CENTRO DE INVESTIGACION Y DE ESTUDIOS AVANZADOS
DEL INSTITUTO POLITECNICO NACIONAL
UNIDAD ZACATENCO
DEPARTAMENTO DE INGENIERIA ELECTRICA
SECCION COMPUTACIÓN

SAWF-HW: Sistema de Administración de Workflow Basado en el Modelo de Coordinación de Hyperworlds

Tesis que presenta el

Lic. Juan Manuel Correa Hernández

Para obtener el grado de

Maestro en Ciencias

En la especialidad de

Ingeniería Eléctrica Opción Computación

Director de la tesis:

Dr. José Oscar Olmedo Aguirre

Ciudad de México, Octubre de 2002

Agradecimientos

Quiero agradecer…

A CONACyT por la beca que me otorgó para la realización de mis estudios de maestría

A mi director de tesis, el Dr. José Oscar Olmedo Aguirre, por el apoyo y los conocimientos aportados al desarrollo de esta tesis

A mis sinodales, la Dra. Ana Maria Martínez Enríquez y la Dra. Xiaoou Li, por sus comentarios y el tiempo dedicado a la revisión de este trabajo

A mis profesores, por los conocimientos que recibí en sus clases

A la Sección de Computación del CINVESTAV, por los recursos proporcionados para el desarrollo de mi trabajo de tesis

Dedico este trabajo…

A mi madre, Maria, por el gran ejemplo de esfuerzo y constancia que eres para mí

A mis hermanos, Paty, Lupita, Angel y Toñi, por apoyarme y hacerme sentir cerca en todo momento

A Mónica, por tu amor y paciencia ;)

A Eduardo, por tus consejos y tu apoyo al momento de tomar la decisión de iniciar la maestría

A Perla, por tu amistad y consejos en los momentos difíciles

A mis amigos y compañeros de la maestría, por la amistad y los buenos momentos que existieron a lo largo de estos dos años que siempre recordaré

A todos los que creyeron que podía llevar a cabo este proyecto, gracias!!!

Resumen

Este trabajo describe el desarrollo de un prototipo experimental de un sistema de administración de *Workflow* llamado SAWF-HW que muestra la conveniencia de un enfoque de coordinación para resolver los problemas que se presentan en el diseño de sistemas de este tipo. La tecnología de *Workflow* tiene por objetivo la automatización de las actividades de grupos de personas que colaboran persiguiendo metas comunes. La coordinación de dichas actividades es tan compleja como la meta en común, por ello es necesario proponer alternativas de coordinación de actividades que faciliten el proceso de trabajo en grupo.

Los modelos de coordinación ofrecen un marco abstracto para resolver los problemas que se generan en ambientes de colaboración o de trabajo en grupo. El prototipo descrito en este documento usa la definición del lenguaje de coordinación *Hyperworlds* así como la implementación actual de la arquitectura de coordinación de *Hyperworlds*, como base de su instrumentación. Las actividades son conducidas bajo este modelo, el cual realiza de manera uniforme la administración de actividades distribuidas, la colaboración síncrona y asíncrona entre usuarios y agentes de software.

Para mostrar la capacidad del enfoque propuesto y del prototipo SAWF-HW desarrollado, se describe también, la solución al problema de la coordinación de actividades en una conferencia. Con lo anterior no se pretende competir con los sistemas de administración *Workflow* existentes, si no considerar la complejidad que tienen este tipo de sistemas para proponer el enfoque de coordinación como una solución viable de diseño para la integración uniforme de diferentes paradigmas de colaboración.

Abstract

In this thesis, the development of a research prototype Workflow management system is described. The system, called SAWF-HW, shows the convenience of the system coordination approach to address the problems involved in designing this kind of systems. One of the aims of Workflow technology is in providing methodologies and tools for automating the activities of people that works pursuing common goals. The management of such activities is as complex as the common goal, and for that reason, it is necessary to consider alternative proposals for the appropriate ordering of activities that simplify the collaboration among the participants.

Coordination models offer an abstract framework to approach the problems involved in computer mediated collaboration. The prototype described in this thesis has adopted the HyperWorlds coordination model as the basis for its system design and the current implementation of the HyperWorlds coordination architecture as the basis for its development and deployment in Internet. The activities among participants in this model approach uniformly the management of distributed interactions and the synchronous and asynchronous collaboration among users and software agents.

In order to show the convenience of the proposed approach and also to validate the implementation of the SAWFH-HW system, a solution for the conference management problem is formally described, simulated and executed. The purpose in developing such an application is mainly to demonstrate in practical terms that system coordination is a viable approach for designing and developing Workflow systems in a powerful framework that can uniformly integrate other approaches of collaboration.

Contenido

Figuras

Tablas

Capítulo 1 Introducción

1.1 Area de estudio

El concepto de *Workflow* (*WF*) evolucionó de la noción de proceso en la industria de manufactura y la oficina. Este tipo de procesos existen desde la industrialización y son producto de la investigación para incrementar la eficiencia mediante la concentración de aspectos rutinarios en actividades de trabajo. Para lograr esto, las actividades de trabajo se separan en tareas bien definidas, roles, reglas y procedimientos que regulan la mayor parte del trabajo en la industria de manufactura y la oficina. Inicialmente, los procesos eran llevados acabo por personas que manipulaban objetos físicos. Con la introducción de la tecnología de la información, tales procesos en el lugar de trabajo se automatizaron de manera parcial o total mediante el uso de sistemas de información, es decir, los programas de computadora ejecutan tareas y hacen cumplir las reglas que antes eran llevadas a cabo por personas.

Un *WF* puede verse como una colección de tareas organizadas para llevar a cabo un proceso de negocio, por ejemplo, procesar ordenes de compra o solicitudes de seguros. Una tarea puede ser ejecutada por uno o más sistemas de software, una o más personas, o una combinación de ambos. Las tareas humanas o las que son llevadas a cabo por personas, incluyen una interacción con las computadoras que puede ser cercana, esto se ejemplifica al proporcionar comandos de entrada, o lejana al usar la computadora solo para verificar el avance de una tarea. Por ejemplo, las tareas pueden ser actualizar un archivo o base de datos, crear una factura o enviarla por correo electrónico, etc.

Además de tener una colección de tareas, un WF define el orden de invocación de las tareas o las condiciones sobre las cuales las tareas se invocan, la sincronización de las tareas y el flujo de información.

La tecnología de *WF* facilita la automatización, ya que provee metodologías y software necesarios para apoyar la modelación de procesos de negocios, de esta forma un *WF* describe las tareas de un proceso a un nivel conceptual para entenderlo, evaluarlo y rediseñarlo.

1.2 Motivación

El problema que se plantea en esta tesis es encontrar nuevas formas de aplicar la infraestructura actual de las comunicaciones y la computación para facilitar y extender el proceso de trabajo en grupo. Para conseguir esto, es necesario encontrar mecanismos para coordinar las actividades de grupos de personas que colaboran persiguiendo metas comunes. La coordinación, como la define Wegner [2], es la administración de las interacciones entre los componentes de un sistema compuesto. Evitando la composición de componentes síncronos, la coordinación entre entidades de procesamiento débilmente acoplados da lugar a arquitecturas con comportamientos más variados y complejos.

El estudio de la coordinación puede justificarse por diferentes razones, en particular, por el hecho de que las computadoras ya no están aisladas sino que se encuentran generalmente conectadas en red. De ésta forma un gran número de personas pueden usar las capacidades de cómputo y comunicaciones para realizar su trabajo, esto es lo que se denomina comúnmente como *trabajo cooperativo asistido por computadora* o *groupware*. Una forma de trabajo cooperativo es la tecnología de *Workflow* (*WF*) que se caracteriza por la interacción asíncrona de sus participantes.

La tecnología de *WF* tiene como idea principal la automatización de procesos de negocios; la descripción precisa de estos procesos se realiza mediante un *lenguaje de definición de procesos*. La interpretación de un proceso de negocio descrito en el lenguaje de definición de procesos se realiza por un *Sistema de Administración de Workflow* (*SAWF*).

1.3 Trabajos relacionados

La coordinación de actividades en ambientes de colaboración puede llevarse a acabo usando un modelo de coordinación, el cual es un marco abstracto que sirve para estudiar y entender problemas de diseño de programas concurrentes y programas distribuidos. Un lenguaje de coordinación, el cual es la representación lingüística de un modelo de coordinación, incluye mecanismos claramente definidos para coordinación, sincronización, distribución y control de concurrencia. A la fecha se han propuesto varios modelos y lenguajes de coordinacion como *Linda* [3], *GAMMA* [4], *CHAM* [5] y *PoliS* [6]. El más conocido de ellos es el lenguaje *Linda* que consiste de un conjunto de operaciones simples que tienen efecto sobre una estructura de datos llamada espacio de tuplas, y permiten la comunicación entre agentes o procesos. Una propuesta nueva en lenguajes de coordinación es *Hyperworlds* [32](*HW*), que extiende a *Linda*, el cual es simbólico y distribuido, cuyo modelo de coordinación se uso en este trabajo como la base del diseño e implementación arquitectura de un prototipo de *SAWF* con la finalidad de administrar las interacciones que se presentan en este tipo de sistemas.

Este trabajo incluye ademas una revisión del estado del arte de los *SAWFs* más conocidos, como son Domino [17], IBM FlowMark [24] y SAP Business Workflow [27], de los cuales se hace un analisis de sus elementos de modelacion, herramientas, arquitectura y área de aplicación. Tambien se describe el modelo de *WF* propuesto por Jablonski [30], el cual incluye las perspectivas más importantes las cuales debe tener un *SAWF*, cabe mencionar que este modelo es usado en la mayoría de los *SAWFs*. La revisión del estado del arte, así como la descripción del modelo de *WF*, no tienen el objetivo de que el prototipo propuesto compita con los sistemas ya existentes, pero si conocer los aspectos de diseño e implementación más significativos los cuales permitan tener un prototipo sobre el cual mostrar la conveniencia del enfoque de coordinación y que tenga características básicas que permitan seguir la investigación en los aspectos diseño de sistemas de este tipo que no se incluyen en este trabajo.

Además, que en el futuro se avance en aspectos de implementación que permitirán agregar más funcionalidad al prototipo, lo cual permita tener un *SAWF* cada vez más robusto.

1.4 Objetivos

El objetivo de esta tesis es desarrollar una arquitectura de coordinación distribuida en Internet basada en el modelo y lenguaje de coordinacion *HW* que facilite la colaboración asíncrona (*WFs*) entre usuarios, agentes y herramientas de software.

Partiendo de la definición del lenguaje y de la implementacion actual de la arquitectura de coordinacion de *HW*, la contribución de este trabajo se describe en las siguientes actividades:

- Actualizar la correspondencia entre las nociones de procesos de negocios y las nociones básicas del lenguaje de coordinacion *HW*
- Utilizar la arquitectura de coordinacion del lenguaje *HW* como base del diseño de un *SAWF*
- Utilizar el lenguaje *HW* como lenguaje de coordinación de procesos en la administración de actividades de una conferencia.

1.5 Comentarios finales

Durante la realización de este trabajo, se analizó la correspondencia entre las nociones de entidad activa del lenguaje *HW* y de participante de un proceso de negocios, entidad pasiva y de documento electronico, por último, las reglas de coordinacion y de actividades en procesos de negocios asi como las estructuras de control básicas en los procesos de negocios como secuencia, alternativa y concurrencia; con la finalidad de poder describir los problemas de administración de interacciones en términos del lenguaje. Después se adapto la arquitectura de coordinación al modelo cliente servidor de Internet, de la cual se usa su infraestructura como medio de comunicación entre sus componentes, en particular usa los protocolos *HTTP*, *SMTP* y *POP*. Para mostrar el uso y la capacidad del prototipo implementado se desarrollo una aplicación como caso de estudio que muestra la interacción entre los participantes y las actividades de una conferencia. Para esto se describieron las

actividades así como la interacción entre los participantes de la conferencia, usando el lenguaje *HW*.

En resumen, ésta tesis se aplican conceptos de coordinación y tecnología de *WF*. Con tal fin, se estudiaron bs modelos de coordinación y su uso para solucionar los problemas de ambientes de coordinación, así como la descripción del diseño e implementación del prototipo; finalizando con un manual de usuario.

1.6 Organización de la tesis

Este documento se organiza en 3 partes, la primera describe la teoría que forma la base de ésta tesis así como el estado del arte, esta parte consta de los siguientes capítulos:

- El Capítulo 1 es ésta introducción.

- En el Capítulo 2 se describen los conceptos que forman la base de este trabajo. La descripción incluye conceptos de coordinación, así como la tecnología de *WF*.

- El Capítulo 3 presenta una revisión de los *SAWFs* más conocidos, también se hace una análisis de sus aspectos de diseño más sobresalientes así como de sus áreas de aplicación.

La segunda parte describe los modelos de coordinación, particularmente el modelo de HW y su aplicación para solucionar el caso de estudio que describe la coordinación de actividades en una conferencia, esta parte contiene:

- El Capítulo 4 se describen las características de los modelos y lenguajes de coordinación, en particular, el modelo del lenguaje de coordinación *HW*.

- El Capítulo 5 se presenta el caso de estudio seleccionado para mostrar el uso del enfoque que se planteo en capítulos anteriores. Se describen a los participantes y las interacciones que existen en una conferencia, también se explican las soluciones empleando el lenguaje de coordinación *HW*.

La tercera y última parte presenta la implementación del *SAWF*, el cual fue diseñado e implementado en esta tesis, esta parte consta de:

- El Capítulo 6 describe la arquitectura del *SAWF-HW*, sus componentes y su forma de comunicación, así como su funcionamiento.

- El Capítulo 7 es básicamente el manual de usuario del prototipo del *SAWF-HW*, se explica como ponerlo en ejecución y la manera en la que los usuarios pueden interactuar con él.

- Por último, en el Capítulo 8 se describen las conclusiones de este trabajo y se menciona el trabajo futuro.

Capítulo 2 Teoría de la coordinación y tecnología de Workflow

En este capítulo se describen los conceptos que forman la base de este trabajo. La descripción incluye conceptos de coordinación y tecnología de *WF* así como su relación con algunas otras áreas de investigación. También se describen las perspectivas necesarias en un modelo de *WF* y una clasificación de los tipos de *SAWFs*. Por último se describen algunas herramientas útiles en los *SAWFs*.

2.1 Coordinación

La teoría de la coordinación es un área de investigación reciente, en la cual se usan y extienden ideas acerca de la coordinación en diferentes disciplinas tales como la ciencia de la computación, la teoría de organización, investigación de operaciones, economía, lingüística y psicología [1]. Coordinación puede definirse como el proceso de administración de dependencias entre actividades. En años recientes creció el interés en conocer cómo actividades de sistemas complejos pueden ser coordinadas. Los enfoques que se presentan pueden ser en coordinación sistemas de cómputo paralelo y distribuido, en coordinación de sistemas humanos, o en sistemas complejos que incluyen personas y computadoras.

Debido a los cambios tecnológicos que ocurren en las computadoras y a la necesidad de tener herramientas más poderosas que permitan solucionar problemas cada vez más complejos, los paradigmas de computación tienen una evolución constante, ésta evolución también incrementó el interés por el estudio de la coordinación. En un principio se uso la programación en lenguaje máquina, la cual era difícil usar y entender. Luego la programación orientada a procedimientos trajo un cambio en la granularidad de las

operaciones. Después, con la programación orientada a objetos ocurrió un verdadero aumento de expresividad de los programas de algorítmicos a sistemas interactivos. Recientemente tenemos tecnología de software basado en componentes, promueve un cambio de la interacción secuencial a la interacción concurrente.

Los algoritmos que han dominado al mundo de la computación por muchos años resultan menos indicados para describir el comportamiento de sistemas cada vez más abiertos aunado a la necesidad de describir la interacción de sus componentes internos y con el ambiente que los rodea. La coordinación puede usarse para describir ésta interacción; de ésta forma la coordinación se refiere al manejo de interacciones entre componentes de un sistema compuesto. Las estructuras de control de los algoritmos son una forma altamente restringida de coordinación; en cambio, la coordinación de interacciones entre objetos y componentes de un sistema de software ésta sujeto a pocas restricciones y por lo tanto da lugar a mejores arquitecturas.

El estudio de la coordinación puede justificarse con diferentes razones, en particular por el hecho de que las computadoras ya no están aisladas, ahora contamos con una red para comunicarlas. De ésta forma un gran número de personas pueden usar las capacidades de cómputo y comunicaciones para realizar su trabajo, esto es lo que ahora denominamos *trabajo cooperativo asistido por computadora* o *groupware*. Estos sistemas ejecutan funciones tales que ayudan a personas colaborando en escribir el mismo documento, manejar proyectos, mantener la pista de tareas y encontrar, ordenar y priorizar mensajes electrónicos. Los conceptos de coordinación se usan para sugerir nuevas ideas de diseño.

2.2 Tecnología de Workflow

Una forma de trabajo cooperativo es la tecnología de *WF* que se caracteriza por la interacción asíncrona de sus participantes. La tecnología de *WF* tiene como idea principal la automatización de procesos de negocios, que tiene una gran importancia en las organizaciones, teniendo gran impacto en sus operaciones, debido a que incrementa la productividad y reduce los costos de operación. Desde el punto de vista de la coordinación

de actividades entre organizaciones humanas, un *proceso de negocio* se refiere a las reglas de interacción entre los participantes de un grupo que colaboran para alcanzar objetivos comunes. En 1996, la Coalición para la administración basada en *WF* (Workflow Management Coalition, WfMC) define el término *WF* [35] de la siguiente manera:

La automatización de un proceso de negocio, en su totalidad o en parte, durante el cual documentos, información o tareas pasan de un participante a otro para realizar alguna acción de acuerdo a un conjunto de reglas de procedimiento.

La descripción precisa de las actividades y sus interdependencias se realiza mediante un *lenguaje de definición de procesos*. De acuerdo con la WfMC un lenguaje de definición de procesos [35] es:

La representación de un proceso de negocio en una forma que permita la manipulación automatizada, tal como su modelación o su interpretación por un Sistema de Administración de Workflow. La definición de un proceso consiste de una red de actividades y sus interrelaciones, criterios para indicar el inicio y la terminación del proceso, y la información sobre las actividades individuales, tales como los participantes, las aplicaciones de tecnología de información asociadas, etc.

La interpretación de un proceso de negocio descrito en el lenguaje de definición de procesos se realiza por el *Sistema de Administración de Workflow* (*SAWF*) [35]definido por la WfMC como:

Un sistema que define, crea y administra la ejecución de un WF mediante el uso de programas, corriendo en una o más máquinas virtuales de WF, las cuales son capaces de entender la definición del proceso, la interacción entre los participantes, y siempre que sea necesario, emplearán herramientas de la tecnología de la información así como de diversas aplicaciones.

Un usuario importante en un *SAWF* es el *modelador del WF*, quien tiene la tarea de describir, analizar y administrar el *WF* en su etapa de construcción y ejecución.

2.3 Tipos de Workflows

El tipo de *WF* a ser utilizado dependerá de los objetivos que una empresa u organización quiera alcanzar. En general, los *SAWFs* pueden dividirse en los siguientes tipos:

- *Producción*. Su objetivo es administrar un número grande de tareas similares y optimizar su productividad. Esto se logra mediante la automatización de tareas tantas como sea posible, hasta conseguir que la intervención humana se requiera sólo para administrar excepciones.
- *Administrativo.* Se caracteriza por su facilidad para definir procesos. Normalmente existen varias definiciones ejecutándose de forma concurrente por lo que se involucra a un número grande de empleados. En este tipo de sistemas lo importante es la flexibilidad y no la productividad.
- *Colaborativo.* Los sistemas de este tipo están orientados a los grupos que trabajan para lograr metas comunes. El tamaño de los grupos puede variar de equipos pequeños, equipos de proyecto, hasta grupos grandes de personas con intereses comunes. El uso de la Internet y la Web son un factor crítico para la comunicación del equipo. La definición de los procesos no es rígida y puede cambiar con frecuencia.
- *Ad-hoc*. Estos sistemas permiten a los usuarios crear y modificar sus definiciones de procesos rápidamente. Se maximiza la flexibilidad en áreas donde el rendimiento y la seguridad no son importantes.

Es importante mencionar que la clasificación anterior es muy general, ya que para cada tipo de *WF* podemos tener varios subtipos y características diferentes.

2.4 Areas relacionadas con la tecnología de Workflow

Como menciona Jablonski [30], existen áreas de investigación relacionadas con la tecnología de *WF*, como la administración de negocios, la arquitectura y modelación de empresa, la modelación de proceso de software y la teoría de la coordinación, de las que aprende y obtiene información sustancial acerca de los elementos fundamentales de un modelo de *WF*. A continuación se presentan los aspectos más importantes de éstas áreas.

2.4.1 Administración de Negocios

La administración de negocios tradicionalmente se ha dividido en organización estructural y organización operacional. La primera se refiere a la interrelación de elementos organizacionales básicos de una empresa. La segunda tiene como tarea principal la de ordenar la ejecución de los procesos de trabajo, es decir, ordenar los pasos de trabajo para formar procesos de trabajo globales. La organización estructural trata a la empresa como una institución, mientras que la operacional trata con el trabajo que ejecuta dicha institución.

El resultado principal de la organización estructural es la definición de elementos organizacionales (*ranuras, posiciones, instancias, departamentos, etc.*) y la asociación de responsabilidades. Una estructura organizacional también determina el flujo formal de información y objetos de trabajo (*documentos*).

La organización operacional estructura los procesos de trabajo en 4 subtareas:

- *Definición del contenido del trabajo.* Se refiere a como se determinan los objetos de trabajo y a las herramientas disponibles y de ayuda para llevarlos a cabo.
- *Definición de tiempo de ejecución de trabajo.* Trata con 3 temas, primero la ordenación de los pasos de trabajo que constituyen un proceso de trabajo. Segundo, se especifica el tiempo de ejecución total/ promedio/ mínimo /máximo de un paso de trabajo para después calcular el tiempo de ejecución acumulado de un proceso de

trabajo. Tercero, si es necesario se especifican el calendario de tiempos para la ejecución de los procesos de trabajo.

- *Definición del lugar de trabajo.* Para cada proceso de trabajo y cada paso de trabajo se determina donde se ejecuta.
- *Asociación del trabajo a elementos organizacionales*. El enlace más estrecho entre la organización estructural y la operacional es la asociación del trabajo a los elementos de la organización. Para cada paso de trabajo y para cada proceso de trabajo, se seleccionan uno o más elementos organizacionales deben para ejecutar el trabajo juntos o de forma exclusiva.

2.4.2 Arquitectura y Modelación de Empresa

Hay muchos métodos en el área de arquitectura y modelación de una empresa: CIM-OSA [8], Purdue Enterprise Reference Architecture [9], GRAI [10], IDEF [11], SADT [12], SSADM [13], TOVE [14], etc. Algunos de ellos se refieren a la especificación de CIM (Manufactura Integrada por Computadora). En este caso sólo se analizan los elementos de los modelos de empresa CIM-OSA y TOVE.

CIM-OSA usa los modelos de empresa para monitorear y controlar las operaciones diarias de la empresa. Para lograr esto CIM-OSA se enfoca en dos componentes:

- *Un modelo de empresa ejecutable* para la descripción total de la empresa, incluyendo todas sus funciones, información, recursos y organizaciones. También se cubren aspectos dinámicos y estáticos,
- *Una infraestructura integrada* conteniendo servicios de tecnología de la información y máquinas para la ejecución de un modelo bajo el control de los eventos diarios de la empresa.

CIM-OSA considera cuatro vistas de la empresa, a continuación se listan los elementos de cada una de ellas:

- *Vista funcional*. Evento, proceso de dominio, proceso de negocio, actividad de empresa y operación funcional. Estos elementos se usan para modelar la

funcionalidad de la empresa y su comportamiento, es decir el flujo de control. Los elementos orientados a la actividad de ésta vista están relacionados cercanamente a los *WFs*.

- *Vista de Información*. Elemento de información, objeto empresa, objeto vista, relaciones objeto y reglas de integridad. Estos elementos describen principalmente la perspectiva de información de una empresa. Ejemplos de estos objetos de empresa son los documentos, formas, archivos y ordenes.
- *Vista de Recursos.* Recurso y capacidad. Los recursos sirven de soporte para una o más actividades; por ejemplo, herramientas, pruebas, carros de transporte, robots y operadores.
- *Vista Organizacional.* Unidad organizacional, célula organizacional, responsabilidad y autoridad. Estos elementos definen la estructura organizacional de una empresa.

El otro enfoque de modelación de empresa es TOVE (Toronto Virtual Enterprise). TOVE se enfoca en la creación de un ambiente de diseño de una empresa que permite la exploración de deferentes diseños o modelos de una empresa a lo largo de varias perspectivas de calidad, costo y agilidad. A estas perspectivas se les llama ontologías. Una ontología es una descripción formal de entidades y sus propiedades. Las ontologías de TOVE son: actividades, estados, tiempo, recursos, calidad, costo, inventario, órdenes, partes, estructuras organizacionales y causalidades.

2.4.3 Modelación de Proceso de Software

Un proceso de software se define como un conjunto de pasos parcialmente ordenados para alcanzar una meta. De acuerdo a ésta definición los componentes de un proceso se denominan *elementos de proceso*; si tales elementos son atómicos, es decir, no tiene una subestructura entonces se denominan *pasos de proceso*. La descripción de un proceso la complementa Curtis [15], con una lista de conceptos básicos entre los cuales se encuentra *agente*, *role* y *artefacto.* Los *agentes* ejecutan los componentes de un proceso. Los pasos del proceso son representados de forma tal que juntos alcancen la meta. Típicamente, los

pasos de proceso manipulan *artefactos*. Curtis nombra cuatro perspectivas de un modelo de proceso:

- Perspectiva funcional
- Perspectiva de comportamiento
- Perspectiva organizacional
- Perspectiva de información

Los elementos de modelación antes descritos en las secciones 2.4.1 y 2.4.2 se pueden mapear fácilmente a éstas perspectivas.

2.4.4 Teoría de la Coordinación

Los enfoques anteriores están orientados a la práctica, en este caso se tiene un enfoque orientado a la teoría. La teoría de la coordinación provee una vista abstracta sobre problemas de coordinación. Debido que la administración de *WF* se puede considerar un problema específico de coordinación, la teoría de la coordinación puede ser de ayuda.

Coordinación es el acto trabajar juntos cooperativamente. El trabajo ésta encaminado a lograr una meta. De ésta manera, debe haber uno o más actores ejecutando actividades que están dirigidas a algún fin. Las actividades no son independientes unas de otras; la relaciones entre las actividades se llaman interdependencias.

La teoría de la coordinación tiene como idea principal entender los procesos básicos involucrados en la coordinación. Son cuatro las tareas principales de la coordinación:

- *Identificación de metas*. Se selecciona las metas de aquellas que valen la pena seguirse.
- *Mapear metas a actividades*. Las metas son descompuestas para identificar actividades. Las actividades son ejecutadas para lograr las metas.

Tabla 2-1 Areas relacionadas con la tecnología de administración de *WF*.

Area relacionada / **Perspectiva**	*Administración de Negocios*	*Arquitectura y modelación de empresa*	*Modelación de Proceso de Software*	*Teoría de la Coordinación*
Función	Pasos de trabajo	Procesos de dominio Procesos de negocio Operaciones funcionales Eventos Estados	Procesos Elementos de proceso Pasos de proceso	Actividades
Operación	Herramientas Dispositivos Computadoras	Recursos Capacidad Inventario	Artefactos	Recursos
Comportamiento	Orden de	Flujo de control en procesos	Flujo de control en procesos	Interdependencias entre actividades
Información	Flujo de información Objetos de trabajo	Elementos de información Objetos de empresa	Artefactos	Recursos
Organización	Elementos organizacionales Estructura organizacional Responsabilidad Asociación del trabajo	Unidades organizacionales Células organizacionales Responsabilidades Autoridades	Agentes Roles	Actores Recursos
Causalidad		Causalidades		Metas
Integridad y tolerancia a fallos		Reglas de integridad		
Calidad	Tiempo de ejecución Tiempo de inicio /fin	Tiempo Costo Calidad		
Historia				
Seguridad	Permisos Responsabilidades			
Autonomía	Lugares de ejecución			

- *Mapear actividades a actores*. Las tareas, es decir, las actividades en ejecución, son asignadas a actores. Los actores son responsables de la ejecución de las tareas de forma precisa y adecuada.
- *Administrar interdependencias.* Las interdependencias tiene diferentes causas. Por ejemplo, las actividades tienen que ser ejecutadas en un orden de tiempo específico. Secuenciación y sincronización son ejemplos de interdependencias de tiempo. Otro tipo de interdependencia resulta del uso de los mismos recursos, es decir dos o más actividades pueden estar en conflicto al competir por un mismo recurso. Los datos y aplicaciones son ejemplos de recursos.

Las reflexiones teóricas de la coordinación justifican muchas de las perspectivas discutidas anteriormente.

La Tabla 2-1 resume el análisis hecho a los modelos anteriores. Los elementos discutidos en las secciones anteriores son la base del modelo de *WF* que se describe a continuación.

2.5 Modelo de Workflow

Según Jablonski [30], un modelo de *WF* debe considerar al menos cinco perspectivas que tienen una importancia fundamental:

- *Perspectiva funcional*. Representa operaciones de alto nivel o unidades funcionales que se ejecutan. Las unidades funcionales son básicamente los *WFs* elementales, los cuales representan una tarea básica asignada a un usuario humano o agente automático; o *WFs* compuestos que representan una tarea compuesta de tareas pequeñas las cuales se representan por otros *WFs*, también llamados *subworkflows.*
- *Perspectiva operacional*. Los *WFs* son simplemente unidades que permiten estructurar procesos u operaciones de bajo nivel. El trabajo real es realizado por programas de aplicación cuyos detalles técnicos son transparentes desde el punto de vista de la administración del *WF*.

- *Perspectiva de comportamiento.* El comportamiento del *WF* se determina por el flujo de control entre las unidades funcionales; en otras palabras, ésta perspectiva representa el orden de ejecución del *WF*.
- *Perspectiva de información*. Representa los datos que consumen y producen los *WFs*, los cuales en conjunto determinan el flujo de datos.
- *Perspectiva Organizacional.* Cada *WF* se ejecuta por uno o más miembros de una organización. Quien es elegible para ejecutar un *WF* se describe en ésta perspectiva.

Para la mayoría de las áreas de aplicación, las perspectivas anteriores son obligatorias y suficientes, ya que con ellas se representa lo que se tiene que hacer o ejecutar (*perspectiva funcional*), cuando se ejecuta el *WF* (*perspectiva de comportamiento*), los datos que se producen y se procesan (*perspectiva de información*), cómo se llevan a cabo las tareas (*perspectiva operacional*) y quién es responsable de llevar las tareas (*perspectiva organizacional*).

Sin embargo, si se desea tener un modelo de *WF* más completo se necesita considerar otras perspectivas:

- *Perspectiva de causalidad*. La ejecución de un *WF* depende de los requerimientos de los negocios. Si tales requerimientos siguen siendo necesarios, se justifica la ejecución de un *WF*.
- *Perspectiva de integridad y recuperación de fallas*. En ésta perspectiva se describen las restricciones que deben cumplirse para la ejecución de un *WF*. Además se describen medidas de seguridad para regresar al *WF* de un estado inconsistente o excepcional a un estado consistente.
- *Perspectiva de calidad*. El tiempo y costo de ejecución son dos criterios potenciales para determinar la calidad de la ejecución de un *WF*. La calidad se determina antes, durante y después de que el *WF* terminó.
- *Perspectiva histórica*. Esta perspectiva representa un registro de todos los aspectos de la ejecución del *WF*.
- *Perspectiva de seguridad*. Determina quién o quienes tiene permiso para la ejecución de un *WF* específico.

- *Perspectiva de autonomía*. Describe si parte de un *WF* puede ejecutarse independientemente sin la intervención de personas, agentes u otros *WFs*. También se refiere a aspectos de movilidad y distribución; por ejemplo, si una parte de un *WF* puede ejecutarse en una computadora móvil y después depositar su resultado en una estacionaria

La lista de perspectivas anterior no es exhaustiva y puede crecer dependiendo de las características que quieran analizarse.

2.6 Herramientas de los sistemas de administración de Workflow

El uso de un *SAWF* se facilita con la ayuda de herramientas que permiten a los usuarios interactuar de forma amigable con él. Estas herramientas se dividen en dos grupos: *Herramientas de tiempo de construcción y herramientas de tiempo de ejecución*. Las herramientas se describen brevemente en las secciones siguientes; no se trata de dar una descripción acerca de las interfaces graficas de las herramientas, pero si una descripción de aspectos funcionales que cada herramienta debe proporcionar.

2.6.1 Herramientas de tiempo de construcción

Estas herramientas facilitan la definición, análisis y administración del *WF*. La Figura 2-1 muestra el lugar que ocupan las herramientas para ayudar en las tareas que se presentan en la etapa de construcción del *WF*. Las tareas principales en ésta etapa son: *definición*, *análisis* y *administración*.

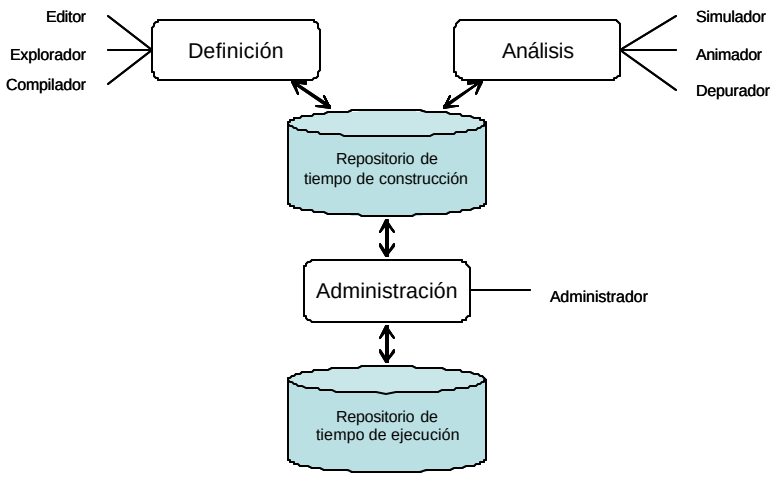


Figura 2-1 Herramientas de tiempo de construcción

El *editor* permitirá definir el *WF* en sus perspectivas, es decir, ésta herramienta definirá el *WF*, los datos, la organización, el flujo de control, etc. Esta descripción será almacenada en el *repositorio de tiempo de construcción*. Un *SAWF* también debe ofrecer un *explorador* que permita revisar los elementos existentes en los *repositorios de tiempo de construcción y ejecución*. Un *compilador* que traducirá la especificación del *WF* en un formato interno para que se almacene en el *repositorio de tiempo de construcción*.

Para el análisis en un *SAWF* se debe contar con un *simulador* que permitirá la ejecución del *WF* y generará un registro de las acciones realizadas durante el simulacro de ejecución. Dicho registro permitirá al modelador del *WF* revisar el comportamiento del *WF* en sus diferentes acciones, permitiéndole optimizar la descripción del mismo.

Un *animador* permitirá una simulación animada del *WF*, lo cual permitirá al modelador del *WF* probar cuando el *WF* se comporta como él esperaba. Cuando el *simulador* y el *animador* no permiten reconocer porque el *WF* no se comporta de la manera deseada, entonces es necesaria otra herramienta llamada *depurador*, la cual permitirá la ejecución paso a paso del *WF* facilitando la búsqueda de los errores.

La descripción del *WF* no se realiza en un sólo paso, normalmente se realiza en partes como todo proceso de diseño. Las partes resultantes deben ser administradas, ésta será una

de las tareas del *administrador*. Además, cuando el *WF* esté descrito completamente, el *administrador* moverá esa descripción del *repositorio de tiempo de construcción* al *repositorio de tiempo de ejecución* para que pueda usarse.

2.6.2 Herramientas de tiempo de ejecución

Las herramientas de tiempo de ejecución ayudan al procesamiento de *WF*. La Figura 2-2 muestra las herramientas principales de tiempo de ejecución. Las tareas principales son la administración de la ejecución, el análisis y la administración del trabajo de los usuarios.

