilidad de desarrollar sistemas de workflow basados en la arquitectura WODIS, así como probar que los requerimientos propuestos fueron cumplidos.

En el diseño de ambos prototipos se utilizó CORBA como la tecnología para implementar la distribución entre nodos, el lenguaje de programación JAVA para proveer la funcionalidad de los prototipos y XML en el desarrollo de los documentos en los cuales se almacenan los datos útiles para la ejecución del proceso.

En la sección de pruebas se pudo ver que el sistema si cumple con los requerimientos de distribución propuestos, como ejemplo que la coordinación y ejecución de las actividades se hace de forma descentralizada. Asimismo, la información es almacenada en el nodo que la necesitará para la ejecución de una tarea y no en un solo nodo centralizado. Por otro lado, los prototipos fueron provistos con un mecanismo de recuperación de errores, y el último de estos prototipos también permite ejecutar procesos basados en la forma en como éstos son llevados a cabo.



## **Capítulo V. Conclusiones y trabajo futuro.**

---

### ***V.1 Conclusiones.***

En este trabajo se abordaron los problemas que presentan los sistemas de workflow tradicionales, los cuales funcionan bajo el modelo de cómputo distribuido cliente-servidor. Estos problemas están relacionados con sobrecarga de trabajo en el servidor, flujo de datos excesivo sobre la red, poca tolerancia a fallos y poca escalabilidad.

Un enfoque utilizado actualmente para abordar estos problemas es a través de sistemas de workflow distribuido, donde la carga de trabajo sea compartida entre varios nodos. Sin embargo, los trabajos reportados hasta el momento no dan una solución completa a la problemática de estos sistemas, lo cual dio la pauta para diseñar una arquitectura de workflow que sirva de base en el desarrollo de sistemas de workflow distribuido.

La principal característica de esta arquitectura es que cada uno de los nodos que conformen los sistemas de workflow basados en la misma funcionan de manera independiente del resto, puesto que desempeñan funciones similares a las realizadas tanto por el cliente como el servidor en un esquema centralizado, lo que permite aprovechar la capacidad de procesamiento de los nodos, y a la vez, como consecuencia de esto, distribuir la carga y evitar tener un solo punto de falla en el sistema.

La arquitectura presentada está compuesta por dos tipos de nodos. El primero de ellos, denominado nodo distribuidor, tiene como responsabilidad coordinar los aspectos de distribución de los elementos del workflow, como son el flujo que siguen las actividades que corresponden a cada actor de acuerdo al rol o los roles que desempeña, los datos que se requieren para el cumplimiento de estas actividades (documentos, registros de bases de datos) y el estado en que se encuentran las instancias que el actor no ha finalizado de ejecutar entre otros, así como también de dar soporte al proceso de recuperación en caso de errores. El otro tipo de nodo, llamado nodo distribuido, se encarga de coordinar y llevar a cabo la ejecución del proceso. En cada nodo distribuido estará trabajando un actor del proceso, por lo que es necesario que en ellos se tengan disponibles los elementos necesarios para que los actores puedan cumplir con las actividades que les corresponden.

Dentro de las características de la arquitectura propuesta se encuentran que la ejecución y coordinación de las actividades es realizada de forma descentralizada e independiente y los datos que los actores requieren para cumplir sus actividades son distribuidos en el nodo correspondiente. Además, la arquitectura tiene un componente de recuperación de errores, el cual se encarga de dar consistencia a la ejecución del sistema, y también cuenta con algunos componentes que tienen como fin principal reflejar la manera en como los procesos son llevados a cabo de acuerdo a la forma natural de trabajo.

El enfoque de ejecución de las actividades es a través de transacciones y se está basado en el modelo de transacciones flexibles.

El flujo que siguen las actividades del proceso es plasmado en un documento XML, las cuales son agrupadas de acuerdo al rol al que pertenecen.

Para probar la viabilidad de construir sistemas de workflow basados en esta arquitectura, se desarrollaron dos prototipos: el primero de ellos con la finalidad de detectar errores en el diseño de la arquitectura y para probar que la tecnología de desarrollo seleccionada era adecuada para este tipo de sistemas. El segundo de ellos fue desarrollado con la finalidad de verificar si los sistemas de workflow basados en la arquitectura propuesta cumplen con los requerimientos establecidos durante el desarrollo de este trabajo.

## ***V.2 Aportaciones.***

El objetivo principal de este trabajo fue analizar y comprender la problemática que presentan los sistemas de workflow tradicionales basados en el esquema cliente-servidor, con la finalidad de proponer una solución a estos problemas. En este sentido la aportación principal de este trabajo fue el diseño de una arquitectura distribuida que sirve de base para el desarrollo de sistemas de workflow. Estos sistemas funcionan bajo el enfoque de nodos distribuidos, donde la coordinación y ejecución de las actividades es llevada a cabo de forma independiente en cada nodo. Además, el manejo de la información se hace de forma distribuida, lo que minimiza el flujo de datos que se genera sobre la red cada vez que se envían datos entre el cliente y el servidor. Este intercambio de información se da cuando inicia o finaliza alguna actividad que requiere alguno de estos datos.

Otra aportación importante es que la arquitectura cuenta con un mecanismo para la recuperación de errores, el cual es implementado mediante un enfoque transaccional, para lo cual se empleó el modelo de transacciones flexibles, el cual a su vez es parte del modelo de transacciones avanzadas, diseñado con la finalidad de dar soporte a la recuperación de errores en sistemas no convencionales, como los sistemas de workflow.

Finalmente, en el período que comprendió el presente trabajo de tesis, se logró la aceptación de un artículo titulado “Un enfoque distribuido para mejorar la coordinación de los sistemas de workflow“, el cual fue escrito en colaboración de la Dra. Ana Isabel Martínez García, y del Dr. Fernando Rojas Iñiguez, el cual será presentado en la ciudad de Puebla, Puebla en Septiembre del 2005, como parte del 1er Taller en Sistemas Cooperativos Multimedia (SCM), el cuál es desarrollado dentro del marco del Sexto Encuentro Internacional de Ciencias de la Computación 2005 (ENC'05)

### ***V.3 Trabajo futuro.***

El estado de la arquitectura propuesta en este trabajo, así como de los prototipos desarrollados, permiten darse cuenta que aún queda trabajo por realizar en el área. En esta sección se presentan algunas propuestas importantes para continuar con su estudio. Algunas de ellas son de índole técnica, mientras que algunos otros están orientados al campo de la investigación.



- Diseñar e implementar una estrategia que permita tener un control detallado sobre los resultados de la comunicación entre nodos. Es decir, se requiere saber en que momento la comunicación entre nodos no ha podido ser establecida.
- Hacer más robusta la parte referente al manejo de información, ya que actualmente solo se hace manejo de información por medio de paso de parámetros, por lo que se propone el uso de objetos como entidades de información, lo que conlleva que no solo los datos sean pasados entre nodos, sino también los mecanismos para manipular estos datos.
- En la parte de manejo de la disponibilidad de actores, actualmente se tiene contemplado que todos los usuarios estén conscientes del estado en que se encuentra el resto de los usuarios, por lo que se propone que cada usuario solo esté enterado del estado que tienen los actores con los que él tiene interacción.
- La arquitectura propuesta está diseñada tomando en cuenta escenarios donde los actores trabajan únicamente en un tipo de dispositivos, en este caso computadoras personales. Sin embargo, algunos escenarios requieren movilidad, provista usualmente por medio del uso de dispositivos personales (PDAs) los cuales presentan algunas restricciones de acuerdo a su limitada capacidad de procesamiento, que limitarían la implementación del enfoque de nodos independientes. El trabajo realizado por los actores que utilizan estos dispositivos es coordinado normalmente por un servidor central. En este sentido, se propone estudiar la forma de combinar la ejecución de un mismo proceso a través de sistemas de workflow distribuido y sistemas de workflow centralizado.

- Implementar mecanismos de seguridad en el prototipo, de manera que pueda ser factible utilizar en organizaciones reales un sistema de workflow basado en la arquitectura WODIS.
- La rigidez de los sistemas de workflow limitan la forma en como le dan soporte a los procesos reales, por lo que se propone estudiar la manera de dar flexibilidad a los sistemas de workflow distribuido basados en la arquitectura propuesta.
- En la parte de manejo de datos, se consideró únicamente el uso de información almacenada en objetos o en bases de datos, por lo que se propone extender el estudio al manejo de datos almacenados en documentos generados por procesadores de palabras y hojas de cálculo y aplicaciones en general.
- El prototipo fue desarrollado con el fin de probar que los requerimientos del sistema han sido cubiertos. Por ello se propone realizar un prototipo que de soporte a la ejecución de un proceso grande (con más de 10,000 actividades) que pueda ser implementado en la vida real, y hacer pruebas con este prototipo.
- Desarrollar un mecanismo que permita automatizar la instalación de la aplicación de workflow en los nodos distribuidos.
- En el prototipo desarrollado, los objetos CORBA deben ser activados de forma manual, por lo que se propone desarrollar un mecanismo que permita activar los objetos CORBA de manera automática y de forma transparente al usuario.
- El mecanismo de recuperación de errores solo considera errores que surgen cuando un nodo no puede comunicarse con otro, o cuando el sistema es cerrado de manera

incorrecta, por lo que se propone que se haga un estudio más detallado sobre la recuperación de errores en sistemas de workflow distribuidos.

- Desarrollar un mecanismo que permita conocer si los documentos XML almacenados en algún nodo distribuido son consistentes o han sufrido alteraciones.

## Referencias

---

Alonso, G., Agrawal D., A. E. Abbadi., C. Mohan, Gunthor R., M. Kamath, 1995. Exotica/FMQM: A Persistent Message-Based Architecture for Distributed workflow Management, Proceedings of the IFIP WG8.1 Working Conference on Information Systems Development for Decentralized Organizations, Trondheim, Noruega. Agosto 1995: 1-18 p.

Alonso G, B. Reinwald, y C. Mohan, 1997. Distributed data management in workflow environments, Proceedings of the 7th International Workshop on Research Issues in Data Engineering (RIDE'97), Birmingham, Inglaterra. Abril 1995, 2 :82-90 p.

Alonso G. y C. Mohan, 1997. Workflow management systems: The next generation of distributed procesing tools, en Advanced Transaction Models and Architectures, Jajodia y L. Kerschberg (eds.), Kluwer Academic Publishers, Dordrecht, Holanda. 1: 35-62 p.

Arroyo-Sandoval, P. , A.I Martinez-García. y C. Ramirez-Fernandez, 2003. An Architecture for Supporting Disconnected Operation: An XML-Based Aproach, 9th International Workshop, CRIWG 2003, Springer, Autrans, Francia. Septiembre 2003, 1: 277-292 p.



Barbara, D., S. Mehrotra y M. Rusinkiewicz, 1996. INCAs: Managing Dynamic Workflows in Distributed Environments, Journal of Database Management, Special Issue on Multidatabases, 7(1): 5-15 p.

Booch, G., J. Rumbaugh y I. Jacobson. El lenguaje de modelado unificado. Addison Wesley Iberoamericana, Primera edición, Madrid, 432 pp.

Coulouris, G., J. Dollimore y T. Kindberg, 2001. Sistemas Distribuidos: Conceptos y Diseño. Addison Wesley. 3ra. Edición. Madrid. 744 pp.

Fowler y Scott] Fowler, M. y K. Scott, 1997. UML Distilled. Addison Wesley, Quinta edición, Massachusetts, 179 pp.

García Carrillo, F., 2001. Desarrollo de una arquitectura de coordinación de procesos organizacionales en Internet. Tesis de Maestría. CICESE. Ensenada BC. 171 pp.

Gutierrez, A. y R. Martínez, 2001. XML a través de ejemplos, Alfaomega Ra-Ma, Madrid, España. 490 pp.

Kwang-Hoon, K. y H. Dong-Soo , 2001. Performance and Scalability Analysis on Client-Server Workflow Architecture. International Conference on Parallel and Distributed Systems (ICPADS 2001), KyongJu City, Korea. Junio, 2001: 179-186 p.

Lavagno L. y W. Reisig, 1998. Modeling and Analyzing Interorganizational Workflows. Proceedings of the 1998 International Conference on Application of Concurrency to System Design (CSD'98), Fukushima, Japan. Marzo, 1998: 262-272 p.

Mowbray T. y R. Malveau, 1997. CORBA Design Patterns. John Wiley and Sons Ltd, New York. 333 pp.

Muth P., D. Wodtke, J. Weissenfels, D. Kotz, y G. Weikum, 1998. From Centralized workflow Specification to Distributed workflow Execution. Journal of Intelligent Information Systems (JIIS), Special Issue on workflow Management, 10(2): 159-184 p.

Orfali, R., D. Harkey y P. Shidharan, 1997, Advanced Java networking, Prentice Hall PTR, Indiana, USA, 399 pp.

Ould, M., 1995. Bussiness Process. John Wiley and Sons Ltd. 1a Edición. Londres. 216 pp.

Paul, S., E. Park, D. Hutches y J. Chaar, 1998. RainMaker: workflow Execution Using Distributed, Interoperable Components. En Second European Conference, ECDL'98, Londres, Inglaterra. Mayo, 1998: 801 - 818 p.

Prior, C., 2002. Workflow and Process Management, En: L. Fisher (ed), Workflow Handbook 2002, Future Strategies Inc, Florida, 1-20 p.

Prior, C., 2003. Workflow and Process Management, En: L. Fisher (ed), Workflow Handbook 2003, Future Strategies Inc, Florida, 118-141 p.

Rojas Iñiguez, F. y Martínez-García A. I., 1998. Process Modeling to Enactment and Simulation. En: Proceedings of the European Simulation Multiconference'98 (ESM), Manchester, Inglaterra. Junio, 1998: 844-848 p.

Silva, R., J. Wainer, E. Madeira y C. Ellis. 1999. CORBA Based Architecture for Large Scale workflow. En: 4th International Symposium on Autonomous Decentralized Systems (ISADS '99), Tokyo, Japón. Marzo, 1999: 276-281 p.

Sommerville, I., 2002. Ingeniería de Software. Addison Wisley, Sexta Edición. México. 712 pp.

Steflick, D. y J. Edwards, 1998, Cliente/Servidor: Guía de supervivencia, Mc Graw Hill, México, D.F, México, 676 pp.

Warboys, B., P. Kawalek, I. Robertson y M. Greenwood, 1998. Business Information Systems: a process approach. Editorial McGraw-Hill Company. 262 pp.

Wodtke, D., J. Weissenfels, G. Weikum y A. Kotz. 1996. The Mentor Project: Steps Toward Enterprise - Wide workflow Management. En: Proceedings of 12th IEEE

International Conference on Data Engineering, New Orleans, Louisiana. Marzo, 1996, 16: 556-565 p.

Workflow Management Coalition, 1998. "Workflow and Internet: Catalysts for Radical Change". A WfMC White Paper.

Worah, D. y A. Sheth, 1997. Transactions in Transactional Workflows. En: S. Jajodia and L. Kershberg (eds), Advanced Transaction Models and Architectures, Kluwer Academic Publishers, Boston, Massachusetts. Agosto, 1997: 3-347 p.



## **Apéndice A. Análisis y diseño de la arquitectura y los procesos utilizados.**

---

En este anexo se presenta el análisis y diseño que se realizó durante este trabajo. La primera sección contiene el análisis de la funcionalidad de la arquitectura propuesta. Posteriormente, se presenta el análisis del proceso del juego de la cerveza, el cual fue utilizado en la implementación del prototipo desarrollado en la primera etapa de este trabajo y finalmente se presenta el análisis del proceso de reclutamiento, selección y contratación de personal, utilizado en el desarrollo del prototipo que desarrolló en la segunda etapa de este trabajo.

### ***A.1 Análisis de la funcionalidad de la arquitectura***

En esta sección se presenta el diseño y funcionalidad de los componentes de la arquitectura propuesta por medio de diagramas UML (diagrama de casos de uso y diagrama de secuencia).

#### **A.1.1 Casos de uso.**

Un caso de uso es en esencia, una interacción típica entre un usuario y un sistema de cómputo. Los casos de uso son llevados a cabo por. Un mismo actor puede llevar a cabo varios casos de uso, lo mismo que a la inversa, es decir, un caso de uso puede ser llevado a cabo por varios actores [Fowler y Scott, 1997].

Enseguida se presenta el diagrama de casos de uso global (figura 49) de la arquitectura WODIS y la descripción de ellos.

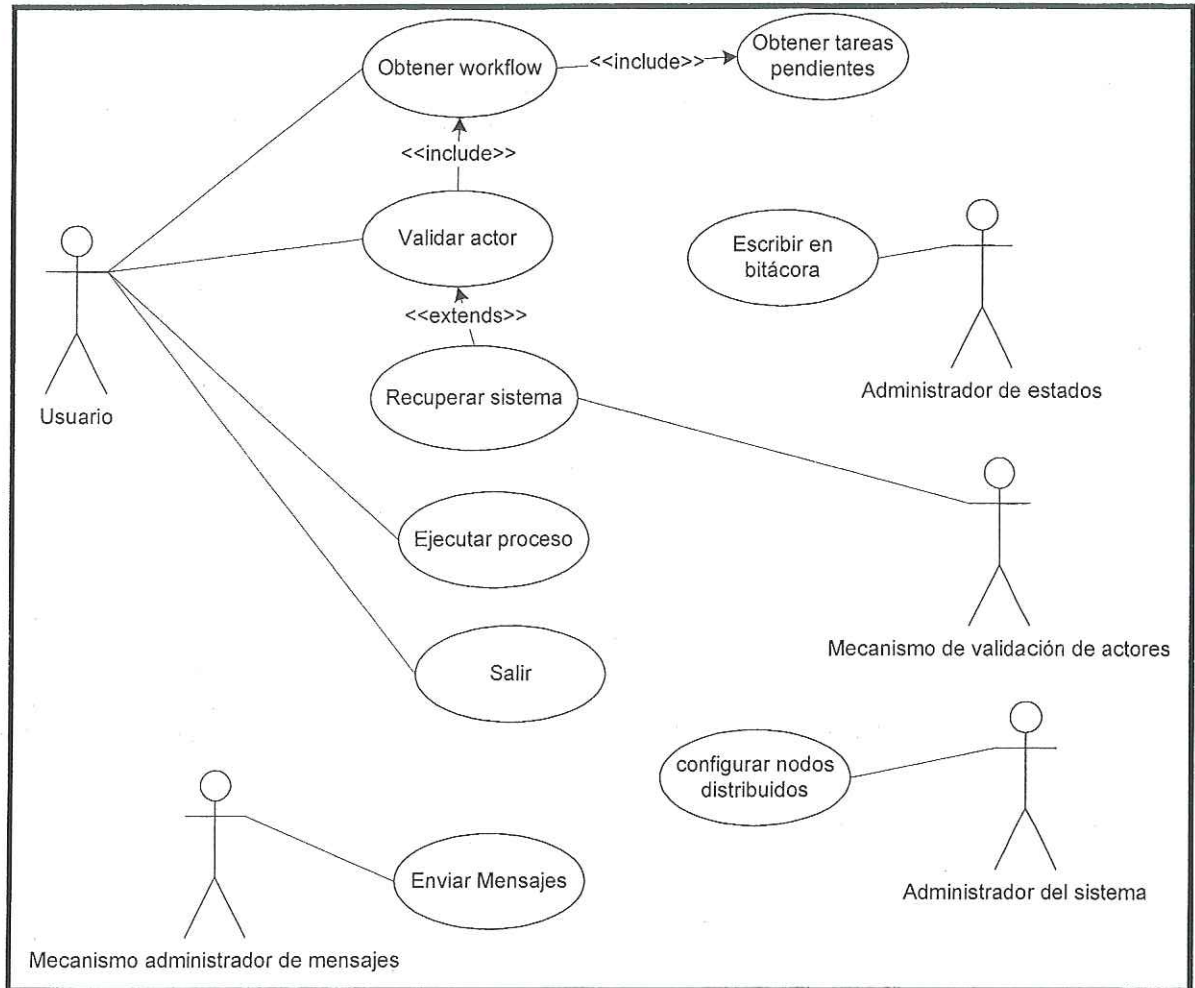


Figura 49. Diagrama de casos de uso de la arquitectura propuesta.

### Descripción de los casos de uso.

**Caso de uso:** Configurar nodos distribuidos.

**Actor:** Administrador del sistema.

**Propósito:** Establecer la aplicación distribuida en los nodos en los que trabajarán los actores.

**Descripción:**

El caso de uso inicia cuando el administrador del sistema desea instalar la aplicación distribuida en los nodos que utilizarán los actores. El administrador del sistema introducirá las direcciones IP y el sistema se encargará de enviar la aplicación a los nodos indicados.

**Caso de uso:** Validar actor

**Actor:** Usuario

**Propósito:** Verificar que los usuarios del sistema estén autorizados a utilizarlo.

**Descripción:**

Este caso de uso inicia cuando el usuario se da de alta en el sistema. El usuario introduce su nombre de cuenta y su contraseña y el mecanismo *Validación de actores* en el *nodo distribuidor* se encarga de revisar si el actor está autorizado a utilizar el sistema. En caso de que así sea, se revisa si el nodo en que trabajará el actor es el mismo en el cual lo hizo la vez anterior. Si el actor trabajó en ese nodo, se envía al nodo distribuido el status que guardan los actores en el proceso y se comienza la ejecución del proceso. En caso de que el nodo no sea el último en el que trabajó el actor, se ejecuta el caso de uso *Obtener workflow* y el caso de uso finaliza. En caso de que el actor no esté autorizado a utilizar el sistema, el caso de uso finaliza.

**Caso de uso:** Obtener workflow

**Actor:** Usuario

**Propósito:** Obtener los elementos que el actor necesita para la ejecución del proceso.

**Descripción:**

Este caso de uso inicia una vez que el actor ya ha sido identificado como usuario autorizado. Se obtienen los roles que desempeña el usuario y se obtiene el flujo que siguen las actividades que corresponden a estos roles, la definición de las actividades (código) y los documentos que se necesitan para la ejecución de las actividades. Además, se revisa en el administrador de tareas pendientes si hay actividades pendientes para alguno o algunos de los roles del actor, y en caso de haberlas, éstas se agregan a la información obtenida. Finalmente se obtiene el estado global de los procesos que lleva a cabo el actor y se envía la información obtenida al nodo distribuido del actor.

**Caso de uso:** Escritura en bitácora

**Actor:** Mecanismo administrador de estados.

**Propósito:** Guardar en bitácora los datos que corresponden a los estados que ya han sido ejecutados o están en ejecución.

**Descripción:**

Este caso de uso inicia cuando una actividad comienza o termina su ejecución. Si la actividad comienza a ejecutarse, el administrador de estados envía los datos al mecanismo escritor de bitácora en el nodo distribuido y en el nodo centralizado y les indica que la actividad se encuentra "en proceso". En caso contrario se envían los datos a ambos escritores de bitácora, pero se indica que la actividad se encuentra "finalizada".



**Caso de uso:** Recuperar el sistema.

**Actor:** Usuario

**Propósito:** Llevar a los nodos a estados consistentes en caso de que ocurra alguna falla.

**Descripción:**

Primeramente, el actor se da de alta nuevamente en un *nodo distribuido (ND)*, y se manda validar al actor al *nodo distribuidor (NC)*. En el *NC*, el mecanismo de *validación* se encarga de revisar que el actor esté autorizado a utilizar el sistema, y en caso de que así sea, revisa si el *ND* en cual está accedando el actor es el mismo en el que lo hizo la vez anterior que utilizó el sistema. En caso de que no sea el mismo nodo, se envía al *ND* toda la información necesaria para que pueda iniciar la ejecución del proceso.

En caso contrario, se revisa si el sistema fue cerrado correctamente. Si así fue, se obtiene de la *bitácora local* el estado en que se encuentran las instancias del proceso y se comienza la ejecución del proceso. Si el nodo no fue cerrado correctamente, se obtiene tanto la *bitácora local* como la *bitácora global* y se obtiene de ambas el estado en que se encuentran las instancias del proceso, y finalmente se comienza la ejecución del proceso.

**Caso de uso:** Ejecución del proceso

**Actor:** Usuario

**Propósito:** Llevar a cabo la ejecución del proceso.

**Descripción:**

El caso de uso inicia cuando el actor desea ejecutar alguna de las actividades que le corresponden. El *administrador de estados (AE)* verifica en el *administrador de instancias*

(AI) los estados actuales de las instancias y le muestra al actor las actividades de las instancias activas que puede realizar, o le pregunta si desea iniciar una nueva instancia. El actor selecciona la actividad que desea realizar y se ejecuta el caso de uso Escribir en bitácora estableciéndose la actividad como “en ejecución”. En caso de ser necesario, el AE le indica al AI que se ha iniciado una nueva instancia. Una vez que el actor ha seleccionado la actividad, se invoca a la actividad por medio del AA, quien a su vez se encarga de obtener la interfaz de usuario del AG, los documentos que necesitará del AD y en caso de necesitar una aplicación en específico se lo notificará al AP. Cuando la actividad ha concluido, se ejecuta nuevamente el caso de uso Escribir en bitácora estableciéndose la actividad como “ejecutada” y se vuelve a iniciar este caso de uso.

El caso de uso termina cuando el actor decide salir del sistema.

**Caso de uso:** Salir.

**Actor:** Usuario.

**Propósito:**

Enviar al nodo distribuidor las entidades de información almacenadas en el nodo distribuidor, con la finalidad de que el actor pueda utilizar el sistema de forma consistente en cualquier nodo.

**Descripción:**

Este caso de uso comienza cuando el actor decide salir del sistema. El mecanismo de *Salida* es notificado, y se encarga de enviar al nodo distribuidor los documentos y datos que se encuentren almacenados en el nodo. Además, el mecanismo de salida debe indicarle al mecanismo *Disponibilidad de actores* que se encargue de avisar a los nodos disponibles en

el sistema que el actor dejará de estar disponible. Así mismo, también obtiene del mecanismo *Administrador de tareas pendientes* los documentos, datos y notificaciones que no han podido ser entregadas a otros actores y los envía al mecanismo *Administrador de tareas pendientes* del nodo distribuidor. Finalmente, el mecanismo *Salida* indica de forma permanente que el nodo ha sido cerrado de forma correcta.

**Caso de uso:** Obtener tareas pendientes

**Actor:** Mecanismo Distribución inicial

**Propósito:** Obtener las tareas que no han podido ser ejecutadas para asignárselas a algún actor.

**Descripción:**

Este caso de uso inicia cuando un actor se da de alta en el sistema. El mecanismo *distribución inicial* revisa en el mecanismo *Administración de tareas pendientes (ATPC)* si hay tareas pendientes para uno o más de los roles que desempeña el actor. En caso de haberlas, el *ATPC* obtiene las entidades de información necesarias para que se puedan llevar a cabo las actividades, y una vez obtenidos estos datos, el *DI* los envía al nodo del actor por medio del mecanismo *Emisor de mensajes*.