El *administrador* es necesario para asegurar el procesamiento del *WF*, evitar los conflictos que se generan por aplicaciones no disponibles, datos que no pueden accederse, usuarios sin permiso para ejecutar tareas, etc. El *configurador* tiene tareas como describir la comunicación entre los módulos, la creación de procesos para una tarea específica, la asignación de las bases de datos, etc.

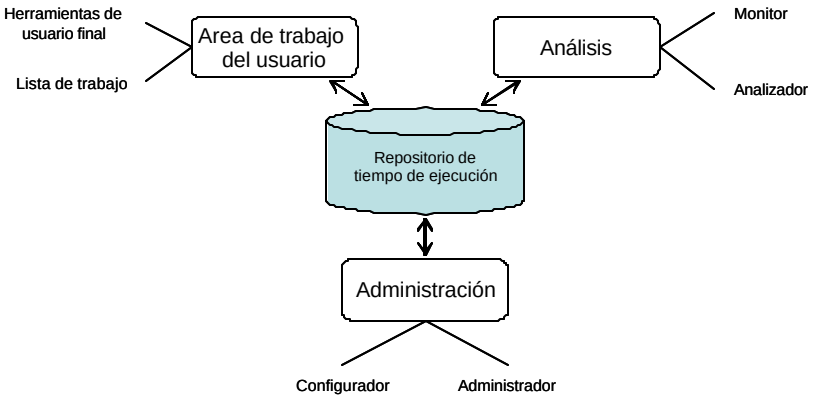


Figura 2-2 Herramientas de tiempo de ejecución

Una herramienta *monitor* es necesaria para el análisis, la cual obtiene información relevante del procesamiento del *WF* que servirá para solucionar problemas como los cuellos de botella o errores de procesamiento. No todos los datos obtenidos por el *monitor* se usan para el análisis, por ello es necesario simplemente observar con frecuencia el

procesamiento del *WF*, con el objetivo de obtener comportamientos que permitan mejorarlo, ésta será la tarea del *analizador*.

Una *lista de trabajo* contiene todas las actividades que el usuario final debe realizar y es el elemento de más importancia para el usuario final. La *lista de trabajo* puede implementarse mediante el correo electrónico o en la base de datos del sistema. La función principal de ésta herramienta es comunicar al usuario final las actividades de trabajo que se le han asignado. También son necesarias las *herramientas de usuario final*, como editores de texto, hojas de cálculo, etc., para llevar a cabo el trabajo. Estas herramientas tienen que estar integradas en el área de trabajo del usuario y son fácilmente invocadas por él.

2.7 Conclusiones

El modelo propuesto por Jablonski se ha utilizado como la base del diseño de la mayoría de los *SAWFs* creados para investigación, por lo que se toma como referencia para el diseño del prototipo de *SAWF-HW* propuesto en ésta tesis, lo cual será descrito más adelante. Por último se mencionaron las herramientas que permiten interactuar con el *SAWF* en las etapas de construcción y tiempo de ejecución.

En el capitulo siguiente se describirán las características más importantes de los *SAWFs* más conocidos así como sus áreas de aplicación, además se analizarán y discutirán los aspectos de diseño más representativos.

Capítulo 3 Sistemas de administración de Workflow

Este capítulo presenta una revisión de los *SAWFs* más conocidos incluyendo los prototipos de experimentación, como los sistemas comerciales; en particular se discuten las características de los tres más importantes: Domino, IBM FlowMark y SAP Business Workflow.

3.1 Sistemas de administración de Workflow

Durante la investigación de la tecnología de *WF* se han desarrollado distintos prototipos por parte de universidades y centros de investigación, así como algunos productos comerciales por compañías de desarrollo de software grandes. Los prototipos de investigación más conocidos son:

- ConTract (University of Stuttgart, Germany) [16]
- Domino (Gesellschaft für Mathematik und Datenverar-beitung, Bonn, Germany) [17]
- Melmac (University of Dortmund, Germany) [18]
- OfficeTalk (Xerox Palo Alto Reserarch Center, Palo Alto, California) [19]
- Pegasus (Hewlett-Packard Laboratories, Palo Alto, California) [20]

Entre los productos comerciales se encuentran:

- Action Wokflow (Action Technologies) [21]
- COSA (Software-Ley) [22][23]
- FlowMark (IBM) [24]
- InConcert (XSoft) [25]
- ProMInanD (IABG) [26]

- SAP Business Workflow (SAP) [27]
- WorkParty (Siemens Nixdorf) [28]

A continuación se presenta una discusión de los sistemas *Domino*, *IBM FlowMark* y *SAP Business Workflow*, se seleccionaron estos tres sistemas por ser los más representativos además de que una discusión de la totalidad de los sistemas listados anteriormente sería muy extensa lo cual no es el objetivo de ésta tesis. Esta discusión incluye aspectos como elementos de modelación, herramientas, lenguajes, API's, arquitectura del sistema y área de aplicación.

3.2 Domino

Domino [17] es un prototipo desarrollado en Alemania, contiene un grupo reducido de elementos para modelar la estructura de una organización. Los elementos de modelación del *Domino* son:

- *Forma*. Es una estructura de datos que es utilizada por el usuario. Los campos de una forma pueden ser definidos libremente por el modelador del *WF*.
- *Acción*. Se define como la tarea que un usuario debe realizar. Para llevar a cabo la tarea, el usuario utilizará un programa de aplicación y los datos necesarios para realizar la acción se obtienen de una forma que sirve como entrada de la acción.
- *Rol*. Existen dos tipos de roles: los roles atados a una acción y los roles dentro de una organización. *Domino* tiene dos tipos de roles predefinidos, el *iniciador* que determina el usuario quien debe realizar la acción, y la *oficina* que se considera un usuario automatizado, de tal forma que si una acción es atada a este rol, la acción se ejecutará automáticamente.
- *Organización*. Los elementos usados para modelar una organización son *unidades organizacionales*, *proyectos*, *trabajos* y *personas*. También existen las relaciones entre objetos *pertenece a* (empleado-unidad organizacional), *responsable de* (empleado-trabajo), *miembro de* (empleado-proyecto) y *supervisor de* (empleado-unidad organizacional, empleado-empleado). Los roles forman parte de la estructura de una organización. En tiempo de ejecución un rol que está atado a una acción es

comparado con los roles de la organización a fin de determinar quienes serán los usuarios habilitados para ejecutar la acción.

- *Procedimiento.* Es una red de acciones relacionadas. Cada acción esta relacionada con al menos dos formas, una que contiene los datos de entrada y una que será usada para almacenar los datos de salida. Las acciones podrán tener más de una forma de entrada y podrá estar conectada a formas de salida diferentes. Las acciones podrán ser ejecutadas solamente si están disponibles los datos o formas de entrada.

Las relaciones entre formas y acciones puede ser fácilmente mapeadas a redes de Petri representando las formas como lugares y las acciones como transiciones, este es el modelo que *Domino* usa internamente para representar los procedimientos. Después de que un usuario termina su trabajo, puede elegir entre enviar la forma a la siguiente acción, enviarla a la acción anterior o delegar el procedimiento. Este comportamiento es parecido al paradigma de migración de carpetas o formas.

Domino tiene la capacidad de asignar acciones a los usuarios de acuerdo a su rol y de verificar la correctitud de los procedimientos mediante las siguientes reglas:

- Un procedimiento no debe detenerse
- Un procedimiento no debe tener acciones muertas
- Una forma es necesitada por una acción o es producida por un procedimiento
- Una forma no debe ser producida dos veces
- Al menos una entrada o salida alterna de las acciones debe ser seleccionada

Domino proporciona un lenguaje de especificación de procedimientos y estructuras de organización, llamado *CoPlanS*. La representación de procedimientos en redes de Petri no es visible a nivel del lenguaje, pero *Domino* provee de un compilador que permite la compilación de procedimientos en redes de Petri. También se proporcionan interfaces de usuario para indicar el estatus de cada procedimiento desde la perspectiva del usuario (por ejemplo terminado, ninguna acción requerida, los datos han cambiado, etc.). Otra interfase es usada para iniciar procedimientos, trabajar con datos y capturar datos.

La arquitectura de *Domino* implementa la estructura cliente/servidor entre sus componentes. La administración y supervisión de procedimientos son realizadas por un componente llamado *mediador*, que tiene acceso a bases de datos para almacenar el estado de los procedimientos en ejecución y también la estructura de la organización. La arquitectura también proporciona un compilador para el lenguaje *CoPlanS* que es usado para compilar e instalar procedimientos de oficina. *Domino* usa el correo electrónico como medio de comunicación y notificación a los usuarios acerca de las acciones y para transportar formas entre los usuarios y el componente *mediador*. El área de aplicación de *Domino* es la implementación de procedimientos de oficina.

3.3 IBM FlowMark

Un producto comercial es *FlowMark* [24], un *SAWF* de IBM, su modelo de divide en cuatro áreas:

- *Programas*. Es una aplicación de computadora utilizada por un usuario dentro de *FlowMark* para llevar a cabo una tarea asignada.
- *Estructuras de datos*. Es una lista de variables que pueden ser usadas en tiempo de ejecución pero no pueden ser borradas. Las estructuras de datos son usadas para modelar parámetros de entradas (contenedor de datos de entrada) y de salida (contenedor de datos de salida) de un proceso.
- *Staff*. Los procedimientos son asignados a los usuarios para su ejecución. *FlowMark* permite la modelación de la estructura de una organización usando los objetos *niveles*, *personas*, *roles*, *organizaciones* y las relaciones *personas-role* y *persona-organización*.
    - Las *personas* son los empleados que llevan a cabo el trabajo.
    - Los *roles* tienen un nombre y miembros.
    - Las *organizaciones* representan grupos como mercadotecnia o investigación, estos objetos tienen una persona asignada como administrador y otros como miembros de la organización. Puede tener una estructura jerárquica que la descompone en suborganizaciones.

- Los *niveles* se usan para distinguir a las personas, estos se basan en alguna propiedad como su nivel de educación (*principiante, experto* o *guru*), los niveles son usados en tiempo de ejecución para seleccionar a los usuarios para realizar alguna actividad.

- *Procesos*. La especificación de los procesos se lleva a cabo con los objetos:
  - *Procesos,* definen todas las actividades que se deben llevar a cabo para completar una tarea. Tiene una duración, si el proceso no se termina en ese tiempo se notifica al administrador del proceso.
  - *Actividad programa,* se ejecuta por un usuario con la ayuda de un programa.
  - *Actividad proceso*, que no tiene que ejecutarse por un sólo usuario, ésta tiene atado un proceso que se inicia cuando la actividad ésta lista para su ejecución, este proceso es llamado subproceso.
  - *Conectores de control,* relacionan dos actividades definiendo un orden de control entre ellas. Existen los conectores *condición de entrada*, *de salida y de transición*. El control pasa a la siguiente actividad conectada si la condición de transición es verdadera. Una actividad termina si su condición de salida es verdadera, y una actividad inicia si su condición de entrada es verdadera.
  - *Conectores de datos*, especifican el flujo de datos entre los contenedores de entrada y salida de las actividades conectadas.
  - *Bloques*, definen grupos de actividades dentro de un proceso para su mejor legibilidad.
  - *Paquete,* se usa para iniciar un grupo de actividades del mismo tipo.

A cada actividad tiene una regla de asignación que define a cuales personas del staff se les tiene que notificar. Esta regla puede ser un rol.

FlowMark agrupa sus herramientas en dos grandes grupos: *herramientas de tiempo de construcción* y *herramientas de tiempo de ejecución*. Las *herramientas de tiempo de construcción* son usadas para definir procesos, y son las siguientes:

- Herramientas para definir el staff como un editor de definición de personas, un editor de definición de roles, un editor de definición de organización y un editor de definición de niveles.
- Un editor gráfico de procesos.
- Un editor de registro de programas.
- Un editor de estructuras de datos.
- Una herramienta de animación de procesos.

Las *herramientas de tiempo de ejecución* son usadas durante el proceso de ejecución y son:

- Una lista de trabajo (*Worklist*) utilizada por los usuarios para ver y ejecutar las tareas asignadas a ellos.
- Una herramienta de planeación de paquetes que se usa para iniciar varias actividades del mismo tipo.

FlowMark proporciona un conjunto de API's para controlar y acceder a los procesos en tiempo de ejecución, esto permite iniciar, terminar, detener o reiniciar los procesos. Los API's pueden usarse en distintos lenguajes de programación.

FlowMark es un *SAWF* centrado en una base de datos, toda la información de la definición de los procesos y su estado en tiempo de ejecución se almacena en una base de datos orientada a objetos. *FlowMark* usa el paradigma cliente/servidor en donde las listas de trabajo (*Worklist*) son los clientes de los servidores *FlowMark*. El servidor responde a las peticiones de los clientes, acceder a la base de datos para ejecutar los servicios que se ofrecen a los clientes. Es posible tener varios servidores *FlowMark* accediendo a una misma base de datos. También es posible operar en distintas bases de datos, cada una funcionando como una esfera que no comparte su información. El usuario selecciona la base de datos al entrar al sistema.

FlowMark está construido para modelar distintos tipos de *WF* dependiendo del área de aplicación.

3.4 SAP Business Workflow

Otro de los productos más conocidos es *SAP Business Workflow* [27] es un *SAWF* de la compañía alemana SAP AG. *SAP Business Workflow* es parte de la versión 3.0 del sistema *R/3* de SAP.

Los elementos de modelado de *SAP Businnes Workflow* (SAP) son:

- *Objeto. R/3* es un sistema base que provee un modelo orientado a objetos el cual soporta programación orientada a objetos. Se usa el lenguaje interno de SAP llamado *ABAP/4*. La funcionalidad de *R/3* se altera agregando nuevos objetos y métodos.
- *Organización*. *R/3* soporta administración de personal, usando el modulo *HR-ORG*, el cual implementa objetos para construir una organización. En este modulo se pueden modelar usuarios, posiciones, lugares de trabajo (*Workplaces*), grupos así como sus relaciones a tareas y capacidades.
- *Tarea*. Una tarea específica una unidad de trabajo que realiza un usuario. Una tarea está compuesta de un componente, el cual es un objeto con un método que es invocado por él al ejecutar la tarea, un componente que describe quien debe llevar a cabo tareas de este tipo, un componente role que se usa para detallar que usuario llevara a cabo la tarea, un texto plano para describir los contenidos de la tarea con propósitos explicativos y finalmente un conjunto de eventos, cada uno de los cuales finaliza la ejecución de una tarea.
- *Aplicación*. Se pueden integrar aplicaciones externas a *SAP Business Workflow* a través de la invocación de métodos estándar como OLE y ODBC. La idea básica es encapsular a las aplicaciones externas mediante objetos.
- *Paso* (Elementos de trabajo). Los pasos son bloques de construcción básica de *WFs*. Hay 5 tipos de pasos:
    - *Actividad*. Una actividad es una tarea, una aplicación externa, una transacción o una acción manual.
    - *Decisión de usuario*. es una acción manual de un usuario.

    - *Paso de espera*. Se usan para unir ramas ejecutándose paralelamente dentro de un *WF*.
    - *Subworkflow*. Son partes de un *WF* pueden ser modeladas para luego ser reutilizadas.
    - *Condición*. Un *WF* puede ramificarse en varias rutas para esto se usan las construcciones `if-then-else` y `case`.
- *Workflow*. Los *WFs* en *SAP Business Workflow* se basan en el concepto de cadenas de procesos manejados por eventos. Un *WF* se dispara por un evento. Cuando un *WF* termina se dispara otro evento que podría iniciar otro paso. Los pasos pueden ejecutarse paralelamente dependiendo de como se enlazan. Los datos necesarios son obtenidos de contenedores que mantienen datos internos del *WF*. El flujo de datos entre contenedores de diferentes *WFs* se especifica por definiciones fuente/destino.

SAP Business Workflow tiene 3 categorías de herramientas:

- *Herramientas de definición*. Para definir *WFs*, tareas, objetos, roles y la conexión de eventos, y sus consumidores y productores.
- *Herramientas de tiempo de ejecución*. Estas se refieren a la ejecución del *WF* (*Administrador del WF*), manejo de elementos de trabajo (*Workitem Manager*), interfaces de usuario (*Worklist Client*).
- *Herramientas para reportes*. Son herramientas para extraer información del *WF*, análisis y funciones estadísticas para evaluaciones.

SAP Business Workflow es una parte integral del sistema *R/3*. El ambiente de ejecución tiene tres tareas principales:

- *Administración de trabajo.* Asignar y supervisar pasos.
- *Administración de flujo.* Evaluación de *WFs* para derivar pasos ejecutables.
- *Administración de lista de trabajo.* Maneja la interacción del ambiente de ejecución con los usuarios.

3.5 Conclusiones

Los sistemas presentados en este capítulo muestran en común un grupo de elementos de modelación con los cuales el modelador del *WF* puede representar una empresa u organización y las tareas que sus integrantes deben realizar. La Tabla 3-1 muestra la comparación de las características de los sistemas analizados.

Tabla 3-1 Comparación de caracteristicas de los SAWFs analizados

Sistema / ***Característica***	***Domino***	***IBM FlowMark***	***SAP Business Workflow***
Clasificación	Administrativo	Producción/Administrativo	Producción/ Administrativo
Modelo	Forma, acción, role y organización	Programas, estructuras de datos, staff y procesos	Agentes, espacios de términos y leyes de coordinación
Cliente/Servidor	Si	Si	Si
Lenguaje de script	Si	No	No
Mecanismo de transporte	Correo electrónico	Base de datos	Sistema R/3
Herramientas de analisis y pruebas	Si	Si	Si
Herramientas de reportes	No	Si	Si
Seguimiento de procesos	Si	Si	Si
Integración con aplicaciones de oficina	No	Si	Si

El sistema *Domino* muestra un grupo de elementos de modelación más sencillo y flexible que los otros dos sistemas analizados; ésta característica hace más útil a *Domino* ya que permite al sistema adaptarse a la empresa y no viceversa. En contraste, *IBM FlowMark* como *SAP Business Workflow* tiene grupos de elementos de modelación más rígidos con los cuales un modelador del *WF* tendría algunas dificultades para adaptarlo a la empresa. Sin embargo, las compañías creadoras de estos sistemas los complementan con otras características que los hacen más útiles y confiables.

En el capitulo siguiente se describirán las características de los modelos de coordinación. En particular, se describirá el modelo de coordinación del lenguaje *HW*.

Capítulo 4 Modelos y lenguajes de coordinación

En este capítulo se describen los conceptos de *modelo de coordinación* y *lenguajes de coordinación*. También se describen las características del modelo de coordinación de *Linda*. Por ultimo, se describen los fundamentos teóricos y elementos de modelación con los que cuenta el lenguaje de coordinación *HW*.

4.1 Modelo de coordinación

Un modelo de coordinación [34] es un marco abstracto que sirve de ayuda para estudiar y entender problemas en el diseño de programas concurrentes y programas distribuidos. En otras palabras, un modelo de coordinación proporciona un área de trabajo en la cual se puede expresar la interacción de agentes individuales.

Un modelo de coordinación ofrece al menos control en la generación y destrucción de componentes, comunicación y flujos múltiples de comunicación. Usualmente un modelo de coordinación ofrece mecanismos de coordinación que son agregados a un lenguaje anfitrión.

Un modelo de coordinación puede verse como una tripleta (*E, M, L*), donde:

E: son las entidades coordinables, que pueden ser pasivas (registros de base de datos) o activas (agentes o procesos).

M: es el medio de coordinación; este medio permite la coordinación entre entidades. (Ej. Canales, variables compartidas, espacios de tuplas, etc.)

L: son las leyes de coordinación que regulan las acciones de las entidades coordinables (Ej. Guardias, restricciones de sincronización, etc.)

4.2 Lenguaje de coordinación

Un lenguaje de coordinación es la representación lingüística de un modelo de coordinación, y consiste de algunos mecanismos de coordinación que son agregados a un lenguaje anfitrión (secuencial). En la práctica un lenguaje de coordinación incluye mecanismos claramente definidos para coordinación, sincronización, distribución y control de concurrencia.

Algunos lenguajes de coordinación son:

C + Unix

E: procesos

M: archivos, tuberías, señales

L: llamadas de sistema Unix

C + Linda

E: tuplas

M: espacio de tuplas

L: operaciones sobre el espacio de tuplas

GAMMA

E: Reacciones químicas representadas por pares condición-acción

M: Multiconjunto

L: Aplicación de punto fijo de los pares condición-acción.

CHAM

E: gramática para generar las configuraciones

M: Conjunto de configuraciones

L: Conjunto de reglas

PoliS

E: Espacios de tuplas

M: Espacio de tuplas raíz

L: Linda + leyes entre espacios de tuplas

C + Unix

El sistema operativo Unix más el lenguaje C [34] permiten la coordinación de procesos, los cuales representan las entidades coordinables. Como medio de coordinación se usan los archivos, las tuberías o las señales, usando como leyes de coordinación las llamadas al sistema operativo.

Linda

Linda [3] consiste de un conjunto pequeño de operaciones simples que tiene que ser embebidas en un lenguaje secuencial anfitrión, como el lenguaje *C*, para obtener un lenguaje de programación paralelo. La Figura 4-1 muestra el modelo del lenguaje de coordinación de Linda.

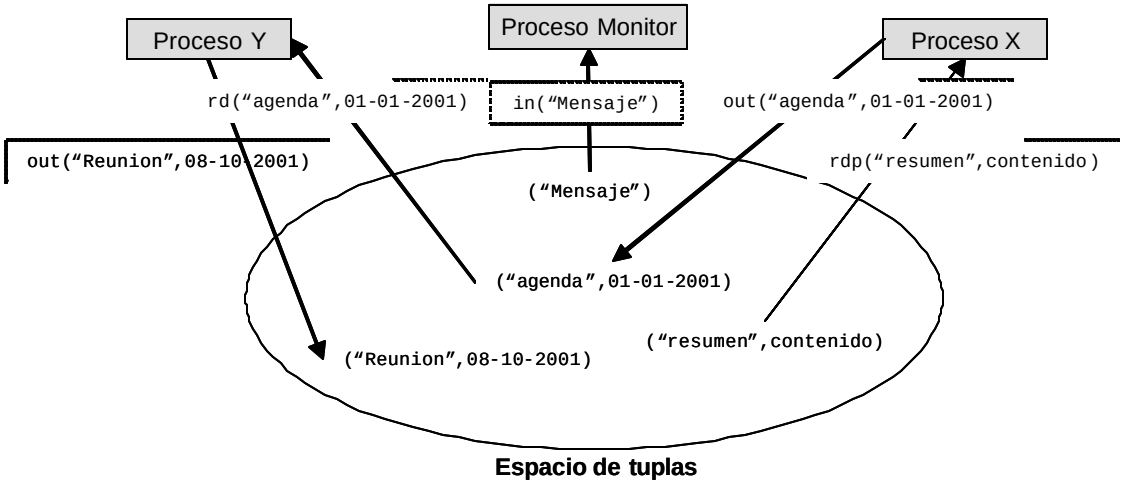


Figura 4-1 Modelo del lenguaje de coordinación Linda

Las características del lenguaje de coordinación Linda son:

- Un conjunto reducido de operaciones simples,
- Una memoria asociativa, persistente y distribuida llamada espacio de tuplas,

- Una tupla es una secuencia de campos con un tipo determinado. Todas las tuplas pueden ser creadas, leídas o borradas,
- Tuplas pasivas que contienen datos y
- Tuplas activas que contiene procesos se reducen a tuplas pasivas al término de su ejecución.

En Linda los agentes o procesos se comunican indirectamente usando las siguientes operaciones primitivas:

- *out(t).* Escritura la tupla pasiva *t*,
- *rd(t).* Lectura bloqueante de la tupla *t*,
- *rdp(t).* Lectura no bloqueante de la tupla *t*,
- *in(t).* Lectura bloqueante con eliminación de la tupla *t*,
- *inp(t).* Lectura no bloqueante con eliminación de la tupla *t* y
- *eval(t*). Creación de los procesos incluidos en la tupla activa *t*.

Un lenguaje de coordinación que extiende el modelo de Linda es *Hyperworlds* [32] (*HW*), el cual se basa en reglas, es simbólico y distribuido que usa la satisfacción de restricciones. *HW* se describe en la sección siguiente.

GAMMA(General Abstract Model for Multiset mAnipulation)

GAMMA [4] es un modelo de coordinación cuya estructura de datos principal es el multiconjunto y la única estructura de control es el operador Γ. El siguiente pseudo código muestra el funcionamiento de GAMMA:

$$\Gamma\left(\left(R_1, A_1\right),...,\left(R_m, A_m\right)\right)\left(M\right) =$$
$$\text{if } \forall i \in [1,m], \forall x_1,...,x_n \in M, \sim R_i\left(x_1,...,x_n\right)$$
$$\text{then } M$$
$$\text{else let } x_1,...,x_n \in M, \text{let } i \in [1,m] \text{ such that } R_i\left(x_1,...,x_n\right) \text{ in}$$
$$\Gamma\left(\left(R_1, A_1\right),...,\left(R_m, A_m\right)\right)\left(\left(M - \{x_1,...,x_n\}\right) + A_i\left(x_1,...,x_n\right)\right)$$

La notacion {…} representa a los multiconjuntos; (R_i, A_i) son pares de funciones cerradas (sin variables globales) que especifican las reacciones. El efecto de (R_i, A_i) sobre el multiconjunto M se remplaza por un subconjunto $\{x_1,...,x_n\}$ tal que $R_i(x_1,...,x_n)$ es verdadero para los elementos de $A_i(x_1,...,x_n)$.

Γ es un operador de punto fijo, es decir, las reacciones continúan hasta que ninguna reacción nueva sea posible. Todas las reacciones posibles se disparan, esto es la fuente del paralelismo de GAMMA.

CHAM(The Chemical Abstract Machine)

CHAM [5] es una tripleta donde:

- G es una gramática que genera a C
- C es un conjunto de configuraciones (moleculas)
- R es un conjunto de reglas: *condition* x bag(C) x bag(C)

Donde *condition* es una expresión lógica sobre las moléculas que están en las bolsas (*bag*) o multiconjuntos, las cuales tienen que borrarse (*preconfiguracion*) y agregarse (*postconfiguracion*). Las reglas pueden dispararse concurrentemente si no interfieren entre ellas.

Se puede imaginar a CHAM como una bolsa que contiene una solución química activa, en la cual las moléculas aparecen y desaparecen.

PoliS

PoliS [6] intenta mejorar a Linda, agregando una operación nueva que permite espacios de tuplas vacios a identificados por un nombre:

```
tsc(tuple_space);
```

Polis extiende la operación out de Linda, para crear tuplas de manera remota:

```
out(tuple)@tuple_space;
```

Un espacio de tuplas cerrado (closed PoliSpace) es similar a un programa en linda, en el cual un número de espacios de tuplas se inician y se ejecutan hasta completar una tarea global. Un PoliSpace abierto es un programa interactivo abierto, el cual incluye un número de espacios de tuplas, algunos de ellos llamados “shells”. Un “shell” es un espacio de tuplas “interactivo” donde el usuario puede agregar tuplas directamente, en otras palabras, un “shell” es un punto de interacción entre el PoliSpace y un ambiente externo.

4.3 Hyperworlds

HW [32] es un lenguaje de coordinación que extiende el modelo de coordinación *Linda* introduciendo términos con variables lógicas, satisfacción de restricciones y ámbito dinámico de nombres. El modelo se basa en una abstracción de un depósito de datos compartido, transaccional y persistente que se representa por multiconjuntos de términos. Un multiconjunto es una colección de elementos que retienen su multiplicidad pero que no poseen entre sí otras relaciones causales que aquellas que se establecen explícitamente mediante reglas de interacción. Así un multiconjunto de términos corresponde intuitivamente a un espacio de objetos cuya distribución no afecta la facilidad a su acceso. *HW* extiende el modelo original de *Linda*, al particionar al multiconjunto en una colección disjunta de multiconjuntos, en donde cada uno de ellos es designado por un nombre. Los multiconjuntos representan el medio de coordinación y comunicación entre agentes que inducen una forma de colaboración desacoplada tanto en tiempo (no requieren hora de reunión) como en espacio (no requieren lugar de reunión). La colaboración entre los agentes participantes está determinada por un conjunto de reglas de interacción que accesan y modifican el contenido del medio de coordinación. Cada regla de interacción es una regla de reescritura de multiconjuntos de términos que relaciona una condición con una acción. La condición describe los elementos sobre los que se va a aplicar la regla, en tanto que la acción indica los nuevos elementos en que serán transformados. La sintaxis y semántica del lenguaje se formaliza dentro del marco de una lógica de reescritura que describe la

estructura sintáctica de las entidades de coordinación y de las reglas de transformación de multiconjuntos de entidades coordinables.

Un aspecto notable del lenguaje de coordinación *HW* es su poder expresivo que es suficiente para describir modelos de concurrencia extensamente estudiados como las redes de Petri. En [37] se pueden encontrar algunos conceptos y definiciones básicos de la teoría de redes de Petri. Una red de Petri es un grafo bipartito con dos tipos de nodos llamados *lugares* y *transiciones* conectados por arcos dirigidos que definen una *relación de flujo.* Los lugares se visualizan por círculos, en tanto que las transiciones por rectángulos, como se muestra en la Figura 4-2.

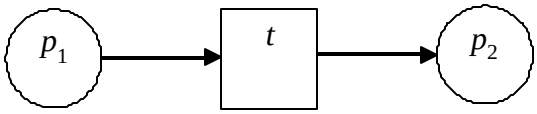


Figura 4-2 Red de Petri elemental

Formalmente, una red de Petri (P, T, F) consiste de:

- un conjunto finito P de lugares,
- un conjunto finito T de transiciones $((P \cap T) = \varnothing)$ y
- un conjunto $F \subseteq (P \times T) \cup (T \times P)$ de arcos dirigidos.

Los lugares que conducen a una transición se le llaman de entrada y a los lugares que salen de una transición se les llama de salida. Los lugares contienen un número finito de elementos llamados marcas. La distribución de las marcas en los lugares constituye el estado de la red y está determinado por una función de marcado. La distribución de las marcas puede cambiar por la habilitación y activación de las transiciones. Una transición está habilitada cuando existe una marca en cada uno de sus lugares de entrada. Cuando una transición habilitada se activa, consume las marcas de sus lugares de entrada y produce nuevas marcas en sus lugares de salida.

El escenario descrito por las redes de Petri se puede describir en el modelo de coordinación *HW*. Los lugares y las marcas que contienen corresponden con los multiconjuntos de términos, las transiciones corresponden con los agentes y la relación entre lugares y transiciones corresponden con las reglas de reescritura. De ésta forma, además de la bien conocida notación gráfica de las redes de Petri, el lenguaje *HW* provee una notación para describir no solo redes de Petri elementales sino también redes de alto nivel como coloreadas y predicado-transición.

La descripción formal del lenguaje *HW* que se presenta a continuación es un subconjunto de aquella presentada en [32]. La principal simplificación consiste en eliminar el uso de variables lógicas en los términos que habitan los espacios de términos como se describió en [7]. No obstante, ésta versión simplificada del lenguaje es suficientemente expresiva para modelar *SAWFs* como el que se analiza en el capítulo siguiente.

4.3.1 Sintaxis de las entidades y reglas de coordinación

En ésta sección se introduce la sintaxis abstracta de las expresiones bien formadas del lenguaje de coordinación *HW*. Una teoría de segundo orden consiste de un alfabeto, un lenguaje de segundo orden, un conjunto de axiomas y un conjunto de reglas de inferencia. El lenguaje de segundo orden consiste del conjunto de expresiones bien formadas de la teoría. Los axiomas y las reglas de inferencia se usan para derivar los teoremas de la teoría.

Un alfabeto consiste de las siguientes clases de entidades lexicográficas o símbolos:

a. *constantes* y *funciones* (*operaciones*) son denotadas por nombres simbólicos que comienzan con una letra minúscula *a*, *b*, *c*, ..., e incluyen además al símbolo de igualdad (=), al operador de composición concurrente coma (,), al operador **new** y a los nombres de agentes *A*, *B*, *C*, ...,
b. el operador ~> indica hacer concreta una abstracción,
c. *variables* son denotadas por nombres simbólicos que comienzan con una letra minúscula precedidas por una subraya *_a*, *_b*, *_c*, ...,

d. *predicados* son denotados por nombres simbólicos que comienzan con una letra mayúscula *P, Q, R, ...*, e incluyen las constantes *True* y *False*,
e. *variables de predicados* son denotados por nombres simbólicos que comienzan con una letra precedidos por una subraya *_p, _q, _r, ...*,
f. *conectivos* son denotados por $\wedge$ para la conjunción, por $\neg$ para la negación, por $\Leftarrow$ para la implicación y por $\rightarrow$ para la relación de reescritura.
g. *cuantificador existencial* denotado por el símbolo ^ (circunflejo),
h. *palabras reservadas* como **`initially`**, **`rule`**, **`constraint`**, **`end`**,
i. *símbolos de puntuación* que separan (o combinan) expresiones bien formadas del lenguaje, como los paréntesis, los dos puntos (:), la raya (|) y el punto (.).

Para evitar el uso excesivo de paréntesis se pueden simplificar la escritura de las frases del lenguaje adoptando la siguiente jerarquía de precedencias:

$\neg$, ^ (precedencia más alta)

$\wedge$

,

|

$\Leftarrow$, $\rightarrow$ (precedencia más baja)

Al igual que Prolog, *HW* usa el símbolo coma (,) tanto como operador de composición concurrente como separador en una lista de elementos. La interpretación del símbolo coma (,) dependerá del contexto donde ocurra.

Los nombres simbólicos como las constantes, variables, funciones y predicados poseen una *aridad* que es el número de subtérminos que admite para formar un término por composición. Las constantes y las variables de primer orden poseen aridad cero mientras que las variables de segundo orden poseen en general aridad mayor que cero. La presencia de variables lógicas que toman sus valores en símbolos de predicados hacen al lenguaje *HW* un lenguaje de segundo orden. Sin embargo, la operación de unificación de términos de

segundo orden es elemental ya que se reduce a equivalencia de patrones (*pattern matching*) como se explicará más adelante.

Las frases bien formadas del lenguaje se obtienen por la composición de entidades sintácticas de acuerdo a las reglas de formación dadas por la sintaxis abstracta de la Figura 4-3.

$$
\begin{array}{lcl}
n & ::= & A \mid c \mid \_x \\
N & ::= & n \mid n, N \\
t & ::= & n \mid n(T) \mid A(T^0)(N) \mid \mathbf{new}(N)(T) \mid t \mid P \\
T & ::= & t \mid t, T \\
p & ::= & True \mid False \mid t = t \mid q(T) \mid c(T) \\
P & ::= & p \mid \neg p \mid P \wedge P \mid \_x \wedge P \\
B & ::= & p \Leftarrow \mid p \Leftarrow P \mid \Leftarrow P \\
R & ::= & T \rightarrow T^0 \mid \mathbf{initially}\ T
\end{array}
$$

Figura 4-3 Sintaxis abstracta del lenguaje de coordinación *HW*

Un nombre simbólico *n* es una unidad básica de representación del conocimiento y puede ser cualquier agente *A*, constructor *c* o variable *_x*. En general, un agente es un constructor por lo que nos referiremos a ellos también como constructores. Una lista *N* de nombres simbólicos es una secuencia que contiene al menos un nombre.

Las frases del lenguaje son entidades abstractas llamadas *términos* que se forman por la composición de constructores y variables. En la descripción sintáctica, un término se denota por *t* mientras que una lista de términos, una secuencia que contiene al menos un término, se denota por *T*. Cuando en la lista de términos el orden de éstos es irrelevante, la lista representa a un multiconjunto de términos y puede interpretarse, en éste caso, al operador coma (,) como el *operador de composición concurrente* de entidades coordinables. Las propiedades asociativa, conmutativa que posee éste operador serán discutidas más adelante

cuando se describe la semántica formal del lenguaje. En lo que sigue, emplearemos los términos espacio de términos y multiconjunto de términos como sinónimos.

Un término puede ser un nombre simbólico n (es decir, puede ser una constante o una variable, inclusive de segundo orden), la composición $n(T)$, el estado de un agente $A(T)(N)$, la restricción dinámica de ámbito **new**$(N)(T)$ o la restricción lógica $t|P$.