**Caso de uso:** Enviar mensajes

**Actor:** Mecanismo Emisor de mensajes

**Propósito:** Interactuar con otros actores del proceso.

**Descripción:**

Este caso de uso inicia cuando el alguno de los mecanismos requiere interactuar con otro nodo. El mecanismo *Emisor de mensajes (EM)* obtiene la ubicación del nodo y también el método que va a invocar. Una vez que obtiene estos datos, se establece la comunicación y el caso de uso finaliza.

## A.1.2 Diagramas de secuencia

Los diagramas de interacción son modelos que describen la manera en que colaboran grupos de objetos para cierto comportamiento. [Fowler y Scott, 1997].

Los diagramas de secuencia son parte de los diagramas de interacción. Habitualmente, un diagrama de interacción capta el comportamiento de un solo caso de uso.

En un diagrama de secuencia, un objeto se muestra como caja en la parte superior de una línea vertical punteada. Esta línea vertical se llama línea de vida del objeto y representa la vida del objeto durante la interacción. Cada mensaje se representa mediante una flecha entre las línea de vida de dos objetos.

A continuación se presentan los diagramas de secuencia de los casos de uso descritos anteriormente.



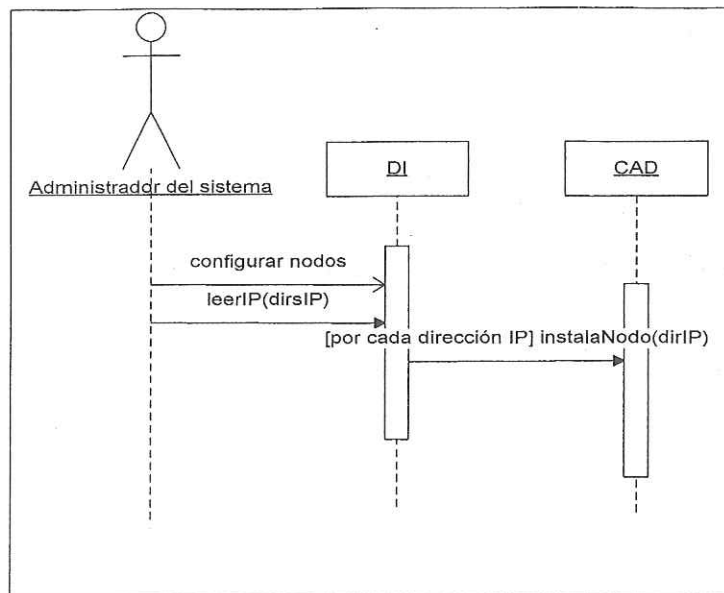


Figura 50. Diagrama de secuencia del caso de uso Configurar nodos distribuidos.

La figura 50 muestra el diagrama de secuencia del caso de uso *Configurar nodos distribuidos*, el cual inicia cuando el **administrador del sistema (AS)** desea configurar los nodos que utilizarán los usuarios del sistema (actores del proceso). El nodo distribuidor tendrá visible en pantalla una opción en pantalla, la cual al ser seleccionada, activará el método *leerIP()* del **mecanismo de distribución inicial (DI)**. Dicho método consiste en solicitarle al AS las direcciones IP de cada una de las PC's que funcionará como nodo distribuido. Una vez que el AS ha terminado de introducir las direcciones IP, el DI activa el método *instalarNodo(dirIP)* del **mecanismo de configuración de aplicación distribuida (CAD)** una vez por cada dirección IP introducida por el AS. El CAD toma la dirección IP y envía la aplicación distribuida a la PC correspondiente.

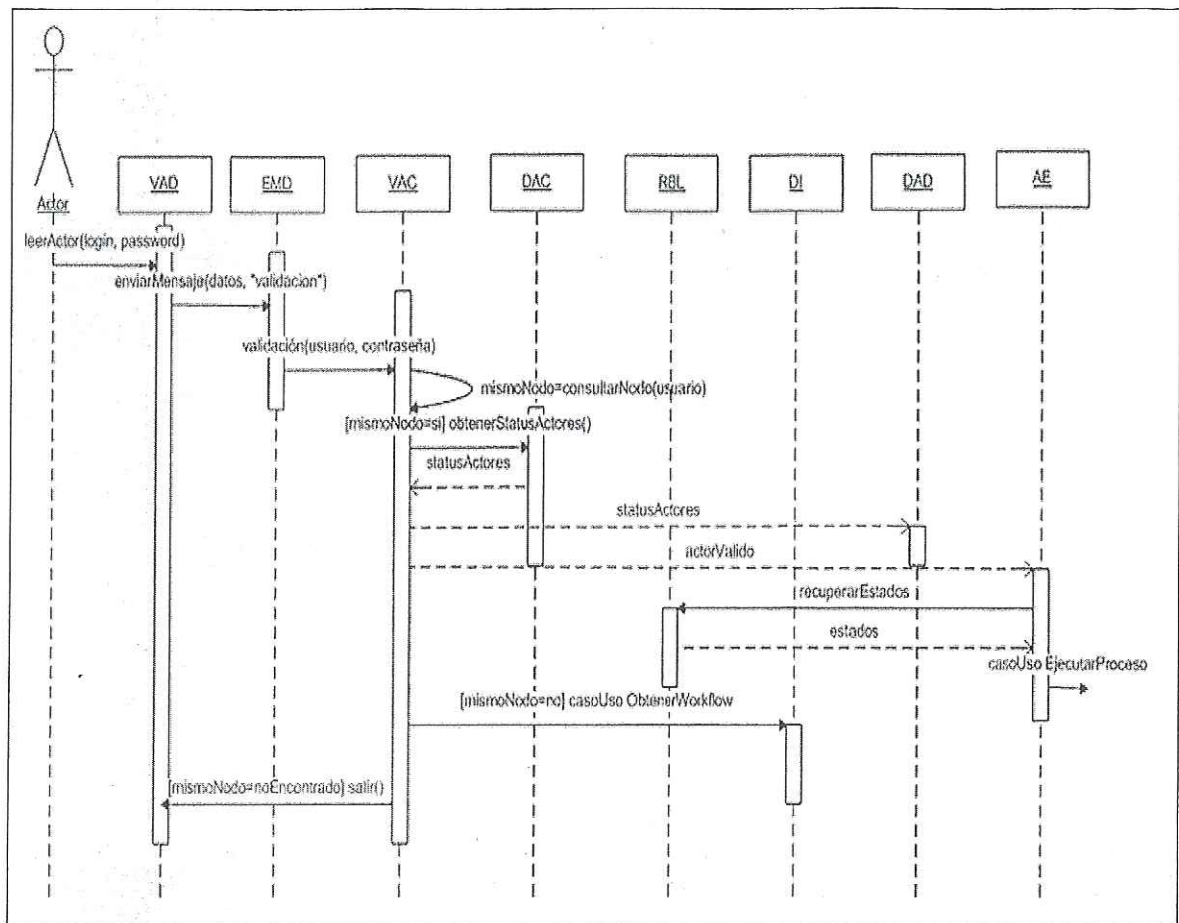


Figura 51. Diagrama de secuencia del caso de uso Validar actor.

La figura 51 muestra el diagrama de secuencia del caso de uso *Validar actor*, que inicia una vez que el usuario vuelve a darse de alta en el sistema. El usuario introduce sus datos de identificación (nombre de usuario y contraseña) al **mecanismo de validación de actores del nodo distribuido (VAD)** por medio del método *validarUsuario(login, password)*. El método consiste en primeramente revisar si el usuario que está entrando es el mismo usuario que utilizó por última vez ese nodo mediante la llamada al método *actorAnterior(login, password)* y almacenando el resultado que arroja este método en la variable *anterior*. Si los datos no corresponden con los datos del usuario que utilizó el nodo previamente, el VAD le indica al **mecanismo emisor de mensajes del nodo distribuidor**

(EMD) que envíe los datos del usuario al **mecanismo de validación de actores del nodo distribuidor (VAC)** para que este se encargue de revisar si el usuario está autorizado a utilizar el sistema. Para esto, el **VAC** revisa en el **directorío de actores del nodo distribuidor** mediante el método *revisarUsuario(login, password)* y este método retorna un arreglo donde indica los roles que le corresponden al actor (en caso de que el usuario no esté autorizado a utilizar el sistema se retorna el arreglo vacío). Una vez hecha la comprobación, se envía *false* al nodo distribuido si el arreglo está vacío y *true* en caso contrario. Para finalizar este caso de uso, se ejecuta el caso de uso **mostrar actores disponibles**.

Por otro lado, si los datos del usuario coinciden con los del usuario anterior, se obtiene el estado global del proceso y el caso de uso finaliza.

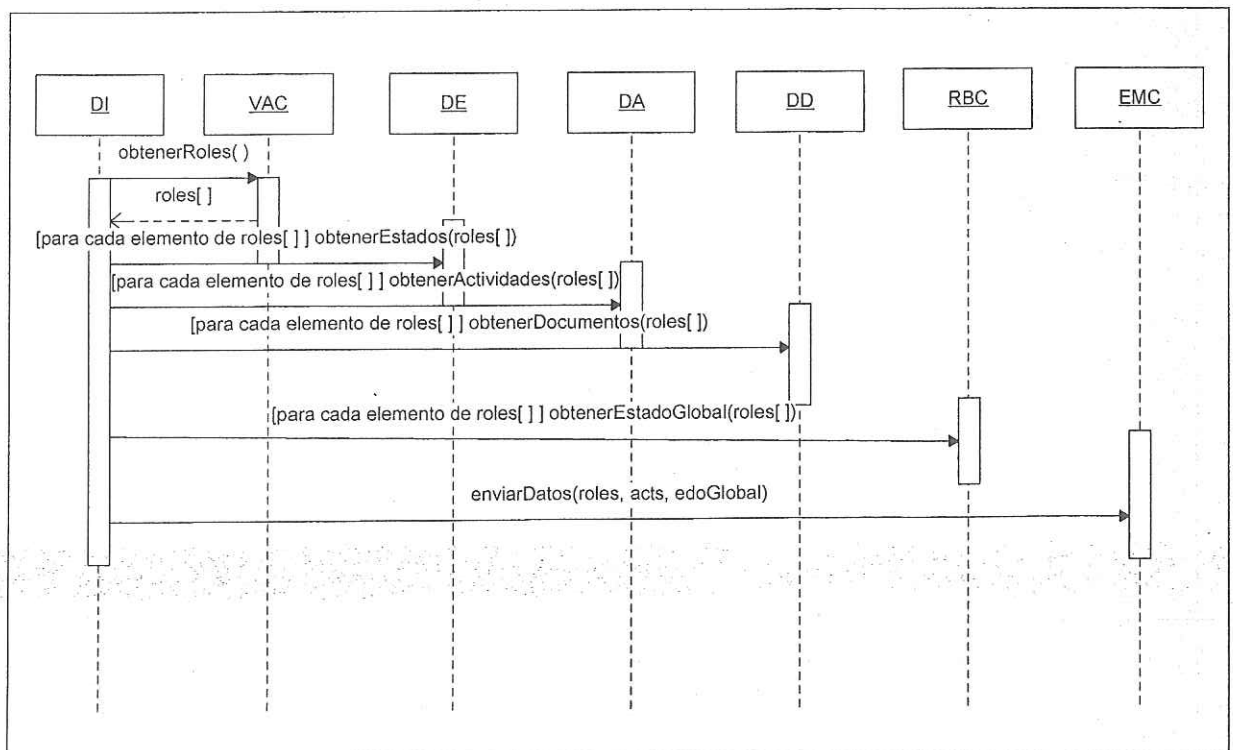


Figura 52. Diagrama de secuencia del caso de uso Obtener workflow.

La figura 52 muestra el diagrama de secuencia del caso de uso *Obtener workflow*, el cual inicia una vez que el actor ya ha sido identificado como usuario autorizado (*caso de uso Validar actor*). El **mecanismo de distribución inicial (DI)** obtiene los roles que desempeña el actor mediante la invocación al método *obtenerRoles()* en el **mecanismo de validación de actores del nodo distribuidor (VAC)**, los cuales quedaron almacenados ahí al momento de ejecutarse el caso de uso Validar actor. Una vez obtenidos los roles, el **DI** obtiene el flujo que siguen las actividades que le corresponden a cada rol del actor mediante la activación del método *obtenerEstados(roles[ ])* en el **mecanismo de distribución de estados (DE)**. Este método accesa al **repositorio global de workflow** para consultar el flujo de trabajo global del proceso y de ahí obtener solo las actividades que le corresponden a cada rol. Después de haber obtenido el flujo de las actividades, el **DI** obtiene la definición de las actividades (código) a través del método *obtenerActividades(roles[ ])* del **mecanismo de distribución de actividades (DA)**, el cual se encarga de serializar los objetos que corresponden a cada rol. Asimismo, ya habiendo obtenido la definición de las actividades, el **DI** obtiene los documentos que el actor necesitará para llevar a cabo las actividades. Para esto, invoca al método *obtenerDocumentos(roles[ ])* del **mecanismo de distribución de documentos (DD)**. Este método accesa al **repositorio global de documentos** y obtiene todos los documentos que le corresponden a los roles del actor. Una vez obtenidos los documentos, el **DI** obtiene el estado global que guarda cada rol mediante el método *obtenerEstadosGlobales(roles[ ])*, el cual accede a la **bitácora global** por medio del **mecanismo recuperador de bitácora (RBC)** y obtiene el estado en que se encuentran las instancias de cada uno de los roles del actor.



Finalmente, el **DI** envía los datos obtenidos a través del **mecanismo emisor de mensajes del nodo distribuidor (EMC)**.

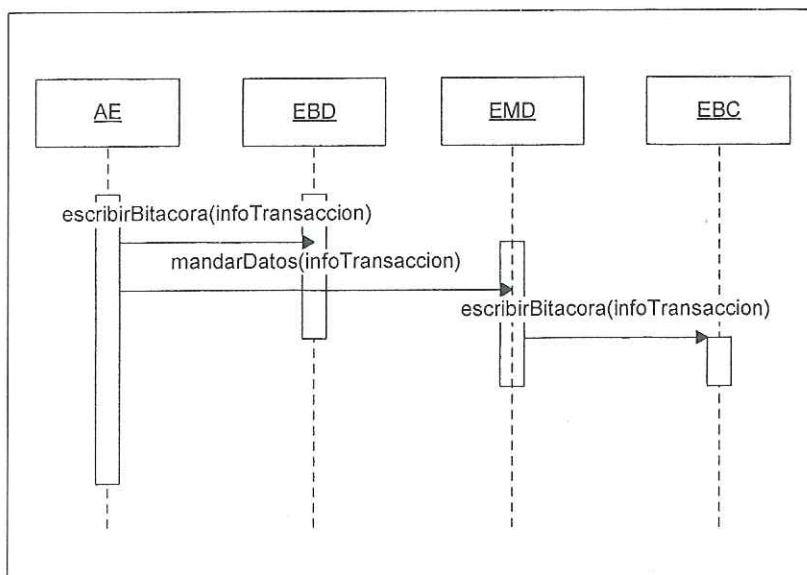


Figura 53. Diagrama de secuencia del caso de uso Escribir bitácora.

La figura 53 muestra el diagrama de secuencia del caso de uso *Escribir bitácora*, que inicia cuando una actividad comienza a ejecutarse o termina su ejecución (actividad actual). El **mecanismo administrador de estados (AE)** le indica al escritor de bitácora local (EBD) los datos que identifican la transacción que contiene la actividad actual, y éste lo registra en **bitácora local** mediante el método *escribirBitacora(infoTransaccion)*. Además, el **AE** envía los datos de la transacción mediante el **mecanismo emisor de mensajes del nodo distribuido (EMD)** al **mecanismo escritor de bitácora global (EBC)**, y el caso de uso finaliza cuando el **EBC** almacena los datos de la transacción la **bitácora global** a través del método *escribirBitacora(infoTransaccion)*.

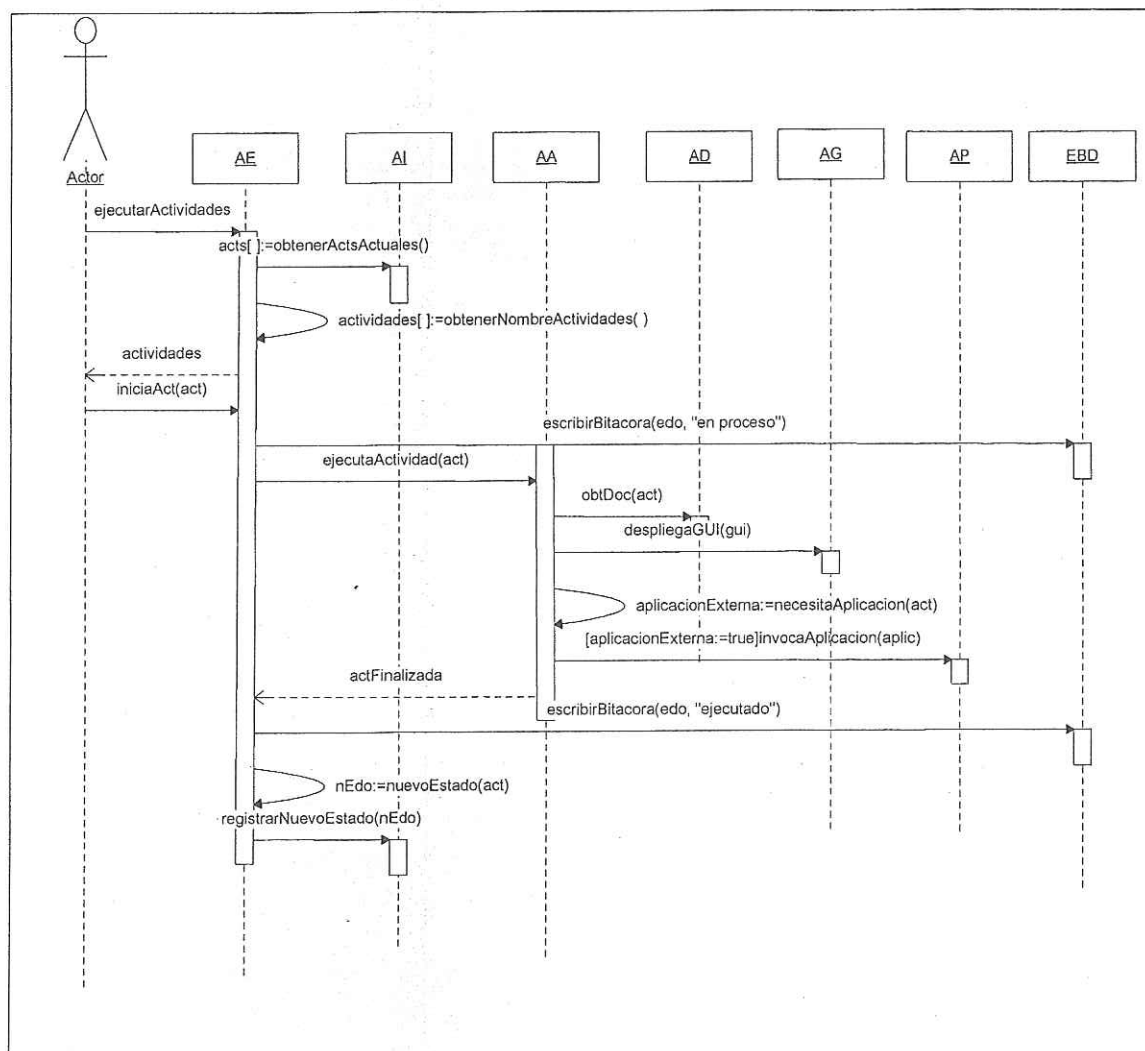


Figura 54 Diagrama de secuencia del caso de uso Ejecutar proceso.

La figura 54 muestra el diagrama de secuencia del caso de uso *Ejecutar proceso*, que inicia cuando el **usuario** desea ejecutar alguna de las actividades que le corresponden. El **mecanismo administrador de estados (AE)** obtiene del **mecanismo administrador de instancias (AI)** los estados correspondientes a las actividades que el actor puede realizar en ese momento a través del método *obtenerActsActuales()*. Una vez obtenidos los estados, el **AE** obtiene el nombre de las actividades que corresponden a los estados y se las muestra

al **usuario**. El **usuario** selecciona la actividad y se inicia el método *iniciaActividad(act)* en el **AE**. El método *iniciaActividad(act)* se encarga de iniciar el caso de uso *Escribir en bitácora* para registrar que la actividad ha comenzado en encuentra “en proceso” de ser ejecutada. Después, el **AE** invoca al método *ejecutarActividad(act)* y es este método el encargado de la ejecución de la actividad. Para esto, obtiene del **mecanismo de administración de información (AD)** los documentos que necesita para poder llevar a cabo la actividad mediante la invocación al método *obtenerDocumentos(act)*, le indica al **mecanismo administrador de interfaz gráfica (AP)** cual interfaz mostrar mediante el método *despliegaInterfaz(interfaz)*, y invoca a alguna aplicación externa al sistema por medio del método *invocaActividad(aplicación)* del **mecanismo Administrador de Aplicaciones**. Una vez que ya ha obtenido los elementos anteriores (datos, interfaz y aplicación) se ejecuta el *enactment* y al finalizar se le informa al **AE**. El **AE** al recibir la notificación de que la actividad ya ha sido ejecutada, nuevamente inicial el caso de uso *Escribir Bitácora*, para registrar que la actividad ya ha sido “ejecutada”. Finalmente, el **AE** determina el nuevo estado (método *nuevoEstado(act)*) en que se encuentra la instancia a la que pertenece la actividad que se ha ejecutado e indica al **AI** estos cambios por medio del método *registrarNuevoEdo(nEdo)*.

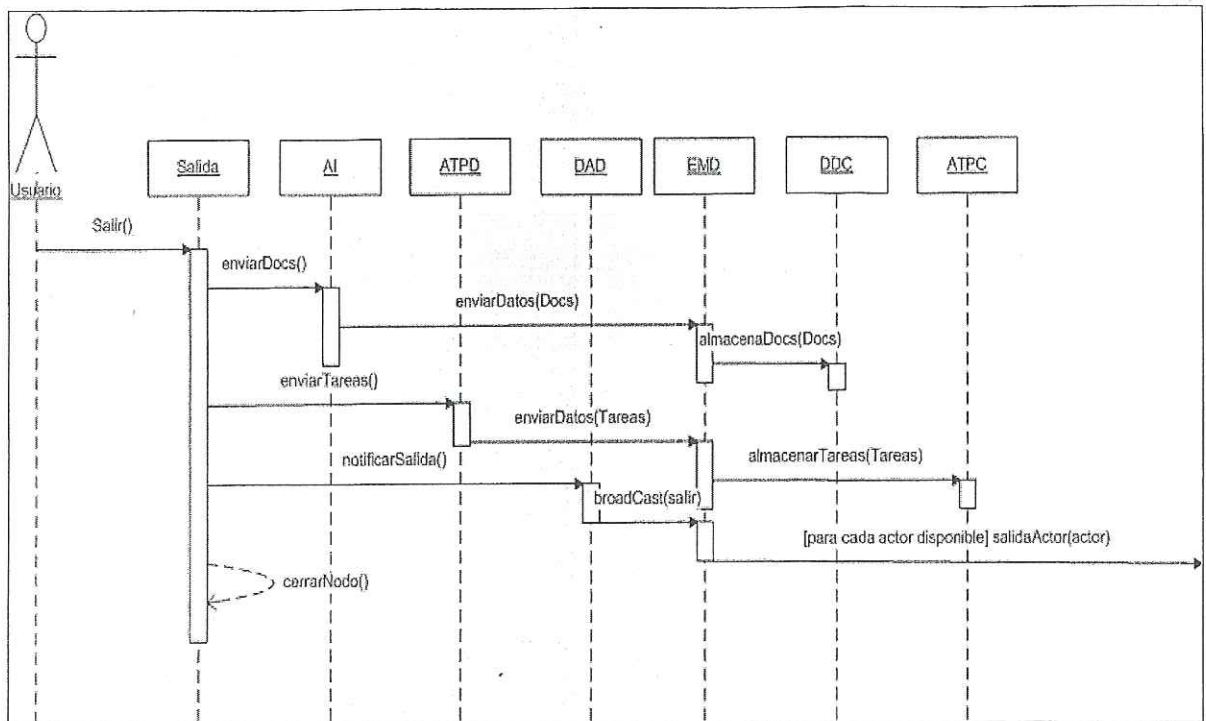


Figura 55 Diagrama de secuencia del caso de uso Salir.

La figura 55 muestra el diagrama de secuencia del caso de uso *Salir*, que inicia cuando el actor desea salir del sistema. Se activa el método *Salida* del **mecanismo Salida**, el cual se encarga coordinar la ejecución de las tareas que permitan que el actor pueda usar el sistema en forma consistente en cualquier nodo que se de alta la próxima vez que utilice el sistema. Para lograr ese propósito, el **mecanismo Salida** le indica al **mecanismo Administrador de la información (AI)**, mediante el mensaje *enviarDocs()*, que envíe al nodo distribuidor los documentos que están almacenados en el nodo distribuido, y éste los envía a través del **mecanismo Emisor de mensajes (EMD)**. El EMD envía los datos al **mecanismo Distribuidor de documentos (DDC)** para que los almacene en el repositorio global de documentos.



Por otra parte, el **mecanismo Salida** le indica al **mecanismo Administrador de tareas pendientes (ATP)**, mediante el método *enviarTareas()*, que envíe las tareas pendientes al nodo distribuidor, y éste lo hace por medio del **EMD**, quien a su vez le indica al **mecanismo Administrador de tareas pendientes (ATP)** del nodo distribuidor que almacene las tareas que le ha enviado. Así mismo, el **mecanismo Salida** le indica al **mecanismo Disponibilidad de actores (DAD)** que notifique a los demás actores que el nodo dejará de estar disponible. Para esto, el **DAD** le pide al **EMD** que envíe un mensaje a todos los nodos disponibles para notificarles que el actor saldrá del sistema.

Finalmente, el **mecanismo Salida** ejecuta el método *cerrarNodo()*, el cual establece en una variable permanente que el nodo ha sido cerrado correctamente y finalmente cierra el sistema.