La composición $n(T)$ permite obtener un término a partir de un nombre simbólico n y una lista de términos T siempre que sean iguales la aridad del nombre simbólico y la longitud de la lista de términos. Un término de especial importancia, llamado *conexión*, está formado por la composición del nombre simbólico de un espacio de términos con uno de los términos que aparecen en él. En este caso, $n(t)$ se interpreta como $t \in n$ cuando n es el nombre de un espacio de términos. Desde el punto de vista de la programación lógica que se discutirá más adelante, la conexión $n(t)$ puede interpretarse como la cláusula unitaria $n(t) \Leftarrow$ con el propósito de formular y responder a consultas sobre el contenido de los espacios de términos. Por otra parte, en la teoría de redes de Petri, la composición $n(t)$ representa al término t que denota a la marca contenida en el lugar n. Para denotar el contenido de una parte del espacio de términos, los elementos se combinan mediante el operador de composición concurrente coma (,) y por lo tanto el orden en el que se escriben no es importante. Por ejemplo, el término *Mesa*(*lapiz*(*rojo*)) indica que el lugar *Mesa* contiene al término *lapiz*(*rojo*). La lista *Mesa*(*lapiz*(*rojo*)), *Mesa*(*lapiz*(*rojo*)) indica que el lugar *Mesa* contiene dos términos idénticos puesto que la multiplicidad de elementos se conserva en multiconjuntos. Por último, en la lista *Mesa*(*lapiz*(*rojo*)), *Mesa*(*lapiz*(*azul*)), *Mesa*(*lapiz*(*negro*)) el orden en el que aparecen los elementos en la lista es irrelevante.

Un agente $A(T^0)(N)$ consiste del nombre de una constante A, una lista posiblemente vacía (indicado por el superíndice 0 de T) de términos T y una lista de nombres simbólicos N. La constante A denota el nombre del agente, la lista de términos T representa su estado interno y la lista de nombres N denota los nombres de los depósitos de datos (multiconjuntos de términos) que el agente conoce y, por lo tanto, a los que puede tener acceso. En comparación con las redes de Petri, el estado de un agente $A()(N)$ se puede comparar a una

transición *A* conectada a un conjunto de lugares representados por la lista *N* de sus nombres simbólicos. Puesto que las transiciones en la teoría de redes de Petri son en general inmutables (no poseen estado interno), la descripción del estado interno del agente no se usará aquí, dejando vacía la primera lista de términos. Intuitivamente, la lista de nombres representa los lugares a los que la transición tiene acceso. Por ejemplo, en la lista de términos *Ana*()(*Mesa*), *Mesa*(*lapiz*(*rojo*)), *Silla*(*lapiz*(*rojo*)), la transición *Ana* puede tener acceso al término *lapiz*(*rojo*) que se encuentra en el lugar *Mesa* pero no aquel que se encuentra en el lugar *Silla*.

En la descomposición modular de transiciones en redes de Petri de mayor complejidad, la restricción dinámica de ámbito es una característica fundamental de éste lenguaje. La restricción dinámica de ámbito **new**(*N*)(*T*) de la lista de nombres *N* en la lista de términos *T* hace disponibles los nombres *N* solamente en los términos *T* que forman su ámbito pero no afuera. Para conseguir este efecto, el operador **new** hace uso del renombramiento uniforme de los nombres *N* en *T* por otros nuevos. Por ejemplo, en la lista de términos **new**(*Mesa*)**(***Ana*()(*Mesa*), *Mesa*(*lapiz*(*rojo*))**)**, *Eva*()(*Mesa*), *Ana* puede tener acceso al objeto *lapiz*(*rojo*) porque se encuentra dentro del ámbito del nombre *Mesa*; en cambio, *Eva* no puede accesarlo porque está fuera del ámbito de *Mesa*. Esto se hace evidente cuando se reemplaza el nombre *Mesa* por otro. Por ejemplo, en la lista equivalente de términos **new**(*Escritorio*)**(***Ana*()(*Escritorio*), *Escritorio*(*lapiz*(*rojo*))**)**, *Eva*()(*Mesa*), el nombre *Mesa* fue reemplazado por *Escritorio* en las ocurrencias que tienen lugar dentro de su ámbito. Así se hace evidente que *Eva* no puede tener acceso al objeto porque se encuentra en un lugar que no conoce. Este mecanismo es similar al de Prolog, ya que para obtener todos los términos que satisfacen un predicado se deben conocer necesariamente el nombre de dicho predicado.

La restricción $t|P$ del término *t* a la condición lógica *P* restringe los términos *t* a aquellos que satisfacen *P*. Las restricciones lógicas surgen naturalmente al describir sistemas con información incompleta como descripciones declarativas en los conjuntos de valores que pueden tomar las variables. La información se determina progresivamente por satisfacción de restricciones, un procedimiento que decide cuando una restricción lógica tiene solución.

Sintácticamente, las restricciones lógicas pueden ser simples o compuestas. Las restricciones simples pueden ser las constantes lógicas (*True* y *False*), los predicados primitivos (*member*, *append*, etc.), las ecuaciones o las consultas (búsquedas) de un término en un espacio de términos. La constante lógica *True* siempre se satisface en tanto que *False* nunca lo hace. Una ecuación $t = t$ determina la igualdad sintáctica entre dos términos usando el procedimiento de unificación de primer orden. Un predicado primitivo puede ser cualquiera de aquellos definidos en el Prolog de Edinburgo tales como *member*/2, *append*/3, *is*/2, *forall*/2, *findall*/3, etc.. Una consulta *c*(*t*) determina cuando un término *t* pertenece a un multiconjunto de términos *c*, en cuyo caso, encuentra la solución a las variables que satisfacen la consulta. Siguiendo los ejemplos anteriores, cuando el término *Mesa*(*lapiz*(*_x*)) aparece en el contexto de una consulta al espacio de términos *Mesa* (es decir, cuando el término *Mesa*(*lapiz*(*rojo*)) ha sido introducido previamente aplicando una regla de reescritura), la variable *_x* puede tomar el valor *rojo*. Como en el ejemplo anterior, la vinculación de una variable con un término se denota por $\_x \mapsto rojo$.

Las restricciones compuestas se obtienen de las restricciones simples por negación, conjunción y cuantificación existencial. La negación $\neg p$ de un átomo *p* no puede usarse para generar soluciones en una consulta ya que de otro modo puede producir soluciones inconsistentes [31]. Por tal motivo, la negación se aplica únicamente a predicados con términos sin variables. La conjunción de restricciones $P \wedge P'$ corresponde con la satisfacción de una meta en cláusulas de Horn, en donde la meta se satisface solamente si existe una solución que satisface tanto a *P* como a P'. La cuantificación existencial *_x*^*P* determina, si existe, una solución, que es un asignamiento a una variable lógica que puede usarse en términos y expresiones lógicas. Por ejemplo, en la consulta $\_x \wedge (Mesa(lapiz(\_x)) \wedge \neg(\_x = azul))$, tiene una solución en el espacio de términos *Mesa*(*lapiz*(*azul*)), *Mesa*(*lapiz*(*rojo*)), obteniendo la solución $\_x \mapsto rojo$. Observe que la negación no se usa para generar valores distintos de azul sino que, por el contrario, verifica únicamente que el valor generado para la variable *_x* por la consulta *Mesa*(*lapiz*(*_x*)) también satisfaga la condición $\neg(\_x = azul)$.

Además de los predicados primitivos, el lenguaje permite introducir otros nuevos definidos por el usuario. Los nuevos predicados se definen mediante cláusulas de Horn y pueden usarse en cualquier restricción lógica. Una *cláusula de Horn* es una cláusula meta $\Leftarrow P$, una cláusula unitaria $p \Leftarrow$ o una cláusula $p \Leftarrow P$ con antecedente *P* y consecuente *p*. Una cláusula meta $\Leftarrow P$ es una consulta general sobre el contenido del espacio de términos que puede realizar un usuario en cualquier momento. Esencialmente, el concepto de *meta* corresponde con el de restricción lógica definido anteriormente. La cláusula unitaria $p \Leftarrow$ introduce una definición para el predicado de *p* estableciendo que es un hecho, es decir, que es incondicionalmente cierto. Por otra parte, una cláusula genérica $p \Leftarrow P$ introduce una definición para el predicado de *p* que depende de la validez de la condición *P*. Como en la programación lógica, un predicado puede consistir de varias cláusulas para introducir una definición por casos, con el orden usual de selección (textual, de arriba abajo y de izquierda a derecha) de reglas.

Finalmente, una regla puede ser una relación $T \rightarrow T^0$ entre un par de configuraciones *T* y T^0 o la configuración inicial del sistema **`initially`** T. Una regla $T \rightarrow T^0$ es una relación entre dos configuraciones válidas llamadas *antecedente* (o *condición*) T y *consecuente* (o *acción*) T^0. Más explícitamente, la sintaxis de una regla es:

`rule` *nombre* :
$términoA_1, términoA_2, ..., términoA_m$
$\rightarrow$
$términoC_1, términoC_2, ..., términoC_n$.

ó para el caso de uan regla condicional:

`rule` *nombre* :
$términoA_1, términoA_2, ..., términoA_m \mid condición$
$\rightarrow$
$términoC_1, términoC_2, ..., términoC_n$.

en donde:

- nombre es el nombre de la regla introducido por la palabra reservada **`rule`,**
- dos puntos (:) es un delimitador indicando el inicio de la descripción de la regla,

- $términoA_1, términoA_2, ..., términoA_m$ es el antecedente formado por la composición concurrente de uno o más términos,
- barra (|) es un delimitador que indica el principio de la condición que es una restricción lógica
- flecha $\rightarrow$ denota la relacion de reescritura e indica el final del antecedente y el inicio del consecuente,
- $términoC_1, términoC_2, ..., términoC_n$ es el consecuente formado por la composición concurrente de uno o más términos y
- punto (.) es un símbolo de puntuación que indica el final de la regla.

En el lenguaje *HW*, la única restricción adicional que se impone sobre la estructura sintáctica de una regla es que debe aparecer exactamente el estado de un agente en el antecedente, como se ejemplifica en la sección siguiente. El propósito del antecedente de una regla es determinar, si existen, los términos que serán seleccionados para la aplicación de la regla, mientras que el propósito del consecuente es transformar a los elementos seleccionados. La aplicación de una regla sobre el espacio de términos es atómica y transaccional, ya que los términos que aparecen en el antecedente se eliminan mientras que los que aparecen en el consecuente se insertan en una sola acción indivisible (*principio de todo o nada*). Finalmente, **`initially`** T establece la configuración inicial T del sistema.

La configuración inicial describe el estado interno de todos los agentes así como el contenido de los espacios de términos introducidos mediante el operador **`new`**.

Para ilustrar intuitivamente la forma de las reglas, en la siguiente sección se describirán estructuras de flujo de control usadas en los *WF* prescriptivos tanto en la notación textual de *HW* como en la notación gráfica de las redes de Petri. Estos ejemplos permitirán introducir la intuitivamente la semántica del lenguaje *HW*.

4.3.2 Construcciones de Flujo de Control

La *perspectiva de comportamiento* de un *WF* se caracteriza mediante el *flujo del control* que establece relaciones de dependencia causal en el orden de ejecución entre flujos de tareas. Las interdependencias pueden describirse en niveles de abstracción que van desde los más detallados tipos de control prescriptivos a los tipos más descriptivos. Los *tipos de control prescriptivos* se basan en formas predefinidas que establecen con precisión la manera en que los *WFs* serán ejecutados, en tanto que los *tipos de control descriptivos* establecen las condiciones que esperan alcanzarse como resultado de la cooperación y colaboración de los *WFs* participantes.

La ejecución secuencial, alternativa y concurrente son las tres formas básicas de flujo de control prescriptivo que permiten obtener nuevos *WFs* por composición de otros más simples.

4.3.2.1 Ejecución secuencial

La ejecución secuencial *Seq* de dos *WFs A* y *B* se define como el comportamiento de *A* (hasta su terminación) seguido del comportamiento de *B*. En la representación de *WFs* por redes de Petri, la ejecución secuencial se obtiene por composición de los *WFs* de *A* y *B*, como se ilustra en la Figura 4-4.

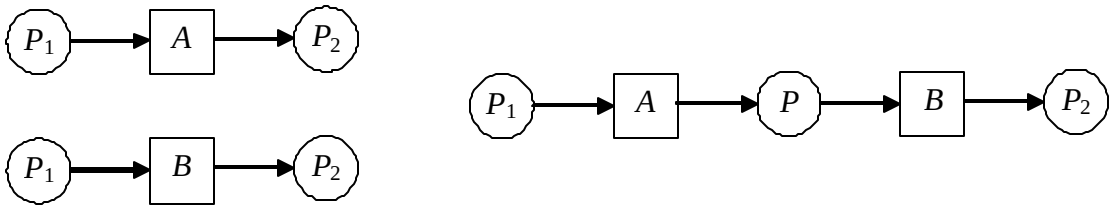


Figura 4-4 Ejecución secuencial de *A* y *B*

En el lenguaje de coordinación *HW*, la composición secuencial se obtiene en forma similar a las redes de Petri, al hacer coincidir el lugar que representa el término de *A* con el que representa el inicio de *B*.

Dadas las transiciones $A()(P_1, P_2)$ y $B()(P_1, P_2)$ y la función de flujo dada por las reglas:

rule $R_1 : A()(P_1, P_2), P_2(x) \mid y = f_1(x) \rightarrow P_2(y), A()(P_1, P_2)$ y

rule $R_2 : B()(P_1, P_2), P_1(x) \mid y = f_2(x) \rightarrow P_2(y), B()(P_1, P_2)$,

en donde f_1 y f_2 son dos funciones. La composición secuencial:

$Seq()(P_1, P_2) \rightarrow \mathbf{new}(P)(A()(P_1, P), B()(P, P_2))$

se obtiene al compartir el lugar *P* de salida de *A* con el lugar de entrada de *B*.

La composición secuencial es muy restrictiva en el orden de ejecución que *A* y *B* pueden tomar ya que a lo más uno de ellos puede estar en ejecución y que *B* puede comenzar solamente hasta que *A* termine. Esta descripción corresponde con la interpretación de la semántica del lenguaje de coordinación ya que la regla R_2 sólo puede habilitarse cuando existe al menos un elemento en P_2.

Como ejemplo de composición secuencial considere el problema de contar el número de veces que un artículo ha sido revisado como parte del proceso de selección de una conferencia. Dado el término *art*(*_ref*) que identifica al artículo por el número de referencia *_ref* que le fue asignado y cuyo contenido se encuentra depositado en una base de datos, el *WF Assoc* establece la asociación del artículo *_ref* con su contador *_cnt* de revisiones se representa con el término *artcnt*(*_ref*,*_cnt*). Por otra parte, el *WF Incr* incrementa simplemente el contador de lecturas. El comportamiento de ambos *WF*s está dado respectivamente por las siguientes reglas:

rule R_1: $Assoc()(\_A, \_B), \_A(art(\_ref)) \rightarrow \_B(artcnt(\_ref, 0))$

rule R_2: $Incr()(\_A, \_B), \_A(artcnt(\_ref, \_cnt_0)) \mid \_cnt_1 \ is \ \_cnt_0 + 1 \rightarrow \_B(artcnt(\_ref, \_cnt_1))$

La composición secuencial de *Assoc* e *Incr* está dada por la configuración:

$\mathbf{new}(A_1, A_2, A_3)(Assoc()(A_1, A_3), Incr()(A_3, A_2), A_1(art(123)))$

en donde, el operador **new** introduce los lugares A_1, A_2 y A_3 siendo A_3 el lugar compartido por ambos *WF*s. Observe que el término $A_1(art(123))$ habilita la regla R_1 para que inicie el *WF Assoc*.

4.3.2.2 Ejecución alternativa

La ejecución alternativa Alt de dos *WFs* A y B se define como la ejecución o bien de A o bien de B pero no de ambos. En la representación de *WFs* por redes de Petri, la ejecución alternativa se obtiene por composición de los *WFs* A y B, como se ilustra en la Figura 4-5.

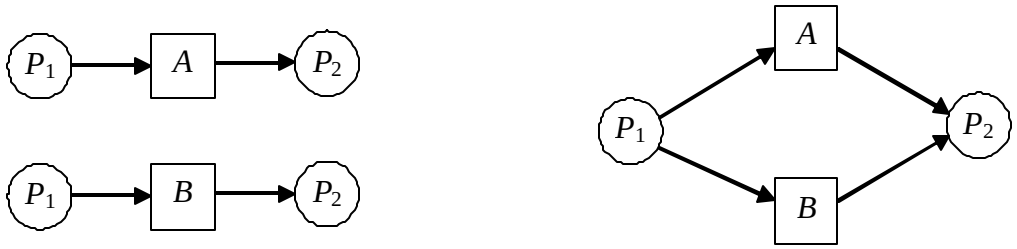


Figura 4-5 Ejecución alternativa de A y B

La composición alternativa se obtiene al compartir los lugares de entrada y de salida de ambos A y B. El comportamiento de la ejecución alternativa de dos *WFs* A y B es o bien como el de A o bien como el de B pero no ambos. En ésta forma de composición, las rutas alternativas del flujo del control son mutuamente exclusivas y no están limitadas a solamente dos procesos.

Como antes, dadas las transiciones $A()(P_1,P_2)$ y $B()(P_1,P_2)$ y la función de flujo dada por las reglas:

rule $R_1 : A()(P_1,P_2), P_1(x) \mid y = f_1(x) \rightarrow P_2(y), A()(P_1,P_2)$ y

rule $R_2 : B()(P_1,P_2), P_1(x) \mid y = f_2(x) \rightarrow P_2(y), B()(P_1,P_2),$

en donde f_1 y f_2 son dos funciones. La composición alternativa:

$Alt()(P_1,P_2) \rightarrow A()(P_1,P_2), B()(P_1,P_2)$

se obtiene al compartir el lugar P_1 de entrada y el lugar P_2 de salida para ambas transiciones A y B.

Cómo se mencionó antes, la ejecución de una regla en *HW* es transaccional, rigiéndose por el principio de "todo o nada". Aún cuando las reglas R_1 y R_2 puedan habilitarse simultáneamente por compartir el mismo lugar de entrada, solamente una de ellas se elegirá imparcial y no determinísticamente para reducción. La ejecución alternativa, en combinación con la ejecución nula y la secuencial, es la base para obtener las construcciones de control clásicas como bifurcación e iteración.

Para ejemplificar la composición alternativa considere el problema de determinar si un artículo ha sido revisado un número máximo de veces como parte del proceso de selección de artículos para una conferencia. Dado el término *artcnt*(*_ref*,*_cnt*) que asocia al artículo *_ref* con su contador de revisiones *_cnt*, cuando el contador ha alcanzado el número máximo de tres revisiones, el *WF Max* modifica la estructura del término a *artrev*(*_ref*) para indicar que el artículo ya ha sido revisado. Por otra parte, el *WF Inc* incrementa simplemente el contador de lecturas siempre que no exceda al máximo. El comportamiento de ambos *WF*s está dado respectivamente por las siguientes reglas:

rule R_3: $Max()(\_A, \_B), \_A(artcnt(\_ref, \_cnt)) \mid \_cnt = 3 \rightarrow \_B(artrev(\_ref))$

rule R_4: $Inc()(\_A, \_B), \_A(artcnt(\_ref, \_cnt_0)) \mid \_cnt_0 < 3 \wedge \_cnt_1 \; is \; \_cnt_0 + 1 \rightarrow \_B(artcnt(\_ref, \_cnt_1))$

La composición alternativa de *Max* e *Inc* está dada por la configuración:

new$(A_1, A_2)(Max()(A_1, A_2), Inc()(A_1, A_2), A_1(artcnt(123,2)))$

en donde, el operador **new** introduce los lugares A_1 y A_2 compartidos por ambos *WF*s. Observe que el término A_1(*artcnt*(123,2)) habilita únicamente a la regla R_2 ya que es la única que satisface su restricción lógica cuando se aplica la substitución *_A*→A_1, *_B*→A_2, *_ref*→123, *_cnt*$_0$→2, *_cnt*$_1$→3.

4.3.2.3 Ejecución concurrente

La ejecución concurrente Par de dos *WFs A* y *B* se define como la ejecución simultánea de ambos. En la representación de *WFs* por redes de Petri, la ejecución concurrente se obtiene por composición de los *WFs A* y *B*, como se ilustra en la Figura 4-6.

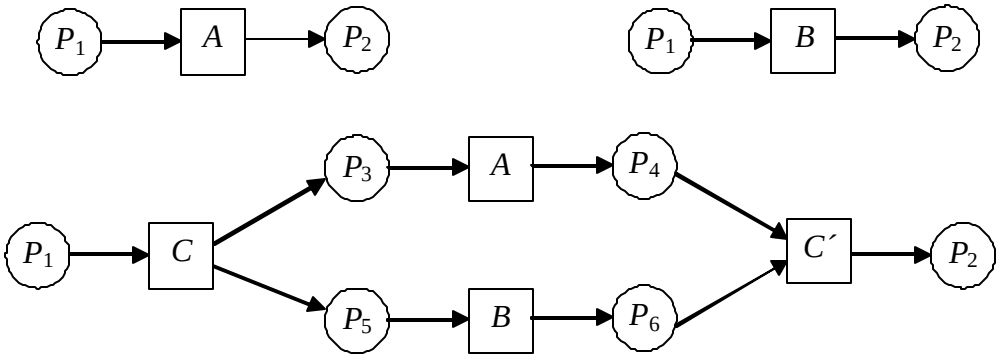


Figura 4-6 Ejecución concurrente de *A* y *B*

La composición concurrente se obtiene al permitir la ejecución simultánea de los *WFs A* y *B*. Más precisamente, dadas las transiciones $A()(P_1,P_2)$ y $B()(P_1,P_2)$ y la función de flujo dada por las reglas:

rule $R_1 : A()(P_1,P_2), P_1(x) \mid y = f_1(x) \rightarrow P_2(y), A()(P_1,P_2)$ y

rule $R_2 : B()(P_1,P_2), P_1(x) \mid y = f_2(x) \rightarrow P_2(x), B()(P_1,P_2),$

en donde f_1 y f_2 son dos funciones. La composición concurrente:

$Par()(P_1,P_2) \rightarrow \textbf{new}(P_3,P_4,P_5,P_6)\big(C()(P_1,P_3,P_5), A()(P_3,P_4), B()(P_5,P_6), C'()(P_4,P_6,P_2)\big)$

se obtiene al introducir los lugares P_3 y P_5 como lugares de entrada de *A* y *B*, respectivamente, los lugares P_4 y P_6 como lugares de salida de *A* y *B*, respectivamente, y las transiciones *C* y *C´* para conectar el lugar de entrada P_1 con P_3 y P_5 y el lugar de salida P_2 con P_4 y P_6, respectivamente. La regla R_3 tiene por objetivo el habilitar al mismo tiempo las transiciones *A* y *B*, mientras que la regla R_4 tiene por objeto esperar a que ambos *WFs* hayan terminado. En otras palabras, la regla R_3 divide el flujo de control en dos, mientras que la regla P_5 sincroniza ambos flujos de control en uno solo.

$R_3 : C()(P_1,P_2,P_3), P_1(x) \rightarrow P_2(x), P_3(x), C()(P_1,P_2,P_3)$

$R_4 : C'()(P_1,P_2,P_3), P_1(x), P_2(x) \rightarrow P_3(x), C'()(P_1,P_2,P_3)$

En ésta sección se describió las tres construcciones básicas de control para *WF* prescriptivos, discutiendo informalmente algunos aspectos de su comportamiento. En la siguiente sección se describirá formalmente el modelo de coordinación *Hyperworld* .

4.3.3 Leyes de coordinación

En el modelo *HW*, las leyes de coordinación se formalizan como teorías en una lógica de reescritura. Una teoría de reescritura comprende un conjunto de reglas de reescritura que describen las transformaciones locales válidas que pueden tener las entidades coordinables. La teoría de reescritura es la representación formal de los principios que subyacen en el comportamiento del sistema. La importancia de una teoría de reescritura consiste en proporcionar un procedimiento para calcular la evolución de sistemas concurrentes lógicamente distribuidos. Así como el estado distribuido del sistema corresponde a un conjunto de proposiciones lógicas, las transiciones entre estados locales corresponden a la aplicación de reglas de reescritura. En la teoría de reescritura la relación de derivación lógica entre dos estados puede obtenerse por un procedimiento que calcula el siguiente estado local válido del sistema distribuido.

La *signatura ecuacional* $(E, \approx)$, en donde el símbolo $\approx$ denota la relación de equivalencia sintáctica, contiene los axiomas de asociatividad, conmutatividad e identidad $E = \{(h, \varnothing) \approx h, (h, (h', h'')) \approx ((h, h'), h''), (h, h') \approx (h', h)\}$, con respecto a la composición concurrente. Una regla de reescritura define una relación entre dos patrones significativos de entidades coordinables que describen los estados previo y posterior de la transición local. Una regla $[h]_E \rightarrow [h']_E$ de reescritura de multiconjuntos de términos se define como una relación entre dos patrones de términos en el medio de coordinación. Sin embargo, preferimos escribir $h \rightarrow h'$ por brevedad tomando los representantes de las clases de equivalencia. En tal caso, el elemento equivalente de h se llama término o expresión reducible (redex) y su substitución por el correspondiente elemento equivalente de h' se llama reducción. La semántica operacional de *HW* está dada por una teoría de reescritura que induce una relación de derivación o de transformación de regiones disjuntas del medio de coordinación.

Una *teoría de reescritura* $\mathsf{R}(E, R)$ consiste de una signatura ecuacional E y de un conjunto de reglas R de reglas reescritura, como la que se muestra en Figura 4-7.

$$\frac{E \vdash h \approx h'}{\mathsf{R} \vdash h \rightarrow h'} \quad \textbf{(R1)} \qquad \frac{\mathsf{R} \vdash h \rightarrow h' \qquad \mathsf{R} \vdash h' \rightarrow h''}{\mathsf{R} \vdash h \rightarrow h''} \quad \textbf{(R2)}$$

$$\frac{\mathsf{R} \vdash h \rightarrow h'}{\mathsf{R} \vdash h, h'' \rightarrow h', h''} \quad \textbf{(R3)} \qquad \frac{\mathsf{R} \vdash h \rightarrow h'}{\mathsf{R} \vdash \textbf{new}(n)(h) \rightarrow \textbf{new}(n)(h')} \quad \textbf{(R4)}$$

$$\frac{p \Leftarrow P \in B}{\mathsf{B} \vdash p \Leftarrow P} \quad \textbf{(R5)} \qquad \frac{h \rightarrow h' \in R}{\mathsf{R} \vdash h \rightarrow h'} \quad \textbf{(R6)}$$

$$\frac{\mathsf{B} \vdash p \Leftarrow P \qquad \mathsf{B} \vdash q \Leftarrow p \wedge Q}{\mathsf{B} \vdash q \Leftarrow P \wedge Q} \quad \textbf{(R7)} \qquad \frac{\mathsf{R} \vdash h \mid P \rightarrow h' \qquad \mathsf{B} \vdash \Leftarrow P}{\mathsf{R} \vdash h \rightarrow h'} \quad \textbf{(R8)}$$

Figura 4-7 Teoría de reescritura

La relación de derivación $\mathsf{R} \vdash h \rightarrow h'$ es válida si la proposición $h \rightarrow h'$ se puede demostrar en la teoría de reescritura R en un número finito de aplicaciones de las reglas de inferencia **R1** a **R8**. El conjunto de reglas R contiene un subconjunto de cláusulas de Horn las cuales definen una teoría deductiva $\mathsf{B} \vdash p \Leftarrow P$ mediante la aplicación de las reglas de inferencia **R5** y **R7**. La teoría deductiva B es completa por refutación y usa negación por fallo.

En el axioma de **reflexividad** (**R1**), cualesquiera dos regiones sintácticamente equivalentes del medio de coordinación h y h' pueden rescribirse libremente una en substitución de la otra. En otras palabras, una región h del medio de coordinación puede rescribirse como h' (y recíprocamente) si h es sintácticamente equivalente a h' módulo asociatividad y conmutatividad. La reflexividad se usa para reordenar las entidades del medio de coordinación para probar la equivalencia sintáctica con el lado izquierdo de una regla de reescritura. En el axioma de **transitividad** (**R2**), una región del medio de coordinación h puede rescribirse como h'' si existe un estado intermedio h' tal que h pueda rescribirse

como h' y, a su vez, h' pueda rescribirse como h''. La transitividad provee un medio de construir cadenas de transiciones a partir de transiciones elementales. En la **congruencia bajo composición concurrente** (**R3**), las regiones del medio de coordinación que no se traslapan pueden evolucionar simultánea e independientemente. Una región del medio h puede transformarse en h', independientemente del ambiente externo h'', considerando condiciones puramente locales (principio conocido también como *principio de localidad*).

Dado que las reglas de reescritura no introducen nombres libres, en la **congruencia bajo restricción de nombres** (**R4**), una región del medio de coordinación h puede rescribirse como h' dentro del ámbito del nombre simbólico n ya que no pueden ocurrir vinculaciones inválidas con n. A diferencia de las reglas en los sistemas de reescritura clásicos, en HW se pueden introducir nombres de variables que no aparecen en el lado izquierdo de una regla; sin embargo, dichos nombres no son libres ya que sólo se pueden introducir mediante el operador de restricción el cual limita su ámbito. Las reglas de inferencia (**R5**) y (**R6**) son las reglas de **instanciamiento**. Dada la substitución $: X \rightarrow X$, la substitución extendida $: X \rightarrow \mathsf{T}(X)$ permite obtener instancias $p \Leftarrow P$ de cláusulas en B y $h \rightarrow h'$ de reglas bajo . El propósito de las reglas de instanciamiento es de obtener una substitución que permita derivar instancias que puedan usarse en las reglas de inferencia de resolución.

Las reglas de inferencia (**R7**) y (**R8**) son llamadas reglas de **resolución**. La regla (**R7**) puede compararse con la interpretación procedural de la resolución SLD sobre cláusulas determinadas. La interpretación procedural es un procedimiento computacional muy efectivo que permite realizar deducción en teorías escritas en cláusulas de Horn. Dicha interpretación consiste en considerar a cada cláusula de la teoría como la definición de un procedimiento y a cada meta como la llamada a un procedimiento. Formalmente, dado el procedimiento asociado a la cláusula $p \Leftarrow P$, un paso en la computación de una meta $\Leftarrow q \wedge Q$ se obtiene:

1. obteniendo una instancia $p \Leftarrow P$ apropiada del procedimiento mediante la regla de instanciamiento **R5**, en donde θ es el unficador de p y q ($p = q$),

2. substituyendo el encabezado p por el cuerpo P, reduciendo así la meta a $\Leftarrow P \wedge Q$.

La aplicación de la regla de instanciamiento permite instanciar las nuevas variables que vayan apareciendo en la meta reducida. De esta forma, la unificación de términos provee un mecanismo uniforme para el paso de parámetros, comparación, selección y construcción de términos. La computación dada por la meta continúa hasta que se deriva la meta nula, dando como resultado un conjunto de variables instanciadas.

La regla (**R8**) puede compararse con los sistemas de reescritura condicional de términos. Cuando en una regla con restricciones lógicas en el antecedente, se ha derivado la meta nula mediante la regla de resolución (**R7**), se concluye que la restricción lógica del antecedente de la regla es consecuencia lógica de los axiomas de la teoría.

La interpretación de procedimiento de servicio es un método computacional efectivo que permite realizar deducción en teorías escritas con reglas de reescritura. Dicha interpretación consiste en considerar a cada regla de reescritura como la definición de un procedimiento de servicio (*call-back*) y la aparición de un término en un espacio como la ocurrencia del evento que invoca al procedimiento de servicio. Formalmente, dado el procedimiento de servicio asociado a la regla $h \mid P \rightarrow h'$, un paso en la computación de una meta $\rightarrow g,G$ se obtiene:

1. obteniendo una instancia $h \mid P \rightarrow H$ mediante la regla de instanciamiento **R6** tal que $\Leftarrow P$ sea consecuencia lógica de la teoría (aplicando la regla de resolución **R7**), en donde θ es el unficador de h y g ($h = g$),
2. substituyendo el encabezado h por el cuerpo h', reduciendo así la meta a $\rightarrow h,G$.

Las reglas de instanciamiento (**R5** y **R6**) permiten instanciar las nuevas variables que aparezcan en la expresión reducida. La computación dada por el multiconjunto continúa

hasta que no es posible aplicar más reglas, dando como resultado además del conjunto de variables instanciadas, nuevos multiconjuntos de términos.

La presencia de términos de segundo orden en el antecedente no hace más compleja la unificación de términos. La razón es que el antecedente contiene el estado de (exactamente) un agente el cuál incluye a todas las variables de predicado usadas en la regla, por lo que al unificar el agente con un término sin variables, las variables de predicado quedan inmediatamente instanciadas. Para garantizar que esto simpre sea posible, el modelo de coordinación *HW* establece que los únicos términos representantes de estados de agentes que pueden participar en una reducción son aquellos que no contienen variables. Con esta restricción a la regla **R8**, una vez instanciadas todas las variables de predicados, la regla de resolución **R7** se puede aplicar normalmente.

Asumiendo que una región h del medio de coordinación pueda transformarse en otra h', la misma transformación puede ocurrir en el contexto de una restricción lógica. Puesto que la relación de reescritura no introduce nombres libres, los nombres de las variables lógicas se mantienen vinculadas apropiadamente. Al mismo tiempo, la operación de renombramiento sólo puede ocurrir dentro del ámbito de un operador de restricción de nombres, tal como en $\mathbf{new}(n)(t \mid P)$ por lo que no es posible renombrar n en el término t sin renombrar al mismo tiempo a n en la restricción lógica P.

Para concluir la descripción del modelo de coordinación *HW*, a continuación se mostrará cómo se calcula la configuración del estado de un *SAWF* por la composición alternativa de dos *WF*s, *Max* e *Inc*, introducidos en la sección 4.3.2.2.

rule R_3: $Max()(\_A, \_B), \_A(artcnt(\_ref, \_cnt)) \mid \_cnt = 3 \rightarrow \_B(artrev(\_ref))$

rule R_4: $Inc()(\_A, \_B), \_A(artcnt(\_ref, \_cnt_0)) \mid \_cnt_0 < 3 \wedge \_cnt_1 \; is \; \_cnt_0 + 1 \rightarrow \_B(artcnt(\_ref, \_cnt_1))$

Suponiendo que los predicados = e *is* son predicados primitivos, para propósitos ilustrativos se considera al predicado < (menor que) definido por el usuario mediante las siguientes cláusulas:

rule R_5: $\_x < \_y \Leftarrow (\_x + 1) \leq \_y$.

rule R_6: $0 \leq \_x \Leftarrow$.

rule R_7: $\_x + 1 \leq \_y + 1 \Leftarrow \_x \leq \_y$.

La composición de *Max* e *Inc* que analizaremos aquí está dada por la configuración:

new$(A_1, A_2)(A_1(artcnt(123,2)), Max()(A_1, A_2), Inc()(A_1, A_1))$

la cual, de acuerdo a la regla de reflexividad (**R1**), es equivalente a:

new$(A_1, A_2)(Max()(A_1, A_2), Inc()(A_1, A_1), A_1(artcnt(123,2)))$

aplicando las propiedades asociativa y conmutativa de la composición concurrente.

Una breve inspección sobre las restricciones lógicas de las reglas *Max* e *Inc* revela que solamente ésta última podrá aplicarse exitosamente. Por lo tanto, aplicando la substitución $\_A \mapsto A_1, \_B \mapsto A_1$ al término $Inc()(\_A, \_B)$ se obtiene la siguiente instancia parcial de la regla R_4:

$$Inc()(A_1, A_1), A_1(artcnt(\_ref, \_cnt_0)) \mid \_cnt_0 < 3 \wedge \_cnt_1 \; is \; \_cnt_0 + 1 \rightarrow A_1(artcnt(\_ref, \_cnt_1))$$

quedando aún por instanciar las variables *_ref*, $\_cnt_0$ y $\_cnt_1$. Para determinarlas, se forma la siguiente meta:

$\Leftarrow A_1(artcnt(\_ref, \_cnt_0)) \wedge \_cnt_0 < 3 \wedge \_cnt_1 \; is \; \_cnt_0 + 1$.