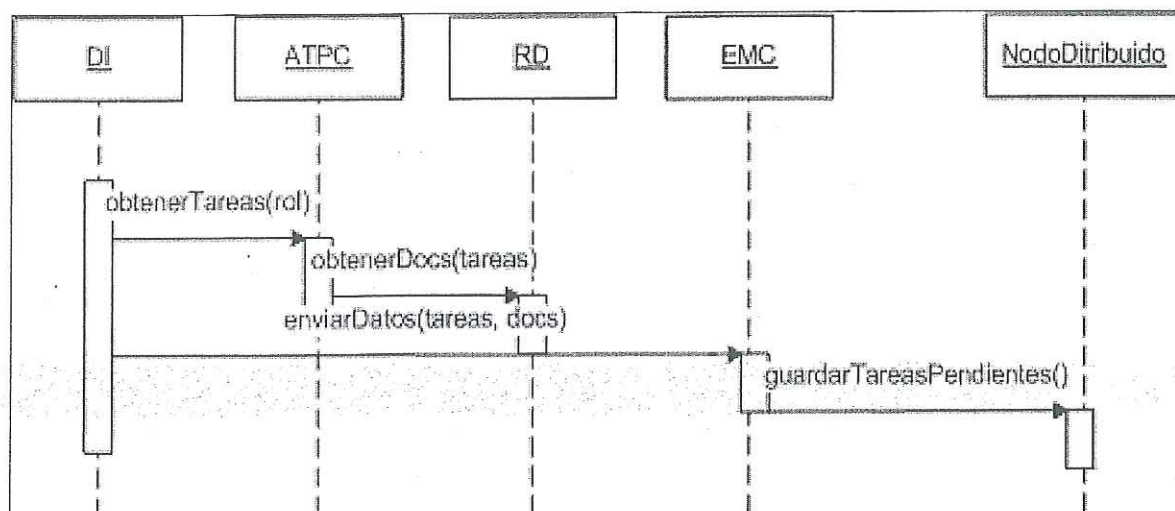


Figura 56. Diagrama de secuencia del caso de uso Obtener tareas pendientes.

La figura 56 muestra el diagrama de secuencia del caso de uso *Obtener tareas pendientes*, que inicia cuando algún actor de da de alta en el sistema. El **mecanismo Distribución Inicial** le indica al **mecanismo Administración de tareas pendientes (ATPC)** del nodo distribuidor que obtenga las tareas pendientes para los roles que desempeña el actor. El ATPC obtiene las tareas pendientes y los documentos que estás requieren (del **repositorio de documentos**) para ser llevadas a cabo, y retorna los datos al **DI**. El **DI** le indica al **mecanismo Emisor de mensajes (EMC)** que envíe los datos al nodo en el cual se encuentra el actor.

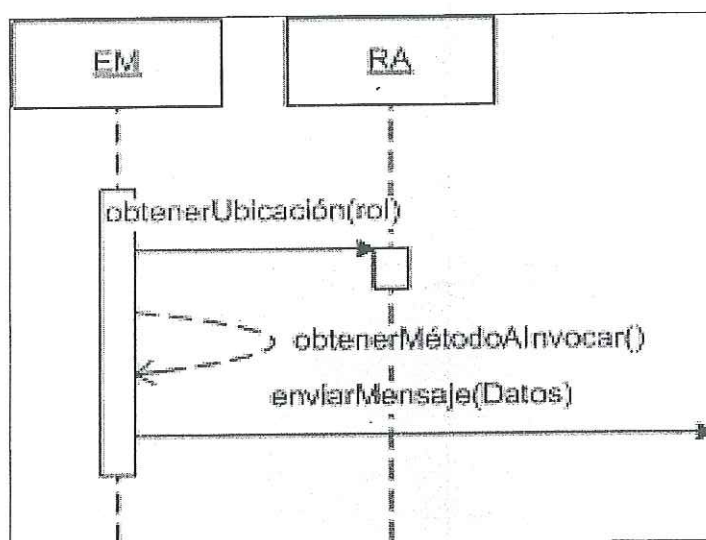


Figura 57. Diagrama de secuencia del caso de uso Enviar mensaje

La figura 57 muestra el diagrama de secuencia del caso de uso *Enviar mensajes*, que inicia cuando algún mecanismo necesita interactuar con otro nodo. Para esto, el **mecanismo Emisor de mensajes (EM)** debe obtener la ubicación del nodo en el cual se encuentra el actor, por lo que busca esta dirección en el **Directorio de actores (RA)**. Una vez que obtiene la ubicación del nodo, debe saber también que método debe invocar y una vez que ya tiene estos datos, se establece la comunicación.

## A2. Análisis del proceso del juego de la cerveza.

Para el prototipo que implementa el proceso del juego de la cerveza se identificaron los siguientes casos de uso:



Figura 58. Diagrama del caso de uso *Hacer pedido de producto*.

El caso de uso *Hacer pedido de producto* (figura 58) es llevado a cabo de la misma manera por el rol cliente y por el rol minorista. Tiene como propósito indicar al proveedor de producto (minorista o mayorista, según sea el caso) la cantidad de éste que se desea adquirir.

### Descripción:

El caso de uso inicia cuando el actor desea adquirir algún artículo. Para este fin, el actor elabora una hoja de pedido donde se especifica el número de artículos que desea, así como la identificación del cliente. Una vez que ha sido elaborado el pedido, el actor lo envía al proveedor correspondiente (el cliente lo envía al minorista y el minorista lo envía al mayorista).

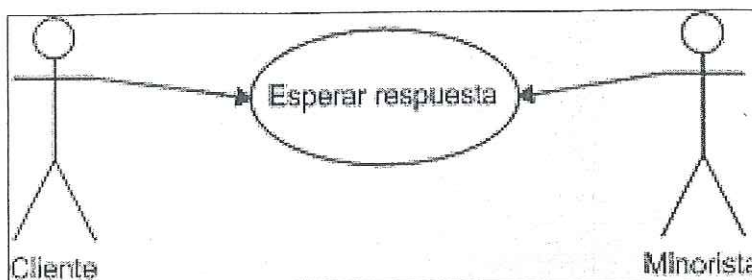


Figura 59. Diagrama del caso de uso *Monitorear respuesta*.

El caso de uso *Monitorear respuesta* (figura 59) es también llevado a cabo por el rol cliente o por el minorista. La finalidad de este caso de uso es saber el momento en que el pedido realizado ha sido atendido por el proveedor de producto.

#### Descripción:

El caso de uso inicia una vez que el actor ha enviado la solicitud de pedido. El actor revisa el estado del pedido: si sufre un cambio, se ejecuta el caso de uso obtener respuesta. Si no sufre cambio se vuelve a ejecutar nuevamente este caso de uso.

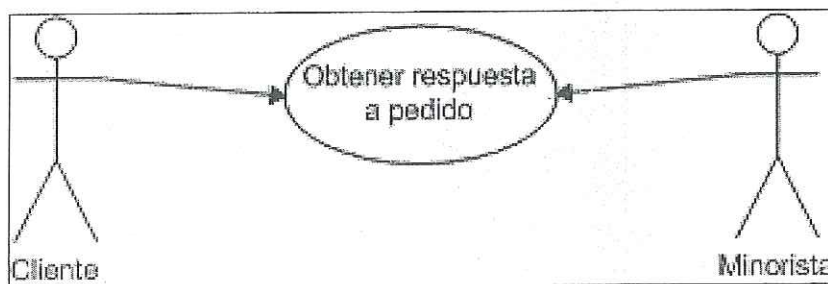


Figura 60 Diagrama del caso de uso *Obtener respuesta a pedido*.

El caso de uso *Obtener respuesta a pedido* (figura 60) tiene como propósito obtener la respuesta que el proveedor de producto ha dado al pedido realizado por el actor o por el minorista según sea el caso. Se ejecuta cuando finaliza el caso de uso *Monitorear respuesta*.



### Descripción:

El caso de uso inicia una vez que el actor se da cuenta que el pedido ha sido atendido. El actor solicita al proveedor de servicio conocer la respuesta al pedido realizado, y el caso de uso finaliza una vez que la respuesta ha sido enviada al actor.

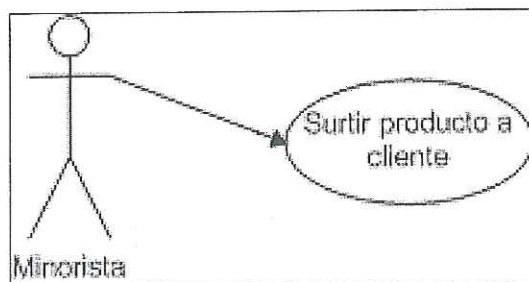


Figura 61 Diagrama del caso de uso *Surtir producto a cliente*.

El caso de uso *Surtir producto a cliente* (figura 61) es llevado a cabo por el minorista. La finalidad de este caso de uso es atender las solicitudes de producto que han realizado los clientes.

### Descripción:

El caso de uso inicia cuando el actor desea atender las solicitudes de pedido hechas por el cliente. El minorista revisa la lista de pedidos, y en caso de haber alguno, revisa en el inventario para saber si cuenta con el producto suficiente para cumplir con la demanda de producto indicada en la hoja de pedido. Si el pedido es cumplido, se marca el pedido como atendido y se ejecuta el caso de uso Actualizar inventario. Si por el contrario el pedido no puede satisfacerse, se marca el pedido como atendido y se ejecuta el caso de uso Hacer pedido de producto.

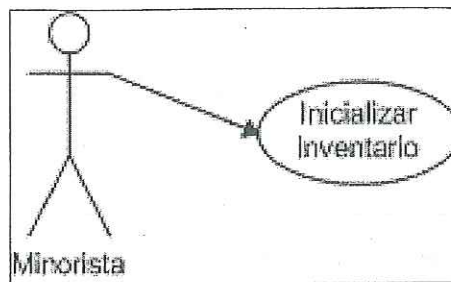


Figura 62. Diagrama del caso de uso *Inicializar inventario*.

El caso de uso *Inicializar inventario* (figura 62) es llevado a cabo por el minorista y tiene como fin establecer la cantidad inicial de producto que tendrá el almacén del minorista.

#### **Descripción:**

El caso de uso inicia cuando el minorista se declara disponible para dar servicio por primera vez. En este caso, el minorista establece la cantidad de producto que tiene disponible en el almacén.

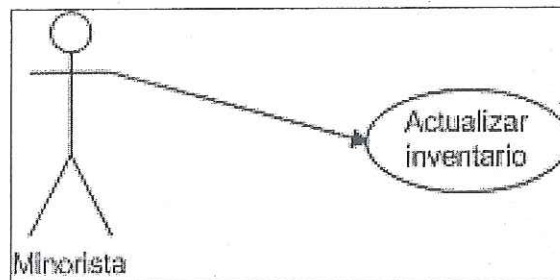


Figura 63. Diagrama del caso de uso *Actualizar inventario*.

El caso de uso *Actualizar inventario* (figura 63) tiene como finalidad reflejar en el inventario las entradas y salidas de producto en el almacén. Es llevado a cabo por el minorista.

**Descripción:**

El caso de uso inicia cuando hay compra o venta de productos por parte del minorista.

Cuando esto sucede, se añade o resta la cantidad correspondiente al inventario.

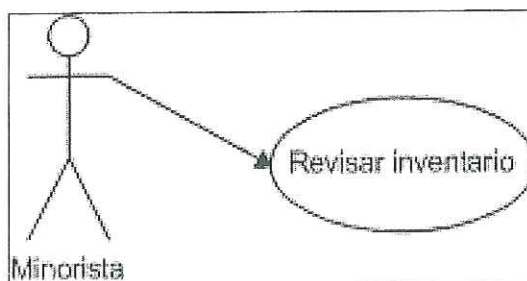


Figura 64. Diagrama del caso de uso *Revisar inventario*.

El caso de uso *Revisar inventario* (figura 64) es llevado a cabo por el minorista. La finalidad de este caso de uso de revisar si existe en el almacén la cantidad de producto suficiente para atender las demandas de los clientes.

**Descripción:**

El caso inicia cuando el minorista desea saber si la cantidad de producto existente en el almacén es suficiente para atender las demandas de los clientes. Para este fin, el minorista compara la cantidad existente con otra cantidad (ya sea el mínimo establecido o la cantidad solicitada por el cliente) y el caso de uso termina una vez que se tiene el resultado de la comparación.

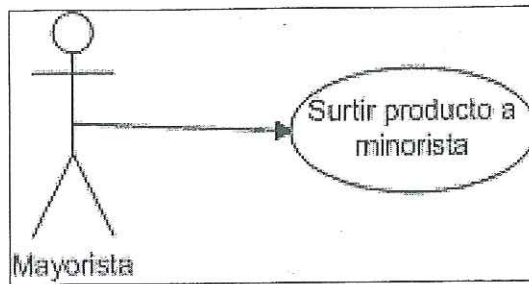


Figura 65. Diagrama del caso de uso *Surtir producto a minorista*.

El caso de uso *Surtir producto a minorista* (figura 65) tiene como finalidad atender las solicitudes de producto que ha hecho el minorista.

### Descripción:

El caso de uso inicia cuando el actor desea atender las solicitudes de pedido. El mayorista revisa la lista de pedidos, envía la respuesta al minorista y marca el pedido como surtido.

Una vez presentados los casos de uso del proceso del juego de al cerveza, se muestran también los diagramas de secuencia que corresponden a estos casos de uso.

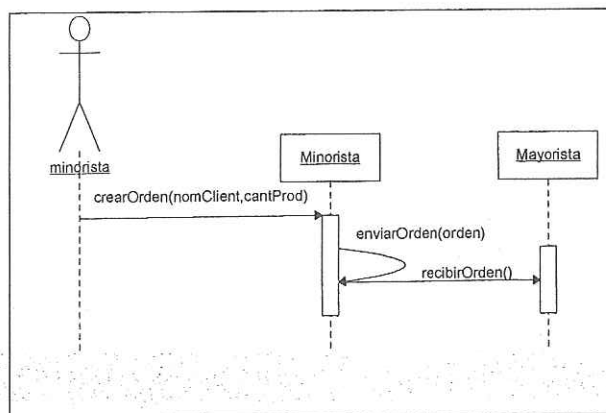


Figura 66. Diagrama de secuencia *Hacer pedido de producto*.



En la figura 66 se puede apreciar el diagrama de secuencia perteneciente al caso de uso *Hacer pedido de producto*. En este diagrama se puede apreciar que el actor minorista invoca al método `crearOrden(nomClient, cantProd)` del *objeto minorista*, y envía como parámetro el nombre que identifica al minorista (por medio de la variable *nomClient*) y la cantidad de producto que desea obtener (a través de la variable *cantProd*). Una vez que se crea la orden, se ejecuta el método `enviarOrden(orden)` en el mismo objeto, y posteriormente se invoca al método `recibirOrden()` en el *objeto mayorista*, el cual debe recibir e interpretar los datos que contiene la variable *orden*.

El diagrama de secuencia que pertenece al caso de uso *Surtir pedido a cliente* (figura 67) comienza cuando el actor minorista decide dar servicio al cliente. Para esto revisa los pedidos existentes por medio del método `revisarPedidos()` del objeto minorista. El objeto minorista retorna una lista con el detalle de los pedidos existentes y el actor minorista elige el pedido que va a atender por medio de la invocación del método `atenderPedido(numPedido)`, y envía como parámetro el número de pedido que desea atender (por medio de la variable *numPedido*). Este método a su vez obtiene, por medio del método `obtenerNomCli()`, el nombre del cliente que realizó el pedido. Asimismo, obtiene también la cantidad de producto solicitada, esta vez por medio del método `obtenerNumProd()` y almacena este valor en la variable *cantProd*.

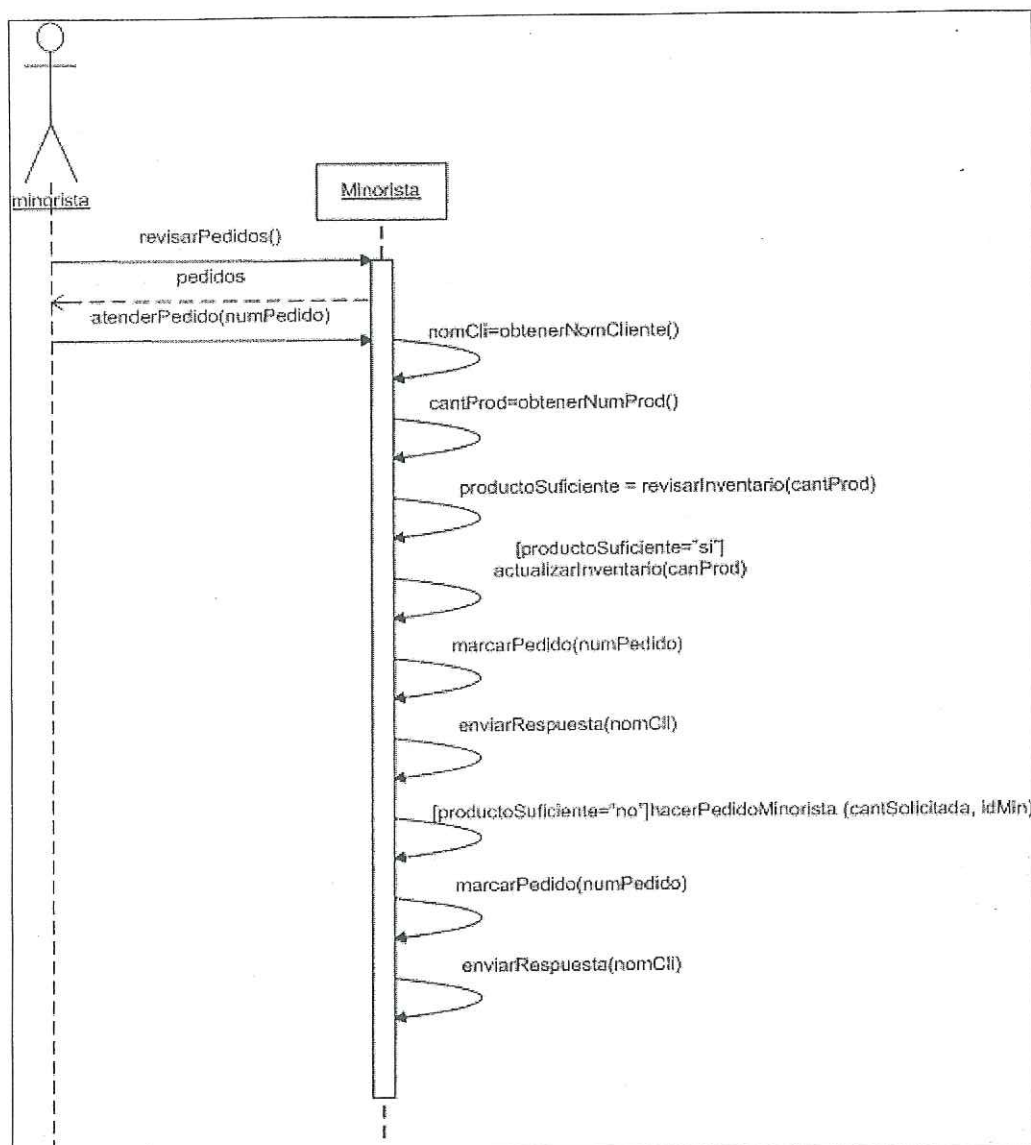


Figura 67 Diagrama de secuencia *Surtir pedido a cliente*.

Una vez que obtiene la cantidad de producto solicitada, el método *revisarInventario(cantProd)* verifica si cuenta en el almacén con la existencia de producto suficiente para cumplir con el pedido. En caso de que la existencia de producto sea necesario, el método *actualizarInventario(cantProd)* se encarga de actualizar el inventario. Enseguida, el método *marcarPedido(numPed)* se encarga de etiquetar el pedido como

“atendido” y finalmente se envía la respuesta al cliente mediante el método *enviarRespuesta(nomCli)*.

Por otra parte, si la cantidad de producto existente en el almacén no es suficiente para cubrir la demanda del cliente, se hace un pedido al minorista por medio del método *hacerPedidoMinorista(cantSolicitada, idMin)*, se etiqueta el pedido como atendido y se envía la respuesta al cliente.

La figura 68 presenta el **diagrama de secuencia** del caso de uso *Surtir producto a minorista*, el cual comienza cuando el *mayorista* decide atender las solicitudes que ha realizado el minorista.

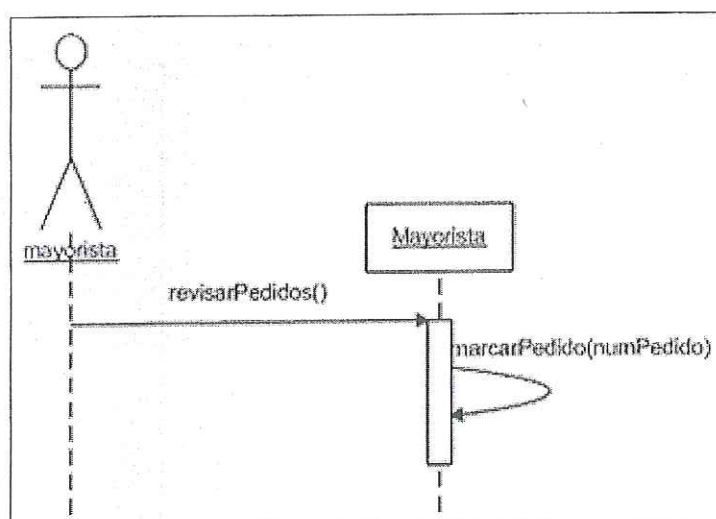


Figura 68. Diagrama de secuencia *Surtir producto a minorista*.

El mayorista revisa los pedidos existentes por medio del método *revisarPedidos()*, atiende el pedido e indica que éste fue atendido por medio del método *marcarPedido()*.

El **diagrama de secuencia** del caso de uso *Monitorear respuesta()* mostrado en la figura 69 comienza cuando el cliente desea saber si su pedido ya ha sido atendido, para lo cual invoca al método *revisarPedido(numPedido)* y pasa como parámetro el número que identifica al pedido. Posteriormente, se obtiene el estado del pedido a través del método *obtenerEdoPed(numPedido)*.

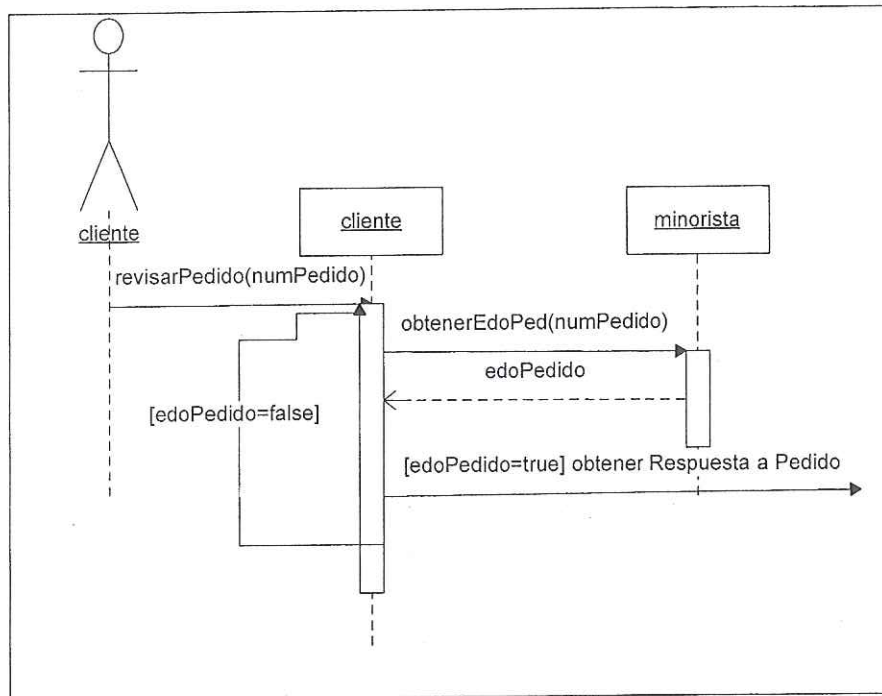


Figura 69. Diagrama de secuencia *Monitorear Respuesta*.

Si el pedido ya fue atendido, se ejecuta el caso de uso *Obtener respuesta a pedido*. En caso contrario, se revisa nuevamente el pedido.

Por otra parte, la figura 70 muestra el diagrama de secuencia del caso de uso *Obtener respuesta a pedido* el cual comienza cuando el cliente desea obtener la respuesta al pedido realizado, para lo cual invoca al método *obtenerRespuesta(numPedido)* del objeto *Minorista*, y este método retorna la cantidad de producto solicitada y el estado de pedido.



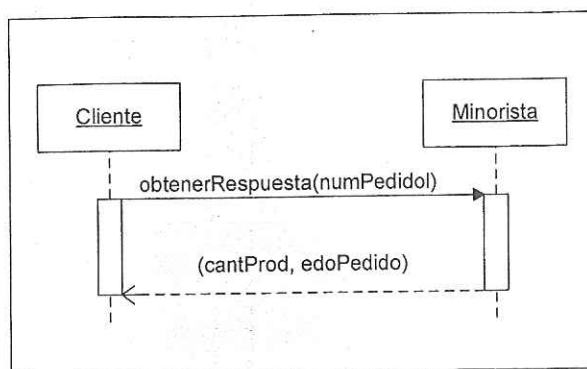


Figura 70. Diagrama de secuencia *Obtener respuesta a pedido*.

La figura 71 muestra el **diagrama de secuencia** del caso de uso *Actualizar inventario* el cual contiene un solo método que es invocado por el actor y consiste en invocar el método *actualizarInventario(cantProd, operación)*, donde se pasa la cantidad de producto con la cual se va a actualizar el inventario y la operación que se realizó, de tal manera que si se realizó una venta, se reduzca el inventario y si realizó una compra se aumente el mismo.

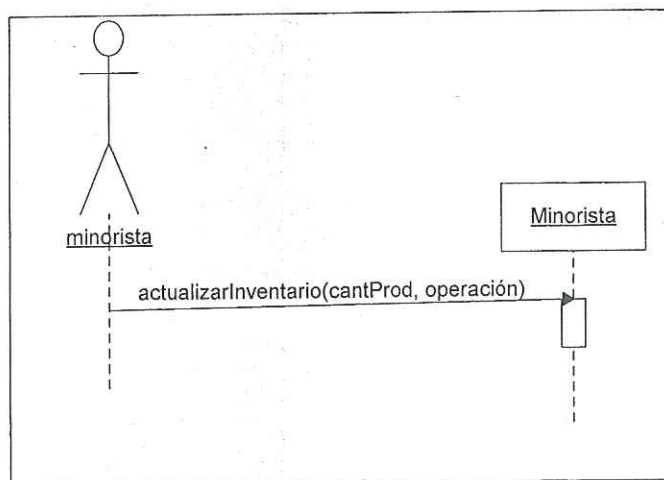


Figura 71. Diagrama de secuencia *Actualizar inventario*.

## **Apéndice B. Aspectos técnicos de la implementación de los prototipos basados en la arquitectura WODIS.**

---

En este apéndice se muestra la implementación del mecanismo *Emisor de mensajes*, el cual emplea CORBA. Además, se presenta una guía para poner en funcionamiento los prototipos desarrollados y finalmente se muestra la definición de los documentos utilizados en ellos.

### ***B.1 Desarrollo de la aplicación.***

En esta sección se muestra la manera en como se construye un objeto CORBA que contiene los métodos de un rol, las instrucciones en lenguajes JAVA para establecer la comunicación.