Dado que la cláusula unitaria $A_1(artcnt(123,2)) \Leftarrow$ es el único hecho en la teoría, al aplicar resolución a ambas se obtiene la substitución $\_ref \mapsto 123, \_cnt_0 \mapsto 2$ junto con la meta parcial:

$\Leftarrow 2 < 3 \wedge \_cnt_1 \; is \; 2 + 1$.

Las siguientes metas parciales se obtienen al usar la definición de la relación < (menor que)

$\Leftarrow (2 + 1) \leq 3 \wedge \_cnt_1 \; is \; 2 + 1$ (por **R7** con $2 < 3 \Leftarrow (2 + 1) \leq 3$ instancia de R_5)

$\Leftarrow 2 \leq 2 \wedge \_cnt_1 \; is \; 2 + 1$ (por **R7** con $2 + 1 \leq 2 + 1 \Leftarrow 2 \leq 2$ instancia de R_7)

$\Leftarrow 1 \leq 1 \wedge \_cnt_1\ is\ 2+1$ (por **R7** con $1+1 \leq 1+1 \Leftarrow 1 \leq 1$ instancia de R_7)

$\Leftarrow 0 \leq 0 \wedge \_cnt_1\ is\ 2+1$ (por **R7** con $0+1 \leq 0+1 \Leftarrow 0 \leq 0$ instancia de R_7)

$\Leftarrow \_cnt_1\ is\ 2+1$ (por **R7** con $0 \leq 0 \Leftarrow$ instancia de R_6)

$\Leftarrow$ (cláusula vacía) (por **R7** con $\_cnt_1\ is\ 2+1 \Leftarrow$ produciendo $\_cnt_1 \mapsto 3$)

Recuerde que *is* es un predicado primitivo que evalua la operación aritmética que está a su derecha y asigna el resultado a la variable que está a su izquierda. Una vez instanciadas las variables faltantes, se puede instanciar completamente la regla *Inc*:

$Inc()(A_1, A_2), A_1(artcnt(123,2)) \mid 2 < 3 \wedge 3\ is\ 2+1 \rightarrow A_1(artcnt(123,3))$

la cual puede reescribirse sin restricciones lógicas:

$Inc()(A_1, A_2), A_1(artcnt(123,2)) \rightarrow A_1(artcnt(123,3))$

ya que la restricción $2 < 3 \wedge 3\ is\ 2+1$ se satisface.

Aplicando la regla de congruencia bajo composición concurrente (**R3**) con ésta instancia, se puede inferir que:

$Max()(A_1, A_2), Inc()(A_1, A_2), A_1(artcnt(123,2)) \rightarrow Max()(A_1, A_2), A_1(artcnt(123,3))$

En forma similar, se puede demostrar que:

$Max()(A_1, A_2), A_1(artcnt(123,3)) \rightarrow A_2(artrev(123))$

Aplicando la regla de transitividad (**R2**) a las dos inferencias anteriores se deduce:

$Max()(A_1, A_2), Inc()(A_1, A_2), A_1(artcnt(123,2)) \rightarrow A_2(artrev(123))$

Finalmente, aplicando la regla de congruencia bajo restricción de nombres (**R4**), se infiere:

$\mathbf{new}(A_1, A_2)(Max()(A_1, A_2), Inc()(A_1, A_2), A_1(artcnt(123,2))) \rightarrow$

$\mathbf{new}(A_1, A_2)(A_2(artrev(123)))$

que describe la configuración final sobre la cual ya no es posible aplicar más cualquier regla de reescritura.

Esto concluye nuestro ejemplo y con él, nuestra descripción del lenguaje de coordinación *HW*.

4.4 Conclusiones

En este capítulo se describieron los conceptos de modelo de coordinación y lenguaje de coordinación. Se describieron las características del lenguaje de coordinación Linda. Linda cuenta con un conjunto reducido de operaciones que tienen efecto sobre un espacio de tuplas, que permite la coordinación de agentes de software de forma simple.

Se describieron las características del modelo de coordinación *HW*, el cual extiende al modelo de Linda introduciendo términos con variables lógicas y satisfacción de restricciones. Se hicieron comparaciones entre las similitudes del modelo de las redes de Petri y del modelo de *HW* con la finalidad de mostrar su capacidad para describir modelos de concurrencia. Se describió la sintaxis abstracta de *HW* con la que se pueden representar los modelos que de las redes de Petri, las restricciones lógicas y las restricciones dinámicas de ámbito.

Se describieron las construcciones de flujo de control con las cuales se puede modelar la perspectiva de comportamiento de un *WF*. Las estructuras de flujo de control modela la ejecución secuencial, alternativa y concurrente de un *WF*. Por último se describieron los conceptos lógicos de las leyes de coordinación que incluyen la signatura ecuacional y la teoría de reescritura.

La descripción del lenguaje de coordinación *HW* muestra los elementos incluidos en Linda más términos lógicos que permiten aumentar su capacidad para describir la interacción de agentes de software y la representación de abstracta de datos en sistemas de colaboración como son los *SAWFs*. Sus estructuras de control, las cuales extienden al modelo de las redes de Petri, permiten la modelación y descripción de las diferentes formas de ejecución de un *WF*. En general, *HW* presenta un modelo lógico formal con la capacidad de diseñar *SAWFs* de manera uniforme.

En el siguiente capítulo se describen el caso de estudio y la solucion al problema usando el lenguaje *HW*.

Capítulo 5 Caso de estudio: Coordinación de actividades en una conferencia

En este capítulo se presenta el caso de estudio[33] seleccionado para mostrar el uso del enfoque que se planteo en capítulos anteriores. Se describen a los participantes y las interacciones que existen entre ellos, también se explican en detalle las soluciones para administrar éstas interacciones empleando el lenguaje de coordinación *HW*.

5.1 Coordinación de actividades en una conferencia

Para demostrar la aplicación y capacidades del prototipo desarrollado en ésta tesis se presenta un caso de estudio llamado "Coordinación de actividades en una conferencia", ésta aplicación tiene la función de administrar la interacción de tres tipos de participantes: *autor*, *secretaria* y *revisor*; además de las actividades que intervienen en la selección de artículos para la inclusión en la conferencia: *envío de artículos*, *selección* y *revisión* de los mismos, así como la notificación de aceptación. A continuación se describen las actividades de la conferencia usando los pasos ilustrados en la Figura 5-1.

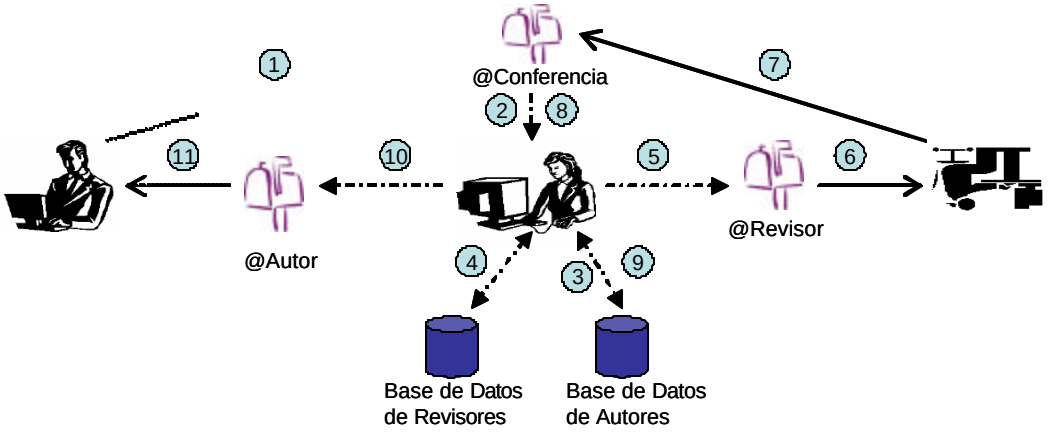


Figura 5-1 Actividades de una conferencia

5.2 Actividades de la conferencia

La interacción de los participantes se inicia cuando el Autor produce un artículo y lo envía al correo electrónico de la conferencia para su revisión (Paso 1). La secretaria recibe el artículo (Paso 2) y lo registra en una base de datos de autores asignándole un número de referencia (Paso 3). Después usa las palabras claves que este incluye para asignarle un revisor adecuado, esto se hace buscando que las palabras claves coincidan con las áreas de interés almacenadas en la base de datos de revisores (Paso 4). Una vez que se encuentra al revisor adecuado la secretaria le envía el artículo a su correo electrónico (Paso 5).

El revisor recibe el artículo (Paso 6), lo lee, y toma una decisión sobre su aceptación. La información sobre la aceptación del artículo se envía nuevamente al correo electrónico de la conferencia (Paso 7). La secretaria revisa los mensajes de respuesta de los revisores (Paso 8) y toma el número de referencia del artículo, el cual usa para consultar la base de datos de autores y obtener el correo electrónico del autor (Paso 9). La secretaria entonces envía al autor el resultado de la revisión del artículo (Paso 10). Finalmente el autor revisa su correo electrónico para conocer el resultado de la evaluación de artículo (Paso 11).

5.3 La solución usando Hyperworlds

El lenguaje de coordinación *HW* provee una notación uniforme para la descripción de procesos que involucran flujo de documentos y de un cálculo simbólico para interpretar dichas descripciones. En *HW*, la arquitectura de coordinación consiste de una colección de máquinas virtuales distribuidas en Internet para dar soporte a la interacción concurrente de programas que dan una interpretación apropiada a las abstracciones del modelo de coordinación.

En la Figura 5-2 aparece un fragmento de la descripción del sistema de conferencia escrita en *HW* la cual incluye números de línea para facilitar su referencia. Las reglas son leídas de

un archivo de texto por la máquina virtual *HW*, se verifican sinctácticamente, se convierten a un programa Prolog, el cual finalmente se interpreta.

```
% Fragmento de la especificación del comportamiento del autor
[ 1]rule Submitting:
[ 2]Author()(_SubmissionForm,_ConferenceMail,_AuthorMail),
[ 3] _SubmissionForm(submit(_author,_title,_keywords,_abstract,_eMail))
[ 4] ->
[ 5] AuthorWaiting(_author,_title,_keywords,_abstract,_eMail)
[ 6] (_SubmissionForm,_ConferenceMail,_AuthorMail),
[ 7]
_ConferenceMail(submitted(_author,_title,_keywords,_abstract,_eMail)).
% Fragmento de la especificación del comportamiento de la secretaria
[ 8]rule Receiving:
[ 9]Secretary(_deadline)(_ConferenceMail,_AuthorsDB,_RefereesDB),
[10]
_ConferenceMail(submit(_author,_title,_keywords,_content,_AuthorMail))
[11] | _keyword ^ (member(_keyword,_keywords),
[12] _RefereesDB(referee(_reviewer,_keyword,_RefereeMail)))
[13] ->
[14] new(Reference)(
[15] Secretary(_deadline)(_ConferenceMail,_ AuthorsDB,_RefereesDB),
[16] _AuthorsDB(author(Reference,_author,_title,_AuthorMail)),
[17] _AuthorMail(received(_author,_title,_Reference)),
[18] _RefereeMail(review(Reference,_title,_keywords,_content))).
[19]
% Fragmento de la especificación del comportamiento del revisor
[20]rule Reviewing:
[21]Referee(_name)(_RefereeMail,_ConferenceMail,_ReviewForm),
[22] _RefereeMail(review(_reference,_title,_keywords,_content)),
[23] _ReviewForm(answer(yes))
[24] ->
[25] Referee(_name)(_RefereeMail,_ConferenceMail,_ ReviewForm),
[26] _ConferenceMail(accepted(_reference,_title,_keywords,_content)).
% Estado inicial
[27]initially
[28] new(SubmissionForm~>httpServer(2000),
[29] ReviewForm~>httpServer(2001),
[30] AuthorMail~>httpServer(2002),
[31] RefereeMail~>httpServer(2002),
[32] AuthorsDB, RefereesDB
[33] )(
[34] Secretary(20)(CIEMail,AuthorsDB,RefereesDB),
[35] Referee(`Dra Li`)(RefereeMail,CIEMail,ReviewForm),
[36] Referee(`Dra Martinez`)(RefereeMail,CIEMail,ReviewForm),
[37] RefereesDB(referee(`Dra Li`,`Petri Nets`,RefereeMail)),
[38] RefereesDB(referee(`Dra Li`,`Workflows`,RefereeMail)),
[39] RefereesDB(referee(`Dra Martinez`,`Artificial Intelligence`,
RefereeMail)),
[40] RefereesDB(referee (`Dra Martinez`,`CSCW`,RefereeMail))
[41] )
```

Figura 5-2 Fragmento de las reglas del sistema conferencia

La maquina virtual *WF* se define en términos de la maquina virtual *HW*, y las reglas que rigen el comportamiento del *WF*, también llamadas reglas de negocios. Es decir:

Maquina Virtual *WF* = Maquina virtual *HW* + reglas de negocios

Las tres clases de participantes, el autor, la secretaria y el revisor, se describen en *HW* mediante agentes o entidades activas que exhiben cierta autonomía y que son capaces de colaborar con otros agentes para realizar tareas en representación de sus usuarios.

Las reglas describen las transiciones validas que el estado de un agente puede tener. Cabe hacer notar que la unica diferencia sintáctica en las reglas que se describen en este capítulo con la dada en la definición del lenguaje HW es el reemplazo del símbolo de conjunción $\wedge$ por la coma (,) en las restricciones; esto debido a que la implementación actual en Prolog del lenguaje HW usa la coma (,) como conectivo de conjunción. Un agente se representa por un término simbólico con una sintaxis caracterizada por su doble lista de términos delimitados por paréntesis. Los términos que aparecen en la primera lista se refieren al estado interno del agente, mientras que los valores de la segunda lista de variables se refieren a los nombres simbólicos de los depositos de datos a los que tiene acceso el agente. Por ejemplo, el agente secretaria se representa por el término:

```
Secretary(_deadline)(_ConferenceMail,_AuthorsDB, _RefereesDB)
```

en donde `Secretary` es el nombre del agente, `_deadline` es la información sobre la fecha límite para la recepción de artículos, los espacios de términos `_ConferenceMail` que representa la dirección de correo electrónico de la conferencia, `_AuthorsDB` es la base de datos de los autores participantes y `_RefereesDB` es la base de datos de los revisores de los artículos. Como se describió en el Capítulo 4, las variables usan una subrayan (_) como prefijo de su nombre. Los valores de las variables que aparecen en la primera lista se refieren al estado interno del agente, mientras que los valores de la segunda lista de variables se refieren a los nombres simbólicos de los depositos de datos a los que

tiene acceso el agente. Para que una regla se pueda aplicar a un agente, todas las variables de estado deben estar completamente instanciadas.

Puesto que *HW* es un lenguaje de reescritura de términos, una regla relaciona la región del espacio de términos que satisface la condición lógica del antecedente con la transformación indicada en el consecuente. Por lo tanto, no se requiere usar en las reglas *HW* los nombres de las operaciones sobre los espacios de términos ya que el contexto en el que ocurre el término determina la operación. Cuando el término ocurre en el antecedente de una regla, la operación corresponde a la de lectura. Si el término ocurre en el consecuente, la operación corresponde a la escritura. Por último, cuando el término ocurre en una restricción lógica (al lado derecho de la barra vertical), la operación corresponde a la de consulta, como ocurre en el ejemplo en (línea [12]) donde se hace una consulta a la base de datos `RefereesDB` sin modificarla. Podemos resumir diciendo que la reescritura de términos se consigue eliminando del espacio los términos que se encuentran a la izquierda de una regla e insertando los que aparecen a la derecha.

Los espacios de términos son una abstracción de varias formas conocidas de repositorios compartidos, persistentes y transaccionales de datos tales como el correo electrónico y las bases de datos. Dependiendo de la instancia particular de la abstracción del espacio de términos, las operaciones sobre los espacios de términos se seleccionan correspondientemente. En nuestro ejemplo, la regla `Receiving` (líneas [8-18]):

```
[ 8]rule Receiving:
[ 9]Secretary(_deadline)(_ConferenceMail,_AuthorsDB,_RefereesDB),
[10]_ConferenceMail(submit(_author,_title,_keywords,_content,_AuthorMail)
)
[11] | _keyword ^ (member(_keyword,_keywords),
[12] _RefereesDB(referee(_reviewer,_keyword,_RefereeMail)))
[13] ->
[14] new(Reference)(
[15] Secretary(_deadline)(_ConferenceMail,_ AuthorsDB,_RefereesDB),
[16] _AuthorsDB(author(Reference,_author,_title,_AuthorMail)),
[17] _AuthorMail(received(_author,_title,_Reference)),
[18] _RefereeMail(review(Reference,_title,_keywords,_content))).
```

indica que busca un mensaje con tópico `submit` (línea [10]) en el correo electrónico `_ConferenceMail` cuyo contenido debe asignarse a las variables `_author, _title, _keywords, _content y _AuthorMail`. Cuando se localiza un mensaje con tales características, éste se remueve de la bandeja de entrada y se determinan los valores de las variables con el contenido del mensaje. Por otra parte, el término `_AuthorsDB(author(Reference,_author, _title, _AuthorMail))` (línea[16]) indica que el registro `author(Reference, _author, _title, _AuthorMail)` se debe insertar en la base de datos `_AuthorsDB`. Finalmente, el término `_RefereesDB(referee(_reviewer, _keyword, _RefereeMail))` (línea [12]) es parte de una restricción lógica y significa que se hace una consulta para obtener instancias de las variables y, por lo tanto, el contenido de la base de datos queda inalterado.

5.4 Reglas de interacción

A continuación daremos una breve descripción de las reglas de interacción. La regla `Submitting` (líneas [1-7]):

```
[ 1]rule Submitting:
[ 2]Author()(_SubmissionForm,_ConferenceMail,_AuthorMail),
[ 3] _SubmissionForm(submit(_author,_title,_keywords,_abstract,_eMail))
[ 4] ->
[ 5] AuthorWaiting(_author,_title,_keywords,_abstract,_eMail)
[ 6] (_SubmissionForm,_ConferenceMail,_AuthorMail),
[ 7]_ConferenceMail(submitted(_author,_title,_keywords,_abstract,_eMail)).
```

describe el comportamiento del agente `Author` que representa al autor. El agente (línea [2]) carece de estado local (su primer lista está vacía) pero tiene acceso a la forma `_SubmissionForm` de registro de artículos que puede ser una forma CGI de captura. También tiene acceso a una cuenta de correo electrónico `_ConferenceMail` para enviar los datos del autor que acompañan al artículo y a una cuenta de correo propia `_AuthorMail`. Cuando el autor completa la forma de registro, la máquina virtual *HW* obtiene los datos y con ella construye el término que aparece en (línea [3]) para insertarlo en el espacio de términos, donde podrá habilitar otras reglas, por ejemplo, las que describen el comportamiento de la secretaria y el revisor. Una vez recibidos los datos, el agente

adquiere un estado interno formado con los datos personales del autor (línea [5]). El agente cambia su estado a `AuthorWaiting` (línea [5-6]), esperando los resultados de la evaluación del artículo. Al mismo tiempo, el agente envía al correo electrónico de la conferencia `_ConferenceMail` (línea [7]), el artículo (aquí solamente el abstract) y los datos del autor representados por el término `submitted(_author,_title, _keywords,_abstract,_eMail)`.

La regla `Receiving` (líneas [8-18]):

```
[ 8]rule Receiving:
[ 9]Secretary(_deadline)(_ConferenceMail,_AuthorsDB,_RefereesDB),
[10]_ConferenceMail(submit(_author,_title,_keywords,_content,_AuthorMail)
)
[11] | _keyword ^ (member(_keyword,_keywords),
[12] _RefereesDB(referee(_reviewer,_keyword,_RefereeMail)))
[13] ->
[14] new(Reference)(
[15] Secretary(_deadline)(_ConferenceMail,_ AuthorsDB,_RefereesDB),
[16] _AuthorsDB(author(Reference,_author,_title,_AuthorMail)),
[17] _AuthorMail(received(_author,_title,_Reference)),
[18] _RefereeMail(review(Reference,_title,_keywords,_content))).
```

describe el comportamiento del agente `Secretary` que actúa en representación de la secretaria. En (línea [9]), el agente determina los valores de las variables. Con la dirección de correo electrónico dado para `_ConferenceMail`, el agente busca un mensaje con tópico `submit` en dicha dirección de correo y vincula las variables con el contenido del mensaje (línea [10]). Sin embargo, el mensaje extraído deberá cumplir con la restricción lógica (|) dada en (líneas [11-12]). La restricción establece que las palabras clave del artículo denotadas por la variable `_keywords` incluidas en el mensaje deben ser aquellas para las que existe un revisor con dicha especialidad, la cual se denota por la variable `_keyword`. Esta condicion de pertenencia esta dada por la expresión Prolog `member(_keyword,_keywords)` (línea [11]).

En (línea [14]), el agente crea un nombre simbólico único (no reproducible) `Reference`, cuyo ámbito dinámico abarca a los términos que aparecen en (líneas [15-19]). Este nombre, `Reference`, es la referencia del artículo para el autor con el que puede dar seguimiento a su participación. En (línea[15]), el agente `Secretary` completa la transición local

regresando al mismo estado. Los datos del autor `author(Reference,_author,_title, _AuthorMail)` se insertan en la base de datos de autores `_AuthorsDB` en (línea [16]).

En (línea [17]), el agente `Secretary` manda a la dirección de correo electrónico del autor `_AuthorMail` la información de que el artículo con número de referencia `Reference` fue recibido. La última acción de ésta regla (línea [18]), es enviar el artículo sin el nombre del autor al correo electrónico del revisor `_RefereeMail`.

La regla `Reviewing` (líneas [20-26]):

```
[20]rule Reviewing:
[21]Referee(_name)(_RefereeMail,_ConferenceMail,_ReviewForm),
[22] _RefereeMail(review(_reference,_title,_keywords,_content)),
[23] _ReviewForm(answer(yes))
[24] ->
[25] Referee(_name)(_RefereeMail,_ConferenceMail,_ ReviewForm),
[26] _ConferenceMail(accepted(_reference,_title,_keywords,_content)).
```

describe el comportamiento del agente que actúa en representación del revisor. Esta regla se aplica cuando se recibe en la dirección de correo electrónico `_RefereeMail`, un mensaje con tópico `review` y con la información del artículo `_reference`, `_title`, `_keywords` y `_content` como contenido del mensaje (línea [22]). El agente del revisor tiene además acceso a `_ReviewForm` que representa una aplicación interactiva (típicamente a través de un navegador Web) mediante el cual recibe instrucciones al recibir la decisión del revisor en cuanto a la aceptación del artículo (línea [23]). La regla `Reviewing` modela el caso en el que el revisor ha decidido aceptar el artículo. La máquina virtual *WF* recibe la respuesta `answer(yes)`, enviada, por ejemplo, por una forma interactiva CGI. El agente del revisor regresa entonces a su estado inicial `Referee(_name)(_RefereeMail,_ConferenceMail,_ ReviewForm)`, listo para recibir más artículos para revisión (línea [25]). En el antecedente de la regla `Reviewing` se indica la lectura con eliminación del término que representa al agente revisor (línea [23]), con lo cual se deja a la máquina virtual *HW* sin información acerca del agente revisor; el consecuente de la regla indica la escritura del término `Referee(_name)(_RefereeMail,_ConferenceMail,_ ReviewForm)` que

representa al revisor (línea [25]), con lo cual se consigue que la máquina virtual *HW* tenga la información necesaria para poder ejecutar la regla `Reviewing` nuevamente si es necesario. Por último, el agente revisor envía un mensaje al correo electrónico de la conferencia `_ConferenceMail` con la decisión favorable que tomó para el autor representado por el término `accepted(_reference, _title, _keywords, _content)`.

De esta forma, cada vez que se recibe un artículo para revisión, la regla `Reviewing` se aplica. En cada aplicación de la regla se produce una instanciación de valores a las variables que aparecen en ella, en forma similar a la instanciación de variables en una llamanda a un procedimiento Prolog. Puesto que el termino `Referee(_name)(_RefereeMail,_ConferenceMail,_ ReviewForm)` aparece en el antecedente como en el consecuente, el regreso del agente revisor a su estado inicial queda asegurado permitiendo repetir la ejecución de esta regla un número arbitrario de veces.

En el caso en el que el revisor no acepte el artículo la regla `Reviewing` se escribiría de la manera siguiente:

```
[20]rule Reviewing:
[21]Referee(_name)(_RefereeMail,_ConferenceMail,_ReviewForm),
[22] _RefereeMail(review(_reference,_title,_keywords,_content)),
[23] _ReviewForm(answer(no))
[24] ->
[25] Referee(_name)(_RefereeMail,_ConferenceMail,_ ReviewForm),
[26] _ConferenceMail(rejected(_reference,_title,_keywords,_content)).
```

en donde el término `answer(no)` (línea [23]) representa la respuesta negativa y el término `rejected(_reference,_title,_keywords,_content)` (línea [26]) representa la decisión desfavorable que tomó el agente revisor para el artículo del autor.

Finalmente el estado inicial del sistema (líneas [27-41]) se describe de la manera siguiente:

```
[27]initially
[28] new(SubmissionForm~>httpServer(2000),
[29] ReviewForm~>httpServer(2001),
[30] AuthorMail~>httpServer(2002),
[31] RefereeMail~>httpServer(2002),
```

```
[32] AuthorsDB, RefereesDB
[33] )(
[34] Secretary(20)(CIEMail,AuthorsDB,RefereesDB),
[35] Referee(`Dra Li`)(RefereeMail,CIEMail,ReviewForm),
[36] Referee(`Dra Martinez`)(RefereeMail,CIEMail,ReviewForm),
[37] RefereesDB(referee(`Dra Li`,`Petri Nets`,RefereeMail)),
[38] RefereesDB(referee(`Dra Li`,`Workflows`,RefereeMail)),
[39] RefereesDB(referee(`Dra Martinez`,`Artificial
Intelligence`,RefereeMail)),
[40] RefereesDB(referee(`Dra Martinez`,`CSCW`,RefereeMail))
[41] )
```

se inicia con la palabra reservada **`initially`** (línea [27]); después se crean los espacios de términos usando la palabra reservada new (líneas [28-33]) a los cuales tendrán acceso los agentes. Los términos como `ReviewForm~>httpServer(2001)` (línea [29]) indican que `ReviewForm` representa la comunicación con el exterior a través del protocolo HTTP usando el puerto `2001`. Se continua con la descripción de los agentes `Secretary` y `Referee` (líneas [34-36]) representados por los términos `Secretary(20)(CIEMail, AuthorsDB,RefereesDB), Referee(`Dra Li`)( RefereeMail, CIEMail, ReviewForm), Referee(`Dra Martinez`)(RefereeMail, CIEMail, ReviewForm)` respectivamente. Por ejemplo, el término `Referee(`Dra Martinez`)(RefereeMail, CIEMail, ReviewForm)` describe al revisor con nombre `` `Dra Martinez` `` y que tiene acceso a los espacios de términos llamados `RefereeMail, CIEMail, ReviewForm`.

Por último se almacena información en la base de datos de revisores que esta representada por términos como `RefereesDB(referee(`Dra Martinez`,`Artificial Intelligence`,RefereeMail))`, el cual indica que el revisor `` `Dra Martinez` `` tiene especialidad en `` `,`Artificial Intelligence` `` y su correo electrónico es `RefereeMail`.

5.5 Conclusiones

En este capítulo se describió la solución al problema de la coordinación de actividades de una conferencia usando el lenguaje de coordinación *HW*. En este caso de estudio, debido

que sólo el agente secretaria decide a cual revisor se le asigna el artículo, la distribución de artículos se podría llamar *tiránica*. En esta forma de distribución se tiene la ventaja de que no se dejan artículos sin revisar ya que cada artículo se asigna a algún revisor. Pero se tiene la desventaja de que se asignan los artículos a los revisores aunque dichos artículos no sean de su interés.

En contraste se puede proponer otra forma de asignación de artículos, a la cual llamamos *democrática*, que consiste en que cada revisor selecciona el artículo que le interese revisar. La manera de implementar esta asignación es más simple ya que cada revisor indica cual es el artículo mediante el número de referencia, lo recupera y se incrementa un contador de número de revisiones asociado a dicho artículo. Al término de un plazo se revisan los contadores para determinar los artículos sin revisión. Esta forma de distribución de artículos tiene la ventaja de que no impone la revisión de los artículos a los revisores, pero tiene la desventaja de que algunos artículos no sean seleccionados y queden sin revisar.

Otra forma de distribución de artículos que evita dejar artículos sin seleccionar es una mezcla de las distribuciones *democrática* y *tiránica*. Es decir, los revisores tienen la posibilidad de seleccionar los artículos de su preferencia durante un plazo. Al término de este plazo los revisores dejan de seleccionar articulos y la secretaria interviene, mientras esto no ocurra la selección de artículos es *democrática*. Cuando la secretaria interviene, su tarea es determinar cuales artículos no fueron seleccionados y asignarlos a los revisores, es decir, la distribución de artículos se vuelve *tiránica*.

En el caso de estudio de la conferencia se considera el uso del correo electrónico como medio de comunicación. Las características del correo electrónico lo hacen propicio para este tipo de sistemas en donde la colaboración es asíncrona, lo cual sería difícil con otras tecnologías como WebDAV [61]. WebDAV también permite la colaboración, pero para ello cada usuario necesita ejecutar el software de WebDAV para poder participar del ambiente de colaboración, es decir, siendo un protocolo petición-respuesta, cuando un usuario ejecuta un comando de este protocolo, éste debe esperar por una respuesta inmediata, por lo que el tipo de colaboración se vuelve síncrona. Con el correo electrónico

no es necesario que los participantes de un ambiente de colaboración estén ejecutando su software de correo para poder participar, los participantes emiten los comandos de correo electrónico, por ejemplo, al enviar un mensaje y no tiene que esperar por una respuesta inmediata. Los participantes recibirán la notificación de sus mensajes en el momento en que ejecuten su software cliente de correo electrónico.

Otro enfoque para la solución de la administración de actividades es el de las bases de datos activas [45]. Las bases de datos activas utilizan reglas ECA (Evento-Condición-Acción) para reaccionar a eventos; con lo anterior se pueden describir *WFs* simples. Una actividad compuesta se puede descomponer en actividades simples entrelazadas, las cuales pueden ser descritas por reglas ECA cuyo resultados generen eventos que disparen otras reglas ECA que representan a las actividades siguientes hasta completar la actividad completa. La desventaja de las bases de datos activas es la rigidez de su modelo, ya que para cada tipo de tupla se tiene que hacer una definición de la misma en el esquema de la base de datos, es decir, crear una tabla, esto se vuelve complicado cuando se desea manejar información no estructurada o heterogénea en forma dinámica. Lo anterior no ocurre al usar un modelo de coordinación, como el de *HW*, el cual abstrae la información permitiendo la integración y el manejo de datos sin importar su estructura.

La solución descrita muestra el uso del lenguaje *HW* en un ambiente de colaboración asíncrono, como lo es el de una conferencia, pero *HW* tiene la capacidad de describir otros ambientes de colaboración, como la colaboración síncrona. Por ejemplo, en el caso de estudio presentado en este capítulo, un conflicto que genere la colaboración síncrona es la evaluación contradictoria de dos revisores sobre el mismo artículo. Una solución para resolver este conflicto es la comunicación directa entre los revisores a través de un medio síncrono como los sistemas de conversación en línea (*chat*). En este caso, la secretaria puede arreglar la conversación introduciendo enlaces hipermedia apropiados en las páginas Web leídas por los revisores. Cuando se seleccionan en el navegador, los enlaces hipermedia activan el sistema de conversación configurado para comunicar a ambos revisores.

El problema de la integración dinámica de herramientas de colaboración síncrona demuestra la necesidad de adaptación en *SAWFs* [29]. La importancia de los *SAWFs* adaptables radica en el efecto que tienen los cambios dinámicos sobre la disponibilidad y consistencia de la información, así como de la usabilidad, escalabilidad, confiabilidad y complejidad del *SAWF*. Aunque existen modelos teóricos como las redes de Petri usados para tratar éste problema [29], aún existe una distancia considerable para la derivación de sistemas reales a partir de las especificaciones formuladas con tales modelos. Aún cuando el problema general de adaptabilidad en *HW* no ha sido estudiado con profundidad, el lenguaje ofrece mecanismos para modelar arquitecturas dinámicas de software. Por ejemplo, es posible modelar el comportamiento de un agente autor que envíe su artículo a otra conferencia tan pronto como haya recibido una notificación desfavorable. Esto se consigue al recuperar de una base de datos de conferencias, la nueva dirección de correo electrónico que el agente usará para enviar el artículo. Al conectarse al *WF* de otro sistema de conferencia, la estructura entera de la colaboración podría cambiar significativamente. Entre los problemas que se vislumbran se encuentra el manejo de los eventos (operaciones) propios del nuevo *WF*, los cuales no necesariamente coinciden con los del anterior.

En el siguiente capítulo se presenta en detalle el diseño e implementación de la arquitectura del prototipo de *SAWF-HW*, con base en los conceptos que se han descrito hasta el momento en este documento.

Capítulo 6 Diseño e implementación del sistema de administración de Workflow

Este capítulo describe en detalle la arquitectura del sistema de administración de *WF* basado en modelo de coordinación de *HW* (*SAWF-HW*), sus componentes, sus perspectivas, su funcionamiento y los protocolos de comunicación utilizados. Tomando como base la definición del lenguaje y la implementacion actual de la arquitectura de coordinacion de *HW*, en esta tesis se diseño e implemento la arquitectura del *SAWF-HW*, la cual es multicapas con el propósito de que los componentes pudieran residir en una o más computadoras, utilizando los protocolos de Internet como la forma de comunicación entre ellos. Además en el diseño del *SAWF-HW* se consideraron las cinco perspectivas más importantes de un modelo de *WF* propuesto por Jablonski [30]. También se muestra la interacción entre los usuarios y el *SAWF-HW* haciendo referencia a las interfaces Web que pueden ser utilizadas desde un navegador.

6.1 Arquitectura del sistema

La arquitectura del *SAWF-HW* es multicapas y se compone de las capas: *Aplicaciones Workflow*, *Workflow Engine* y *Comunicaciones.*

La Figura 6-1 muestra la arquitectura del sistema, que es común en de los *SAWFs* (ConTract[16], Domino [17], Melmac [18], OfficeTalk [19], Pegasus [20], Action Workflow [21], COSA [22], FlowMark [24], InConcert [25], ProMInanD [26], SAP Business Workflow [27], WorkParty [28]). La comunicación entre las capas podría realizarse mediante un formato de mensajes que se definiera únicamente para este efecto, pero esto tiene la desventaja de que haría rígida la comunicación entre las capas, no

permitiendo la sustitución de alguna de las capas por alguna otra implementación de la misma, debido a que el formato de los mensajes no sería estandarizado ni de uso común.

Por lo anterior, se seleccionó el protocolo HTTP(*Hypertext Transfer Protocol*) para realizar la comunicación por ser un protocolo de comunicación estándar en Internet y de funcionamiento simple, que facilita la comunicación entre las capas ya que no es relevante la plataforma sobre la cual se implementen las capas.

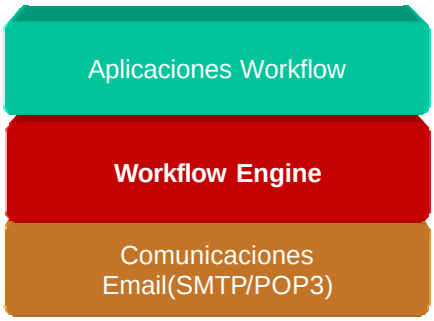


Figura 6-1 Arquitectura del SAWF-HW

6.2 Componentes

Las capas de *Aplicaciones Workflow* y *Comunicaciones* se implementaron usando el modelo de programación orientada a objetos, específicamente se uso la tecnología Java Servlets. La Capa de *Workflow Engine* se implementó usando un modelo de coordinación del lenguaje de *HW* [32]. La Figura 6-2 muestra los diagramas de clases que se utilizaron. En las secciones siguientes se describe en detalle cada capa y su funcionamiento así como la interacción entre sus componentes, usando la metodología UML (*Unified Modeling Language*) [43] específicamente diagramas de secuencia y tarjetas CRC (*Clase-Responsabilidad-Colaboración*) [44], para la descripción del *WF* en sus diferentes perspectivas.

6.3 Capa Aplicaciones Workflow

En la capa *Aplicaciones Workflow* se tienen operaciones de alto nivel que representan una o más operaciones en las capas inferiores. La capa *Aplicaciones Workflow* consistirá de un conjunto de páginas Web dinámicas, con las cuales los usuarios pueden revisar sus tareas o

actividades asignadas, alimentar al sistema con la información que se les solicite y verificar el avance del *WF*.

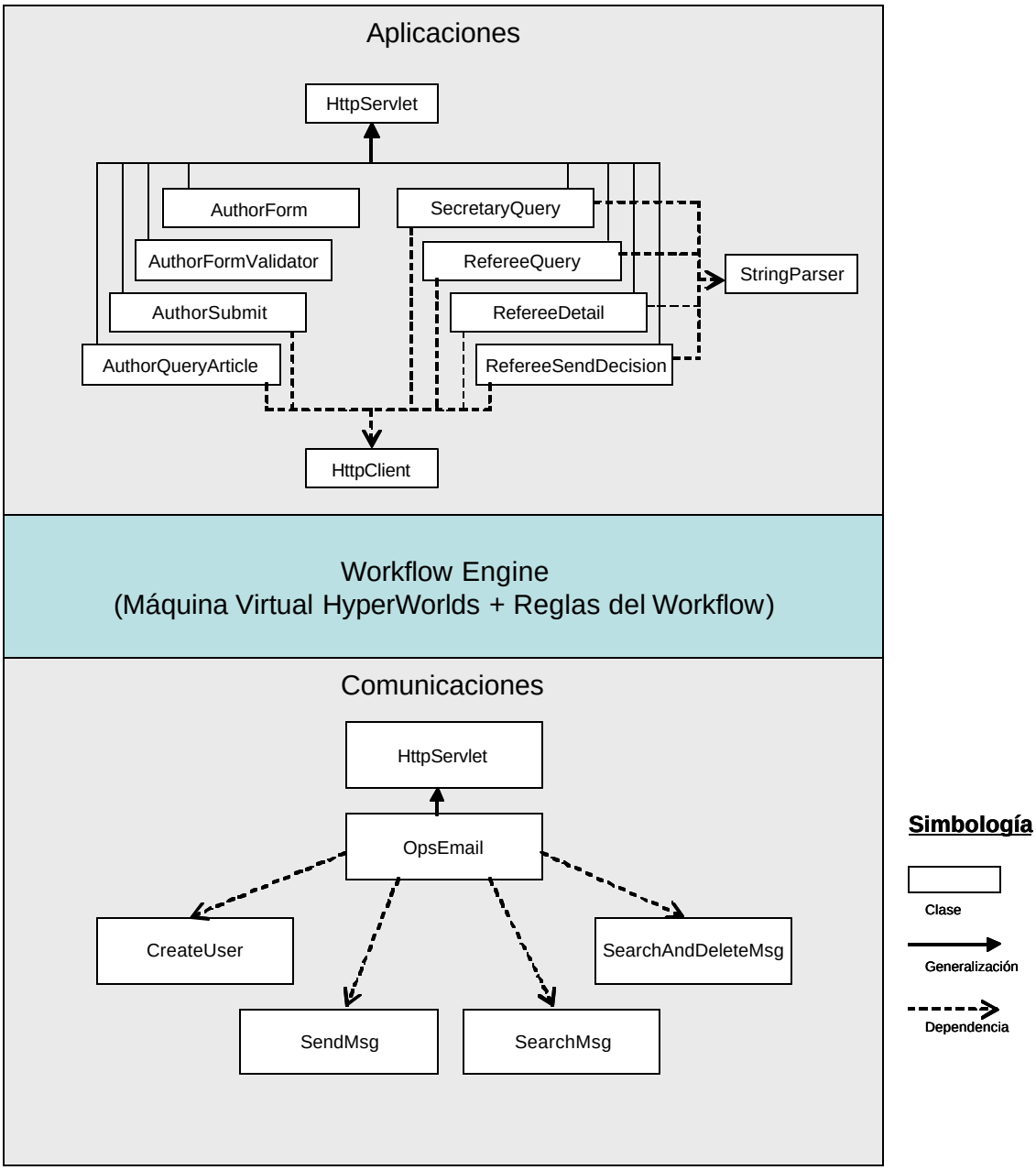


Figura 6-2 Diagrama de clases

Desde la perspectiva organizacional, para el caso de estudio pesentado en el Capítulo 5, el sistema tiene los usuarios que juegan los siguientes roles: *Autor*, *Secretaria* y *Revisor*. Para

obtener información acerca de los servidores Tomcat, James, Java Servlets, JavaMail, y los protocolos SMTP, HTTP y POP consulte el Apéndice A.

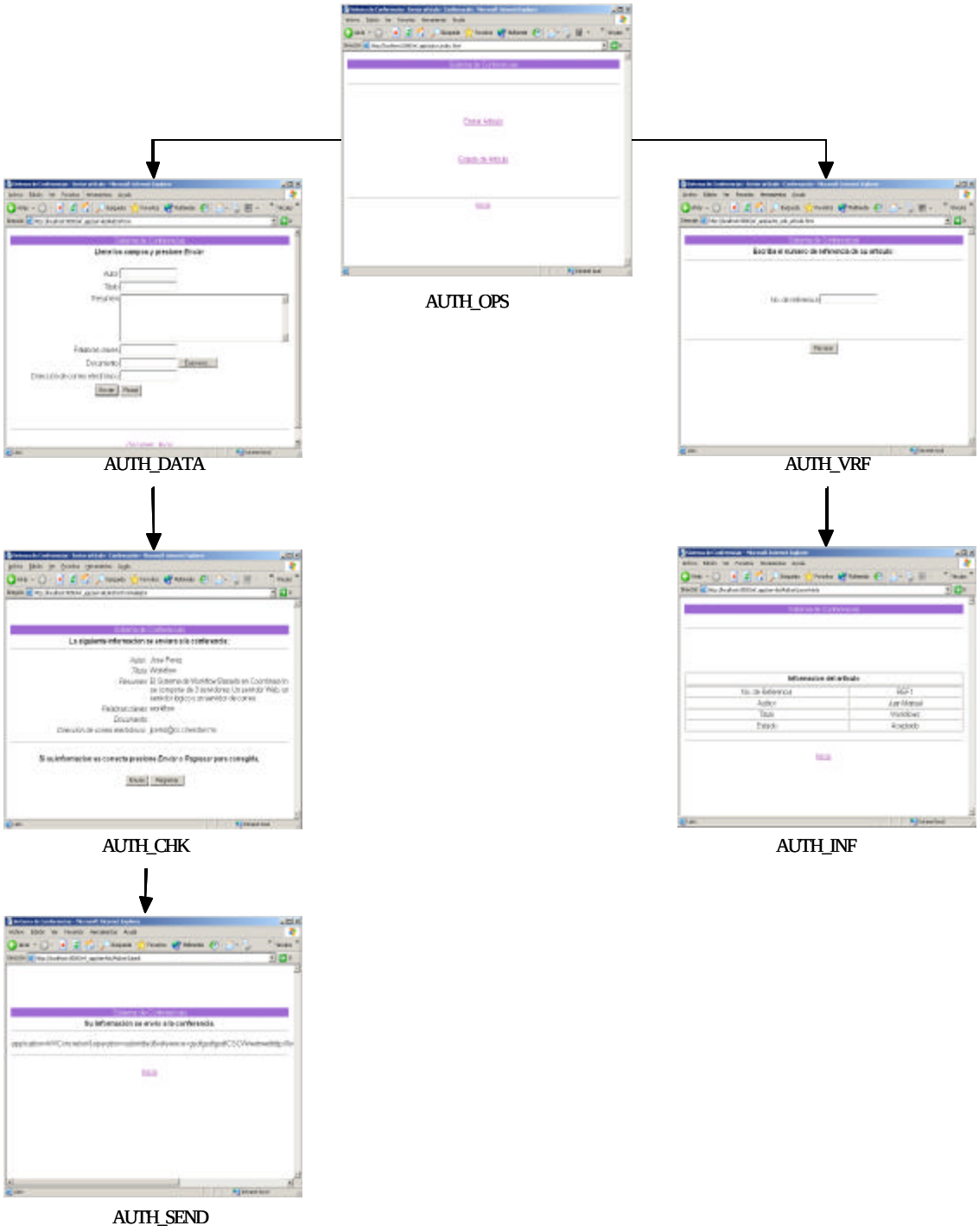


Figura 6-3 Páginas Web del usuario con rol *Autor*

6.3.1 Autor

Desde la perspectiva funcional, el usuario con el rol *Autor*, puede realizar dos acciones: Enviar artículo y Revisar el estado del artículo. La Figura 6-3 muestra la páginas Web que el usuario con el rol *Autor* usa para interactuar con el sistema para llevar a cabo su trabajo. La primera página mostrada al usuario con el rol *Autor* es AUTH_OPS (Figura 6-4) de la cual puede escoger entre las opciones “Enviar artículo” y “Estado del artículo”.

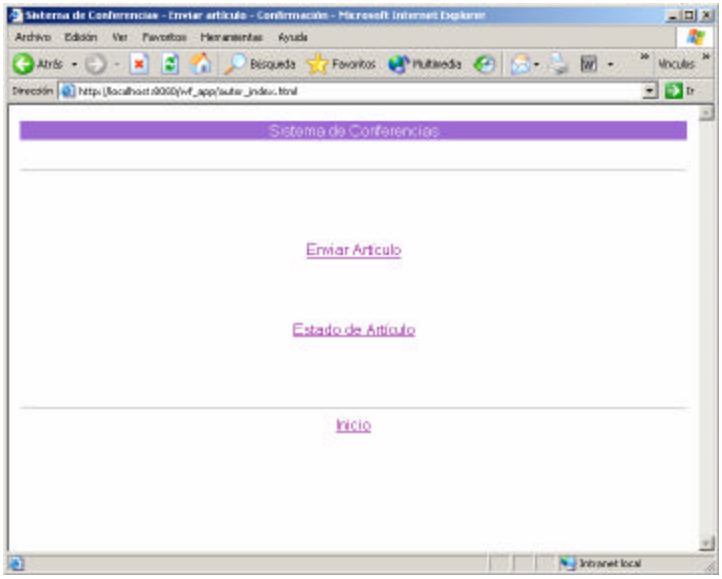


Figura 6-4 Página Web AUTH_OPS

La perspectiva de comportamiento y de información para la operación “Enviar artículo” se muestran en la Figura 6-5, en donde el flujo de control y de datos entre los componentes así como el usuario con el rol *Autor*, interactúan realizando las acciones siguientes:

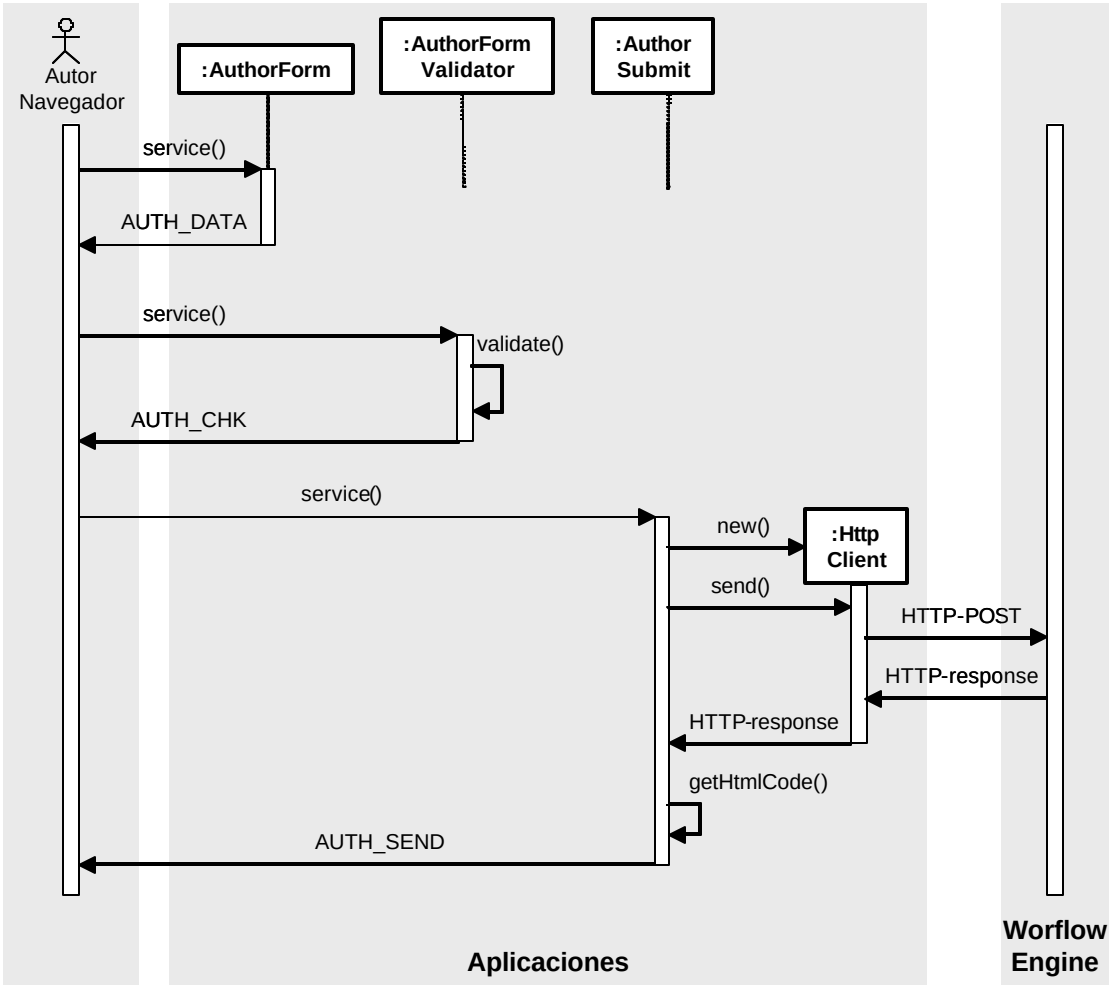


Figura 6-5 Acciones de la opción "Enviar artículo".

1. Llenar la forma de captura mostrada en la página Web AUTH_DATA (Figura 6-6) que genera la clase `AuthorForm`,
2. La información capturada llega a la clase `AuthorFormValidator` la cual valida los datos capturados y almacena el contenido del documento, el cual será referenciado por un enlace de hipertexto que apunta al lugar en donde físicamente se almacenó. Si la información es válida retorna la página AUTH_CHK (Figura 6-7) para que el usuario col el rol *Autor* confirme su información y
3. El usuario con el rol *Autor* envía su información a la conferencia, ésta acción la hace la clase `AuthorSubmit`. `AuthorSubmit` hace uso de la clase `HttpClient` cuya función es hacer una conexión a la capa *Workflow Engine* y

enviar la información del artículo del *Autor*, la comunicación se hace siguiendo el protocolo HTTP. `HttpClient` responde con el resultado de la comunicación y `AuthorSubmit` muestra al usuario con el rol *Autor* la página AUTH_SEND (Figura 6-8) indicando el éxito o el error de la operación.

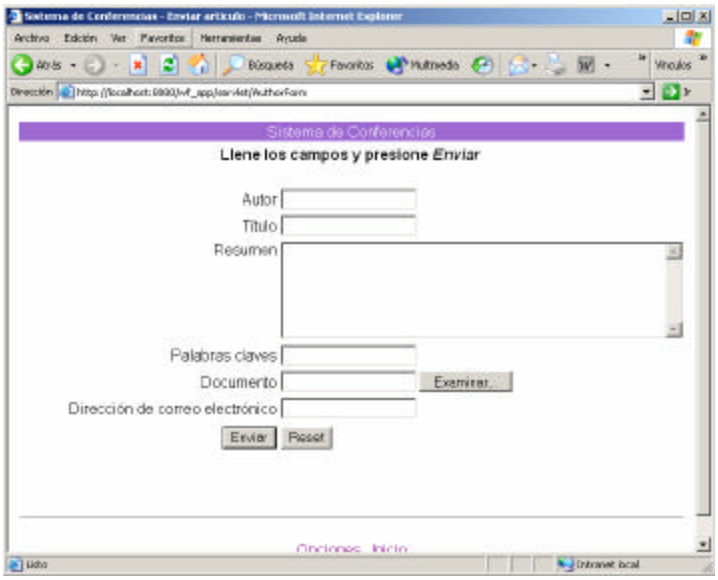


Figura 6-6 Página Web AUTH_DATA

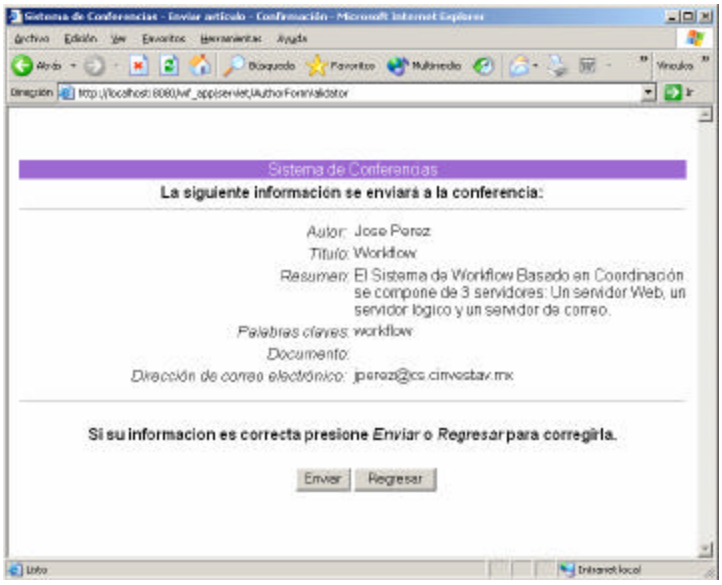


Figura 6-7 Página Web AUTH_CHK

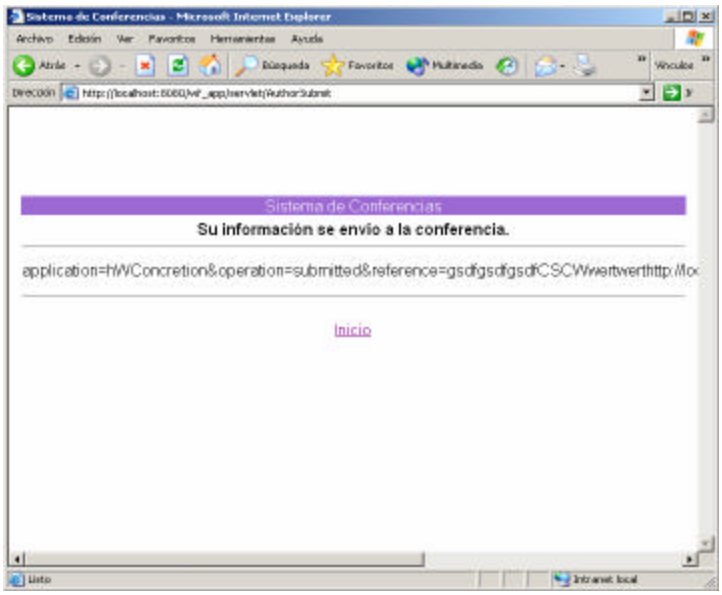


Figura 6-8 Página Web AUTH_SEND

La Figura 6-9 muestra la perspectiva de comportamiento y de información para la operación “Estado del artículo”.

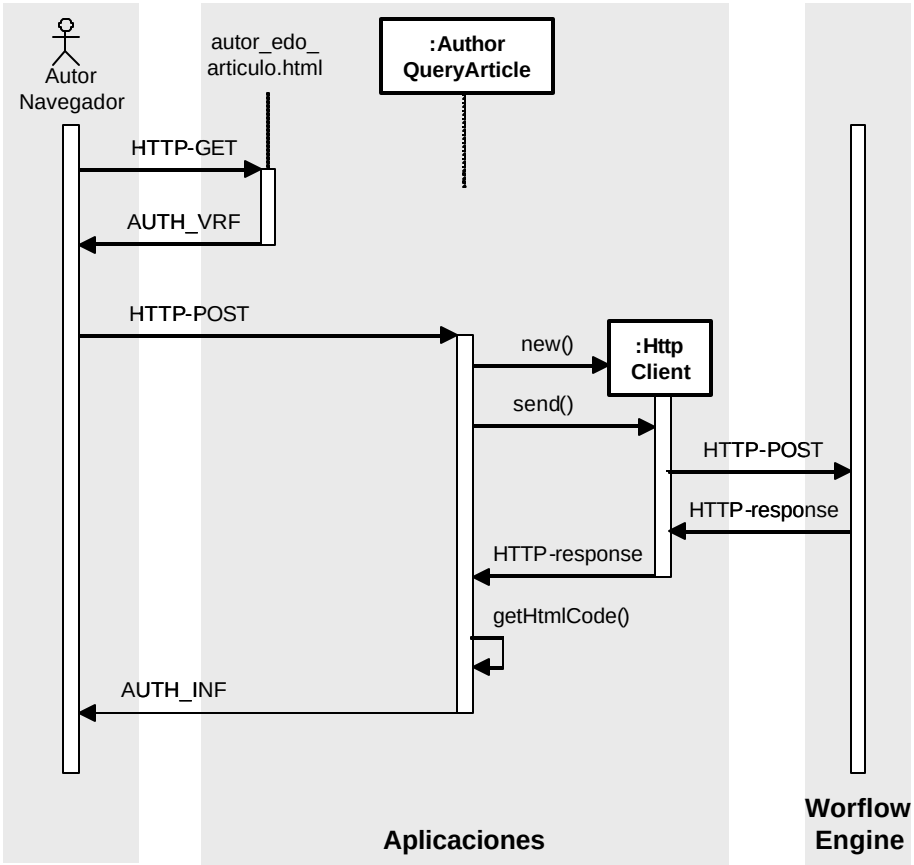


Figura 6-9 Acciones de la opción "Estado artículo"

Las acciones que realizan el usuario con el rol *Autor* y los componentes son las siguientes:

1. El usuario con el rol *Autor* llena el campo de texto mostrado en la forma de captura AUTH_VRF (Figura 6-10) y envía la solicitud al sistema y
2. La solicitud la recibe la clase `AuthorQueryArticle` que hace uso de `HttpClient` cuya función es hacer una conexión a la capa *Workflow Engine* y solicitar la información del artículo del *Autor*, la comunicación se hace siguiendo el protocolo HTTP. `HttpClient` responde con el resultado de la comunicación y `AuthorQueryArticle` genera la página AUTH_INF (Figura 6-11) con la información del artículo.

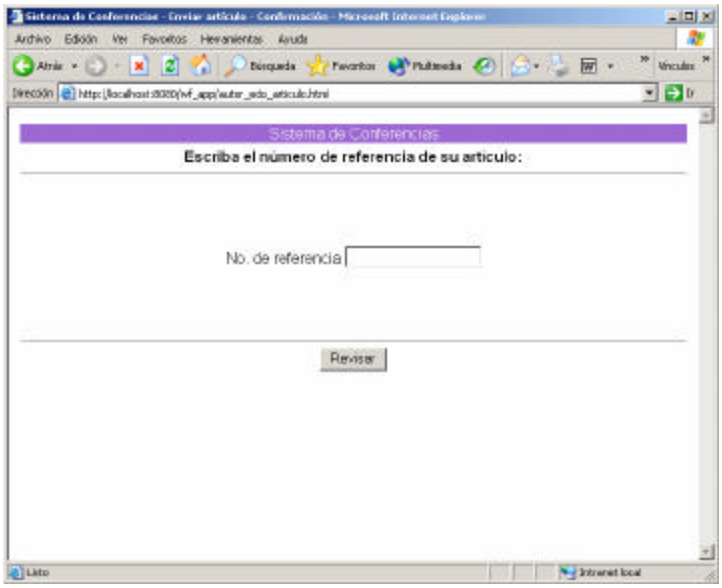


Figura 6-10 Página Web AUTH_VRF

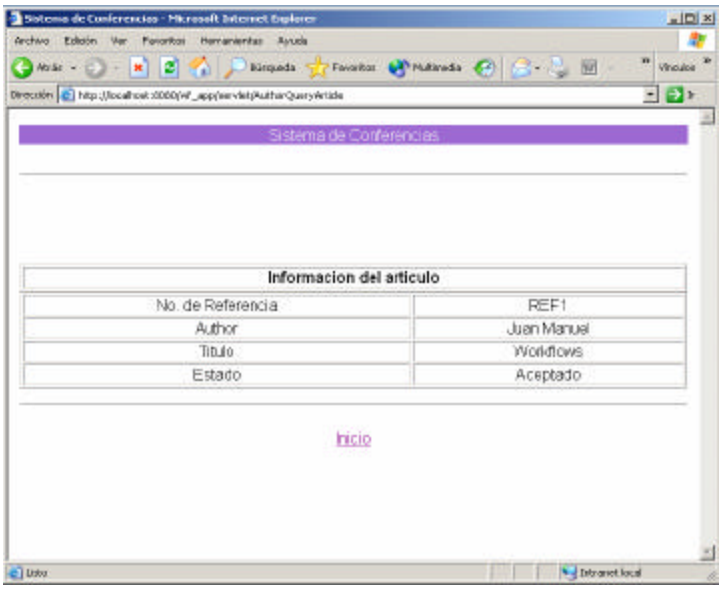


Figura 6-11 Página Web AUTH_INF

6.3.2 Secretaria

El usuario con el rol *Secretaria*, desde la perspectiva funcional, realiza la operación "Consultar artículos" que han entrado a la base de datos de la conferencia. La página Web que se presenta al usuario con el rol *Secretaria* se muestra en la Figura 6-12.

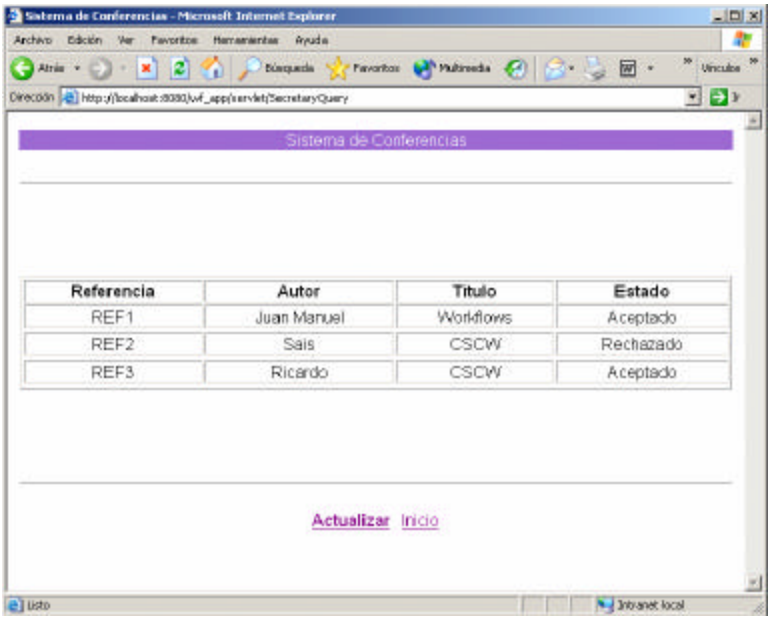


Figura 6-12 Página Web del usuario Secretaria

Las perspectivas de comportamiento y de información que se muestran en la Figura 6-13, presentan la interacción entre el usuario con el rol *Secretaria* y los componentes del sistema mediante las acciones siguientes:

1. El usuario con el *Secretaria* solicita al sistema la información de los artículos, que se atiende por la clase `SecretaryQuery`.
2. La clase `SecretaryQuery` usa la clase `HttpClient` que hace una conexión a la capa *Workflow Engine* y solicita la información de los artículos que han llegado a la conferencia, la comunicación se hace siguiendo el protocolo HTTP. `HttpClient` responde con el resultado de la consulta.
3. La respuesta llega a la clase `SecretaryQuery` en formato HTTP y para convertirla a una tabla HTML hace uso de la clase `StringParser`. Por último `SecretaryQuery` responde al usuario con la página SEC_QRY (Figura 6-12).

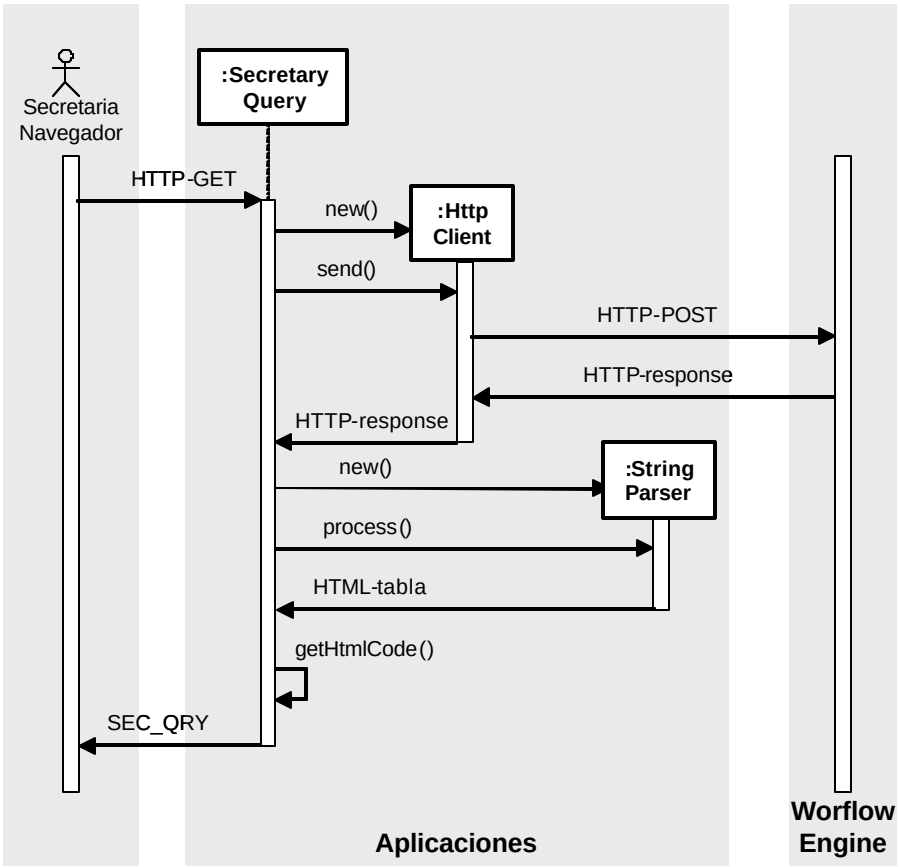
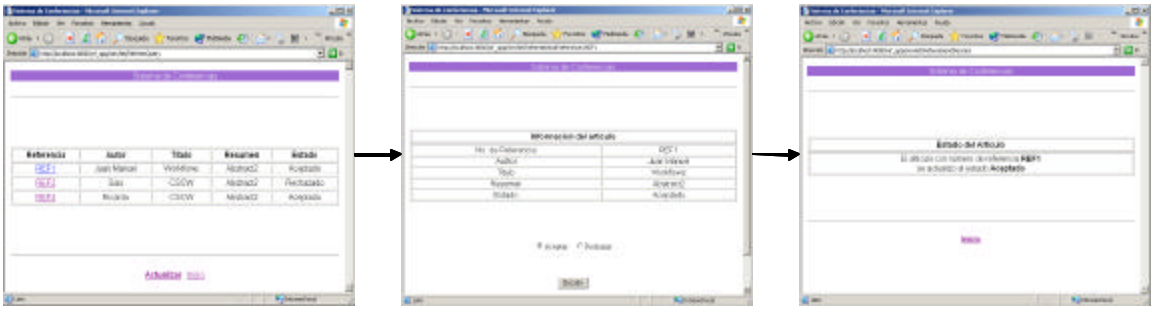


Figura 6-13 Acciones del sistema para el usuario Secretaria.

6.3.3 Revisor

Desde la perspectiva funcional, el usuario con el rol *Revisor* realiza las operaciones "Consultar artículos", "Revisar artículo" y "Decidir sobre un artículo". La secuencia de páginas que usa el usuario con el rol *Revisor* se muestra en la Figura 6-14.



REV_DATA REV_DETAIL REV_DES

Figura 6-14 Páginas Web del usuario Revisor

La Figura 6-15 muestra las perspectivas de comportamiento y de información para la operación "Consultar artículos".

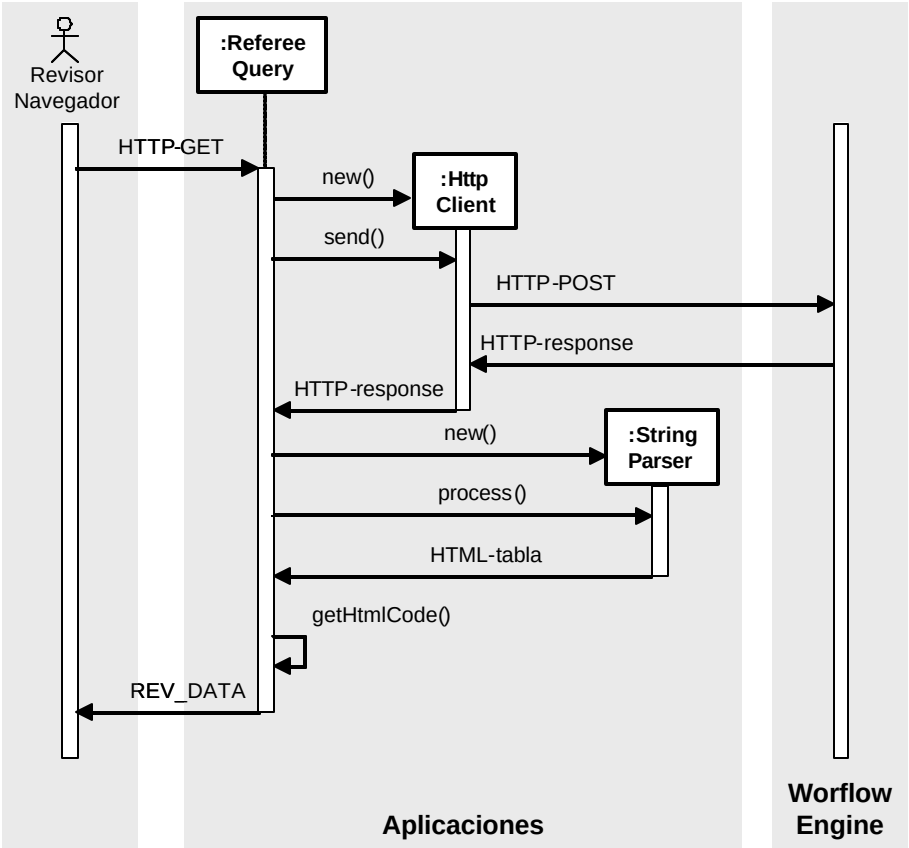


Figura 6-15 Acciones del sistema para la lista de artículos del usuario Revisor

Las acciones que se realizan son las siguientes:

1. El usuario envía la solicitud al sistema la cual se atiende por la clase `Refereequery`.
2. La clase `RefereeQuery` usa la clase `HttpClient` para comunicarse a la capa *Workflow Engine* a través del protocolo HTTP y solicitar la información de los artículos asignados al usuario con el rol *Revisor*. `HttpClient` responde el resultado de la consulta.
3. `RefereeQuery` usa la clase `StringParser` para convertir la respuesta HTTP que obtuvo en una tabla HTML y responde al usuario con el rol *Revisor* con la página REV_DATA (Figura 6-16).