Para desarrollar una aplicación con soporte para la tecnología CORBA, se deben seguir los siguientes pasos:

- *Definir interfaz remota (IDL).*-El objeto que se va a implementar es el correspondiente al *Nodo Distribuidor*. La figura 72 muestra el IDL de este objeto. Cuando se construye un IDL, cada interfaz (*interface*) representa un objeto CORBA, y en ellas se declaran los métodos de dicho objeto que serán invocados de forma remota. Las interfaces deben ir agrupadas en módulos (*module*), y un módulo debe tener uno o más interfaces.

```

module Wodisflow
{
    interface NodoCentral
    {
        typedef string actividades[50][5];
        typedef string roles[8];
        typedef string datos[20][3];
        typedef string actores[5][6];
        roles validacion(in string usuario, in string contraseña);
        actividades retornaWorkflow(in string rol);
        datos retornaDatos(in string rol);
        actores retornaActores ();
        void actualizaActores(in string usuario, in string status);
        /**Métodos agregados con respecto al prototipo anterior
        typedef string tareasPendientes[50][5];
        typedef string eBitacora[1000][7];
        typedef string rBitacora[1000][8];
        tareasPendientes retornaTareasPendientes(in string usuario);
        rBitacora retornaBitacora(in string usuario);
        void escribeBitacora(in eBitacora transacción);
        void escribeTareasPendientes(in tareasPendientes tareas);
    };
};

```

Figura 72. IDL del objeto *Nodo Distribuidor*.

Ya que se han diseñado las interfaces, se guardan en un archivo con extensión IDL (*wodisflow.idl* para este ejemplo).

- *Compilar el IDL.*- El lenguaje IDL no puede ser entendido por los compiladores de los lenguajes de programación existentes hasta hoy, por lo que es necesario mapearlo a alguna forma que si pueda ser entendida por ellos. También es necesario crear los interpretes de las sentencias CORBA (*stub* en el cliente y *skeleton* en el servidor), los cuales tienen como función codificar las peticiones y resultados de la comunicación con algún objeto remoto, de manera objetos CORBA puedan comunicarse entre si, sin importar el lenguaje de programación en el que estén desarrollados.

Existe mucha información disponible sobre la equivalencia de una sentencia IDL con respecto a algún lenguaje de programación, así como también sobre la forma en como construir de forma manual los intérpretes de las sentencias CORBA. Sin embargo, la mayoría de los lenguajes de programación ofrecen herramientas que automatizan este proceso, por lo que en el desarrollo de este prototipo se hizo uso de *JAVA IDL*, ofrecida por el compilador JAVA (J2SE 1.4.2).

La sentencia *idlj -fall wodisflow.idl* crea las interfaces en código JAVA, así como los intérpretes para estos objetos. Los archivos que se generan son:

- *\_NodoDistribuidorPOA.java*.- Esta clase abstracta es el esqueleto del servidor (*skeleton*), proporcionando funcionalidades básicas de CORBA para el servidor. Implementa la interfaz *NodoDistribuidor.java*. La clase servidor *NodoDistribuidorServant* descende de *\_NodoDistribuidorImplBase*.
- *\_NodoDistribuidorStub.java*.- Esta clase es el cliente, proporciona funcionalidad CORBA al cliente. Implementa la interfaz *NodoDistribuidor.java*.
- *NodoDistribuidorOperations.java*.- Esta interfaz contiene la versión Java de nuestra interfaz IDL. Contiene la declaración de los métodos definidos en el IDL.
- *NodoDistribuidor.java*.- La interfaz *NodoDistribuidor.java* descende de *org.omg.CORBA.Object*, proporcionando funcionalidades estándares de CORBA.



- *NodoDistribuidorHelper.java*.- Esta clase final proporciona funcionalidades auxiliares, notablemente el método *narrow* requerido para convertir las referencias de los objetos CORBA a sus propios tipos.
- *NodoDistribuidorHolder.java*.- Esta clase final proporciona operaciones para argumentos de entrada y salida, que CORBA tiene pero que no se mapean fácilmente a la semántica Java.
- *Implementar el servidor*.- Un servidor CORBA requiere una clase *Sirviente*, que es donde se define el comportamiento de los métodos remotos, además de la clase servidor, en la cual se implementa la comunicación.

```
//Se importan los paquetes necesarios
import org.omg.CORBA.ORB;

//Se declara la clase. Es importante notar que Wodisflow se
//refiere al nombre del módulo que contiene la interfaz
//NodoCentral.

public class NodoCentralServant extends Wodisflow.NodoCentralPOA
{
    //Se define el comportamiento de los métodos de igual manera
    //a como se hace en una aplicación convencional JAVA.

    public void validacion(String usuario, String contraseña )
    {
        //Aquí el comportamiento de este método
    }
}
```

Figura 73. Clase *NodoDistribuidorServant* del Prototipo del proceso de reclutamiento, selección y contratación de personal en CICESE.

En la figura 73 se aprecia la implementación de la clase *NodoDistribuidorServant*. El código que define el comportamiento del método *validacion(String usuario, String contraseña)* no es mostrado debido a que no es relevante al funcionamiento de las aplicaciones basadas en CORBA.

```

//Se declara la clase servidor
public class NodoCentralServer {

    public static void main( String args[] ) {
        //Se establece el nombre de la máquina y puerto a utilizar.
        Properties properties = System.getProperties();
        properties.put( "org.omg.CORBA.ORBInitialHost",
            "PC-Coolab6" );
        properties.put( "org.omg.CORBA.ORBInitialPort",
            "1950" );

        try {
            // Se hace una instancia al ORB.
            ORB orb = ORB.init(args, properties);

            // Se hace una instancia a la clase Sirviente
            NodoCentralServant sirviente = new NodoCentralServant(orb);

            //Se crea un adaptador de objetos (POA)
            POA rootPOA = POAHelper.narrow(orb.resolve_initial_references("RootPOA"));

            Policy[] nodoCentralPolicy = new Policy[1];
            nodoCentralPolicy[0] = rootPOA.create_lifespan_policy(
                LifespanPolicyValue.PERSISTENT);

            POA nodoCentralPOA = rootPOA.create_POA("childPOA", null, nodoCentralPolicy );

            persistentPOA.the_POAManager().activate( );

            //Se asocia el POA con el sirviente
            nodoCentralPOA.activate_object( sirviente );

            //Se crea el servicio de nombres
            org.omg.CORBA.Object obj = orb.resolve_initial_references(
                "NameService" );
            NamingContextExt rootContext = NamingContextExtHelper.narrow( obj );

            //Aquí se le establece el nombre por el cual sera
            //localizado el servidor
            NameComponent[] nc = rootContext.to_name(
                "NodoCentral" );
            rootContext.rebind( nc, persistentPOA.servant_to_reference(
                servant ) );

            //Se echa a andar el ORB
            orb.run();
        } catch ( Exception e ) {
            System.err.println( "Exception in Persistent Server Startup " + e );
        }
    }
}

```

Figura 74. Clase NodoDistribuidorServer del Prototipo del proceso de reclutamiento, selección y contratación de personal en CICESE.

Una vez que se tiene la *clase sirviente*, se define la clase servidor como se puede apreciar en la figura 74. La clase servidor se encarga de administrar las peticiones recibidas, de solicitar a la clase sirviente que ejecute los métodos solicitados y de retornar la respuesta de la ejecución del método al cliente que lo invocó.

- *Implementar el cliente.*- Posteriormente a la implementación del servidor CORBA, se desarrolla la aplicación que invoca estos métodos (figura 75).

```
//se importan los paquetes necesarios
import java.util.Properties;
import org.omg.CORBA.ORB;
import org.omg.CORBA.OBJ_ADAPTER;
import org.omg.CosNaming.NamingContext;
import org.omg.CosNaming.NamingContextHelper;
import org.omg.CosNaming.NameComponent;
import org.omg.PortableServer.POA;

import Wodisflow.NodoCentralHelper;
import Wodisflow.NodoCentral;

//se declara la clase cliente.
public class responsableAreaClient
{
    String rolesActor[][];

    public static void main(String args[])
    {
        try
        {
            //se hace una instancia del ORB
            ORB orb = ORB.init(args, null);

            //Se localiza el objeto CORBA por medio del servicio de
            //nombres
            org.omg.CORBA.Object obj = orb.string_to_object(
                "corbaname::localhost:1950#NodoCentral");

            NodoCentral nc = NodoCentralHelper.narrow( obj );

            //se invoca al método validación y los datos que retorna se
            //almacenan en el arreglo rolesActor
            rolesActor = nc.validacion("actor 1", "mi password");

        } catch ( Exception e ) {
            System.err.println( "Exception in PersistentClient.java..." + e );
            e.printStackTrace();
        }
    }
}
```

Figura 75. Clase responsableAreaClient del Prototipo del proceso de reclutamiento, selección y contratación de personal en CICESE.

Aquí se muestra la implementación de una aplicación que invoca el método *validación*( *String usuario*, *String contraseña*), el cual sirve para saber si un actor está autorizado a utilizar el sistema o no.

- *Arrancar las aplicaciones.*- Después de haber desarrollado las aplicaciones, es necesario compilarlas. Para esto se abre una ventana de comandos, se accede al directorio que contiene los archivos de las aplicaciones anteriormente presentadas y se teclea la siguiente instrucción:

```
javac *.java Wodisflow/*.java
```

Posteriormente, es necesario poner a funcionar la implementación del ORB. Para esto se teclea la siguiente instrucción:

```
orbd -ORBInitialPort 1950
```

Nótese que 1950 es el puerto sobre el cual se va a establecer la comunicación. Una vez que el ORB está funcionando, es necesario dar de alta el servidor y ponerlo en funcionamiento. Para esto se utiliza una nueva ventana de comando, se accede al directorio que contiene las clases de estos servidores y se teclea lo siguiente:

```
servertool -ORBInitialPort 1950
```

Con esto se inicia la herramienta *servertool*, la cual se encarga de administrar los objetos CORBA. Para registrar el servidor *NodoDistribuidorServer* se teclea lo siguiente:

```
servertool> register -server NodoDistribuidorServer -applicationName s1  
-classpath D:\WODIS
```



Finalmente es necesario poner en funcionamiento el cliente CORBA. Para esto se ejecuta la siguiente instrucción:

*java responsableAreaClient*

Y con esto la aplicación comienza su ejecución.

## **B.2 Funcionamiento del mecanismo Emisor de mensajes.**

La funcionalidad del mecanismo *Emisor de mensajes (EM)* consiste en establecer la comunicación entre la aplicación cliente y el objeto remoto que implementa algún método remoto al cual la aplicación cliente requiera invocar.

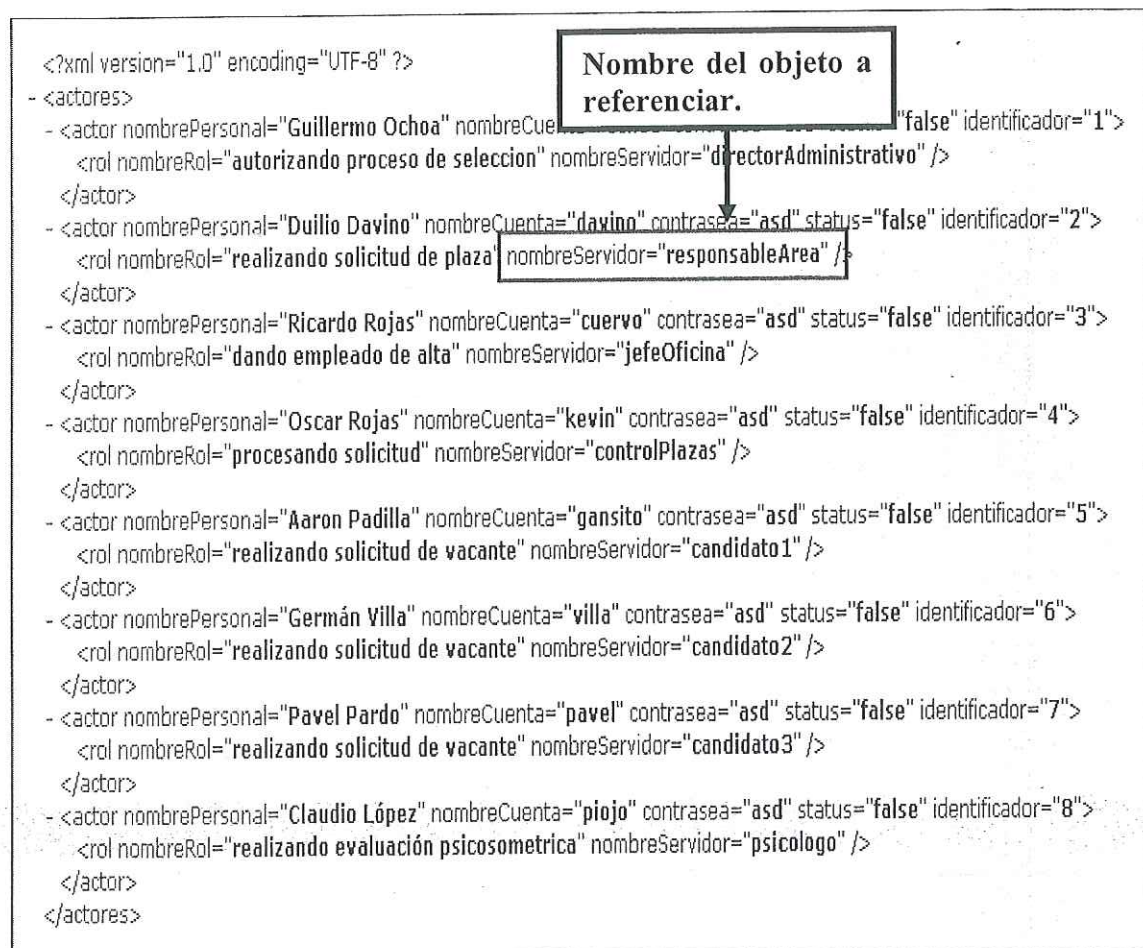


Figura 76. Directorio de actores del prototipo de reclutamiento, selección y contratación de personal.