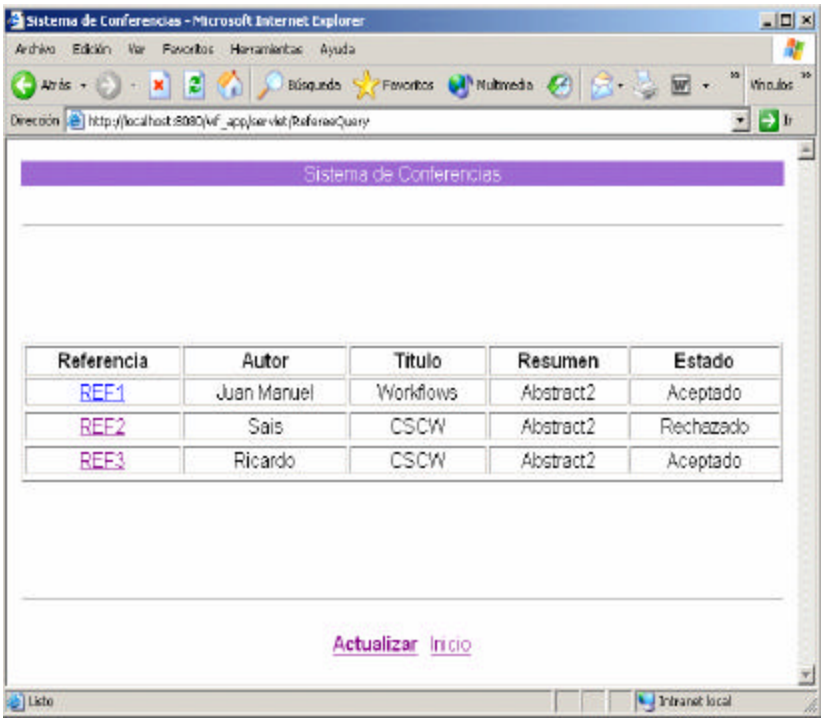


Figura 6-16 Página Web REV_DATA

Cuando el usuario con el rol *Revisor* tiene la lista de sus artículos asignados a él, tiene la opción de realizar la operación "Revisar artículo" para obtener más información de un artículo específico, para ello usa las ligas de hipertexto que se muestran en la página REV_DATA (Figura 6-16).

Las perspectivas de comportamiento y de información para la operación "Revisar artículo" se muestran en la Figura 6-17, describiendo la interacción entre el usuario con el rol *Revisor* y los componentes del sistema mediante las acciones siguientes:

1. El usuario con el rol *Revisor* solicita al sistema la información de un artículo específico, esta solicitud llega a la clase `RefereeDetail`.

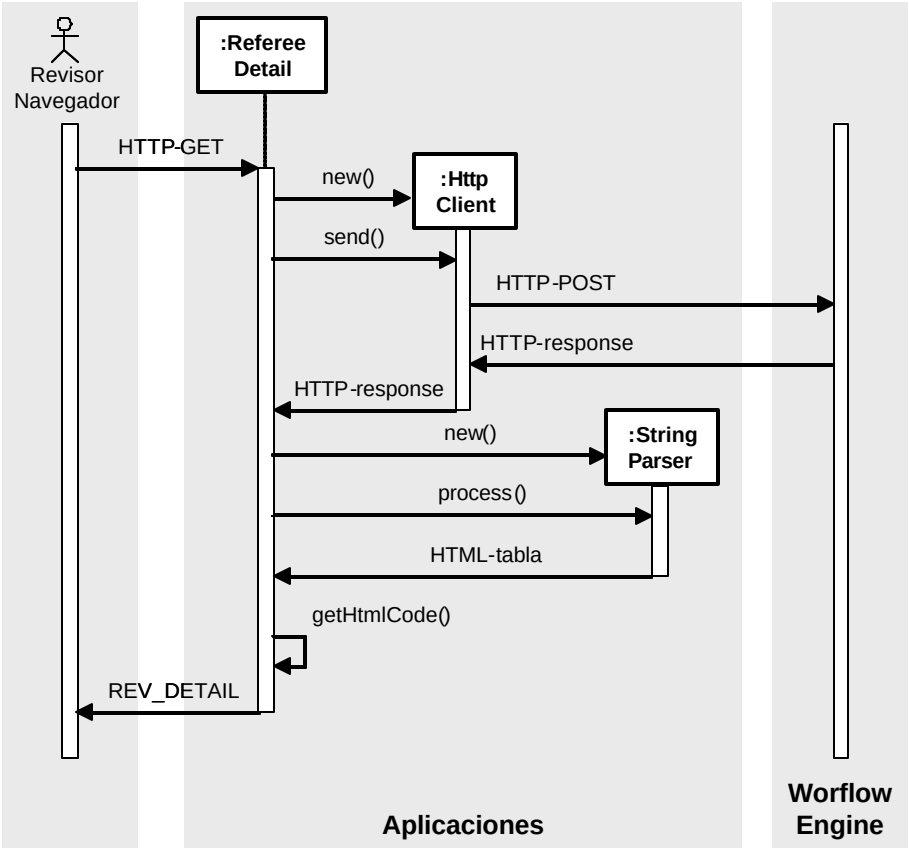


Figura 6-17 Acciones del sistema para la operación "Revisar artículo".

2. La clase `RefereeDetail` usa la clase `HttpClient` para comunicarse a la capa *Workflow Engine* a través del protocolo HTTP y solicitar la información del artículo

seleccionado por el usuario con el rol *Revisor*. `HttpClient` responde el resultado de la consulta.

3. `RefereeQuery` usa la clase `StringParser` para convertir la respuesta HTTP que obtuvo en una tabla HTML y responde al usuario con el rol *Revisor* con la página REV_DETAIL (Figura 6-18).

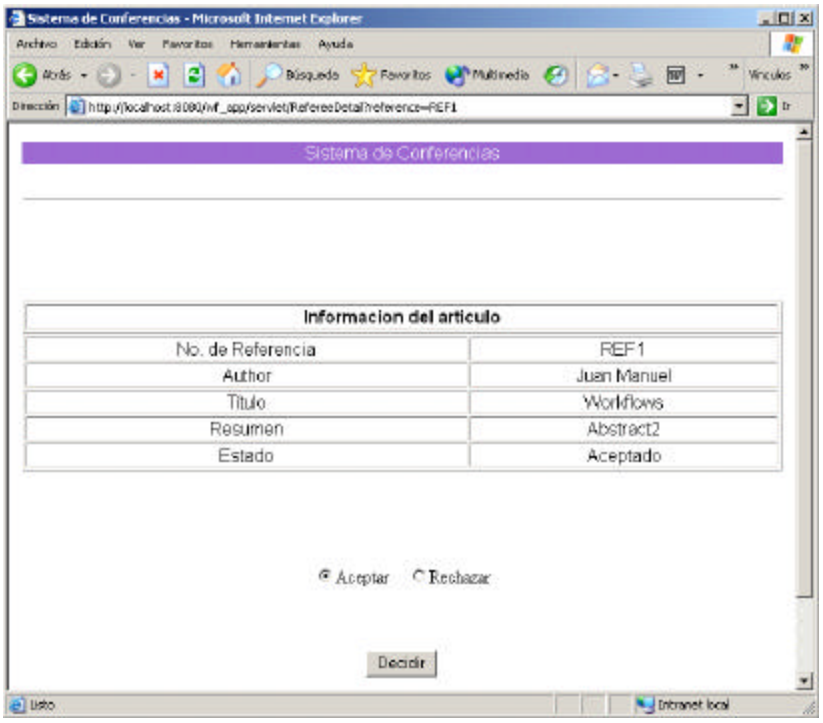


Figura 6-18 Página Web REV_DETAIL

Para realizar la operación “Decidir sobre un artículo”, el usuario con el rol *Revisor* usa la página REV_DETAIL (Figura 6-18) para emitir su decisión, la cual muestra la información del artículo y un enlace de hipertexto que le da el acceso al documento del artículo.

Para ésta operación, las perspectivas de comportamiento y de información del sistema se muestran en la Figura 6-19.

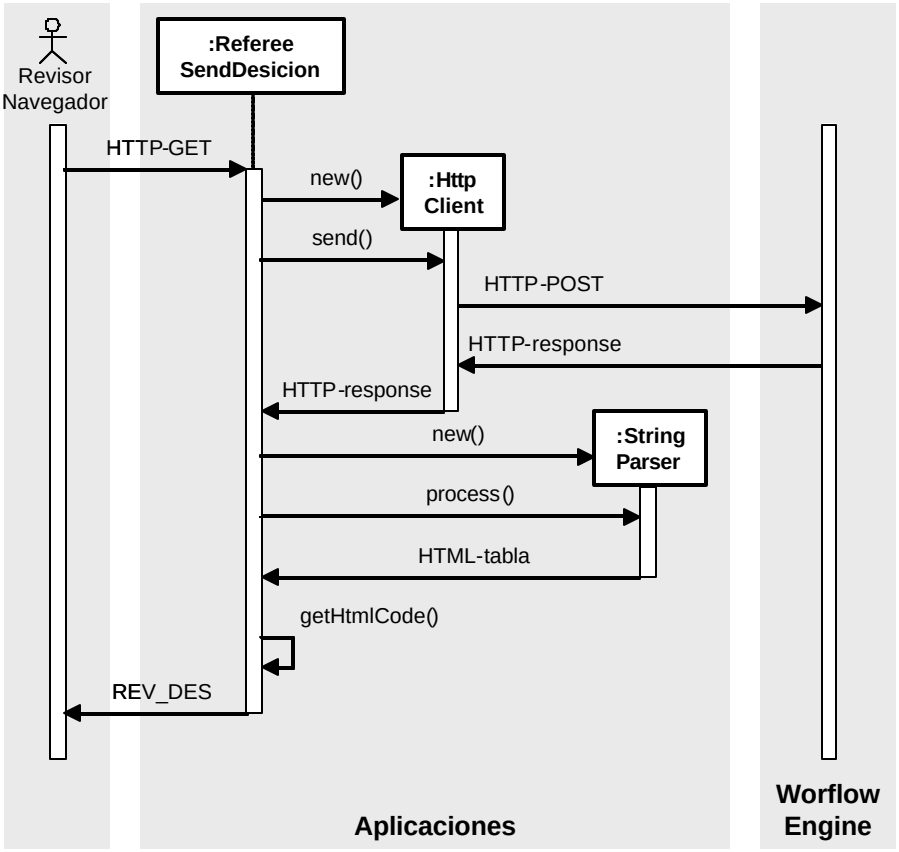


Figura 6-19 Envío de la decisión del usuario Revisor al Sistema

Las acciones que el sistema lleva a cabo para almacenar la información del usuario con el rol *Revisor* son las siguientes:

1. El usuario con el rol *Revisor* envía al sistema su aceptación o rechazo usando la página REV_DETAIL (Figura 6-18), ésta información llega a la clase `RefereeSendDesicion`.
2. La clase `RefereeSendDesicion` usa la clase `HttpClient` para comunicarse a la capa *Workflow Engine* a través del protocolo HTTP y enviar la decisión del usuario con el rol *Revisor*. `HttpClient` responde con el resultado de la acción.

`RefereeSendDesicion` usa la clase `StringParser` para convertir la respuesta HTTP que obtuvo en una tabla HTML y responde al usuario con el rol *Revisor* con la página REV_DES (Figura 6-14), que indica al usuario con el rol *Revisor* si su decisión fue almacenada con éxito o no.

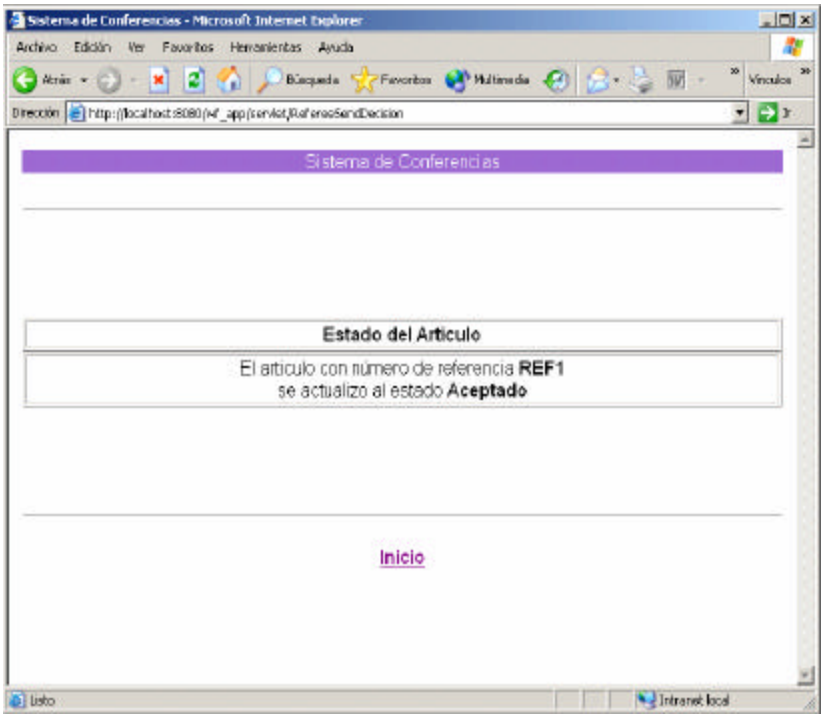


Figura 6-20 Página Web REV_DES

6.3.4 Descripción de las clases

A continuación se presentan la perspectiva de operación de la capa de *Aplicaciones Workflow*. Las clases que contiene esta capa son: `AutorForm`, `AuthorFormValidator`, `AuthorSubmit`, `AuthorQueryArticle`, `SecretaryQuery`, `RefereeQuery`, `RefereeDetail`, `RefereeSendDesicion`, `StringParser` y `HttpClient`. La descripción de cada una de las clases usadas en la capa *Aplicaciones Workflow* se hace con tarjetas CRC en donde se describen las funciones que realizan las clases y sus relaciones con otras.

6.3.4.1 Clase AuthorForm

Muestra la forma de captura al usuario con el rol *Autor*, la cual se llena con la información del artículo que se quiere someter a la conferencia. La forma de captura indica al usuario la información que debe ingresarse y advierte la ausencia de la misma.

AuthorForm	
Responsabilidades	***Colaboradores***
• Mostrar al usuario la forma de captura de la información del artículo en una página HTML • Indicar al usuario la información a capturar • Advertir al usuario sobre la ausencia de información o errores en la captura	• AuthorFormValidator

6.3.4.2 Clase AuthorFormValidator

Recibe la información de la forma de captura del artículo del usuario con el rol *Autor* y valida que sea correcta. La lógica de la clase permitirá mostrar la información del artículo en caso de que sea correcta, en caso contrario comunicará el error a la clase `AuthorForm`.

AuthorFormValidator	
Responsabilidades	***Colaboradores***
• Recibir la información de la forma de captura del artículo del autor • Almacenar el documento del artículo • Validar la información del artículo • Si la validación es correcta, mostrar la información del artículo al usuario para su confirmación en una página HTML • Si la validación es incorrecta, comunicar los errores a la clase AuthorForm	• AuthorForm • AuthorSubmit

6.3.4.3 Clase AuthorSubmit

Recibe la información del artículo que fue validada por la clase `AuthorFormValidator` y usa la clase `HttpClient` para comunicarse a la capa *Workflow Engine*, enviarle la información del artículo y recibir la información que resulte del procesamiento realizado en la capa *Workflow Engine* y mostrarla al usuario.

AuthorSubmit	
Responsabilidades	***Colaboradores***
• Recibir la información del artículo de la clase AuthorFormValidator • Utilizar la clase HttpClient para comunicarse a la capa Workflow Engine • Enviar la información del artículo a la capa Workflow Engine • Recibir la información de respuesta de la capa Workflow Engine • Mostrar la información de respuesta al usuario en una página HTML	• HttpClient

6.3.4.4 Clase AuthorQueryArticle

Esta clase recibe el número de referencia de un artículo y solicita a la capa *Workflow Engine* la información acerca del estado del artículo que el usuario con el rol *Autor* envió a la conferencia, la solicitud se hace a través de la clase `HttpClient`. El resultado de la solicitud se presenta al usuario con el rol *Autor*.

AuthorQueryArticle	
Responsabilidades	***Colaboradores***
• Recibir el número de referencia del artículo • Utilizar la clase HttpClient para comunicarse a la capa Workflow Engine • Enviar el número de referencia del artículo solicitando la información de su estado artículo • Recibir la información de respuesta de la capa Workflow Engine • Mostrar la información de respuesta al usuario en una página HTML	• HttpClient

6.3.4.5 Clase SecretaryQuery

Esta clase solicita a la capa *Workflow Engine* la información de todos los artículos que han ingresado a la conferencia, esto se hace a través de la clase `HttpClient`. El resultado de la solicitud se procesa usando la clase `StringParser` para después mostrarse al usuario con el rol *Secretaria*. Se muestra un enlace de hipertexto que permite al usuario solicitar nuevamente a la capa *WorkFlow Engine* la información de todos los artículos, con el propósito de mantener actualizada la información en el momento que se desee.

SecretaryQuery	
Responsabilidades	***Colaboradores***
• Usar la clase HttpClient para comunicarse a la capa Workflow Engine • Solicitar la información de los artículos que se recibieron en la conferencia • Recibir la información de respuesta a la solicitud • Procesar la respuesta mediante la Clase StringParser. • Mostrar la información procesada al usuario en una página HTML	• HttpClient • StringParser

6.3.4.6 Clase RefereeQuery

Esta clase solicita a la capa *Workflow Engine* la información de los artículos asignados a revisión a un revisor específico, mediante la clase `HttpClient`. El resultado de la solicitud se procesa usando la clase `StringParser` para después mostrarse al usuario con el rol *Revisor*. Se muestra una liga de hipertexto que permite al usuario solicitar nuevamente a la capa *WorkFlow Engine* la información de los artículos, con el propósito de mantener actualizada la información en el momento que se desee.

RefereeQuery	
Responsabilidades	***Colaboradores***
• Usar la clase HttpClient para comunicarse a la capa Workflow Engine • Solicitar la información de los artículos que se asignaron a un revisor • Recibir la información de respuesta a la solicitud • Procesar la respuesta mediante la Clase StringParser. • Mostrar la información procesada al usuario en una página HTML	• HttpClient • StringParser

6.3.4.7 Clase RefereeDetail

Esta clase solicita a la capa *Workflow Engine* la información de uno de los artículos asignados a revisión a un revisor específico, la solicitud del artículo se hace a traves de su número de referencia y la comunicación se realiza mediante la clase `HttpClient`. El resultado de la solicitud se procesa usando la clase `StringParser` para después mostrarse al usuario con el rol *Revisor*.

RefereeDetail	
Responsabilidades	***Colaboradores***
• Recibir el número de referencia de un artículo • Usar la clase HttpClient para comunicarse a la capa Workflow Engine • Solicitar la información del artículo a revisar • Recibir la información de respuesta a la solicitud • Procesar la respuesta mediante la Clase StringParser. • Mostrar la información procesada al usuario en una página HTML	• HttpClient • StringParser

6.3.4.8 RefereeSendDesicion

Esta clase recibe la decisión sobre un artículo que hizo el usuario con rol Revisor y la envía a la capa *Workflow Engine* mediante la clase `HttpClient`. La capa *Workflow Engine* responde indicando el cambio de estado del artículo según la decisión, el resultado se procesa usando la clase `StringParser` para después mostrarse al usuario con el rol *Revisor*.

RefereeSendDesicion	
Responsabilidades	***Colaboradores***
• Recibir la decisión del revisor sobre un articulo • Usar la clase HttpClient para comunicarse a la capa Workflow Engine • Enviar la decisión • Recibir la información de respuesta • Procesar la respuesta mediante la Clase StringParser. • Mostrar la información procesada al usuario en una página HTML	• HttpClient • StringParser

6.3.4.9 Clase StringParser

Esta clase recibe una cadena URL codificada, la decodifica y obtiene la información de los campos. Con la información de los campos se forma el código HTML que representa una tabla y la regresa a la clase que la solicito.

StringParser	
Responsabilidades	***Colaboradores***
• Recibir una cadena URL Codificada • Decodificar la cadena URL • Obtener la información de los campos de la cadena • Formar el código HTML de una tabla con la información de los campos • Regresar el código HTML generado	• SecretaryQuery • RefereeQuery

6.3.4.10 Clases HttpClient

Esta clase realiza la comunicación con la capa *Workflow Engine*. Recibe como parámetros la información del servidor y el puerto donde se encuentra la capa *Workflow Engine* así como la cadena URL codificada que se enviará como parte de la solicitud. La clase realiza la conexión al servidor indicado, se prepara la solicitud siguiendo el formato del protocolo HTTP y envía la misma. La clase espera la respuesta de la solicitud enviada regresando como resultado.

HttpClient	
Responsabilidades	***Colaboradores***
• Recibir información del servidor y puerto donde se encuentra la capa Workflow Engine • Recibir la cadena URL codificada • Realizar la conexión al servidor indicado • Preparar la solicitud en el formato del protocolo HTTP • Enviar la solicitud • Recibir la respuesta a la solicitud realizada • Regresar la respuesta como resultado a la clase que la solicito	• AuthorSubmit • SecretaryQuery • RefereeQuery

6.4 Capa Workflow Engine

La capa de *Workflow Engine* es un servidor lógico que recibe las operaciones *Workflow* de la capa *Aplicaciones Workflow* y las convierte en operaciones Assert/Retract, que se ejecutan sobre los espacios de términos, generando o retirando los términos. La capa *Workflow Engine* mantiene comunicación continua con la capa siguiente, la capa *Comunicaciones*, para hacer coincidir las operaciones y los contenidos de los espacios de términos con los de los almacenamientos de mensajes de correo electrónico.

La traducción de las operaciones de la capa *Workflow Engine* a aquellas de correo electrónico se describe a continuación. La primera que se describe es la operación *Assert* como se muestra en la Figura 6-21; cuando ésta se realiza en la capa *Workflow Engine* se realizan las siguientes acciones:

- Extraer la información del término *t*,
- Determinar el documento *d* asociado a *t*,
- Determinar la dirección *M* de correo electrónico asociado a *S* y
- Solicitar a la capa de comunicaciones la realización de la operación con la información obtenida.

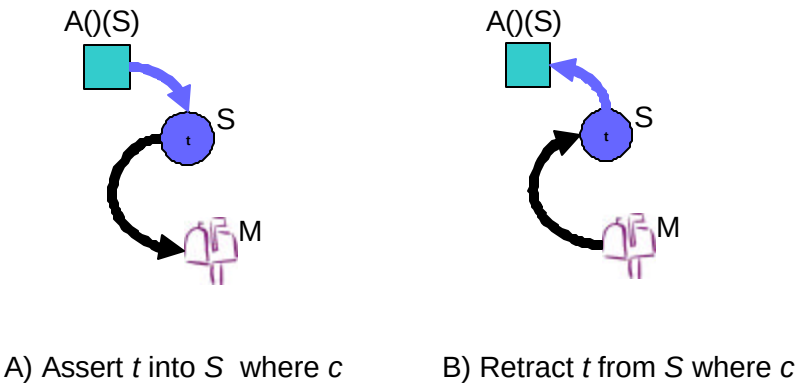


Figura 6-21 Operaciones de la capa Workflow Engine

Cuando la operación *Retract* mostrada en la Figura 6-21, se ejecuta entonces se realizan las siguientes acciones:

- Seleccionar el término t del espacio S que se desea extraer,
- Determinar la dirección *M* de correo electrónico asociado a *S*,
- Solicitar a la capa de comunicaciones la realización de la operación con la información obtenida y
- Retirar el término t del espacio S.

6.5 Capa Comunicaciones

La función de la capa *Comunicaciones* consiste en recibir las solicitudes de la capa inmediata superior, *Workflow Engine*, para hacer corresponder los contenidos de los espacios de términos con los de las cuentas de correo que los representan. Desde la perspectiva funcional, La capa *Comunicaciones* puede realizar las funciones: *create*, *assert*, *select* y *retract*.

La capa *Comunicaciones* se compone de 2 elementos principales: un servidor Web y un servidor de correo electrónico como se presenta en la Figura 6-22. Como servidor Web se utilizó *Tomcat 4.0*, este servidor soporta el uso de *Java Servlets*. Como servidor de correo se utilizó *James 2*. Para acceder y manipular el contenido de los almacenamientos de mensajes de correo electrónico use *JavaMail 1.2*. Para obtener información acerca de los servidores *Tomcat*, *James*, *Java Servlets*, *JavaMail* y los protocolos HTTP, SMTP y POP3 consulte el Apéndice A.

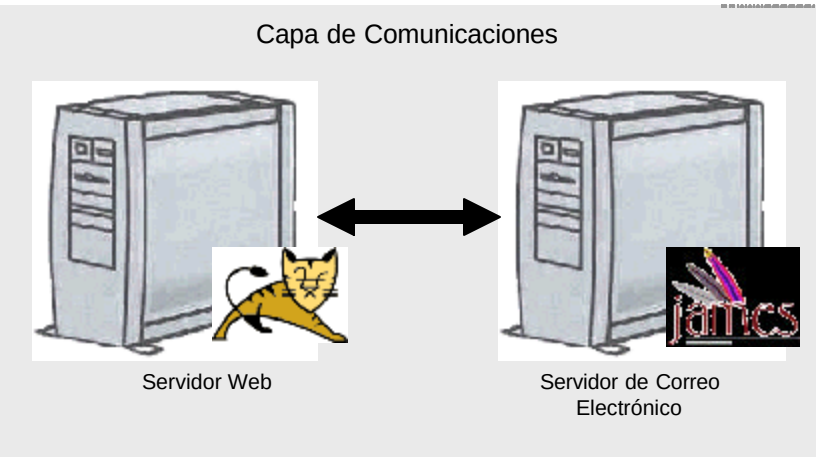


Figura 6-22 Elementos mayores de la capa de comunicaciones.

En el servidor Web se programo e instaló la clase Java Servlet `OpsEmail` que se encarga de recibir las peticiones de la capa de *Workflow Engine*; la solicitud se hace usando el protocolo HTTP, la información de la solicitud se envía utilizando el método `POST` del protocolo. La clase `OpsEmail` determina cual es la operación a realizar las cuales pueden

ser *create, assert*, *select* y *retract*; para llevarlas a cabo, usa las clases `CreateUser`, `SendMsg`, `SearchAndDeleteMsg` y `SearchMsg` respectivamente que acceden y manipulan el almacenamiento de mensajes en el servidor de correo electrónico, como se muestra en la Figura 6-23.

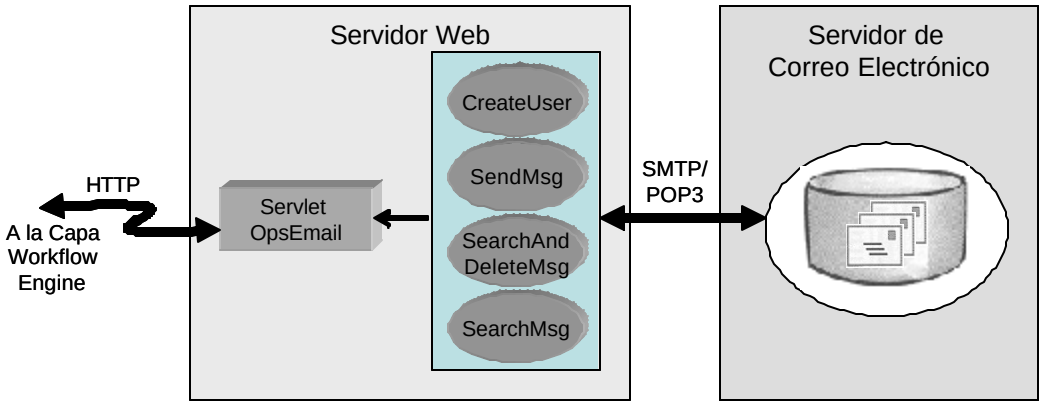


Figura 6-23 Funcionamiento de la capa de comunicaciones

En las secciones siguientes se describen en detalle las operaciones que lleva a cabo la clase `OpsEmail` y su interacción con las clases antes mencionadas, así como el funcionamiento de las mismas.

6.5.1 Create

El primera función es *create,* mostrada en la Figura 6-24, la cual presenta la perspectiva de comportamiento y de información. Esta función indica a la clase `OpsEmail` que cree una cuenta de correo, provocando la invocación del método `doCreate()`. El método `doCreate()` solicita el valor de los campos:

- *host* indica el nombre del servidor de correo donde se quiere crear la cuenta,
- *port* número del puerto de administración del servidor de correo,
- *user* nombre del usuario o de la cuenta que se quiere crear y
- *passwd* contraseña de la cuenta.

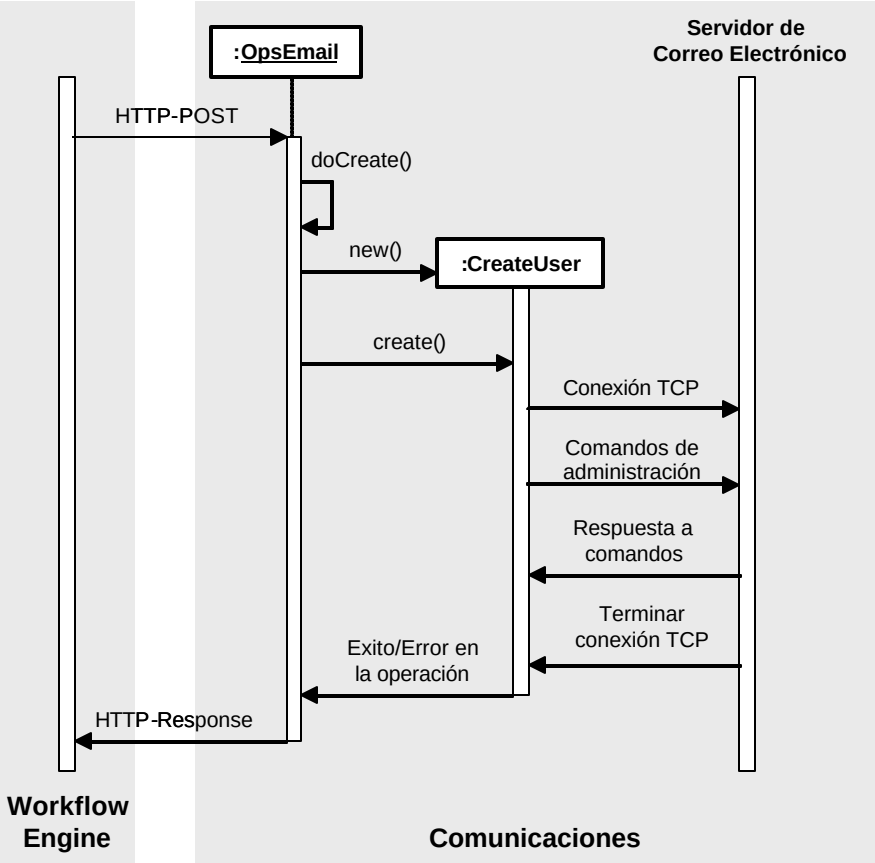


Figura 6-24 Interacción del sistema para la operación Create

Con la información de estos campos se utiliza la clase `CreateUser`; ésta clase usa un socket para realizar la comunicación con el servidor de correo, se autentifica y procede a ejecutar los comandos de administración necesarios para crear la cuenta de correo. Al terminar su ejecución regresa la información de éxito o error que el servidor de correo genero durante la creación de la cuenta.

6.5.2 Assert

La operación *assert* indica la creación de un mensaje de correo electrónico, ésta operación se lleva a cabo con la ejecución del método `doAssert()` de la clase `OpsEmail`. Las perspectivas de comportamiento y de información para ésta operación se muestra en la Figura 6-25, en la cual se presenta la interacción de los componentes, el flujo de control y de información para esta operación.

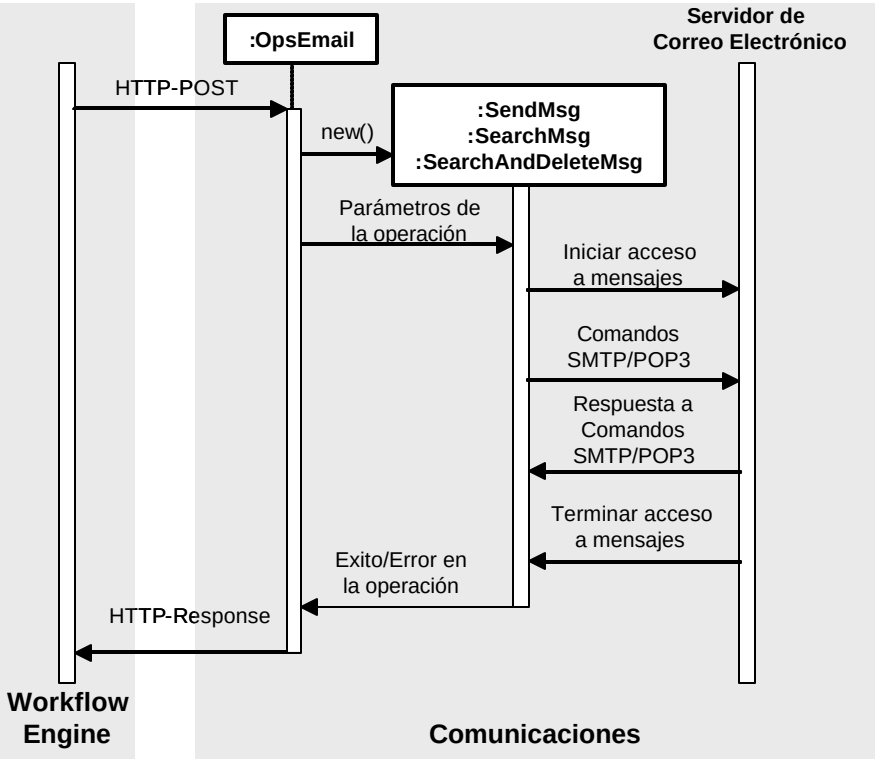


Figura 6-25 Interacción del sistema para las operaciones Assert, Select y Retract.

Esta operación no requiere del acceso al almacenamiento de mensajes, sólo requiere de un servidor de correo que tenga implementado el protocolo SMTP que permitirá el envió o la creación del mensaje de correo electrónico. Esta operación requiere de los parámetros:

- *host*. Nombre del servidor que tiene implementado el protocolo SMTP,
- *from*. Dirección de correo electrónico de quien envía o remitente,
- *to*. Dirección de correo electrónico de quien recibe o receptor,
- *subject*. Titulo o asunto del mensaje y
- *msg*. Contenido del mensaje de correo electrónico.

Con esta información se invoca a la clase `SendMsg`, que usa las clases e interfaces de *JavaMail* para enviar el correo electrónico. Al final de su ejecución, la clase `SendMsg` retorna los mensajes de éxito o error que se generaron durante el envío del mensaje de correo electrónico.

6.5.3 Select

La operación *Select* indica una búsqueda de un mensaje dentro de una cuenta de correo electrónico. Para esto se ejecuta el método `doSelect()` de la clase `OpsEmail`. Las perspectivas de comportamiento y de información se muestran en la Figura 6-25 de la sección 6.5.2.

Esta operación requiere del acceso al almacenamiento de mensajes del servidor de correo, por lo que se necesita información de la cuenta de correo sobre la cual se hará la búsqueda. Los parámetros que se necesitan son:

- *host* es el servidor de correo que contiene la cuenta de correo,
- *user* es el nombre de usuario o cuenta de correo sobre la cual se hará la búsqueda y
- *passwd* es la contraseña de la cuenta de correo.

Además es necesaria la información de los parámetros que restrinjan el resultado de la búsqueda; los parámetros pueden ser:

- *from* indica la búsqueda por una cuenta de correo o remitente específico,
- *subject* indica la búsqueda por un titulo o asunto específico y
- *msg* indica la búsqueda por el contenido del correo electrónico.

Con todos estos parámetros se usa la clase `SearchMsg` que usa las interfaces y clases de *JavaMail* para autentificarse con el servidor de correo y realiza la búsqueda de los mensajes. El acceso al servidor de correo es mediante el protocolo POP3. Al terminar su ejecución la clase `SearchMsg` regresa el resultado de la consulta o los errores que se produjeron.

6.5.4 Retract

La última operación es *retract*, que indica la búsqueda de un mensaje de correo electrónico al igual que la operación *select* y además realiza su eliminación. Para esto se ejecuta el

método `doRetract()` de la clase `OpsEmail`. Las perspectivas de comportamiento y de información para ésta operación se muestran en la Figura 6-25 de la sección 6.5.2.

Al igual que la operación *Select*, la operación *Retract* requiere también del acceso al almacenamiento de mensajes del servidor de correo, por lo que se necesita información de la cuenta de correo sobre la cual se hará la búsqueda y de restricción de la búsqueda. Los parámetros que se necesitan son *host*, *user*, *passwd*, *from*, *subject* y *msg* tal como se describen para la operación *Select* descrita en la sección 6.5.3.

Con todos estos parámetros se usa la clase `SearchAndDeleteMsg` que al igual que la operación *Select* usa *JavaMail* para autentificarse con el servidor de correo y realiza la búsqueda y eliminación de los mensajes y también usa el protocolo POP3 para acceder al servidor de correo. Al terminar su ejecución la clase `SearchAndDeleteMsg` regresa el resultado de la búsqueda y eliminación o los errores que se produjeron.

6.5.5 Descripción de las clases

En esta sección se presenta la perspectiva de operación de la capa *Comunicaciones* mediante la descripción de las clases usadas por medio de tarjetas CRC. Las clases que contiene esta capa son: `OpsEmail`, `CreateUser`, `SendMsg`, `SearchMsg` y `SearchAndDeleteMsg`.

6.5.5.1 Clase OpsEmail

Esta clase recibe las solicitudes de la capa *Workflow Engine*, decide la operación a realizar y la ejecuta, al final de la ejecución responde con la información del éxito o el error resultado de la operación.

OpsEmail	
Responsabilidades	***Colaboradores***
• Recibir las solicitudes de la capa Workflow Engine • Determinar la operación a realizar y ejecutarla • Responder a la solicitud con la información que se produjo por la ejecución de la operación.	• CreateUser • SendMsg • SearchMsg • SearchAndDeleteMsg

6.5.5.2 Clase CreateUser

Esta clase tiene la función de crear una cuenta de correo, para esto recibe como parámetros la información del servidor de correo donde se creara así como el nombre de usuario y contraseña de la nueva cuenta.

CreateUser	
Responsabilidades	***Colaboradores***
• Recibir los parámetros host y port donde se creara la cuenta. • Recibir los parámetros user y passwd para crear la nueva cuenta. • Crear la conexión o socket al servidor de correo. • Enviar los comandos de administración necesarios para crear la cuanta de correo. • Esperar la respuesta a los comandos de administración y devolverlos como respuesta a la clase que solicito la creación de la cuenta.	• OpsEmail

6.5.5.3 Clase SendMsg

Esta clase tiene la función de crear un mensaje de correo electrónico. Para la creación del mensaje son necesarios los parámetros *host*, *from*, *to*, *subject* y *msg* indicando el servidor que ofrece el protocolo SMTP necesario para enviar el mensaje, el remitente del mensaje, el destinatario, el tema del mensaje y el mensaje respectivamente. Al final de su ejecución la clase responde con el mensaje de éxito o error que se produjo durante la creación del mensaje.

SendMsg	
Responsabilidades	***Colaboradores***
• Recibir los parámetros host, from, to, subject y msg, necesarios para la creación del mensaje • Crear el objeto Java que corresponde al mensaje de correo electrónico. • Enviar el mensaje electrónico usando el protocolo SMTP que es proporcionado por el servidor host • Responder con el mensaje de éxito o error resultado del envío del mensaje.	• OpsEmail

6.5.5.4 Clase SearchMsg

Esta clase se encarga de hacer una búsqueda de uno o más mensajes dentro de una cuenta de correo. Para tener acceso a la cuenta de correo se necesitan los parámetros *host*, *user* y *passwd* que representan al servidor de correo, el nombre de la cuenta y su contraseña. También son necesarios los parámetros de búsqueda *from*, *to*, *subject*, *msg* permitiendo limitar la búsqueda por un remitente, destinatario, tema de mensaje o contenido de mensaje específico. La respuesta de esta clase es la lista de los mensajes encontrados.

SearchMsg	
Responsabilidades	***Colaboradores***
• Recibir los parámetros host, user y passwd para tener acceso a la cuenta de correo. • Recibir los parámetros de búsqueda from, to, subject y msg. • Acceder a la cuenta de correo en modo lectura. • Crear el objeto Java que representa los parámetros de búsqueda. • Ejecutar la búsqueda. • Terminar el acceso a la cuenta de correo. • Responder con los mensajes encontrados en la búsqueda.	• OpsEmail

6.5.5.5 Clase SearchAndDeleteMsg

Esta clase tiene la función de buscar y eliminar uno o más mensajes dentro de una cuenta de correo. Al igual que la clase `SearchMsg` la cual se describe en la sección 6.5.5.4, son necesarios los parámetros *host*, *user* y *passwd* para acceder a la cuenta de correo así como los parámetros de búsqueda *from*, *to*, *subject*, *msg*. La respuesta de ésta clase es la lista de los mensajes eliminados.

SearchAndDeleteMsg	
Responsabilidades	***Colaboradores***
• Recibir los parámetros host, user y passwd para tener acceso a la cuenta de correo. • Recibir los parámetros de búsqueda from, to, subject y msg. • Acceder a la cuenta de correo en modo lectura/escritura. • Crear el objeto Java que representa los parámetros de búsqueda. • Ejecutar la búsqueda y posteriormente la eliminación de los mensajes encontrados. • Terminar el acceso a la cuenta de correo. • Responder con los mensajes encontrados en la búsqueda.	• OpsEmail

6.6 Conclusiones

En este capítulo se describió la arquitectura del prototipo de *SAWF-HW* que se propone en ésta tesis. Considerando las perspectivas descritas en la sección 2.5, sólo se consideraron las cinco más importantes: *funcional, operacional, de comportamiento, de información* y *organizacional* con las cuales se cubren los aspectos fundamentales de un *SAWF*, como son sus operaciones de alto nivel, la manera en que éstas se llevan a cabo, la forma como interactúan los componentes del sistema para llevar a cabo el procesamiento del *WF*, como fluye la información en el sistema, así como las usuarios que participan en el *WF*. La descripción se hizo utilizando metodologías como UML y las tarjetas CRC, las cuales tienen elementos gráficos que permiten una descripción adecuada de los elementos del sistema.

Existen otras perspectivas que no están incluidas en este trabajo, debido a que los objetivos de la tesis se dirigen en el sentido de presentar un modelo de coordinación para la solución

de problemas en ambientes de colaboración, así como la conveniencia de su uso en un *SAWF*; pero no se pretende presentar un sistema completo que compita con los ya existentes en el mercado. Las perspectivas que no se incluyen son:

- *Perspectiva de causalidad*, no se quieren cubrir necesidades las necesidades especificas de una empresa u organización,
- *Perspectiva de integridad y recuperación de fallas*, el caso de estudio considerado para mostrar el uso del sistema propuesto es simple por lo que las posibilidades de que el sistema alcance estado de inconsistencia son mínimas. El sistema tampoco cuenta con mecanismos de recuperación de fallas,
- *Perspectiva de calidad*, el prototipo propuesto no tiene el objetivo de tener tiempos de ejecución óptimos,
- *Perspectiva histórica*, el sistema no cuenta con mecanismos que registren aspectos de ejecución del *WF*,
- *Perspectiva de seguridad*, el sistema considera un grupo u organización de personas y agentes reducido y con responsabilidades bien definidas por lo que no se consideraron aspectos referentes a permisos sobre la ejecución del *WF*,
- *Perspectiva de autonomía*, existen tareas automáticas que muestran el uso del modelo de coordinación del lenguaje de *HW*, pero éstas son básicas por lo que por lo que no se describen elementos independientes que sean responsables de la ejecución automática de las partes del *WF*.

Por las características del prototipo propuesto se le puede considerar dentro de la categoría de los *SAWF* de *producción* y *administrativos*. En cuanto a las herramientas de los *SAWFs* sólo se consideran las herramientas de tiempo de ejecución, en particular las *listas de trabajo*, las cuales se implementaron en páginas Web que se pueden ver en un navegador, así como un monitor de la ejecución de las reglas que funciona en la capa *Workflow Engine*.

Los tipos de documentos que pueden ingresar al sistema son prácticamente de cualquier tipo, incluyendo los más comunes como son los documentos *Microsoft Word*, *Postscript*, *PDF*, *HTML*, *XML* y documentos de texto. En un futuro el sistema puede incluir

herramientas XML para manipular y recuperar la información de documentos XML. El manejo de los documentos es muy simple ya que este sistema no es uno de administración de documentos, por lo que los documentos no se consideran como elementos principales del sistema y se les trata igual que a otros datos que consume y genera el sistema.

En siguiente capítulo se describirá como poner en ejecución el sistema y la forma en que los usuarios pueden utilizar las interfaces de usuario para poder utilizar la aplicación de caso de estudio con la que se muestra en uso del *SAWF-HW*.

Capítulo 7 Como usar el sistema

Este es básicamente el manual de usuario del prototipo desarrollado en ésta tesis. A continuación se describe como poner en ejecución el *SAWF-HW* y la manera en la que los usuarios pueden interactuar con él. La interacción con los usuarios se realiza por medio de páginas Web dinámicas que residen en un servidor Web y pueden ser utilizadas por el navegador de Internet que el usuario prefiera.

Para usar el sistema se deben ejecutar los programas siguientes:

1. Iniciar el servidor Web
2. Iniciar el servidor de correo electrónico
3. Iniciar el Workflow Engine

Una vez ejecutados los programas anteriores se podrá cargar la aplicación *WF* y los usuarios podrán utilizarla a través de páginas Web usando un navegador de Internet. Las páginas Web indican al usuario las acciones a realizar mediante enlaces de hipertexto o botones de acción, así también en algunos casos le advierten de posibles errores cometidos; dichas páginas se dividen en 3 grupos: *Autor*, *Secretaria* y *Revisor*.

7.1 Iniciando el servidor Web

El sistema fue desarrollado usando la plataforma Windows y los servidores se inician desde la línea de comandos. Se debe ejecutar una sesión de MS-DOS como se muestra en la Figura 7-1.

La ruta de los comandos para ejecutar los servidores varía dependiendo de los directorios donde se instalen. Para iniciar el servidor Web se ejecuta el siguiente comando:

```
C:\jakarta-tomcat-4.0.3\bin>startup.bat
```

y se obtiene el mensaje:

```
Starting service Tomcat-Standalone
Apache Tomcat/4.0.3
Starting service Tomcat-Apache
Apache Tomcat/4.0.3
```

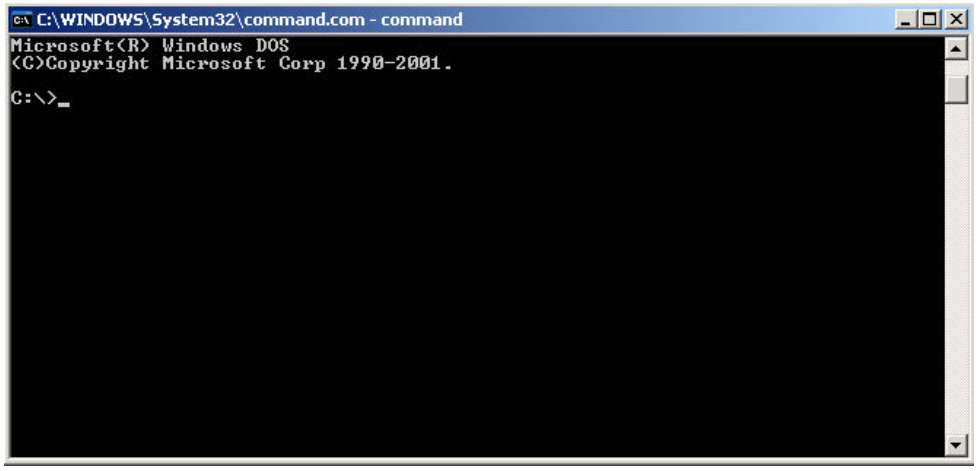


Figura 7-1 Línea de comandos de Windows

indicando que el servidor Web se inició con éxito.

7.2 Iniciando el servidor de correo electrónico

Para iniciar el servidor de corre electrónico se usa el comando:

```
D:\james\bin>run.bat
```

y se obtiene el mensaje:

```
Phoenix 4.0a4
```

```
James 2.0a3-cvs
Started POP3 Server plain:110
Started SMTP Server plain:25
Started NNTP Server plain:119
```

el cual indica que el servidor de correo se inició con éxito.

7.3 Iniciando el Workflow Engine

Para iniciar el Workflow Engine o máquina virtual de *HW* se ejecuta el software de LPA PROLOG como se muestra en la Figura 7-2.

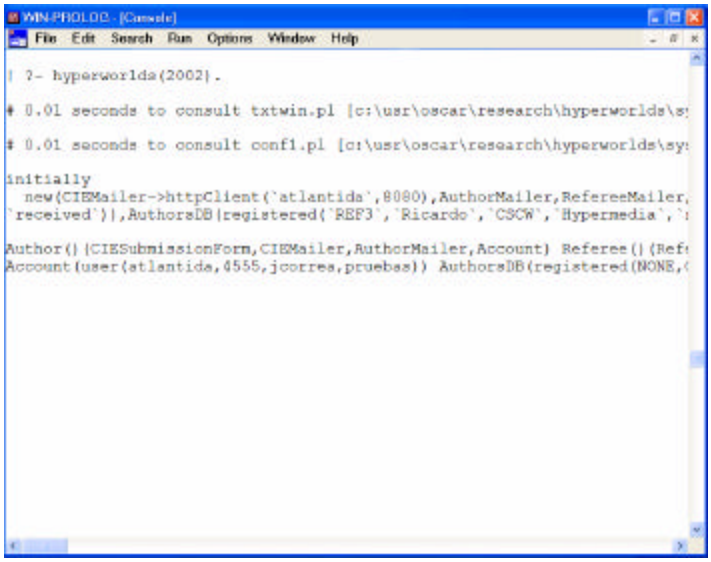


Figura 7-2 LPA PROLOG corriendo HW

al aparecer el indicador se escribe el comando siguiente:

```
? – hyperworlds(2002).
```

En donde el parámetro 2002 indica el puerto TCP/IP en donde será conectado el monitor de peticiones. La acción anterior presentará al usuario la pantalla de la Figura 7-3.

Figura 7-3 Monitor de solicitudes de *HW*

A partir de éste momento se tiene en ejecución la máquina virtual de *HW* que implementa la infraestructura del *SAWF*.

7.4 Seleccionando la aplicación Workflow

Ahora se pueden cargar las reglas que describen el comportamiento de la aplicación, para ello se ejecuta en el navegador la página, la cual es el monitor de las reglas de la aplicación que están en ejecución en la máquina virtual de *HW*, éste monitor se muestra en la Figura 7-4

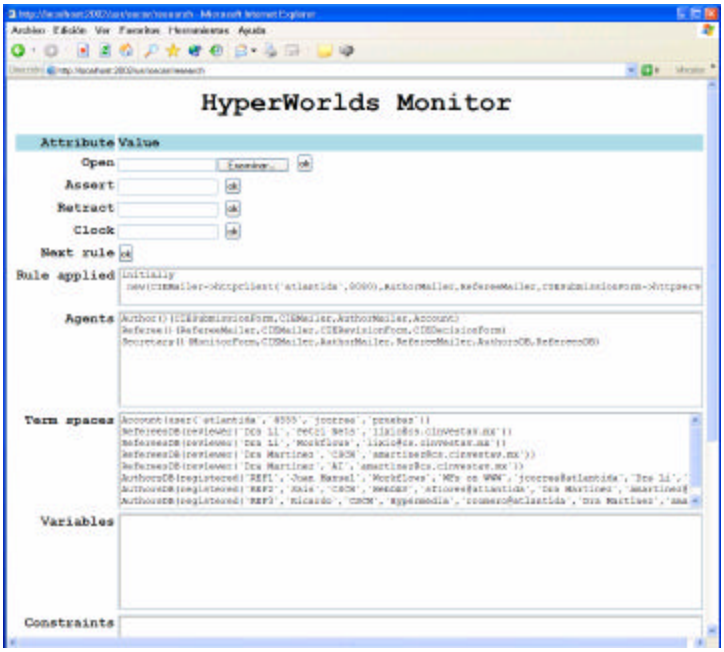


Figura 7-4 Monitor de reglas de *HW*

Desde esta página se cargan el archivo de reglas y con ello se tiene en ejecución la aplicación y está lista para ser utilizada.

Una vez iniciados el *SAWF-HW* y la aplicación, se puede hacer uso de la aplicación a través de las páginas Web con un navegador y escribiendo el URL:

```
http://server:8080/wf_app
```

Después de ésta acción el usuario puede interactuar con el sistema; las operaciones que cada usuario puede llevar a cabo en el sistema se describen en las secciones siguientes.

7.5 Usando la aplicación Workflow

A continuación se describe el uso de la aplicación *WF*. Se muestran las actividades que cada uno puede realizar y cómo pueden interactuar con el sistema para llevarlas a cabo a través de las interfaces de usuario o páginas Web.

7.5.1 Autor

Las opciones del autor se presentan en la Figura 7-5. Las opciones son:

- Enviar artículo y
- Revisar el estado del artículo.

Para acceder a ellas el autor debe presionar los enlaces de hipertexto que se muestran en la página Web. Al presionar la opción “Enviar Artículo” el sistema muestra la página de la Figura 7-6.

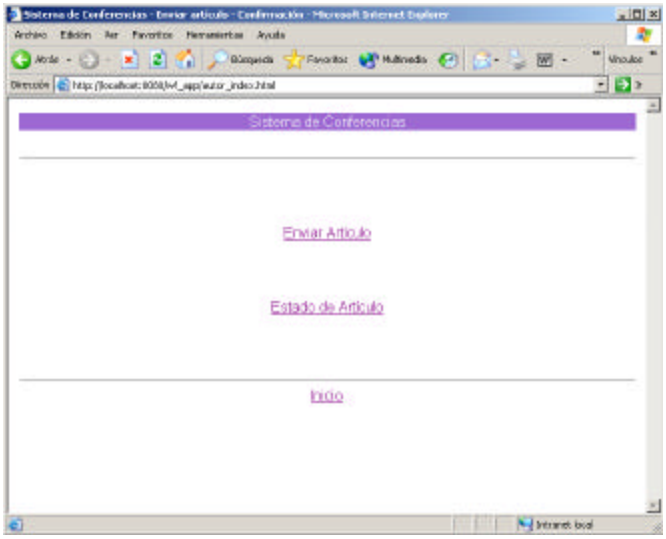


Figura 7-5 Opciones del usuario Autor

En ésta opción el usuario deberá llenar todos los campos con la información del su artículo: *nombre del autor*, *título*, *resumen*, *palabras clave*, *documento* y *dirección de correo electrónico del autor*. Al terminar de capturar la información, deberá presionar el botón "Enviar" para que su información sea validada.

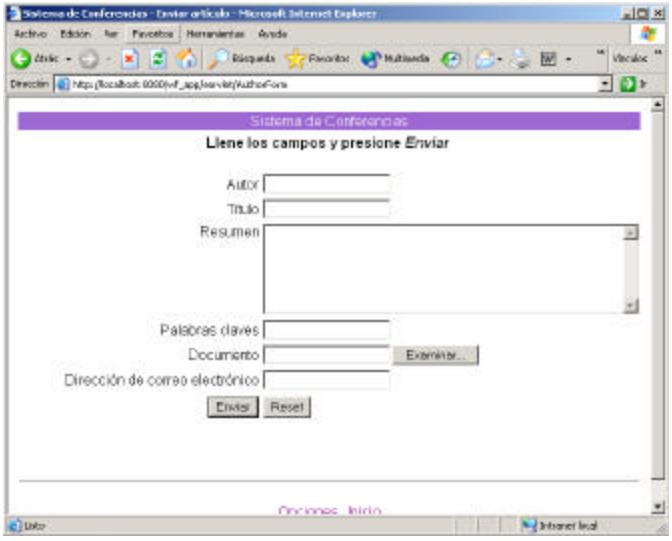


Figura 7-6 Forma de captura del artículo

El sistema validará que no falte información del artículo. Si la información no fuera válida, el sistema presentará al usuario la forma de captura que se muestra en la Figura 7-6 indicando al autor la información faltante. En caso contrario el sistema mostrará al usuario la página que se muestra en la Figura 7-7 permitiendo al autor confirmar su información. El usuario deberá presionar el botón “Enviar”, con ésta acción la información será enviada a la conferencia.

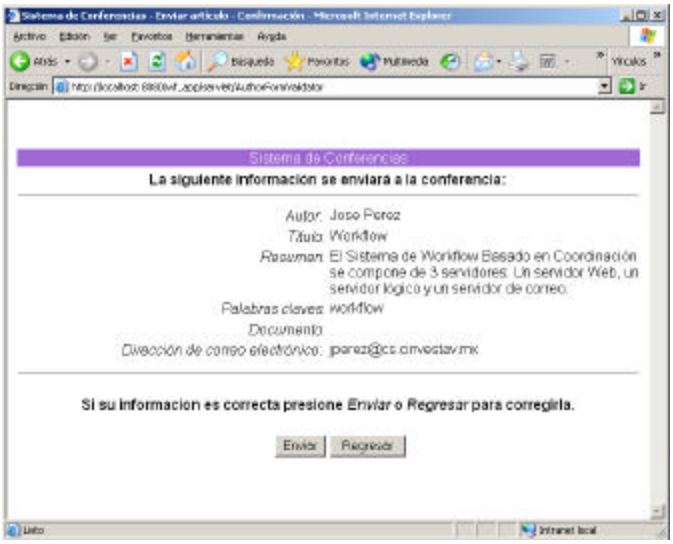


Figura 7-7 Confirmación de información del artículo

La opción “Estado del Artículo” permite al autor consultar al sistema para conocer el estado de su artículo. Esta opción se muestra en la Figura 7-8.

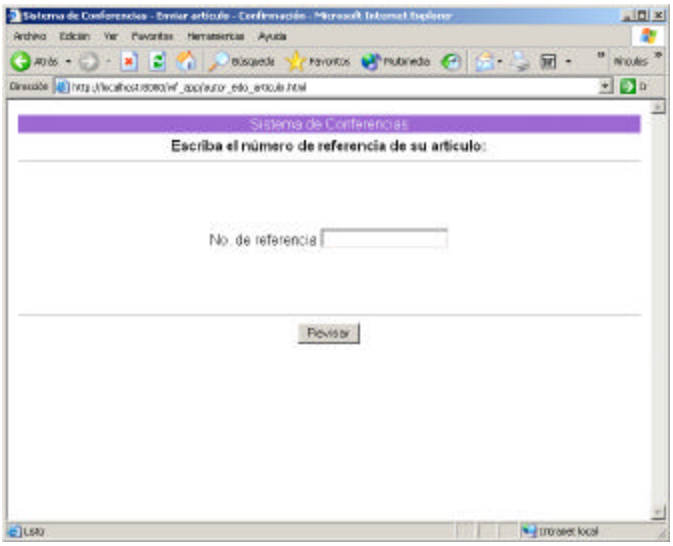


Figura 7-8 Estado del artículo

En ésta opción el autor deberá llenar el campo con el número de referencia de su artículo, el sistema hará la búsqueda de la información del artículo y la mostrará al usuario como aparece en la Figura 7-9; la cual incluye el estado del artículo que indica si es aceptado o rechazado en la conferencia.

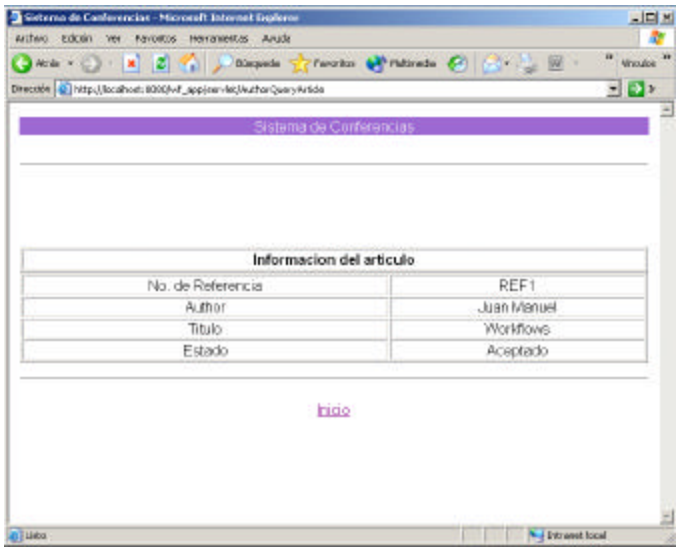


Figura 7-9 Información del estado del artículo

7.5.2 Secretaria

La secretaria tiene acceso a la información de los artículos que se reciben en la conferencia. La página de la secretaria se muestra en la Figura 7-10, la información que se presenta contiene el número de referencia asignado por el sistema, el nombre del autor, el título del artículo y el estado del artículo.

En ésta página la secretaria puede consultar la información de los artículos en el momento que se desee presionando el enlace de hipertexto “Actualizar”, la cual solicita al sistema la información reciente de los artículos que han ingresado a la conferencia.

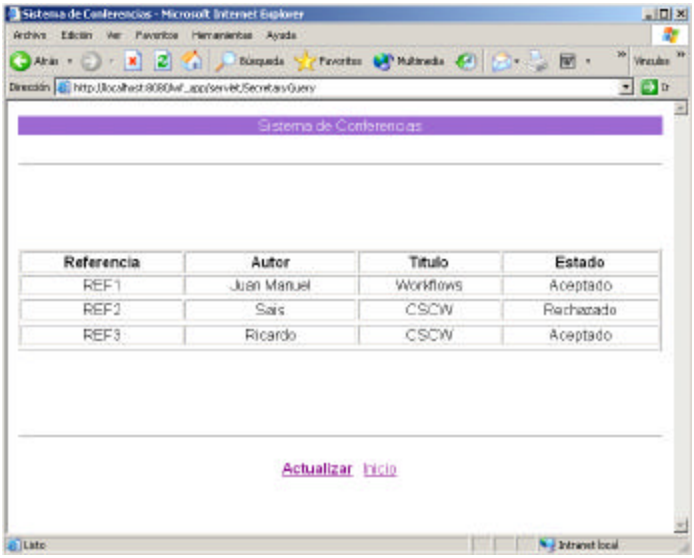


Figura 7-10 Página del usuario Secretaria

7.5.3 Revisor

Las operaciones que el usuario Revisor puede realizar son:

- Obtener información sobre los artículos asignados,
- Obtener información de un artículo específico y
- Rechazar o aceptar un artículo.

La primera página Web del usuario *Revisor* presenta una lista con la información de los artículos que le fueron asignados para revisión como se muestra en la Figura 7-11.

La página presenta un enlace de hipertexto para cada artículo asignado al usuario *Revisor*, esto le permitirá al usuario obtener una página donde se le presentará información detallada del artículo (Figura 7-12).

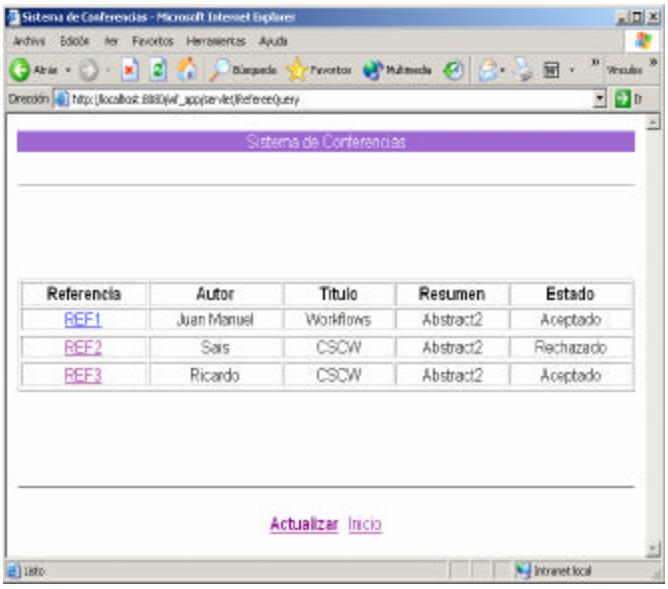


Figura 7-11 Lista de artículos asignados al usuario Revisor

Esta página permite al *Revisor* decidir sobre el artículo, es decir, el *Revisor* podrá determinar el estado del artículo en la conferencia usando los controles "Aceptado" o "Rechazado" y posteriormente presionando el botón "Decidir".

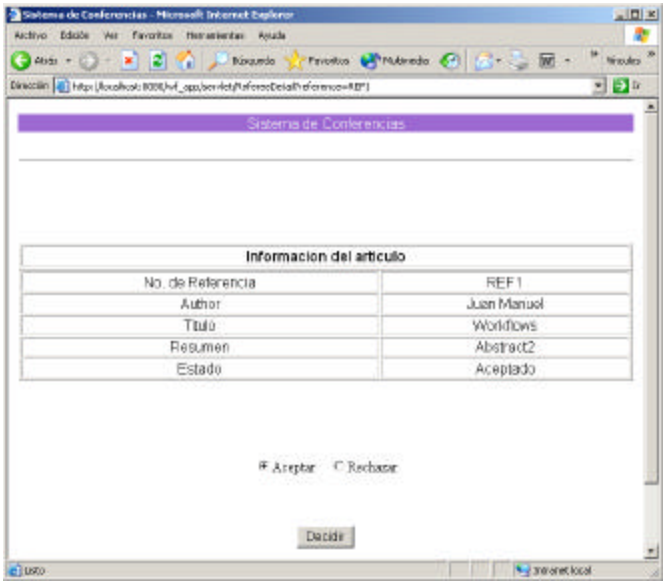


Figura 7-12 Información de un artículo asignado al usuario Revisor

Una vez que el Revisor decidió sobre el estado del artículo el sistema le mostrará la página de la Figura 7-13 de la donde se le informa acerca de la decisión que tomó sobre el artículo.

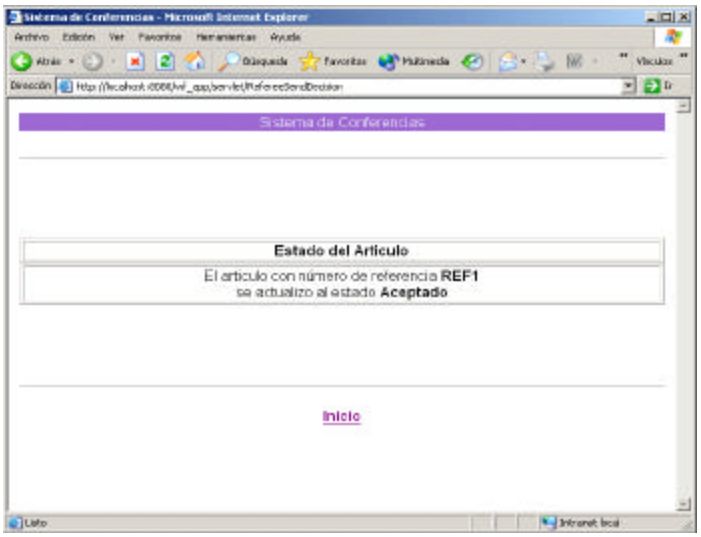


Figura 7-13 Resultado de la decisión del Revisor sobre un artículo

7.6 Conclusiones

Este capítulo es un breve manual de usuario del *SAWF-HW* para la aplicación del caso de estudio mostrada en el Capítulo 5. Se mostró la forma en como se pone en marcha el servidor Web y el servidor de correo electrónico usando las sesiones de MSDOS. También se presentaron las diferentes interfaces de usuario o páginas Web para los diferentes usuarios.

Las páginas Web tiene la funcionalidad básica y común en los sistemas que funcionan en la Web, es decir, se usaron formas con elementos de captura de texto, selección de opciones, botones de acción, etc. Se usaron los enlaces de hipertexto para moverse de una página a otra. La generación de las interfaces de usuario es dinámica con el objetivo de reflejar los cambios en la información que estén ocurriendo en el sistema.

En comparación con el enfoque seguido con la mayoría de los *SAWFs* (Capítulo 3 Sistemas de administración de Workflow), el sistema de conferencia descrito en este capítulo, usa "*Worklists*", para mostrar la secuencia de actividades que deben realizar los participantes.

Los *Worklists* se presentan al usuario como páginas Web usando el mecanismo de seguir enlaces de hipertexto para mostrar la siguiente actividad de la lista. Esto se hace evidente en las páginas del usuario *Revisor* (Figura 7-11), donde se muestran las varias opciones para el procesamiento de los documentos.

En el próximo capítulo se presentan las conclusiones así como el trabajo a futuro de esta tesis, con lo que se concluye este trabajo.

Capítulo 8 Conclusiones y trabajo futuro

Este trabajo propuso un enfoque basado en coordinación para *SAWFs*, el cual permite una descripción abstracta de los procesos de negocios, su interpretación de acuerdo al contexto que brinda la aplicación y su ejecución mediante programas que corren en un entorno distribuido.

En ésta tesis se han mostrado los conceptos de la teoría de coordinación y las ventajas que tiene sobre otros enfoques para la modelacion de *WF* (Administración de negocios, arquitectura/modelación de empresa y modelación de proceso de software), las cuales la hacen conveniente para el diseño de programas paralelos y distribuidos. En particular, se enfatizó en el uso de la coordinación para solucionar problemas que se presentan en el diseño de sistemas colaborativos, en donde la descripción de la interacción entre personas y agentes de software, así como la descripción adecuada de las actividades y de la información que fluye en ellos, es una tarea de administración que llega a ser tan compleja como el sistema mismo.

Una vez que se describió la visión de los problemas que involucran el diseño de los *SAWFs*, se mencionaron los conceptos de modelos de y lenguajes de coordinación, en particular, se describió el modelo de *HW*. *HW* ofrece las características y los elementos de modelación que lo hacen conveniente para describir de manera uniforme a los participantes, las actividades y la información de un *SAWF*.

Se propuso el caso de estudio de coordinación de actividades en una conferencia, el cual fue resuelto mediante la ejecución de las reglas el *WF* usando usando *HW*. El resultado muestra la capacidad del lenguaje para resolver la interacción entre los participantes (autor, secretaria y revisor), así como el manejo uniforme de la información que se maneja en la conferencia (documentos, bases de datos y correo electrónico).

El lenguaje de coordinación de *HW* [32] se uso como base del diseño de un *SAWF*. El resultado del diseño e implementación realizados, es un prototipo de un *SAWF-HW* con las funciones básicas de un sistema de este tipo. El prototipo desarrollado no intenta competir con las características de otros prototipos de investigación o sistemas comerciales como los que se revisaron y analizaron en este trabajo, pero si propone una perspectiva de colaboración que permite dar a los problemas que se presentan al administrar interacciones, un tratamiento uniforme basado en un conjunto de pocas primitivas de interacción. Esto facilita la integración de herramientas de software y la participación de usuarios y agentes de software.

Se probó el uso y la capacidad del prototipo implementando la aplicación del caso de estudio antes mencionado; con ésta aplicación específica no se intentó crear un sistema de administración de conferencia completo. Sin embargo se logró representar y administrar un conjunto de interacciones entre usuarios y actividades básicas que se presentan en un ambiente de colaboración, como lo es el de una conferencia. Las características del trabajo final se comparan en la Tabla 8-1, en la cual se muestra al *SAWF-HW* con un modelo simple y general comparado con los otros *SAWF*, además de tener otros aspectos básicos de diseño que son comunes, como la arquitectura cliente-servidor, lenguaje de script, mecanismo de transporte, seguimiento de procesos e integración con aplicaciones de oficina. Las herramientas de análisis, pruebas y reportes no están implementadas en *SAWF-HW* ya que no se consideran como parte de los objetivos de la tesis, pero dichas herramientas son consideradas como trabajo futuro que describe mas adelante.

HW mostró su capacidad para usarse como base del diseño de un *SAWF*, además mostró ser descriptivo al momento de definir procesos de negocios. Su modelo de coordinación da interpretación a las descripciones y su arquitectura de coordinación controla a los agentes y a los repositorios de datos de forma abstracta, uniforme y consistente. Actualmente se están modelando en *HW* otras formas de colaboración como noticias, discusiones, reuniones y cursos, lo que permitirá su integración en ambientes que puedan conducir nuevas formas de trabajo en grupo.

Tabla 8-1 Comparación de características del trabajo de tesis y los SAWFs analizados.

Sistema / ***Característica***	***Domino***	***IBM FlowMark***	***SAP Business Workflow***	***SAWF-HW***
Clasificación	Administrativo	Producción/Administrativo	Producción/ Administrativo	Producción/Administrativo
Modelo	Forma, acción, role y organización	Programas, estructuras de datos, staff y procesos	Objeto, Organización, Tarea, Aplicación y Workflow	Agentes, espacios de términos y leyes de coordinación
Cliente/Servidor	Si	Si	Si	Si
Lenguaje de script	Si	No	No	Si
Mecanismo de transporte	Correo electrónico	Base de datos	Sistema R/3	Correo electrónico y HTTP
Herramientas de análisis y pruebas	Si	Si	Si	No
Herramientas de reportes	No	Si	Si	No
Seguimiento de procesos	Si	Si	Si	Si
Integración con aplicaciones de oficina	No	Si	Si	Si(HTTP)

En cuanto al trabajo por hacer, se encuentran la implementación de la arquitectura de coordinación de *HW* en una plataforma de uso libre como Linux, la extensión del lenguaje para definir procesos de negocios más complejos y la implementación o integración de herramientas que permitan que el uso del lenguaje sea más amigable.

El trabajo futuro también incluye la continuación de la investigación en el área de la tecnología de *WF*. Como se mencionó antes, el prototipo implementado sólo cubre las perspectivas más importantes del modelo de *WF* propuesto por Jablonski [30], por lo cual queda por analizar, diseñar e implementar sobre el prototipo los aspectos que cubre las perspectivas faltantes (perspectiva de causalidad, perspectiva de integridad y recuperación de fallas, perspectiva de calidad, perspectiva histórica, perspectiva de seguridad y perspectiva de autonomía), con lo cual se lograría proponer estrategias para la solución de el diseño de dichas perspectivas así como tener un *SAWF* robusto. Las herramientas de *WF* también son un trabajo útil que facilitaría el uso y administración del sistema. Existen muchas tecnologías nuevas, como la tecnología XML, que podrían integrarse a este trabajo en sus diferentes perspectivas, ya que XML tiene la filosofía de un manejo uniforme de los datos y la independencia de la plataforma sobre la cual se manejan, por ejemplo, Windows o Unix.

Aunque los grupos de investigación tanto de universidades como de empresas particulares han desarrollado varios prototipos (ConTract[16], Domino [17], Melmac [18], OfficeTalk [19], Pegasus [20], Action Workflow [21], COSA [22], FlowMark [24], InConcert [25], ProMInanD [26], SAP Business Workflow [27], WorkParty [28]) orientados a cubrir situaciones de trabajo cooperativo, las propuestas en este sentido están lejos de agotarse.

Apéndice A. Protocolos y tecnologías

A.1 HTTP

El protocolo de transferencia de Hipertexto (HTTP) [47] funciona a nivel de aplicación para sistemas de información distribuidos, colaborativos y de hipermedia. HTTP no mantiene el estado de la transferencia y que puede usarse para diferentes tareas más allá de su uso para hipertexto, tales como servidores de nombres y sistemas de administración de objetos distribuidos, a través de extensión de sus métodos, sus códigos de error y sus encabezados. HTTP se usa en la Web desde 1990.

El propósito básico de HTTP es acceder y recuperar recursos nombrados con URL's. Un localizador de recursos uniforme o URL (*Uniform Resource Locator*)[51] es una forma de nombrar recursos accesibles desde Internet usando una cadena de caracteres.

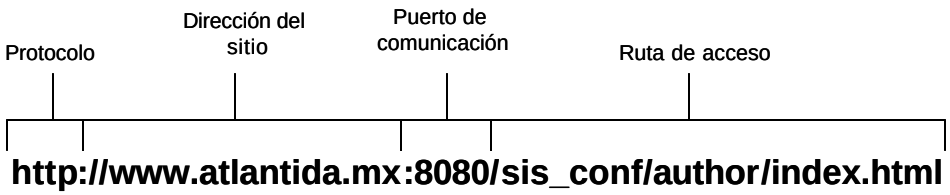


Figura 8-1 Estructura de un URL

La estructura de un URL se muestra en la Figura 8-1 y sus elementos típicos son:

- *Protocolo* le dice al navegador cual protocolo de Internet se usará para acceder al recurso de un servidor; su valor puede ser HTTP [47], FTP [48], Gopher [46], Malito [49], News [50] o WAIS [51],
- *Direccion del sitio* es normalmente el nombre de la computadora donde el servidor se ejecuta,

- *Puerto de comunicación* identifica al programa servidor y
- *Ruta de acceso* representa el recurso que se quiere acceder aunque es común que al nombre del recurso le anteceda un conjunto de directorios que indiquen su localización dentro del servidor.

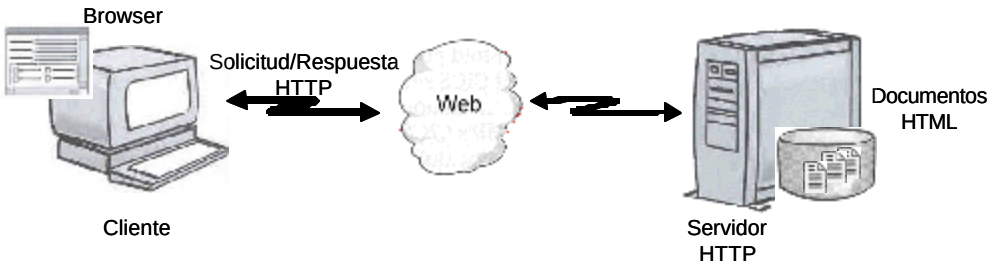


Figura 8-2 Solicitud y respuesta a una solicitud HTTP

HTTP es un protocolo sin estado. Una secuencia común de acciones se muestra en la Figura 8-2. El cliente establece una conexión a un servidor remoto y emite una solicitud. La forma de la solicitud se muestra en la Figura 8-3.

`<método><identificador de recurso><HTTP versión><crlf>`	Inicio de la solicitud

`[<encabezado> : <valor>]<crlf>`
…
…
`[<encabezado> : <valor>]<crlf>`
`Línea en blanco <crlf>`
	Encabezados de la solicitud

`[Cuerpo de la solicitud]`	Cuerpo del mensaje

Figura 8-3 Estructura de una solicitud HTTP

La solicitud puede hacerse usando los métodos `GET` o `POST` como se muestra en el ejemplo de la Figura 8-4.

`POST /wf_app/servlet/OpsEmail HTTP/1.1`	Inicio de la solicitud

`Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/vnd.ms-excel, application/vnd.ms-powerpoint, application/msword, */*`
`Accept-Language: es-mx`
`Content-Type: application/x-www-form-urlencoded`
`Accept-Encoding: gzip, deflate`
`User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)`
`Host: atlantidas.cs.cinvestav.mx`
`Content-Length: 124`
	Encabezados de la solicitud

`operation=assert&host=conf.cs.cinvestav.mx&`
`from=erivera@conf.cs.cinvestav.mx&`
`to=fortega@conf.cs.cinvestav.mx&`
`subject=Sistema+de+workflow&msg=artículo+por+revisar&`
`Submit=Send`
	Cuerpo del mensaje

Figura 8-4 Ejemplo de una solicitud HTTP

El servidor procesa la solicitud, y devuelve una respuesta y cierra la conexión. La sintaxis de la respuesta se muestra en la Figura 8-5.

`<HTTP versión><código de resultado>[explicación]<crlf>`	Inicio de la respuesta

`[<encabezado> : <valor>]<crlf>`
`.`
`.`
`[<encabezado> : <valor>]<crlf>`
`Línea en blanco <crlf>`
	Encabezados de la respuesta

`[Cuerpo del mensaje]`	Cuerpo del mensaje

Figura 8-5 Respuesta a una solicitud HTTP

Un ejemplo de la respuesta a una solicitud HTTP se muestra en la Figura 8-6.

`HTTP/1.1 200 OK`	Inicio de la respuesta

`Content-Type: text/html`
`Content-Length: 308`
`Date: Wed, 24 Jul 2002 21:22:26 GMT`
`Server: Apache Tomcat/4.0.3 (HTTP/1.1 Connector)`
	Encabezados de la respuesta

`<html>`
…
`</html>`
	Cuerpo del mensaje

Figura 8-6 Ejemplo de respuesta a una solicitud HTTP

El cliente, representado por un navegador, solicita una página de hipertexto, comúnmente también emite otras solicitudes de recursos a los que se hacen referencia en el documento, por ejemplo, imágenes. Una vez que se obtuvo el documento el usuario puede presionar alguna de los enlaces de hipertexto que le permiten moverse a otro documento. De ésta forma, HTTP emplea una conexión nueva por cada solicitud. El cliente tiene que esperar una respuesta antes de enviar una solicitud nueva.

El uso del protocolo HTTP es fundamental en esta tesis ya que permite la comunicación entre las capas que componen al sistema.

A.2 HTML

El lenguaje de marcado de hipertexto (HTML) [52] se creó con la idea de publicar información para distribución global. Para esto se necesita de un lenguaje universal, un tipo de lenguaje que todas las computadoras pudieran entender, y surgió HTML como el lenguaje de publicación usado por la Web.

HTML permite a los autores:

- Publicar documentos en línea con encabezados, texto, tablas, listas, fotos, etc.
- Recuperar información en línea mediante ligas de hipertexto.
- Diseñar formas para realizar transacciones con servicios remotos, para uso en la búsqueda de información, hacer reservaciones, ordenar productos, etc.
- Incluir hojas de cálculo, video, sonido y otras aplicaciones directamente en sus documentos.

La estructura de un documento HTML se muestra en la Figura 8-7.

```
<html>
<head>
<title>Saludo</title>
</head>

<body>
<h1>Saludo</h1>
<p>Hola a Todos!!</p>
</body>
</html>
```

Figura 8-7 Estructura de un Documento HTML

El documento comienza con la etiqueta `<html>` y termina con la etiqueta `</html>`. Cada documento HTML debe contener una sección de encabezados que se encierra entre las etiquetas `<head>` y `</head>` la cual describe el contenido del documento, su titulo, etc. La porción del documento que se verá en la parte del cliente es la que se encuentra encerrada entre las etiquetas `<body>` y `</body>`.

Las etiquetas HTML permiten también el uso de *formas*, permitiendo insertar campos de captura de texto, elementos de selección, listas de selección, y botones de eventos dentro de los documentos. Una *forma* se describen entre las etiquetas `<form>` y `</form>`. Un documento puede contener una o más formas pero éstas no pueden anidarse. Una *forma*

tiene dos campos obligatorios: `METHOD` y `ACTION`. El campo `METHOD` puede tener el valor `GET` o `POST` que corresponden a las operaciones del protocolo HTTP, por lo que este campo especifica como se enviara la información al servidor. El campo `ACTION` específica el URL al cual se enviara el contenido de la forma. Un ejemplo de una forma se muestra en la Figura 8-8.

```
<html>
<head>
<title>Sistema de Conferencias - Enviar art&iacute;culo
</title>
<body>
<form name="form1" method="post"
     action="/wf_app/servlet/AuthorFormValidator">
  <p><b>Sistema de Conferencias<br>
    </b>Autor
    <input type="text" name="author" value="">
    <br>
    T&iacute;tulo
    <input type="text" name="title" value="">
    <br>
    Resumen
    <textarea name="abstract" wrap="VIRTUAL"
         cols="50" rows="6">
    </textarea>
    <br>
    Palabras claves
    <input type="text" name="keywords" value="">
    <br>
    Documento
    <input type="file" name="document">
    <br>
    Direcci&oacute;n de correo electr&oacute;nico
    <input type="text" name="eMail" value="">
    <br>
    <input type="submit" name="Enviar" value="Enviar">
    <input type="reset" name="Reset" value="Reset">
```

```
  </p>
</form>
</body>
</html>
```

Figura 8-8 Ejemplo de una forma HTML

y ese documento se muestra en el navegador como en la Figura 8-9. HTML se uso en ésta tesis en la implementación de la capa de aplicaciones para mostrar las interfaces Web que permiten a los usuarios interactuar con el sistema.

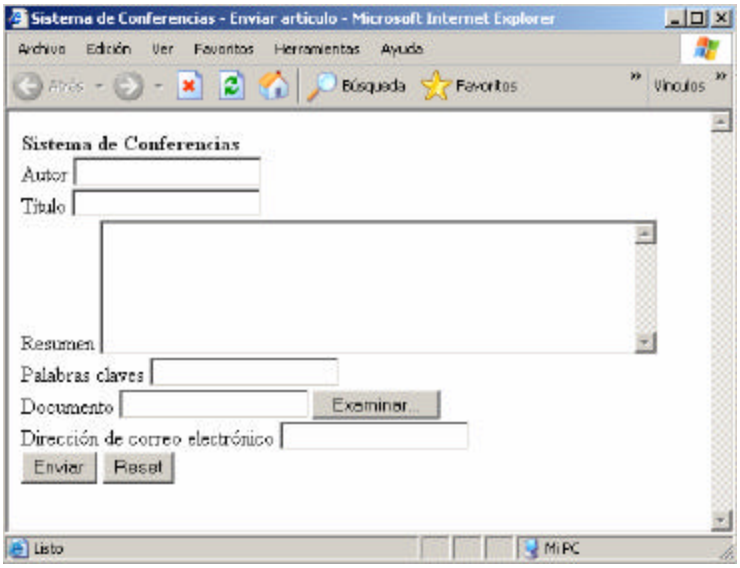


Figura 8-9 Forma HTML

A.3 Java Servlet

Un Servlet [56] es un componente Web basado en tecnología Java [55], que se administra por un contendedor, el cual genera contenido dinámico. Como otros componentes basados en Java, los Servlets son clases Java independientes de la plataforma que son compilados a una plataforma neutral que pueden ser cargadas dinámicamente y ejecutados por un servidor Web que soporte Java. Los contenedores, también llamados motores Servlet (*Servlet Engine*), son extensiones de servidor Web, parte de un servidor Web o servidor de

aplicaciones que proporcionan funcionalidad para el uso de Servlets y proporciona los servicios de red sobre los cuales las solicitudes y respuestas se envían. Los Servlets interactúan con clientes Web mediante objetos `Request` y `Response` que se implementan en el contenedor de Servlets.

Lo siguiente es la secuencia típica de eventos:

1. Un cliente, por ejemplo un navegador, accede a un servidor Web y hace una solicitud HTTP.
2. La solicitud se recibe por el servidor Web y se envía al contenedor de Servlets. El contenedor de Servlets puede correr como proceso en la misma computadora donde corre el servidor Web o en una computadora diferente del servidor Web.
3. El contenedor de Servlets determina cual Servlet invocar basándose en la configuración de sus Servlets, y llamándolo con los objetos que representan la respuesta y la solicitud.
4. El Servlet usa el objeto `Request` para encontrar quien es el usuario remoto, cuales son los parámetros HTTP POST que se enviaron como parte de la solicitud y otros datos relevantes. El Servlet ejecuta la lógica que se programó en él y genera los datos que se envían de regreso al cliente. Envía los datos de regreso al cliente usando el objeto `Response`.
5. Una vez que los Servlets han terminado de procesarse, el contenedor de Servlets asegura que la respuesta se envíe correctamente y retorna el control al servidor Web.

La tecnología Java Servlet se uso en la capa *Aplicaciones Workflow* en ésta tesis permitiendo la generación de páginas Web cuyo contenido dinámico era determinado por el resultado de la interacción entre el usuario y el sistema. También se uso en la Capa *Comunicaciones* para recibir las solicitudes de la capa *Workflow Engine*.

A.4 Tomcat

Tomcat [59] es un contenedor de Servlets que se usa en la referencia oficial para las tecnologías de *Java Servlet* y *Java Server Pages*. Tomcat se desarrollo en un ambiente abierto y de participación de los mejores desarrolladores del mundo, y fue liberado bajo la licencia de Apache Software. Tomcat forma parte del proyecto Jakarta de *Apache Software Foundation* [58] y es de uso libre. Las características de Tomcat son:

- Puede ser usado como un servidor Web o como un motor de Servlets
- Implementa las versiones recientes Servlet 2.3 y JSP 1.2,
- Un motor de Servlets nuevo llamado *Catalina* y
- Se construyó en módulos para mejorar su funcionamiento y flexibilidad

El servidor Tomcat se uso en las capa Aplicaciones y Comunicaciones permitiendo almacenar y generar páginas Web estáticas o dinámicas.

A.5 SMTP

El objetivo del protocolo simple de transferencia de correo (SMTP, Simple Mail Transfer Protocol)[53] es transferir correo eficiente y confiablemente. SMTP es independiente del subsistema de transmisión particular y requiere solamente de un canal de transferencia de datos confiable. Una característica importante es su capacidad para retransmitir correo a través de ambientes de servicio de transporte. Un servicio de transporte proporciona un ambiente de comunicación entre procesos (IPCE, *interprocess communication environment*).

Un IPCE puede cubrir una red, varias redes o un subconjunto de una red. Es importante mencionar que los sistemas de transporte o IPCEs no son uno-a-uno con las redes. Un proceso puede comunicarse directamente con otro proceso a través de IPCE mutuamente conocido. El correo es una aplicación o uso de la comunicación entre procesos. El correo puede comunicarse entre procesos en diferentes IPCEs mediante la retransmisión a través de un proceso conectado a dos o más IPCEs. Más específicamente, el correo puede ser

retransmitido entre servidores sobre diferentes sistemas de transporte mediante un servidor en ambos sistemas de transporte.

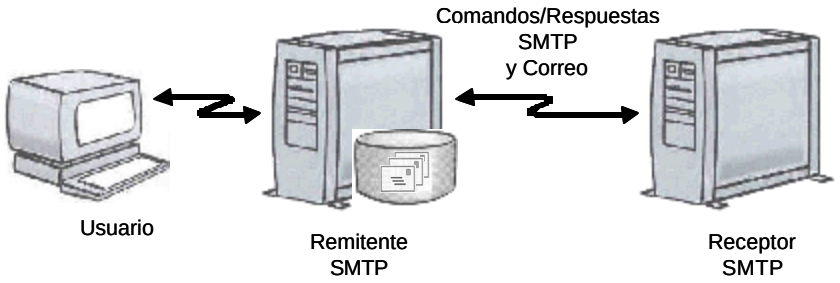


Figura 8-10 Modelo de uso de SMTP

El diseño de SMTP que se muestra en la Figura 8-10, se basa en el siguiente modelo de comunicación: como resultado de una solicitud de correo de un usuario, el remitente SMTP establece un canal de transmisión de dos vías hacia el receptor. El receptor SMTP puede ser el destino final o un intermediario. Los comandos SMTP generados por el remitente son enviados al receptor. Las respuestas SMTP son enviados del receptor al remitente en respuesta a los comandos.

Una vez que el canal de transmisión se establece, el remitente envía el comando MAIL indicando el remitente del correo. Si el receptor puede aceptar el correo responde con una respuesta OK. Entonces el remitente envía el comando RCTP identificando un recipiente de correo. Si el receptor puede aceptar el correo responde con una respuesta OK, si no, responde rechazando el recipiente (pero no toda la transacción de correo completa). El remitente y el receptor pueden negociar diferentes recipientes. Cuando los recipientes han sido negociados el remitente envía los datos del correo terminando con una secuencia especial. Si el receptor procesa con éxito los datos del correo responde con una respuesta OK.

Este protocolo se usa en este trabajo para el envío de mensajes de correo electrónico en la capa *Comunicaciones*.

A.6 POP

El protocolo de oficina postal (POP, *Post Office Protocol*)[54] es un protocolo cliente/servidor por medio del cual el correo electrónico se solicita y se almacena para el usuario. POP se pensó para permitir a una estación de trabajo acceder dinámicamente un buzón de correo de un servidor de una manera útil, esto significa que el protocolo POP es usado para recuperar correo que contiene el servidor. POP no fue pensado para proporcionar operaciones de manipulación extensiva del correo en el servidor; normalmente, el correo es descargado y borrado.

El protocolo POP mostrado en la Figura 8-11 permite que el buzón del usuario resida en una computadora que ejecute un servidor de correo y que el usuario acceda a los elementos del buzón desde otra. Como se muestra en la figura, una computadora con buzón tiene que ejecutar dos servidores, Un servidor convencional de correo que acepta el correo entrante y lo almacena en el buzón adecuado, y un servidor POP que permite al usuario de un equipo remoto acceder al buzón. El usuario ejecuta software de correo electrónico que se vuelve cliente del servidor POP para acceder al contenido del buzón.

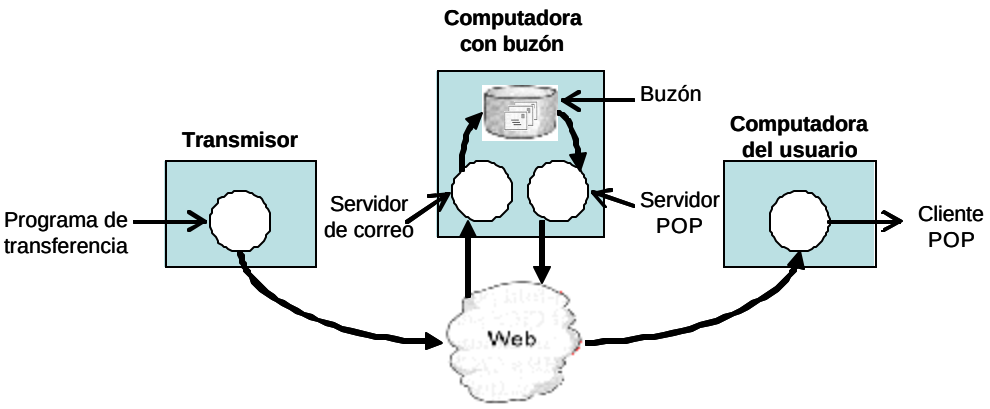


Figura 8-11 Modelo de uso de POP

El protocolo POP se usa en la capa *Comunicaciones* para la comunicación y el acceso a los almacenes de mensajes de correo electrónico así también para la búsqueda y eliminación de los mensajes.

A.7 JavaMail

JavaMail [57] proporciona un API común y uniforme para manejo de correo electrónico. Usando este API, las aplicaciones pueden acceder a los almacenes de mensajes, crear y enviar mensajes. JavaMail API es un conjunto de clases abstractas que definen objetos o partes que comprenden un sistema de correo electrónico; usando éstas clases las aplicaciones pueden crear mensajes, almacenar y recuperar mensajes, enviar mensajes y comunicarse con un cliente (por ejemplo, notificando al cliente de un nuevo correo).

El API permite extender y crear subclases que proporcionen nuevos protocolos y agreguen funcionalidad cuando sea necesario. El conjunto de clases incluyen:

- *`Message`*. Clase abstracta que representa un mensaje de correo electrónico, sus métodos más importantes son:
  - `void setContent(Object obj, String type)` especifica el contenido del mensaje como un objeto Java,
  - `void setText(String text)` especifica el contenido del mensaje como texto plano y
  - `Object getContent()` retorna el contenido de un mensaje como un objeto Java.

- *`Store`*. Clase abstracta que representa una base de datos o almacenamiento de mensajes mantenida por el servidor de correo electrónico y agrupada por usuario, sus métodos más usados son:
  - `boolean protocolConnect(String host, int port, String user, String password)` permite la conexión al servidor de correo electrónico con los parámetros especificados,
  - `Folder getDefaultFolder()` retorna el objeto fólder por defecto, normalmente el fólder INBOX y
  - `Folder getFolder(String name)` retorna el objeto fólder indicado en el parámetro.

- *`Folder`*. Clase abstracta que proporciona una manera de organizar mensajes jerárquicamente. Un fólder puede contener mensajes u otros fólder. Un servidor de correo asigna a cada usuario un fólder por defecto, y los usuarios pueden crear y llenar subfolders. Sus métodos son:
    - `String getName()` retorna el nombre del fólder,
    - `void open(int mode)` abre el fólder en modo lectura y/o escritura,
    - `void close(boolean expunge)` cierra el fólder eliminando los mensajes marcados como borrados si el parámetro lo indica,
    - `Message getMessage(int index)` retorna el objeto Mensaje identificado por `index`,
    - `Message[] getMessages()` retorna todos los mensajes del fólder,
    - `Message[] getMessages(int[] msgnums)` retorna los mensajes con identificadores definidos por `msgnums`,
    - `Message[] getMessages(int start, int end)` retorna los mensajes cuyos identificadores estén en el rango `start – end`,
    - `Message[] search(Search term)` busca y retorna los mensajes que concuerden con el parámetro de búsqueda `term` y
    - `Message[] search(Search term, Message[] msgs)` busca en el conjunto de mensajes `msgs` y retorna los mensajes que concuerden con el parámetro de búsqueda `term`.

- *`Transport`*. Clase abstracta que representa un protocolo especifico de transporte. Esta clase usa un protocolo particular, por ejemplo SMTP, para enviar un mensaje.
    - `void send(Message msg)` envía un mensaje de correo electrónico y
    - `void send(Message msg, Address[] addresses)` envía un mensaje de correo electrónico a las direcciones `addresses.`

JavaMail API se usa en la capa Comunicaciones para acceder y manipular los almacenes de mensajes de correo electrónico usando la programación orientada a objetos.

A.8 James

James (*Java Apache Mail Enterprise Server*) [60] es un servidor desarrollado 100% en Java, se diseño para ser una motor de correo completo y portable basado en los protocolos abiertos SMTP, POP3 y NNTP actuales. James forma parte del proyecto Jakarta de *Apache Software Foundation* y es de uso libre.

Las características de James son:

- *Portabilidad completa*: Apache James es una aplicación 100% Java basada en la plataforma Java 2 y en JavaMail API.
- *Abstracción de protocolo*: A diferencia de otras máquinas de correo, los protocolos se ven solamente como "lenguajes de comunicación" que rigen las comunicaciones entre los clientes el servidor. Apache James no se ata a un protocolo particular pero sigue un diseño de servidor abstracto, como lo hace JavaMail del lado del cliente.
- *Solución completa*: El sistema de e-mail es capaz de manejar correo, transporte y almacenamiento en una sola aplicación servidor. Apache James trabaja sólo sin la necesidad de otro servidor.
- *Soporte de Mailet*: Apache James soporta Apache Mailet API. Un Mailet es una pieza discreta de lógica de procesamiento de correo el cual es incorporado en un servidor de correo que soporte Mailet. Este patrón de fácil uso y escritura permite a los desarrolladores construir sistemas de correo personalizados.
- *Abstracción de recursos*: Al igual que los protocolos, los recursos se abstraen y se accedan a través de interfaces definidas, el servidor es modular y puede reusar soluciones de otros proyectos y aplicaciones.
- *Diseño seguro y multihilos*: se basa en la tecnología desarrollada para el contenedor de Servlets Apache JServ, Apache James tiene un diseño cuidadoso, seguro y multihilos para permitir buen desempeño, escalabilidad y uso crítico.

Actualmente James soporta SMTP, POP3 y una herramienta de administración remota. Este servidor se usa en la capa de comunicaciones para almacenar los mensajes de correo electrónico.

Referencias

[1] Malone T., Crowston K. *The Interdisciplinary Study of Coordination*. ACM Computing Surveys, Vol. 26, No. 1, pp. 87-119, 1994.

[2] Wegner P. 1996. *Interactive Software Technology*. In J. Allen B. Tucker, editor, The Computer Science and Engineering Handbook. CRC Press, in cooperation with ACM, 1997.

[3] Carriero N., Gelernter D., *Linda in context*. Communications of the ACM, 32(4):444–458, 1989.

[4] Banâtre J.-P., Le Métayer D., *The Gamma Model and its Discipline of Programming*. Science of Computer Programming, 15:55–77, 1990.

[5] Berry G., Boudol G., *The Chemical Abstract Machine*. In Proc. 17th ACM SIGACT/SIGPLAN Annual Symp. on Principles of Programming Languages, pags. 81–94, San Francisco, CA, 1990.

[6] Ciancarini P., *Distributed programming with logic tuple spaces.* New Generation Computing, 12(3):251-284, 1994.

[7] Olmedo O., Morales-Luna G., *Indeed: Interactive Deduction on Horn Clause Theories*. Proc. Iberamia 2002. Lecture Notes in Computer Science. Springer-Verlag. Aceptado para publicación.

[8] Kosanke, K. *CIMOSA – A European Development for Enterprise Integration, Part. 1: An Overview.* Proc. Internacional Conference on Enterprise Integration Modeling Technology (ICEIMT), Hilton Head, SC, pp. 179-188, 1992.

[9] Williams, T. J., *The Purdue Enterprise Reference Architecture: A Technical Guide for CIM Planning and Implementation*. Instrument Society of America, Research Triangle Park, NC. 1992.

[10] Vallespir B., Chen D. Zanettin M., Doumeingts G., *Definition of a CIM Architecture within the ESPRIT Project "IMPACS"*, in Computer Applications in Production Engineering: Integration Aspects, Doumeingts G., Browne J., Tomljanovich (Eds), Elsevier, Ámsterdam. 1991. pp 731-738.

[11] IDEF method report. http://www.idef.com/Complete_Reports/idef0. 1981.

[12] D.T.Ross. *Applications and extensions of SADT*. IEEE Computer, 18(4):25–35, 1985.

[13] Ashworth C., Goodland M., *SSADM: A Practical Approach*. McGraw-Hill, 1990.

[14] Fox M.S.; Gruninger M., *Ontologies for Enterprise Integration*. Proceedings 2nd Conference on Cooperative Information Systems, Toronto, Canada, pp. 82-89, 1994.

[15] Curtis B., Kellner M., Over J., *Process Management*. Communications of the ACM, Vol. 35, No. 9, pp. 75-90, 1992.

[16] Wächter H., Reuter A., *The ConTract Model*. Database Transaction Models for Advanced Aplications, Morgan Kaufmann Publishers, pp. 219-263, 1992.

[17] Kreifelts T, Hinrichs E., Klein K., Seuffert P., Woetzel G., *Experiences with the Domino Office Procedure System*. Proceedings of the 2nd. European Conference on Computer-Supported Cooperative Work (ECSCW '91), Amsterdam, Netherlands, pp. 117-130, 1991.

[18] Deiters W., Gruhn V., *Managing Software Processes in The Enviroment Melmac*. In: Cooke D. E., The Impact f CASE Technology on Software Processes. Series on Software Engineering and Knowledge Engineering, Vol. 3, World Scientific Publishers, 1994.

[19] Ellis C., Bernal M., *OfficeTalk-D: An Experimental Office Information System*. 1st. Proc. ACM-SIGOA Conference on Office Information Systems, pp. 131-140, 1982.

[20] Ahmed R., Albert J. Du W. Kent W. Litwin W. Shan M. *An Overview over Pegasus*. Proceedings of the Conference on Research Issues in Data Engineering: Interoperability in Multidatabase Systems (RIDE-IMS '93), Viena, Austria, pp. 273-277, 1993.

[21] Medina-Mora R. Winograd T., Flores R, Flores F. *The Action Workflow Approach to Workflow Management Technology*. Proceedings of the ACM 1992 Conference on Computer Supported Cooperative Work, Toronto, Canada, pp. 281-288, 1992.

[22] COSA *Administration Manual*, Software-Ley GMBH, Pulheim, Germany, 1994.

[23] COSA *Programming Manual*, Software-Ley GMBH, Pulheim, Germany, 1994.

[24] *IBM FlowMark*. *Workflow Modeling*. Internacional Business Machines Corporation, Viena, Austria, 1994.

[25] McCarthy D., Sarin S., *Workflow and Transactions in InConcert*. Bulletin of the Technical Committee on Data Engineering, Vol. 16, No. 2, pp. 53-56, 1993.

[26] Karbe B., Ramsperger N., Weiss P., *Support of Cooperative Work by Electronic Circulation folders*. Proceedings of Conference on Office Information Systems, Cambridge, MA, pp. 109-117, 1990.

[27] *SAP Business Workflow. Funktionen im Detail*. System R/3. SAP AG, February, 1995.

[28] *WorkParty. Technical Information*. Siemens Nixdorf Informationssysteme AG, 1994.

[29] W.M.P. van der Aalst, T. Basten, H.W.M. Verbeek, P.A.C. Verkoulen and M. Voorhoeve. *Adaptive Workflow: On the interplay between flexibility and support*. In J. Filipe and J. Cordeiro, editors, Proceedings of the first International Conference on Enterprise Information Systems, Setubal, Portugal, pp. 353-360, 1998.

[30] Jablonski S., Bussler C. *Workflow Management: Modeling Concepts, Architecture and Implementation*. Internacional Thomson Computer Press, 1996.

[31] Van Le T., *Techniques of Prolog Programming With Implementation of Logical Negation and Quantified Goals*. John Wiley & Sons, Inc. 1993

[32] Olmedo Aguirre J. *Design and Construction of a Coordination Language with Applications to Hypermedia.* PhD. Thesis. University of Southampton, UK. 2000.

[33] Olmedo Aguirre J. O., Correa Hernández J. M., *Definición y Gestión de Workflow en el Lenguaje de Coordinación HyperWorlds.* Octava Conferencia de Ingeniería Eléctrica CIE 2002. CINVESTAV. pp. 458-467.

[34] Ciancarini P. *Lecturas de Coordinación*. Universidad de Bolognia. 2001. http://www.cs.unibo.it/~cianca/wwwpages/seminari/lista.html

[35] Plesums C. *introduction to Workflow*. Computer Sciences Corporation, Financial Services Group In "The Workflow Handbook 2002" Published in association with the Workflow Management Coalition

[36] *Linda, User's Guide & Reference Manual.* Scientific Computing Associates. January 1995.

[37] Aalst, W. and Hee, K., *Workflow Management*. MIT Press. 2002

[38] Shaw, M. and Garlan, D., *Software Architectures*.New Jersey: Prentice-Hall, 1996.

[39] Weber, M., Partsch, G., Höck, S., Schneider, G., Scheller-Houy, A., and Schweitzer, J., *Integrating synchronous multimedia collaboration into Workflow management*. ACM GROUP 97 Phoenix Arizona USA, pp. 281- 290, 1997.

[40] Carriero, N. and Gelernter, D., *Coordination Languages and their Significance*, Communications of ACM, 35(2):96-107. 1992

[41] Ciancarini, P., *Coordination Models and Languages as Software Integrators*. ACM Computing Surveys, 28(2):300-302. 1996

[42] Ciancarini, P. and Wolf, A. L., *Coordination Languages and Models*. In Proceedingsof Third International Conference Coordination' 99. Amsterdam, The Netherlands. LNCS 1594. Berlin: Springer-Verlag. pp. 26-28. 1999

[43] Booch G., Rumbaugh J., Jacobson I., *The Unified Modeling Language, User Guide*, Addison-Wesley, 1999.

[44] Pressman R., *Ingeniería del software, Un enfoque practico*, McGraw-Hill, 1997.

[45] Paton N., Díaz O., *Active Database Systems*, ACM Computing Surveys, Vol.31. No. 1, pp. 63-103. March 1999.

[46] World Wide Web Consortium. http://www.w3.org/

[47] HTTP - Hypertext Transfer Protocol. Network Working Group. June 1999 ftp://ftp.isi.edu/in-notes/rfc2616.txt

[48] A FTP URL Format. Casey, Internet Draft issued 1996-11-26. http://www.ics.uci.edu/pub/ietf/uri/draft-casey-url-ftp-00.txt

[49] The mailto URL scheme. P. Hoffman, L. Masinter, J. Zawinski, Internet RFC 2368 issued 1998-07. http://www.ics.uci.edu/pub/ietf/uri/rfc2368.txt

[50] The hnews URL scheme. T. Stockwell, Internet Draft issued 1998-06-17. http://www.ics.uci.edu/pub/ietf/uri/draft-stockwell-hnews-url-00.txt

[51] Uniform Resource Locators (URL). T. Berners-Lee, L. Masinter, M. McCahill, Internet RFC 1738 issued 1994-12. http://www.w3.org/Addressing/rfc1738.txt

[52] HTML 4.01 Specification. W3C Recommendation December 1999. http://www.w3.org/TR/1999/REC-html401-19991224.

[53] Simple Mail Transfer Protocol. August 1982. http://www.ietf.org/rfc/rfc0821.txt

[54] Post Office Protocol. May 1996. http://www.ietf.org/rfc/rfc1939.txt

[55] Lenguaje Java . http://java.sun.com

[56] Java Servlets Technology. http://java.sun.com/products/servlet/

[57] Java Mail API. http://java.sun.com/products/javamail/index.html

[58] Proyecto Apache Jakarta. http://jakarta.apache.org

[59] Apache Tomcat. http://jakarta.apache.org/tomcat/index.html

[60] Jakarta James. http://jakarta.apache.org/james/index.html

[61] WebDAV. http://www.webdav.org