Para lograr esto, el *EM* consulta en el directorio de actores (figura 76), el nombre del objeto a referenciar (atributo *nombreServidor* del elemento *rol* en dicho documento), y ese documento lo pasa al servicio de nombrado al momento de crear un objeto (figura 77), y con ese objeto se puede invocar el método que se desea.

```
//Se localiza el objeto CORBA por medio del servicio de
//nombres
org.omg.CORBA.Object obj = orb.string_to_object(
    "corbaname::localhost:1950#responsableArea");

//Se crea el objeto para establecer la comunicación
NodoCentral nc = NodoCentralHelper.narrow( obj );

//se invoca al método requerido
rolesActor = nc.cualquierMetodo( );
```

Figura 77. Porción de código del mecanismo *Emisor de mensajes*.

En la siguiente sección se presenta la DTD (Declaración de tipo de documento) de los documentos XML utilizados en este prototipo.

### ***B3. DTD de los documentos XML utilizados.***

Los documentos XML utilizados en este prototipo son explicados en el capítulo IV, por lo que en esta sección solamente se presenta la DTD de estos.

La figura 78 muestra la DTD del documento que define el *flujo de actividades del proceso* (*workflow.xml*).

```

<!DOCTYPE workflow [
  <|ELEMENT workflow (estadosRol)+ >
  <|ELEMENT estadosRol (estado+)>
  <|ATTLIST estadosRol
                                nombreRol CDATA #REQUIRED>
  <|ELEMENT estado (transicion+, subtransaccion*, comunicarseCon*)>
  <|ELEMENT subtransaccion EMPTY>
  <|ELEMENT transicion EMPTY>
  <|ELEMENT comunicarseCon EMPTY>
  <|ATTLIST estado
                                idEstado ID #REQUIRED
                                nombreActividad CDATA #REQUIRED
                                nivel CDATA #REQUIRED
                                estadoCompuesto (true|false) "false"
                                perteneceA CDATA "--"
                                pantalla CDATA #REQUIRED
                                >
  <|ATTLIST subtransaccion
                                nombreST CDATA #REQUIRED
                                estadoInicial IDREF #REQUIRED>
  <|ATTLIST transicion
                                estadoSiguiete CDATA #REQUIRED>
  <|ATTLIST comunicarseCon
                                rol CDATA #REQUIRED>
]>

```

Figura 78. DTD del documento de definición de flujo de trabajo (*workflow.xml*).

La figura 79 muestra la DTD del *directorio de actores* (*directorioActores.xml*).

```

<!DOCTYPE actores [
  <|ELEMENT actores (actor)*>
  <|ELEMENT actor (rol)+>
  <|ATTLIST actor
                                nombrePersonal CDATA #REQUIRED
                                nombreCuenta CDATA #REQUIRED
                                contraseña CDATA #REQUIRED
                                status (true|false) "false"
                                identificador CDATA #REQUIRED>
  <|ELEMENT rol EMPTY>
  <|ATTLIST rol
                                nombreRol CDATA #REQUIRED
                                nombreServidor CDATA #REQUIRED>
]>

```

Figura 79. DTD del documento *directorio de actores* (*directorioActores.xml*).

La figura 80 muestra la DTD del *manejador de instancias* (*instancias.xml*).

```
<!DOCTYPE instancia [
  <!--ELEMENT manejadorInstancias (instancia)*-->
  <!--ELEMENT instancia (estado)+-->
  <!--ELEMENT estados EMPTY-->
  <!--ATTLIST instancia
                                idInstancia ID #REQUIRED
                                rol CDATA #REQUIRED-->

  <!--ATTLIST estado
                                idEstado CDATA #REQUIRED
                                activo CDATA #REQUIRED
                                subtransaccion CDATA #REQUIRED
                                informacion CDATA #REQUIRED-->

1>
```

Figura 80. DTD del documento *manejador de instancias* (*instancias.xml*).

La figura 81 muestra la DTD del documento *bitácora* (*bitácora.xml*).

```
<!DOCTYPE bitacora [
  <!--ELEMENT bitacora (instancia)*-->
  <!--ELEMENT instancia EMPTY-->
  <!--ATTLIST transaccion
                                idInstancia ID #REQUIRED
                                idEstado CDATA #REQUIRED
                                status CDATA #REQUIRED
                                nivel CDATA #REQUIRED
                                paralelas CDATA #REQUIRED
                                subtransaccion CDATA #REQUIRED
                                información CDATA #REQUIRED -->

1>
```

Figura 81. DTD del documento *manejador bitácora* (*bitácora.xml*).



## **Apéndice C. Casos de prueba para el prototipo.**

En este apéndice se muestran los casos de prueba que se aplicaron al prototipo que se desarrolló en este trabajo, y que implementó el proceso de reclutamiento, selección y contratación de personal en CICESE.

### ***Caso de prueba para el Requerimiento 1: Coordinación descentralizada de las actividades***

Este caso de uso fue diseñado para probar si la arquitectura propuesta permite desarrollar sistemas de workflow distribuido, donde la coordinación de las actividades pueda darse de forma descentralizada.

Tabla XXXII. Caso de prueba de requerimiento 1.

| <b><i>Requerimiento 1 – Coordinación Descentralizada</i></b> |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|--------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Condición inicial</b>                                     | Solamente el <i>Nodo Distribuidor</i> está funcionando                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>Flujo de eventos</b>                                      | <ol style="list-style-type: none"><li>1. Se da de alta un actor en cada nodo.</li><li>2. Se comienza la ejecución de una instancia del proceso.</li><li>3. Se da de baja el <i>Nodo Distribuidor</i>.</li><li>4. Se continúa con la ejecución de la instancia.</li><li>5. Se da de baja el nodo <i>Director administrativo</i>.</li><li>6. Se da de baja el nodo <i>Psicólogo</i>.</li><li>7. Se da de baja el nodo <i>Jefe de área</i>.</li><li>8. Se da de baja el nodo <i>Candidato</i>.</li><li>9. Se da de baja el nodo <i>Jefe de oficina</i>.</li></ol> |
| <b>Condición final</b>                                       | <ul style="list-style-type: none"><li>• Cada nodo debe tener el flujo de actividades que le corresponden al actor que trabajó en ese nodo</li></ul>                                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>Observaciones</b>                                         | La condición final se cumplió adecuadamente.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |

### **Caso de prueba para el Requerimiento 2: Ejecución descentralizada de las actividades**

Este caso de uso fue diseñado para probar si la arquitectura propuesta permite desarrollar sistemas de workflow distribuido, donde la ejecución de las actividades pueda darse de forma descentralizada.

Tabla XXXIII. Caso de prueba de requerimiento 2.

| <b>Requerimiento 2 – Ejecución Descentralizada</b> |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|----------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Condición inicial</b>                           | Los actores del proceso ya se dieron de alta en el proceso.                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>Flujo de eventos</b>                            | <ul style="list-style-type: none"> <li>○ Se comienza la ejecución de una instancia del proceso.</li> <li>○ Se da de baja el <i>Nodo Distribuidor</i>.</li> <li>○ Se continúa con la ejecución de la instancia.</li> <li>○ Se da de baja el nodo <i>Director administrativo</i>.</li> <li>○ Se da de baja el nodo <i>Psicólogo</i>.</li> <li>○ Se da de baja el nodo <i>Jefe de área</i>.</li> <li>○ Se da de baja el nodo <i>Candidato</i>.</li> <li>○ Se da de baja el nodo <i>Jefe de oficina</i>.</li> </ul> |
| <b>Condición final</b>                             | <ul style="list-style-type: none"> <li>• Los actores deben poder ejecutar todas las actividades que les aparezcan disponibles.</li> </ul>                                                                                                                                                                                                                                                                                                                                                                       |
| <b>Observaciones</b>                               | La condición final se cumplió adecuadamente.                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |

### **Caso de prueba para el Requerimiento 3: Recuperación de errores.**

Este caso de uso fue diseñado para probar si la arquitectura propuesta permite desarrollar sistemas de workflow distribuido, donde se tenga consistencia en el proceso, aún cuando el sistema sufra algunas fallas.

Tabla XXXIV. Caso de prueba de requerimiento 3.

| <b>Requerimiento 3 – Recuperación de errores</b> |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|--------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Condición inicial</b>                         | Todos los nodos están funcionando normalmente                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>Flujo de eventos</b>                          | <ul style="list-style-type: none"> <li>○ Se comienza la ejecución de una instancia del proceso</li> <li>○ Se registran los estados actuales de las instancias del actor <i>Jefe de área</i>.</li> <li>○ Se apaga de forma incorrecta el nodo <i>Jefe de área</i>.</li> <li>○ Inicia sesión el actor del nodo <i>Jefe de área</i> en la misma PC.</li> <li>○ Se contrastan los estados obtenidos antes de apagar en nodo y después de iniciar el nodo.</li> <li>○ Se registran los estados actuales de las instancias del actor <i>Jefe de área</i>.</li> <li>○ Se apaga de forma incorrecta el nodo <i>Jefe de área</i>.</li> <li>○ Inicia sesión el actor del nodo <i>Jefe de área</i> en diferente PC.</li> <li>○ Se registran los estados actuales de las instancias del actor <i>Jefe de área</i>.</li> <li>○ Se contrastan los estados obtenidos antes de apagar en nodo y después de iniciar el nodo.</li> <li>○ Se apaga de forma incorrecta el nodo <i>Jefe de control de plazas</i>.</li> <li>○ Inicia sesión el actor del nodo <i>Jefe de control de plazas</i> en diferente PC.</li> <li>○ Se contrastan los estados obtenidos antes de apagar en nodo y después de iniciar el nodo.</li> </ul> |
| <b>Condición final</b>                           | <ul style="list-style-type: none"> <li>• Cada vez que se contrasten las listas de estados no debe haber diferencias entre la obtenida antes de apagar el nodo y después de iniciar el nodo.</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>Observaciones</b>                             | La condición final se cumplió adecuadamente.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |



### ***Caso de prueba para el Requerimiento 4: Distribución de las entidades de información.***

Este caso de uso fue diseñado para probar si la arquitectura propuesta permite desarrollar sistemas de workflow distribuido, donde la administración de la información que las actividades requieren pueda darse de forma descentralizada.

Tabla XXXIII. Caso de prueba de requerimiento 4.

| <b><i>Requerimiento 4 – Información Descentralizada</i></b> |                                                                                                                                                                                                                                                                                                                                                                                                                            |
|-------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Condición inicial</b>                                    | Los actores del proceso ya se dieron de alta en el proceso.                                                                                                                                                                                                                                                                                                                                                                |
| <b>Flujo de eventos</b>                                     | <ul style="list-style-type: none"> <li>▪ Se comienza la ejecución de una instancia del proceso.</li> <li>▪ Después de que el actor que desempeña el rol <i>Jefe de área</i> ejecuta la actividad <i>Enviar solicitud a control de plazas</i>, se da de baja el nodo <i>Jefe de control de plazas</i> y se inicia sesión con ese actor en otro nodo.</li> <li>▪ Se continúa con la ejecución normal del proceso.</li> </ul> |
| <b>Condición final</b>                                      | <ul style="list-style-type: none"> <li>• La actividad <i>Recibir solicitud</i> del rol <i>Jefe de control de plazas</i> debe permitir ver el expediente que envió el rol <i>Candidato</i>.</li> </ul>                                                                                                                                                                                                                      |
| <b>Observaciones</b>                                        | La condición final se cumplió adecuadamente.                                                                                                                                                                                                                                                                                                                                                                               |

### ***Caso de prueba para el Requerimiento 5: Reflejar forma natural de trabajo***

Este caso de uso fue diseñado para probar si la arquitectura propuesta permite desarrollar sistemas de workflow distribuido, donde pudiera verse la ejecución del proceso de acuerdo a la forma usual en como es llevado a cabo.



Tabla XXXVI. Caso de prueba de requerimiento 5.

| <b>Requerimiento 5 – Reflejar forma natural de trabajo.</b> |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|-------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Condición inicial</b>                                    | Todos los nodos están funcionando normalmente                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>Flujo de eventos</b>                                     | <ul style="list-style-type: none"> <li>• Se comienza la ejecución de una instancia del proceso</li> <li>• Se registran los estados actuales de las instancias del actor <i>Jefe de área</i>.</li> <li>• Se apaga de forma correcta el nodo <i>Jefe de área</i>.</li> <li>• Inicia sesión el actor del nodo <i>Jefe de área</i> en la misma PC.</li> <li>• Se contrastan los estados obtenidos antes de apagar en nodo y después de iniciar el nodo.</li> <li>• Se registran los estados actuales de las instancias del actor <i>Jefe de área</i>.</li> <li>• Se apaga de forma correcta el nodo <i>Jefe de área</i>.</li> <li>• Inicia sesión el actor del nodo <i>Jefe de área</i> en diferente PC.</li> <li>• Se registran los estados actuales de las instancias del actor <i>Jefe de área</i>.</li> <li>• Se contrastan los estados obtenidos antes de apagar en nodo y después de iniciar el nodo.</li> <li>• Se apaga de forma correcta el nodo <i>Jefe de control de plazas</i>.</li> <li>• Inicia sesión el actor del nodo <i>Jefe de control de plazas</i> en diferente PC.</li> <li>• Se contrastan los estados obtenidos antes de apagar en nodo y después de iniciar el nodo.</li> </ul> |
| <b>Condición final</b>                                      | <ul style="list-style-type: none"> <li>• Cada vez que se contrasten las listas de estados no debe haber diferencias entre la obtenida antes de apagar el nodo y después de iniciar el nodo.</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>Observaciones</b>                                        | La condición final se cumplió adecuadamente.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |