ESCUELA POLITÉCNICA NACIONAL

ESCUELA DE INGENIERÍA

ELABORACIÓN DE UN COMPONENTE DE SOFTWARE REUTILIZABLE PARA LA INTERACCIÓN DE APLICACIONES CLIENTE Y SISTEMAS DE GESTIÓN DE FLUJOS DE TRABAJO (WFMS).

PROYECTO PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN SISTEMAS INFORMÁTICOS Y DE COMPUTACIÓN

SALAZAR VILLACÍS CÉSAR GONZALO

DIRECTOR: ING. CARLOS BONILLA

Quito, Septiembre 2007

DECLARACIÓN

Yo, César Gonzalo Salazar Villacís, declaro bajo juramento que el trabajo aquí descrito es de mi autoría; que no ha sido previamente presentada para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración cedo mis derechos de propiedad intelectual correspondientes a este trabajo, a la Escuela Politécnica Nacional, según lo establecido por la Ley de Propiedad Intelectual, por su Reglamento y por la normatividad institucional vigente.

_____

César Gonzalo Salazar Villacís

CERTIFICACIÓN

Certifico que el presente trabajo fue desarrollado por César Gonzalo Salazar Villacís, bajo mi supervisión.

Ing. Carlos Bonilla

DIRECTOR DE PROYECTO

CONTENIDOS

RESUMEN

El presente proyecto consiste en la elaboración de un componente de software que sea utilizado en el desarrollo de "aplicaciones clientes de flujo de trabajo", para que dichas aplicaciones interactúen con un Sistema de Gestión de Flujos de Trabajo (WFMS).

En el capitulo 1 se hace una descripción más detallada del componente a desarrollarse y el contexto en el que este funcionaría. Luego se realiza una descripción de la metodología de desarrollo y los estándares a utilizarse para la elaboración del componente.

En el capítulo 2 se hace una definición de los requerimientos que debe cumplir el componente, y luego se detalla la documentación generada del análisis y el diseño del componente.

En el capítulo 3 se muestra la elaboración del componente, se detallan las herramientas y el marco de trabajo, así como los estándares de codificación utilizados. Posteriormente se muestra el marco de realización de pruebas y en este mismo capítulo se incluye al final un manual de utilización del componente.

El objetivo del cuarto capítulo es el desarrollo de un aplicativo de demostración de uso del componente, en este capítulo se propone un problema teórico y se diseña y se desarrolla una solución usando el componente generado anteriormente. Finalmente se realizan las pruebas del aplicativo de demostración.

En el capítulo 5 se muestran las conclusiones y recomendaciones obtenidas.

Dado que se requiere un conocimiento claro de conceptos referentes a Sistemas de Gestión de Flujos de Trabajo, y otros temas relacionados, en los anexos se ha

incluido información que ayudará mucho a la comprensión del presente trabajo.

INTRODUCCIÓN

Desde que existen las organizaciones, estas han tenido en sus procesos, además de la planificación y la ejecución, el control de procesos.

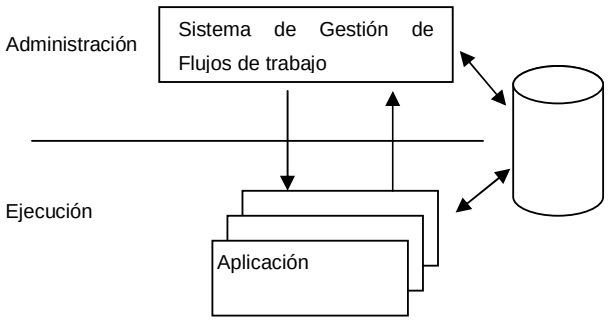
Gran parte del funcionamiento de una empresa no es solo la ejecución de tareas específicas, sino la interacción entre tareas y recursos (personas, sistemas, etc.) para la ejecución de los procesos. De esta problemática nace el concepto de flujo de trabajo.

Dado que los sistemas son desarrollados para automatizar en parte o totalmente uno o varios procesos de la organización, deben implícitamente resolver la problemática de los flujos de trabajo, pero mezclando en el mismo sistema tanto la ejecución como la administración (control) de un proceso. Esto genera el problema de que cambios en los procesos implican cambios en las aplicaciones.

Actualmente esto ya no es lo más conveniente, sino que los sistemas de información deben además del soporte para la ejecución, dar el soporte para las tareas de control, monitoreo, optimización y soporte de los aspectos logísticos de un proceso del negocio; separando la ejecución y el control de un proceso.

Es decir los sistemas deben separar la ejecución de tareas específicas de la lógica de gestión de los flujos de trabajo. De esto nace el concepto de Sistemas de Gestión de Flujo de Trabajo (Workflow Management Systems ó WFMSs).

Haciendo una analogía con los Sistemas de Administración de Bases de Datos, podemos decir que, así como (hace un poco más de 30 años) los datos estaban embebidos en las aplicaciones, así también el control de procesos ha estado también embebido en las aplicaciones. Los Sistemas de Gestión de Flujo de Trabajo (WFMSs) son la alternativa para el desarrollo de sistemas de información en los cuales se desee separar la ejecución y el control de los procesos.



FUENTE: AALST, Wil van der HEE, Kees van; Workflow Management. The MIT Press. Londres Inglaterra. 2002

La presente tesis tiene como objetivo la generación de un componente de software reutilizable que permita la rápida creación de aplicaciones que interactúen con Sistemas de Gestión de Flujo de Trabajo (WFMSs). Estas aplicaciones son conocidas como “aplicaciones clientes de flujo de trabajo”.

El componente a desarrollarse para interacción entre las “aplicaciones cliente de flujo de trabajo” y los WFMS, estará basado en los estándares propuestos por la WfMC (Workflow Management Coalition) y el OMG (Object Management Group). Tanto la WfMC como el OMG son organizaciones ampliamente reconocidas a nivel mundial, y sus estándares tienen una gran aceptación en el mercado.

La mayoría de WFMSs proveen librerías para creación de “aplicaciones clientes de flujo de trabajo” solamente en el mismo lenguaje que fue desarrollado el WFMS pero existen muchas organizaciones especialmente medianas y grandes que no desarrollan todas sus aplicaciones sobre una sola plataforma tecnológica.

Viendo esta necesidad, el componente a desarrollarse tiene como objetivo permitir la interacción de aplicaciones escritas en un lenguaje diferente que el WFMS.

CAPITULO 1. PRELIMINARES

1.1 DESCRIPCIÓN DEL COMPONENTE A DESARROLLARSE

1.1.1 TÉRMINOS UTILIZADOS

Se ha tratado de que los términos utilizados estén en concordancia con los términos utilizados por la WfMC[1] (Coalición para la Gestión de Flujos de Trabajo), pero no se puede asegurar que todos los autores o desarrolladores utilicen los mismos términos, pues no existe una definición formal en español, sino solo en inglés.

Algunas definiciones de los términos referentes a conceptos de flujos de trabajo se detallan en el ANEXO 1.

1.1.2 OBJETIVOS DEL COMPONENTE

Los sistemas de flujos de trabajo han ido evolucionando y cambiando durante la historia, y aún cuando su desarrollo ha sido vertiginoso, todavía no podemos decir que se encuentran en una etapa madura.

La visión actual de los sistemas de flujos de trabajo propone la separación de la lógica de flujos de trabajo del resto de la lógica del negocio. Haciendo una analogía, así como en los años 60's la perspectiva fue separar los datos de las

[1] WfMC, Coalición para la Gestión de Flujos de Trabajo, por sus siglas en inglés (Workflow Management Coalition), es una organización mundialmente reconocida, con una membresía de más de 200 empresas de 25 países, dedicada a desarrollar terminología e interfaces estándar para los componentes de los Sistemas de Gestión de Flujos de Trabajo.

aplicaciones, mediante el uso de los DBMS[2] (Sistemas de Gestión de Bases de Datos), así las últimas perspectivas de los sistemas de flujos de trabajo es separar la definición de procesos de las aplicaciones, mediante el uso de los WFMS[3] (Sistemas de Gestión de Flujos de Trabajo).

Se ha realizado un estudio histórico de la evolución de los sistemas de flujo de trabajo, su estado actual y sus perspectivas. Leer este estudio puede ser útil para tener un mejor entendimiento del objetivo de este componente. Este estudio se encuentra en el ANEXO 2.

El componente a desarrollarse podrá ser utilizado en el desarrollo de aplicaciones cliente de un Sistema de Gestión de Flujos de Trabajo.

Un WFMS es "una herramienta de software genérica que nos permite la definición, ejecución, registro y control de flujos de trabajo." [4]

La explicación de la definición, propósito y la visión de un WFMS no forman parte del alcance de la presente tesis, pero es esencial tener un conocimiento de dichos conceptos para entender el propósito del presente trabajo. Se ha hecho por tanto una explicación de los conceptos concernientes, la cual se puede revisar en el ANEXO 3.

Desarrollar sistemas de flujos de trabajo utilizando WFMS tiene muchas ventajas demostradas. Se puede revisar esto en el ANEXO 4, en el cuál se encuentra un estudio del porqué utilizar un WFMS.

---

[2] DBMS: "Sistemas de Gestión de Bases de Datos", por sus siglas en inglés (Database Management Systems)

[3] WFMS: "Sistemas de Gestión de Flujos de Trabajo", por sus siglas en inglés (Workflow Management Systems)

[4] AALST, Wil van der y HEE, Kees van, "Workflow Management: Models, Methods and Systems", The MIT Press, Cambridge Massachussets, 2002, pag. 1

El objetivo principal del componente a desarrollarse es ser una pieza de software genérica y reutilizable, para un desarrollo más rápido de aplicaciones cliente de un WFMS.

Además debe cumplir con el objetivo de permitir la interacción de aplicaciones cliente escritas en diferentes lenguajes, y utilizar estándares ampliamente aceptados, propuestos por las organizaciones WfMC y OMG[5].

La mayoría de WFMSs presentes en el mercado proveen ya una librería para interacción de las aplicaciones cliente cuando los clientes son escritos en el mismo lenguaje que el WFMS. Uno de los objetivos de este componente es permitir la interacción de clientes escritos en un lenguaje diferente al lenguaje del WFMS. El componente permitirá a clientes escritos en .NET acceder a un WFMS que funciona sobre tecnología J2EE / java.

El componente a desarrollarse debe permitir que la aplicación cliente interactúe con el WFMS, dentro del contexto del modelo de referencia de WFMS propuesto por la WfMC, el cual se detalla a continuación.

1.1.3 MODELO DE REFERENCIA DE WFMS PROPUESTO POR LA WFMC

1.1.3.1 Resumen

La WfMC (Workflow Management Coalition), la cual ha proporcionado estándares altamente aceptados en el mercado, ha hecho una definición de las partes que un WFMS debería tener.

---

[5] Object Management Group, Inc. (OMG) es una organización internacional apoyada por más de 800 miembros, que incluye vendedores de sistemas de información, diseñadores de software y usuarios. Fundado en 1989, el OMG promueve la teoría y práctica de la tecnología orientada a objetos en el desarrollo del software. La carta constitucional de la organización incluye el establecimiento de pautas de industria y especificaciones de manejo de objetos para proporcionar un marco común para el desarrollo de aplicaciones.

El modelo propuesto por la WfMC es el siguiente:

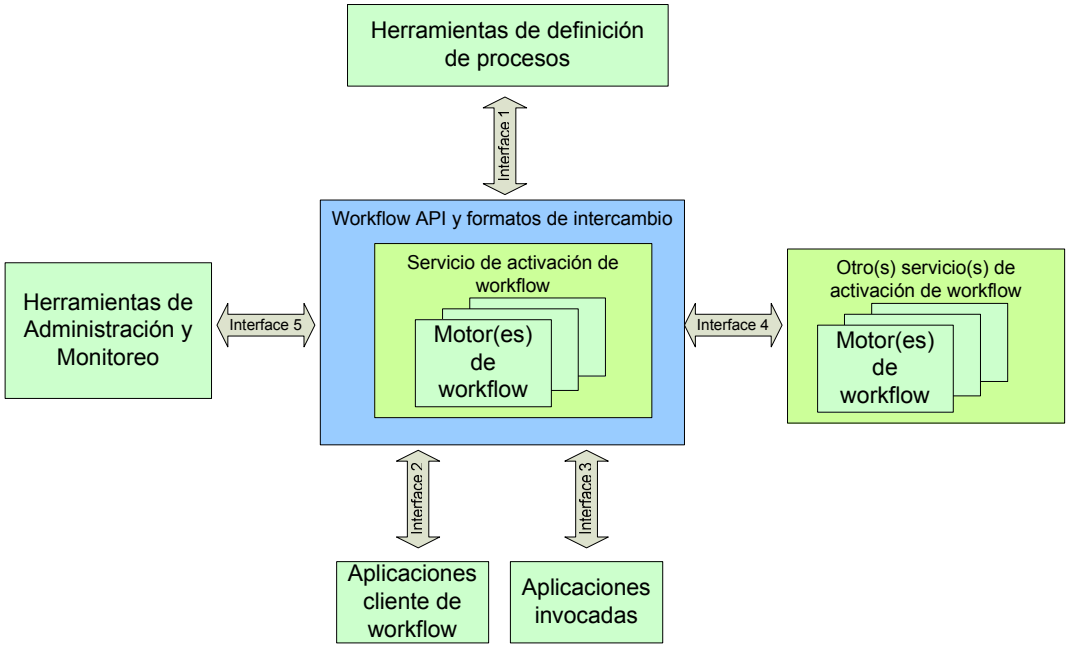


Figura 1-1 Modelo de referencia de la WfMC

Fuente: W.M.P. van der Aalst, “The Application Of Petri Nets to Workflow Management”, p. 10

1.1.3.2 Componentes de un WFMS

Dentro de este modelo, tenemos las siguientes partes:

1.1.3.2.1 Servicio de Activación de Flujo de Trabajo (Workflow Enactment Service)

Es la parte central del WFMS y provee el entorno de ejecución que se encarga del control y la ejecución de los flujos de trabajo.

No se debe confundir un Servicio de Activación de Flujo de Trabajo con un Motor de Flujo de Trabajo, pues un Servicio de Activación de Flujo de Trabajo puede

estar compuesto por más de un Motor de Flujo de Trabajo, aunque el usuario por lo general no va a notar si un sistema usa uno o varios motores.

En las organizaciones pequeñas y en algunas medianas por lo general el Sistema de Activación de Flujo de Trabajo estará compuesto de un solo motor de Flujo de Trabajo, pero en organizaciones más grandes se pueden tener varios motores de Flujo de Trabajo por razones técnicas o por razones administrativas.

1.1.3.2.2 Motor de Flujo de Trabajo

Un motor de Flujo de Trabajo está hecho para manejar una parte de los procesos de la organización con una parte de los recursos. El motor de Flujo de Trabajo debe encargarse de las siguientes actividades: [6]

- "Crear nuevos casos y remover los completados
- Enrutar casos, usando la interpretación de la definición de procesos apropiada.
- Manejar atributos de caso
- Enviar ítems de trabajo a los recursos correctos (empleados), basado en una clasificación de recursos
- Manejar y controlar disparadores
- Comenzar aplicaciones de software durante la ejecución de una actividad
- Grabar datos históricos
- Proveer un resumen de Flujo de Trabajo, y
- Monitorear la consistencia del Flujo de Trabajo"

1.1.3.2.3 Herramientas de Definición de Procesos

Son herramientas usadas por lo general al momento de diseñar o rediseñar los procesos o la estructura de la organización. Son la interfaz del WFMS para:

---

[6] AALST, Wil van der y HEE, Kees van, "Workflow Management: Models, Methods and Systems", The MIT Press, Cambridge Massachussets, 2002, pag. 151.

- Definir los procesos
- Definir los recursos y tipos de recursos
- Validar las definiciones de procesos

La mayoría de WFMSs actuales proveen una o varias herramientas gráficas de definición de procesos. Algunos incluso proveen herramientas para simulación de las definiciones de Flujo de Trabajo.

1.1.3.2.4 Herramientas cliente de Flujo de Trabajo

El usuario final de Flujo de Trabajo se comunica al WFMS mediante aplicaciones que interactúan con el Sistema de Activación de Flujo de Trabajo (Workflow Enactment Service).

La mayoría de WFMS incorporan herramientas de cliente genéricas y una interfaz para herramientas más especializadas.

Las herramientas de cliente que el WFMS incorpora generalmente no son más que bandejas de entrada y salida de ítems de trabajo, en el cual no se incluyen sino ciertas características relevantes del caso, para que el usuario pueda realizar el procesamiento del mismo.

1.1.3.2.5 Herramientas de monitoreo y administración

Uno de los propósitos de un WFMS es proveer la funcionalidad de monitorear los procesos, para esto son necesarias herramientas, las cuales deben interactuar con el Sistema de Activación de Flujos de Trabajo para interpretar los datos registrados por el WFMS.

Generalmente las herramientas de monitoreo y administración sirven para detectar cuellos de botella, para visualizar el progreso de uno o varios casos de un proceso, etcétera.

Además estas herramientas son necesarias para cambiar parámetros, asignar recursos y manejar anormalidades.

1.1.3.2.6 Aplicaciones invocadas

Dentro de los procesos de una organización, no todas las tareas son realizadas por personas, sino que muchas son realizadas por aplicaciones de software creadas para tal fin.

Las aplicaciones invocadas constituyen aplicaciones con las que el WFMS interactúa directamente. Pueden proveer datos y eventos al WFMS o pueden recibir datos y eventos del WFMS, y con éstos realizar tareas específicas.

Dentro de estas aplicaciones, provistas por los WFMS están aplicaciones para envío de correo electrónico, acceso a bases de datos, entre otras.

Además de proveer aplicaciones para tareas comunes, como las mencionadas anteriormente, un WFMS debe proveer una interfaz para que la organización pueda integrar sus propias aplicaciones.

Las aplicaciones invocadas pueden ser de dos tipos: interactivas o totalmente automáticas.

1.1.3.3 Interfaces de un WFMS

La WfMC ha definido varias interfaces para interacción de los diferentes componentes de un WFMS. Uno de los objetivos de la WfMC ha sido la estandarización de dichas interfaces con el fin de facilitar la integración entre

herramientas de varios proveedores y para permitir incluso la integración de varios WFMS's de una manera más simple.

La WfMC ha hecho una definición teórica de 5 interfaces que debe tener un WFMS, como se muestra en la Figura 1-1. Estas interfaces son:

1. Interfaz 1: Herramientas de definición de procesos
2. Interfaz 2: Aplicaciones cliente de Flujo de Trabajo
3. Interfaz 3: Aplicaciones invocadas
4. Interfaz 4: Otros Servicios de activación de Flujo de Trabajo
5. Interfaz 5: Herramientas de monitoreo y administración

La interfaz puede consistir de una definición usando:

- Archivos: Por ejemplo, en la interfaz 1, la herramienta de definición de procesos podría generar un archivo que servirá de entrada para el Servicio de Activación de Flujo de Trabajo, el cual funcionará con dicha definición.
- Bases de Datos. Por ejemplo, en la interfaz 5, las herramientas de monitoreo y administración podrían acceder a una base de datos generada por el Servicio de Activación de Flujo de Trabajo con el fin de interactuar con el mismo.
- API's (Application Programming Interface o Interfaz de Programación de Aplicación), el cual consiste en servicios ofrecidos a un cliente mediante un servidor.

En la actualidad muchos WFMS's utilizan archivos y bases de datos sobre todo para las interfaces 1 y 5, pero dentro de los documentos de la WfMC, se asume que todas las interfaces se deben implementar mediante API's.

Para definir las API's relacionadas al Flujo de Trabajo se utiliza el término WAPI (Workflow Application Programming Interface).

1.1.3.3.1 Interfaz 1: Herramientas de definición de procesos

Esta interfaz es utilizada para que las herramientas de definición de procesos interactúen con el motor de Flujo de Trabajo. El API utilizado para esta interfaz debe proveer funciones que permitan:

- Obtener datos de las definiciones que está manejando el WFMS
- Definir nuevos procesos o cambiar procesos de Flujo de Trabajo del WFMS

La WfMC ha hecho una definición del API de esta interfaz en el documento WfMC TC-1016-O, el cual está disponible en el Internet en el sitio web de la WfMC. [7]

El estándar más ampliamente utilizado para la definición de procesos es XPDL (XML Process Definition Language). XPDL es un estándar propuesto por la WfMC. Se puede obtener más información de XPDL en el sitio web de la WfMC.

1.1.3.3.2 Interfaz 2: Aplicaciones cliente de Flujo de Trabajo

Esta interfaz debe proveer funciones para permitir:

- Creación de nuevos casos
- Recepción de ítems de trabajo, con sus propiedades correspondientes
- Indicar al WFMS el inicio, interrupción y terminación de la realización de un determinado ítem de trabajo.
- Recepción y envío de eventos relacionados al Flujo de Trabajo.

1.1.3.3.3 Interfaz 3: Aplicaciones invocadas

Provee funciones similares a la interfaz anterior. Es por esto, que la WfMC, aunque ha hecho la definición teórica de las interfaces 2 y 3, ha hecho una sola

---

[7] www.wfmc.org Sitio web de la WfMC (Workflow Management Coalition)

definición técnica de una API para ambas interfaces. [8] Este documento será analizado posteriormente de una manera más amplia pues es necesario para el desarrollo de la presente tesis.

1.1.3.3.4 Interfaz 4: Otros Sistemas de activación de Flujo de Trabajo

Debe permitir el intercambio de ítems de trabajo entre varios sistemas de activación de Flujo de Trabajo. Esta funcionalidad puede ser necesitada por ejemplo para interacción en empresas grandes, en las que se implemente más de un WFMS o para la interacción del flujo de trabajo de varias empresas.

Esta es una de las interfaces que tiene más complicaciones, y aunque hay varias definiciones teóricas, y algunas implementaciones en los WFMSs actuales, todavía no existe una implementación que permita una interoperabilidad completa de WFMSs de varios proveedores.

Mientras los WFMSs vayan madurando, seguramente en un futuro esta interfaz se irá consolidando y se tendrá una definición mucho más completa de la misma.

1.1.3.3.5 Interfaz 5: Herramientas de Administración y Monitoreo

La API de esta interfaz debe proveer dos tipos de funciones:

- Funciones para administración del sistema de Flujo de Trabajo
- Funciones para acceso a datos históricos y estadísticos del Flujo de Trabajo

---

[8] WfMC, “Workflow Management Application Programming Interface (Interface 2&3) Specification”, Documento Número WFMC-TC-1009, Julio 1998, Versión 2.0. Disponible también en www.wfmc.org

1.2 METODOLOGÍA DE DESARROLLO

Parte de la definición de requerimientos y parte del diseño ya se encuentran detallados en el documento “Workflow Management Facility Specification” del OMG, sobre el cual se basará el desarrollo de este componente, por tanto:

- Los requerimientos del desarrollo del componente no son cambiantes, son claramente identificables y definibles, y
- La arquitectura del componente es fácilmente definible desde el inicio

Y además, dado que:

- No se necesitan tener productos mostrables durante el desarrollo, sino solo al final
- Se requiere que el componente sea confiable
- Es un proyecto relativamente pequeño

Luego de realizar un rápido análisis entre varias metodologías, se define que la mejor opción, para el desarrollo del presente componente, es la “Metodología de Desarrollo en Cascada”.

En el ANEXO 7 se detalla una breve descripción de las metodologías que se consideraron.

1.2.1 METODOLOGÍA DE DESARROLLO EN CASCADA

Esta metodología consta de las siguientes fases:

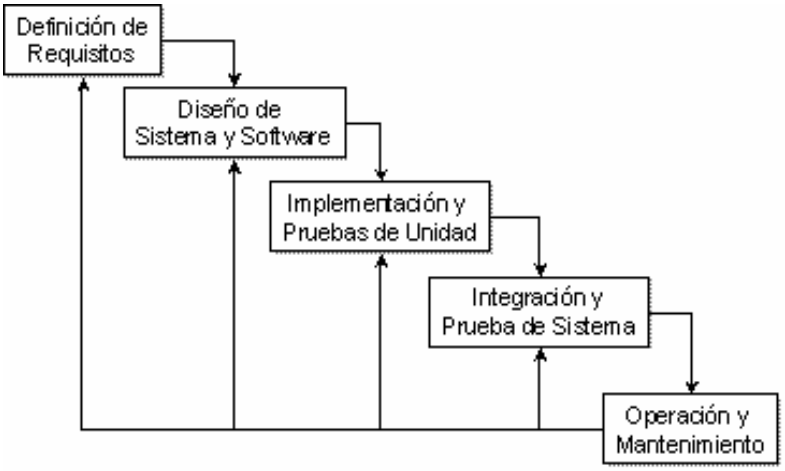


Figura 1-2 Metodología de desarrollo: Desarrollo en cascada

FUENTE: Ingeniería de Software, Somerville, 2002

La metodología de desarrollo en cascada es un enfoque riguroso de las etapas de desarrollo de software, proponiendo que no se debe comenzar una fase sino hasta haber terminado la anterior.

Aunque lo ideal es lograr pasar por cada fase una sola vez, esto no es lo que sucede en la vida real, sino que se vuelve a fases anteriores cuando habiendo ya pasado dicha etapa, se detectan errores que no se los podían detectar anteriormente.

De esta manera, cualquier error en el diseño detectado en la etapa de pruebas conduce al rediseño y a la remodificación. Esta metodología por tanto, debe ser utilizada cuando los requerimientos de usuario son fácilmente identificables, y no son cambiantes.

Dentro de las ventajas de esta metodología es que tiende a ser la base para producir software altamente confiable.

Dentro de las desventajas tenemos que: presenta problemas de desempeño cuando los requisitos y la arquitectura no están bien definidos, es difícil implementar correcciones sobre la marcha y el cliente no va a ver progresos durante el desarrollo, sino solamente al final verá el producto terminado.

1.3 ESTÁNDARES A UTILIZARSE

1.3.1 ESTÁNDARES UTILIZADOS PARA INTERACCIÓN DE LAS APLICACIONES CLIENTE

Desde que se comenzó a utilizar el concepto de WFMS, se vio la necesidad de establecer estándares para la interacción de las aplicaciones con el WFMS.

Un WFMS debe implementar estándares para la definición de procesos y clasificación de recursos (interfaz 1) y estándares para la interacción de las aplicaciones con dichos procesos que fueron definidos (interfaces 2 y 3).

Un WFMS debe tener además estándares para interacción con herramientas de monitoreo y administración (interfaz 5) y estándares para interacción con otros WFMSs (interfaz 4).

Dentro del estudio de la presente tesis, no profundizaremos sino en los estándares para interacción con aplicaciones cliente (interfaz 2).

Para la interacción de las aplicaciones cliente (y las aplicaciones invocadas), se han creado varias definiciones de APIs (Application Programming Interfaces) o Interfaces de programación de aplicaciones, las cuales especifican las funciones que un WFMS debe proveer para que sus aplicaciones cliente puedan interactuar con el mismo.

Aunque las aplicaciones cliente y las aplicaciones invocadas son dos partes diferentes dentro un WFMS, y aunque teóricamente existen dos interfaces (interfaz 2 e interfaz 3), en la práctica, ambas utilizan la misma API para la comunicación con el WFMS.

Para la elaboración del presente componente, se utilizará la especificación "Workflow Management Facility Specification v.1.2" provista por el OMG (Object Management Group), la cual está basada en la "WAPI Interface 2&3 Specification" provista por la WfMC.

A continuación, un resumen y un análisis de ventajas y desventajas de dichas especificaciones.

1.3.2 ESPECIFICACIÓN WAPI PARA INTERFACES 2 Y 3 POR LA WFMC

1.3.2.1 Resumen

Esta especificación está disponible en el documento WFMC-TC-1009 de la WfMC[9].

Este documento está directamente relacionado y hace referencia a otros dos documentos:

- Workflow Management Coalition Glossary (Glosario de la Coalición de Gestión de Flujo de Trabajo)
- Workflow Management Coalition Interface 2 WAPI Name Conventions (Convenciones de Nombre de WAPI de interface 2 de la WfMC)

Las funciones definidas en esta API pueden ser divididas en las siguientes categorías:

- Funciones de conexión
- Funciones de definición de flujo de trabajo

---

[9] WfMC, "Workflow Management Application Programming Interface (Interface 2&3) Specification", Documento Número WFMC-TC-1009, Julio 1998, Versión 2.0. Disponible también en www.wfmc.org

- Funciones de control de proceso
- Funciones de control de actividad
- Funciones de estado de proceso
- Funciones de estado de actividad
- Funciones de lista de trabajo
- Funciones de administración

En este documento contiene, además de la parte introductoria:

1. Especificación de tipos de datos utilizados
2. Definición de códigos de errores
3. Prototipos de las funciones de la WAPI

La definición de tipos de datos, y los prototipos de funciones en esta especificación se muestran en lenguaje C. Pueden sin embargo ser traducidos a otro lenguaje.

1.3.2.2 Ventajas

La ventaja más clara es que es una especificación abierta, lo que permite ser utilizada por cualquier WFMS, ampliando así las posibilidades de estandarización.

Es bastante completa, y ya existen implementaciones de WFMS que la utilizan.

Es una definición que está continuamente mejorando e incorporando nuevas funciones, al mismo tiempo que mantiene la compatibilidad con las versiones anteriores.

1.3.2.3 Desventajas

Las funciones de manejo de datos son limitadas, según el mismo documento de la WfMC[10], dice que se cree que es necesaria la implementación de nuevas

[10] WfMC, “Workflow Management Application Programming Interface (Interface 2&3) Specification”, pag. 92

funciones y la adición de varios parámetros en las funciones de manejo de datos de Flujo de Trabajo.

No contiene funciones para manejo de actividades “ad hoc” [11], este tipo de actividades son utilizadas sobre todo en Flujo de Trabajo colaborativo y para manejo de excepciones en un Flujo de Trabajo.

No existe una definición para manejo de roles y usuarios, aunque se piensa agregar en la próxima versión, aún no disponible.

No provee requerimientos específicos o previsiones de mecanismos de seguridad, sino que esto se lo deja a cargo del desarrollador del WFMS.

No provee requerimientos específicos para el manejo de bloqueos y para asegurar la integridad de procesos.

1.3.3 ESPECIFICACIÓN DE LA OMG PARA UNA FACILIDAD DE GESTIÓN DE FLUJOS DE TRABAJO (OMG’S WORKFLOW MANAGEMENT FACILITY SPECIFICATION) V.1.2

1.3.3.1 Resumen

Es una especificación basada en la propuesta de la WfMC, con adiciones propias del OMG (Object Management Group) para desarrollo de sistemas de Flujo de Trabajo usando estándares propios de la OMG, como son CORBA e IDL (Interface Definition Language) [12].

---

[11] Actividades o procesos “ad hoc”: Se da este nombre a actividades o procesos excepcionales, generalmente son actividades o procesos para los cuales es difícil establecer un diagrama que se aplique para todos los casos, pues el proceso puede variar de acuerdo a cada caso.

[12] El lenguaje de definición de interfaces (IDL) es el estándar para definición de interfaces CORBA.

Este documento está disponible en Internet, en el sitio web del OMG [13] bajo el título de “Workflow Management Facility Specification v.1.2”

1.3.3.2 Ventajas

Es un estándar que ha sido acogido por gran parte de los desarrolladores de sistemas de gestión de flujos de trabajo.

Esta especificación al ser realizada por el OMG, tiene el aval y el soporte de las empresas que conforman este grupo. Alrededor de 50 empresas participaron directamente en la creación de esta especificación, entre las que podemos destacar: Oracle, Siemens, Xerox, Hewlett Packard, IBM Corporation, entre las más conocidas.

Por su naturaleza de organización la WfMC no puede ser parte de la OMG, sin embargo, como se menciona en el mismo documento de la especificación [14], da su aval a este estándar.

Otra de las grandes ventajas de este estándar es que no solo provee un detalle de la interfaz que se provee a las aplicaciones cliente, sino que se da una breve descripción de cómo debería actuar el WFMS ante el uso de las funciones provistas por dicha interfaz.

1.3.3.3 Desventajas

Al igual que el estándar provisto por la WfMC, sobre la que se basa esta especificación, deja algunas funciones necesarias a criterio de las empresas desarrolladoras. Por ejemplo:

---

[13] http://www.omg.org/library/specindx.html. Sitio web de documentación formal provista por el OMG (Object Management Group)

[14] OMG, “Workflow Management Facility Specification v.1.2”, pag. 5

- No especifica las interfaces para conexión y desconexión.
- No especifica un mecanismo para acceso a los objetos base: Es decir, no especifica una interfaz para acceder a la lista de procesos, a la lista recursos y actividades que se encuentran en el WFMS.
- No especifica un mecanismo de manejo de seguridades
- No provee requerimientos específicos para el manejo de bloqueos y para asegurar la integridad de procesos.
- No existe una definición para manejo de roles y usuarios

Dadas estas desventajas, no se puede crear un componente totalmente estándar para todos los WFMSs, dado que debemos adaptar los puntos mencionados anteriormente a lo implementado por el desarrollador del WFMS.

1.3.3.4 Breve descripción del documento

En su parte principal, el documento describe las interfaces para interacción de las aplicaciones cliente con el WFMS, mediante un diagrama de objetos de dichas interfaces, haciendo una descripción de las mismas.

El diagrama de objetos propuesto es el siguiente:

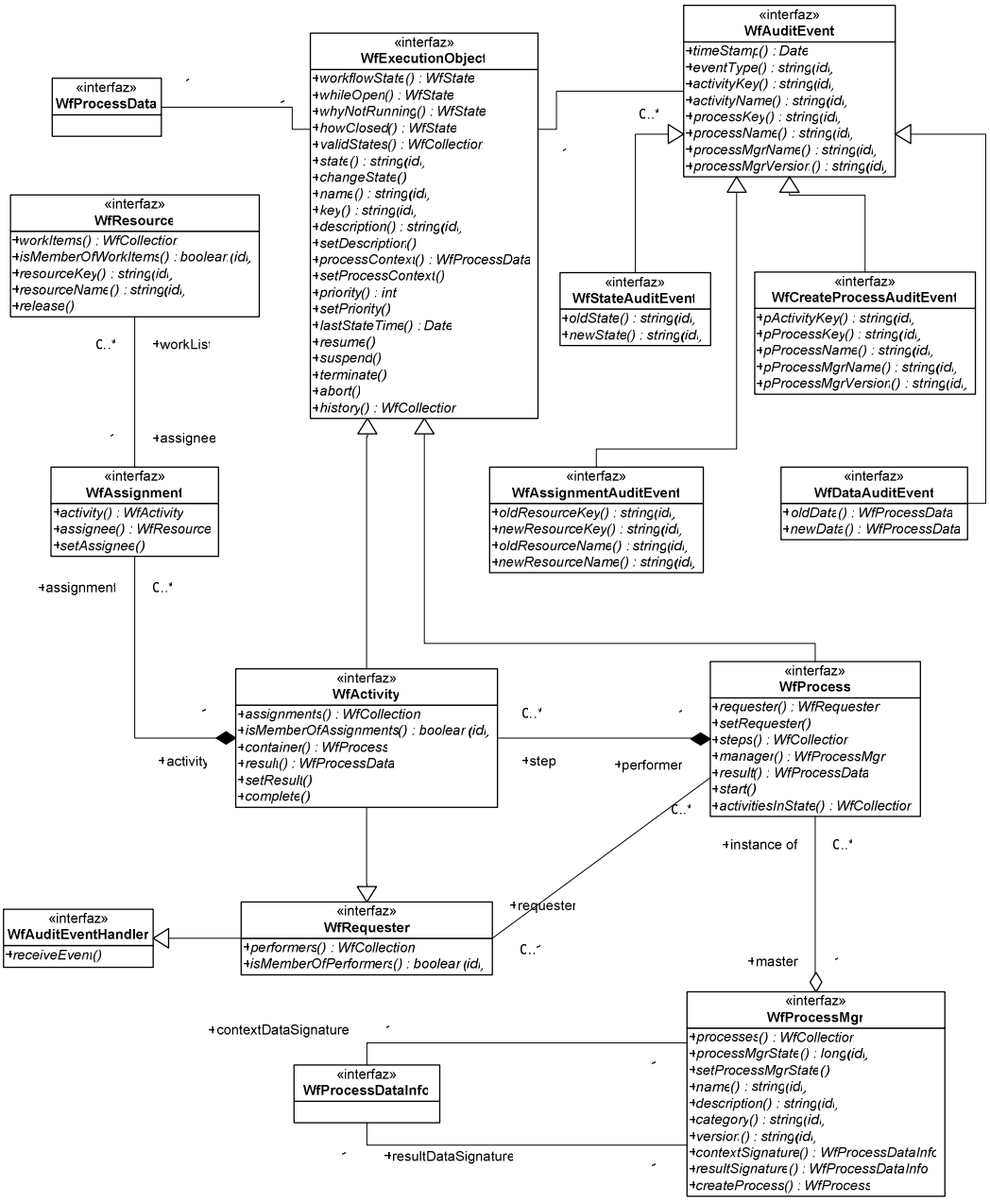


Figura 1-3 Modelo de Facilidad de Gestión de Flujo de Trabajo

FUENTE: OMG, “Workflow Management Facility Specification v.1.2”

Posteriormente, para cada interfaz se describen sus atributos, relaciones, operaciones y conjunto de estados. Haciendo un resumen de las interfaces:

	Descripción
WfRequester	Vincula el propietario inmediato de una petición con un proceso (WfProcess) (para por ejemplo recibir eventos significativos como cuando el proceso se completa). En la sección 2.4 del documento del OMG se detalla esta interfaz.
WfProcessMgr	Constituye la interfaz para localizar procesos y fabricar instancias de los mismos (WfProcess) En la sección 2.6 del documento del OMG se detalla esta interfaz.
WfProcess	Es el realizador de una instancia de un proceso de flujo de trabajo solicitada por un usuario o por un autor automático, como puede ser una actividad (WfActivity) que actúa como solicitante (WfRequester), pidiendo que inicie otro proceso. En la sección 2.7 del documento del OMG se detalla esta interfaz.
WfActivity	Consiste en un paso dentro de un proceso (WfProcess) y puede también ser un solicitante (WfRequester) para la realización de una nueva actividad (WfActivity) o proceso (WfProcess). En la sección 2.8 del documento del OMG se detalla esta interfaz.
WfExecutionObject	Es una clase abstracta base para actividades (WfActivity) y procesos (WfProcess). En la sección 2.5 del documento del OMG se detalla esta interfaz.
WfAssignment	Vincula las actividades a recursos (WfResources) potenciales o actualmente asignados. En la sección 2.9 del documento del OMG se detalla esta interfaz.
WfResource	Representa una persona o cosa que puede realizar y aceptar una actividad (WfActivity).

	En la sección 2.10 del documento del OMG se detalla esta interfaz.
WfEventAudit	Constituye una interfaz común para grabar los eventos de flujo de trabajo. Varios subtipos de esta interfaz son definidos para grabar el cambio de estado de un objeto de flujo de trabajo; datos de proceso asociados, y cambios en la asignación de recursos a las actividades. En la sección 2.11 del documento del OMG se detalla esta interfaz.

Tabla 1-1 Resumen de interfaces especificacion OMG

Es importante recalcar que si somos estrictos en el uso de los términos definidos por la WfMC:

- Un WfProcessMgr servirá para manejo de un PROCESO.
- Un WfProcess servirá para manejo de un CASO.
- Un WfActivity servirá para el manejo de un ITEM DE TRABAJO.

Esto puede prestarse para confusión, pues un WfProcess en realidad no maneja un proceso, sino un CASO, es decir una instancia de un proceso.

Además se debe notar que NO existe ninguna interfaz para:

- Realización de las tareas de conexión / desconexión
- Acceso a la lista de procesos (WfProcessMgr)
- Acceso a la lista de recursos (WfResource).

Esto la especificación lo deja abierto para que lo implemente el desarrollador a su criterio.

CAPITULO 2. ANÁLISIS Y DISEÑO

2.1 DEFINICIÓN DE REQUERIMIENTOS

De acuerdo a una clasificación hecha por Ian Sommerville[15], existen tres tipos de requerimientos de software:

- Requerimientos funcionales
- Requerimientos no funcionales
- Requerimientos de dominio

2.1.1 REQUERIMIENTOS FUNCIONALES

Básicamente, el componente deberá permitir la interacción de aplicaciones cliente con un WFMS.

Debe proveer al desarrollador de software las siguientes funciones:

- Funciones de conexión
    - Conexión (con la respectiva validación de usuario)
    - Desconexión
- Funciones de control de proceso y actividad
    - Obtener la lista de definiciones de procesos
    - Obtener la lista de instancias de procesos o actividades
    - Crear una instancia de un proceso

[15] SOMERVILLE, Ian, “Ingeniería de Software”, Editorial Addison Wesley, 6ta Edición, Inglaterra, 2002.

    - Controlar un proceso: iniciar, suspender, continuar, terminar, abortar.
    - Controlar una actividad: suspender, continuar, terminar, abortar.
    - Obtener y definir datos relativos al proceso o actividad
    - Asignar o reasignar recursos a las actividades
- Funciones de estado de proceso o actividad
    - Obtener estado actual del proceso o actividad
    - Cambiar el estado de un proceso o actividad
    - Obtener el historial de estados del proceso o actividad
- Funciones de auditoria
    - Recibir eventos de cambio de estado
    - Recibir eventos de cambio en los datos de un proceso o actividad
    - Recibir eventos de creación de procesos
    - Recibir eventos de asignación de recursos

El componente NO proveerá:

- Funciones de definición de procesos
- Funciones de administración y monitoreo
- Funciones de interacción con otros WFMSs.

Debido a que dichas funciones no corresponden a las interfaces 2 y 3 de un WFMS, sino a las interfaces 1, 4 y 5 respectivamente. [16]

Esta tesis generará un componente que pueda conectarse a las interfaces 2 y 3 de un WFMS.

[16] Revisar en el capítulo 1: “Modelo de WFMS propuesto por la WfMC” para más información.

2.1.2 REQUERIMIENTOS NO FUNCIONALES

Facilidad de incorporación en una aplicación cliente: El componente debe ser fácilmente integrable a aplicaciones cliente. Debe consistir en una librería que se incorpore en las aplicaciones cliente.

2.1.3 REQUERIMIENTOS DE DOMINIO

2.1.3.1 Compatibilidad con el documento propuesto por el OMG

El componente a desarrollar debe incorporar las interfaces descritas por el OMG dentro de su “Especificación para una Facilidad de Gestión de Flujos de Trabajo”, pero realizará las adaptaciones necesarias para su implementación en lenguaje java (lado del servidor) y C# (lado del cliente).

La versión más reciente publicada a este momento es la versión 1.2. La cual se encuentra disponible en Internet.

2.1.3.2 Mecanismo de comunicación CORBA

El documento del OMG define a CORBA como la arquitectura de comunicación que debe usarse para la implementación de la interfaz 2 y 3 de un WFMS. Sin embargo, algunos desarrolladores han implementado la especificación del OMG sobre otras tecnologías, como son RMI.

Se realizó un análisis de las posibles tecnologías disponibles para interacción de componentes. Este análisis se encuentra en el ANEXO 8.

Luego de analizar opciones como RMI, XML-RPC, SOAP y CORBA, entre otras, dado que se desea la interoperabilidad de aplicaciones escritas en diferentes lenguajes, se ha visto como la mejor opción el uso de CORBA como mecanismo para interacción de los componentes de software.

2.1.3.3 Lenguajes de programación

Dentro de los WFMSs analizados (ver ANEXO 6), todos fueron escritos en java, y ya proveen componentes para interacción de clientes JAVA-JAVA.

Se deberá escribir el componente de tal manera que permita el acceso de clientes escritos en .NET a WFMS que funcionan bajo tecnología JAVA.

Por tanto, parte del componente será escrito en JAVA, para que acceda a un WFMS escrito en dicho lenguaje, y parte será escrita en C# de .NET, para ser incluida en los clientes que deseen acceder al servidor WFMS.

Se usará el framework de .NET más actual (2.0) y el lenguaje más ampliamente aceptado para desarrollo en .NET (C#).

2.2 ANÁLISIS

2.2.1 VISIÓN GENERAL DEL COMPONENTE

El componente debe ser una librería que provea todas las funciones necesarias para la interacción con el servidor de flujos de trabajo, y debe ser además fácil de implementarlo en el proceso de desarrollo.

El componente a desarrollarse deberá comunicarse remotamente al servidor de flujo de trabajo.

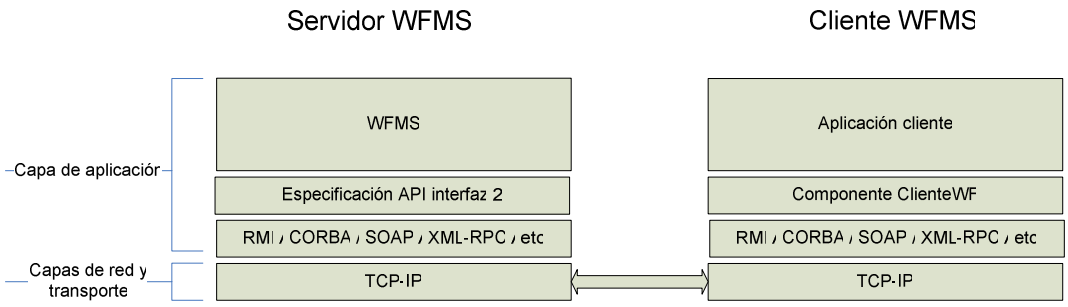


Figura 2-1 Capas de comunicación de aplicaciones cliente de flujo de trabajo con el WFMS

Dentro de las capas de red y transporte, el estándar a utilizarse es sin duda, TCP-IP, dado que los métodos para comunicación remota funcionan ampliamente sobre este protocolo.

Dentro de la capa de aplicación, para la comunicación de los componentes, se utilizará la tecnología CORBA.

El desarrollo principal se encontrará en el lado del servidor, donde el sub-componente se encargará de escuchar las peticiones hechas, las procesará accediendo al WFMS y posteriormente creará los objetos correspondientes y enviará la respuesta al cliente.

Con el fin de que el componente pueda ser ampliado para compatibilidad con otros WFMS, en el lado del servidor se deberá dividir el componente en dos partes, una general para todos los WFMS, y una específica, que se adaptará al WFMS que se desee usar.

Tenemos por tanto para el lado del servidor, dos sub-componentes, que llamaremos:

- CORBAWfServer: Servicio CORBA de flujo de trabajo.
- JMSWfServer: Interfaz entre CORBAWfServer y un WFMS específico

En el lado del cliente se desarrollará una librería para que el programador acceda a las funciones provistas por el WFMS mediante CORBA. Este sub-componente se llamará:

- CORBAWorkfowClient: Librería para acceso al servicio CORBA.

En el capítulo referente al diseño se ampliará la definición de las partes del componente.

2.2.2 WFMS A UTILIZARSE

El componente por si mismo no puede realizar ninguna tarea, sino que sirve de interfaz para comunicación con un WFMS. Por tanto, se hace necesario el uso de un WFMS para la realización del desarrollo y las pruebas correspondientes del componente.

Se realizó un análisis de los WFMS que pudieran utilizarse para el desarrollo de la presente tesis. Este análisis se encuentra en el ANEXO 6.

El WFMS escogido es WfmOpen, debido a que es un WFMS que cumple con las siguientes características:

- Desarrollado bajo estándares de la WfMC y OMG
- Código abierto, disponible gratuitamente en Internet.

WfmOpen provee ya interfaces de cliente, basado en RMI, es decir solo para aplicaciones cliente desarrolladas en JAVA.

El componente complementa al sistema, pues permitirá que clientes desarrollados en otros lenguajes puedan también acceder al WFMS.

2.2.3 ESCENARIOS DE USO DEL COMPONENTE

Todo componente o aplicación que desee interactuar con el WFMS recibe el nombre de “aplicaciones cliente de flujo de trabajo”.

Para interactuar con el WFMS, los componentes o aplicaciones “clientes de flujo de trabajo” utilizarán el componente a desarrollarse en la presente tesis.

El componente a desarrollarse consta de tres partes, que se describirán con más detalle en el diseño.

- CORBAWorkfowClient: Librería para acceso al servicio CORBA.
- CORBAWfServer: Servicio CORBA de flujo de trabajo.
- JMSWfServer: Interfaz entre CORBAWfServer y un WFMS específico

Podemos dividir a los “clientes de flujo de trabajo” en tres tipos:

- Aplicaciones de escritorio.
- Aplicaciones web.
- Componentes o librerías.

Por tanto se pueden tener los siguientes escenarios:

2.2.3.1 Escenario 1: Aplicaciones de escritorio.

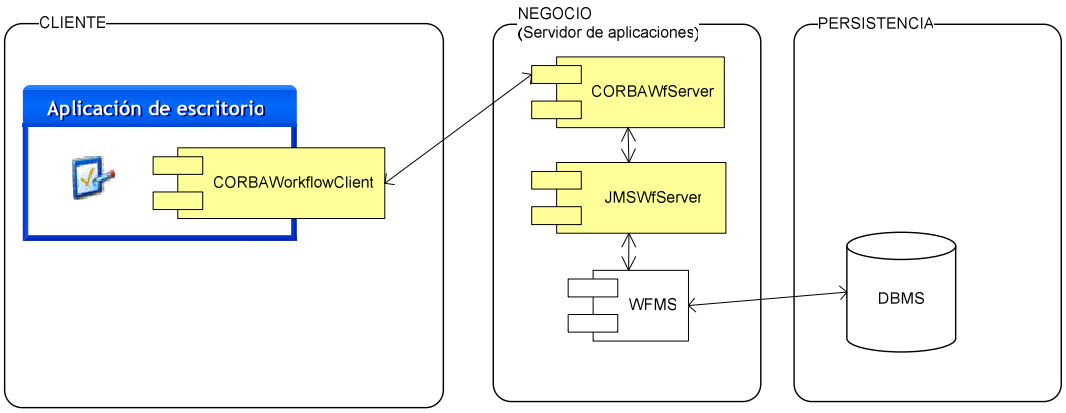


Figura 2-2 Escenarios de uso del componente: Aplicaciones de escritorio

En este escenario, la librería CORBAWorkflowClient debe ejecutarse en el lado del cliente, mientras que los sub-componentes CORBAWfServer y JMSWfServer deben ejecutarse en el servidor de aplicaciones, junto con el WFMS.

2.2.3.2 Escenario 2: Aplicaciones web

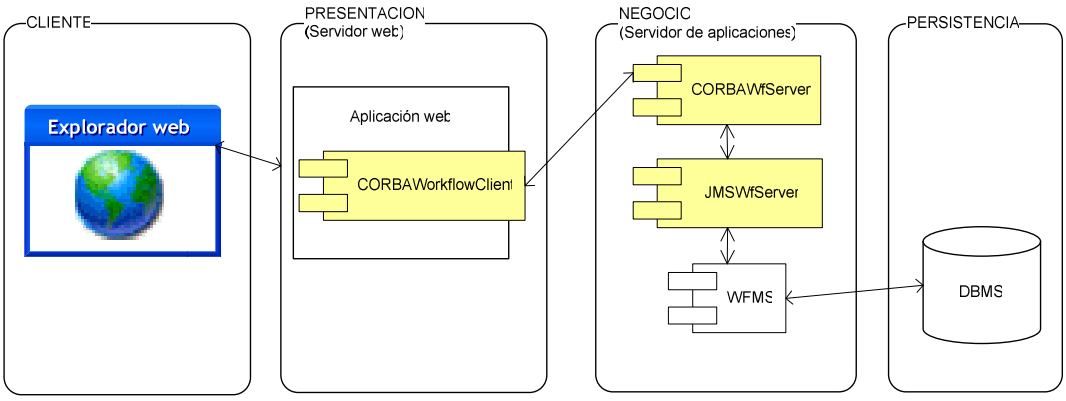


Figura 2-3 Escenarios de uso del componente: Aplicaciones web

En este escenario, en el cliente no es necesario sino un explorador web. La librería CORBAWorkflowClient debe encontrarse en el servidor web, mientras que

los sub-componentes CORBAWfServer y JMSWfServer deben ejecutarse en el servidor de aplicaciones, junto con el WFMS.

2.2.3.3 Escenario 3: Componentes o librerías.

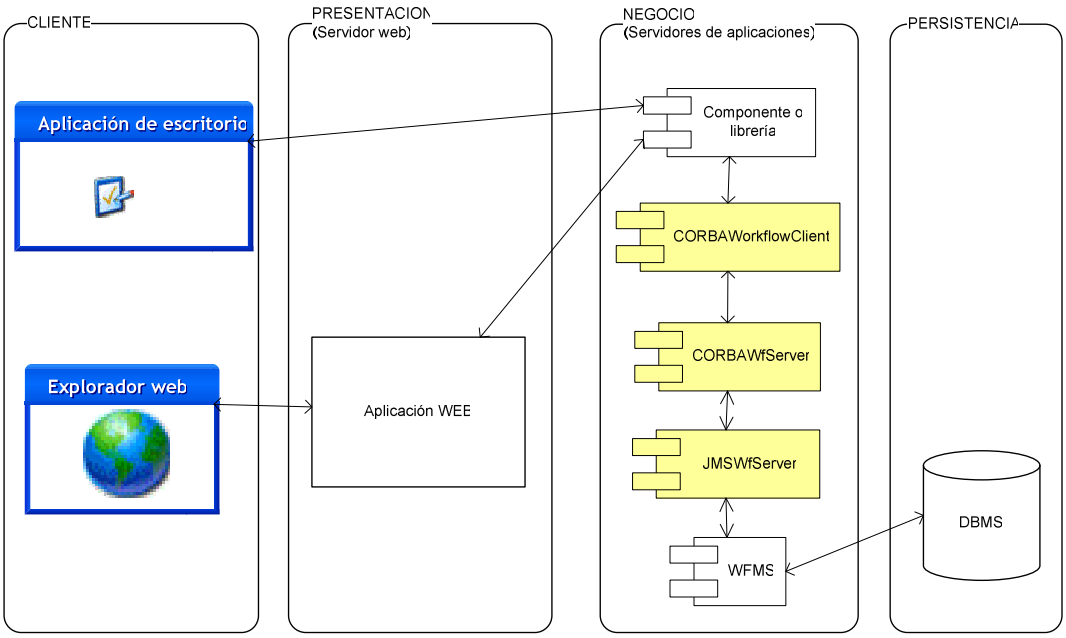


Figura 2-4 Escenarios de uso del componente: Componentes o librerías

En este escenario todos los sub-componentes se encuentran en la capa de NEGOCIO.

Obviamente puede haber variaciones o mezclas de los escenarios presentados anteriormente.

2.3 DISEÑO

2.3.1 DISEÑO DE PARTES DEL COMPONENTE

El componente consta de tres partes:

- CORBAWorkfowClient: Librería para acceso al servicio CORBA.
- CORBAWfServer: Servicio CORBA de flujo de trabajo.

- JMSWfServer: Interfaz entre CORBAWfServer y un WFMS específico

Todo “cliente de flujo de trabajo” deberá utilizar una librería DLL[17] que se desarrollará para .NET (CORBAWorkflowClient.DLL), la misma que permitirá el acceso a las funciones provistas por el servidor CORBA de flujos de trabajo.

En el alcance de la presente tesis solamente se desarrollará la librería para clientes .NET. Si el cliente está desarrollado en otro lenguaje diferente a .NET necesitará desarrollar una librería que defina un cliente CORBA para el componente CORBAWfServer, como se muestra en el siguiente gráfico:

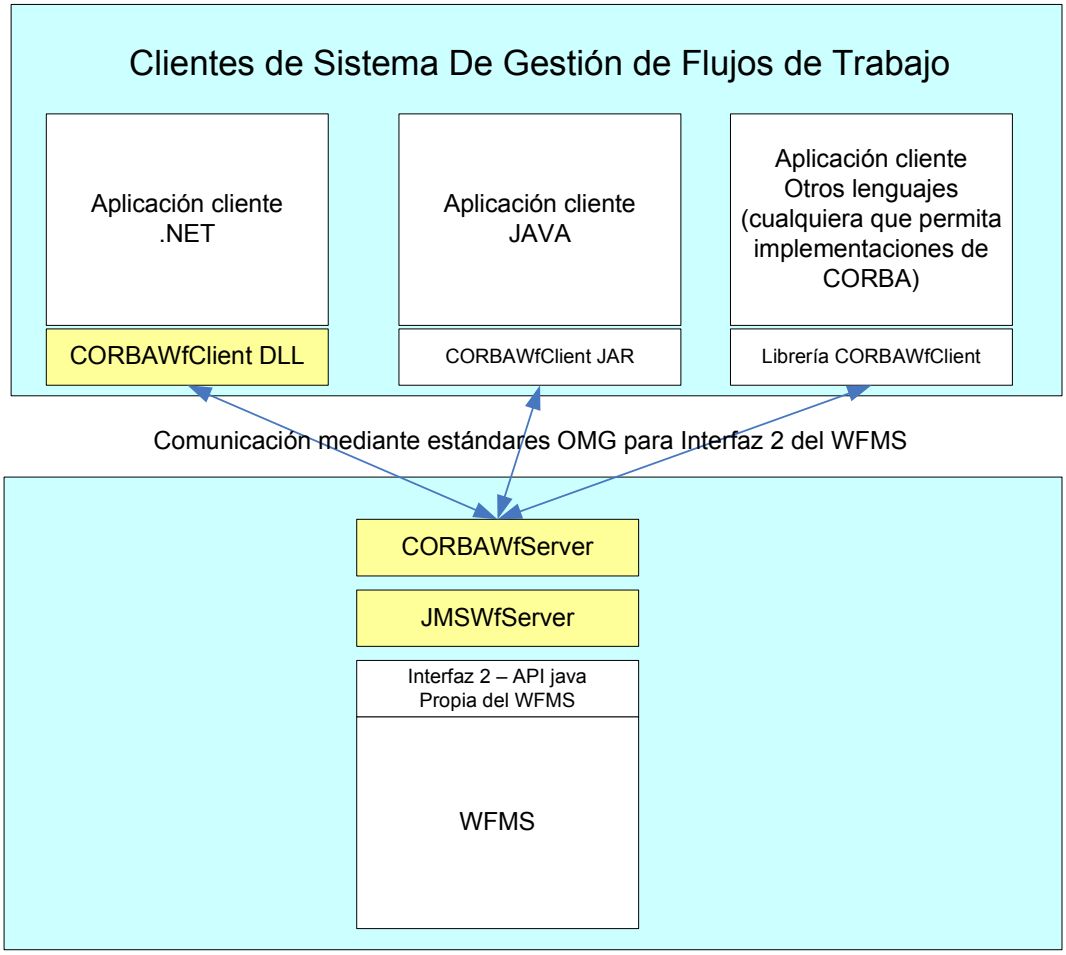


Figura 2-5 Interacción del WFMS con aplicaciones cliente de diferentes lenguajes

[17] DLL: Dynamic Link Library (Librería de enlace dinámico).

En la parte del servidor, dividimos el componente en dos partes claramente identificables, la primera (CORBAWfServer) será totalmente independiente del WFMS, mientras que la segunda será una capa que se adapta a las funciones propias de cada WFMS para el que se quiera implementar el componente.

Se hizo esta división en la parte del servidor con el fin de crear un componente que no se encuentre atado a un WFMS específico, sino que pueda posteriormente ser ampliado para la compatibilidad con otros WFMSs, siendo esto totalmente transparente a los clientes.

La mensajería de los dos componentes de lado del servidor, se realizará utilizando JMS (Java Messaging System), un estándar de J2EE ampliamente difundido.

2.3.2 DISEÑO DE SUBCOMPONENTES

2.3.2.1 CORBAWfServer

Este sub-componente corresponderá la parte central del desarrollo a realizarse.

Debe proveer, para el lado del cliente, la interfaz propuesta por el OMG mediante el documento de especificaciones para interfaz de cliente. [18]

2.3.2.1.1 Interfaces

Las interfaces descritas por este documento se basan en un modelo de objetos, el cual se muestra a continuación.

---

[18] OMG, “Workflow Management Facility Specification v.1.2”

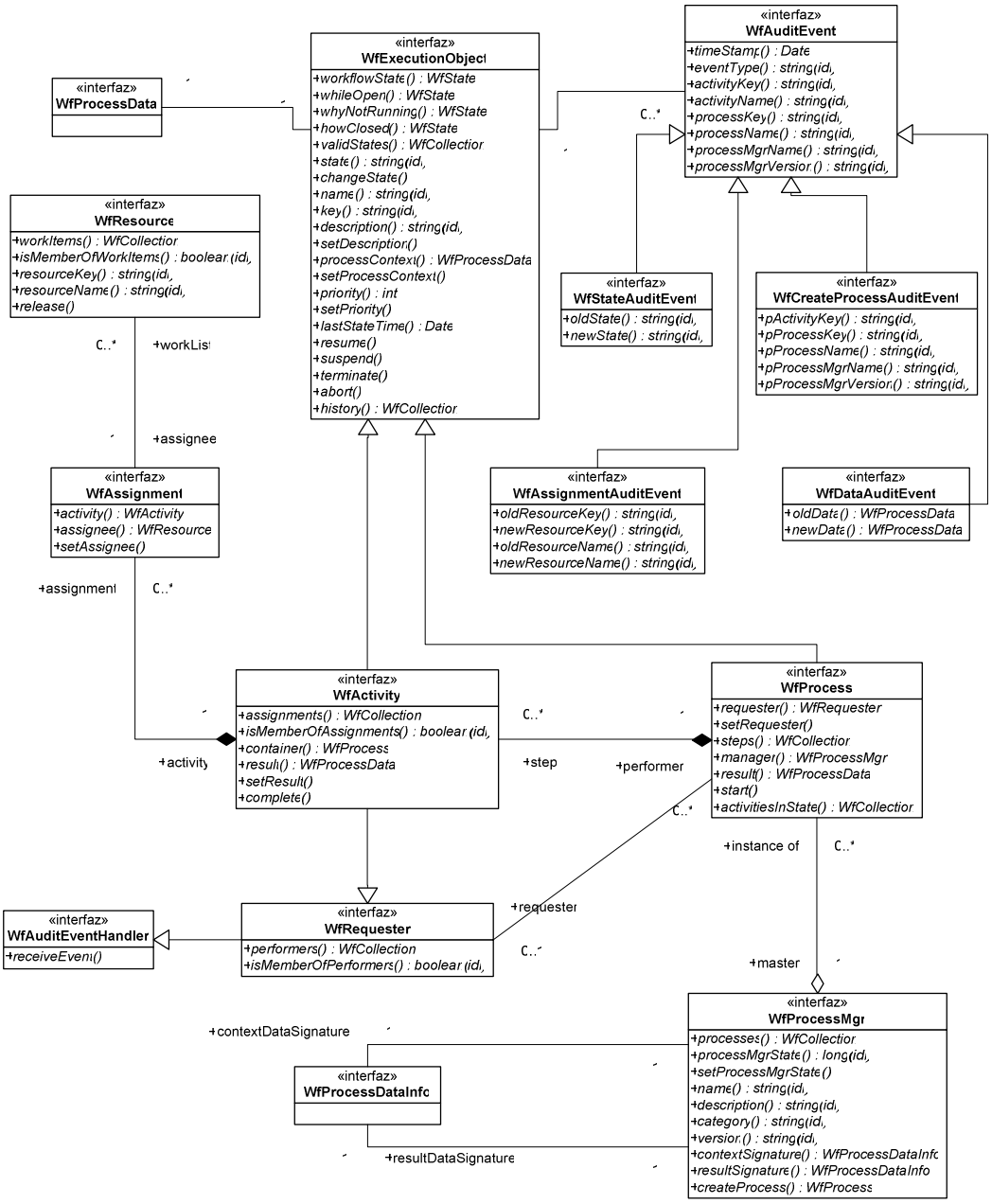


Figura 2-6 Modelo de objetos para gestión de flujos de trabajo

FUENTE: OMG, “Workflow Management Facility Specification v.1.2”

Las interfaces centrales del modelo son WfProcess y WfActivity, las cuales permiten el acceso a las funciones de control de procesos y actividades, respectivamente.

La interfaz WfProcessMgr constituye la “fábrica” de procesos, mediante la cual el usuario puede acceder a un proceso y crear nuevas instancias.

Las interfaces descritas en el documento del OMG son:

- WfExecutionObject (abstracta)
- WfProcess
- WfActivity
- WfProcessMgr
- WfRequester
- WfActivity
- WfAssignment
- WfResource
- WfAuditEvent
- WfStateAuditEvent
- WfDataAuditEvent
- WfCreateProcessAuditEvent
- WfAssignmentAuditEvent

Se hizo en el capítulo 1 una descripción breve de las interfaces, su descripción completa puede ser leída en la especificación propuesta por el OMG. [19]

Adicionalmente, se hace referencia en el documento del OMG a dos interfaces que son necesarias para el almacenamiento de “variables de flujo de trabajo” [20]. Dichas interfaces son:

- WfProcessData
- WfProcessDataInfo

Estas dos interfaces contienen una colección de pares nombre-valor, la primera (WfProcessData) almacenará un arreglo de pares:

---

[19] OMG, “Workflow Management Facility Specification v.1.2”, disponible en internet, www.omg.org

[20] Las “variables de flujo de trabajo” son aquellas que son utilizadas dentro de un proceso para el enrutamiento del flujo de trabajo. Por ejemplo: una variable de flujo de trabajo podría ser: “tipo de cliente”, de acuerdo a esta variable el flujo de trabajo de un proceso puede seguir diferentes rutas, dando por ejemplo preferencia a cierto tipo de clientes. Las “variables de flujo de trabajo” también pueden ser identificadores que diferencian a cada caso o datos necesarios para la ejecución de los ítems de trabajo de cada caso.

1. nombre de variable. (Cadena)
2. el valor correspondiente. (Objeto cualquiera)

La segunda (WfProcessDataInfo) contiene en cambio las definiciones de tipo de datos de las variables de flujo de trabajo, almacenará un arreglo de pares:

1. nombre de variable (Cadena)
2. tipo de variable. (Cadena)

Aunque se aparta un poco de la definición hecha por el OMG, se diseñaron estas interfaces de tal manera que las funciones que proveen sean parecidas a las funciones provistas por la interfaz *Map*[21] del lenguaje java.

Dado que WfProcessData y WfProcessDataInfo son muy parecidas, se decidió crear una clase base para ambas:

- WfMap

WfMap se parece mucho a la interfaz Map de lenguaje java. WfMap hace referencia además a dos interfaces necesarias:

- WfCollection
- WfNameValue

La interfaz WfCollection servirá para almacenar colecciones (listas) de cualquier tipo de objetos. La interfaz WfNameValue en cambio servirá para almacenar un solo par nombre-valor.

[21] La interfaz Map de java es utilizada para la implementación de clases que almacenan colecciones de pares nombre-valor.

Con el fin de hacer que la clase WfCollection tenga funcionalidades parecidas a la interfaz Collection del lenguaje java, se vio la necesidad además de crear las siguientes interfaces:

- WfListItem
- WfIterator

La interfaz WfListItem será utilizada para la creación de ítems de una lista, contiene en sí dos campos: un valor, que puede ser cualquier objeto, y una referencia al siguiente elemento.

La interfaz WfIterator en cambio tendrá las mismas funciones que la interfaz Iterator del lenguaje java, y podrá ser utilizada para la iteración de elementos de una lista simple.

No se utilizaron las interfaces Map, Collection, Iterator, propias de java porque, por razones técnicas, para la implementación del servicio CORBA, se necesita que las interfaces del componente extiendan la interfaz Remote. Fue por tanto necesaria la creación de las interfaces mencionadas (WfMap, WfCollection, WfIterator).

El diagrama de objetos de WfProcessData, WfProcessDataInfo, WfMap, WfCollection, WfNameValue, WfListItem y WfIterator es el siguiente:

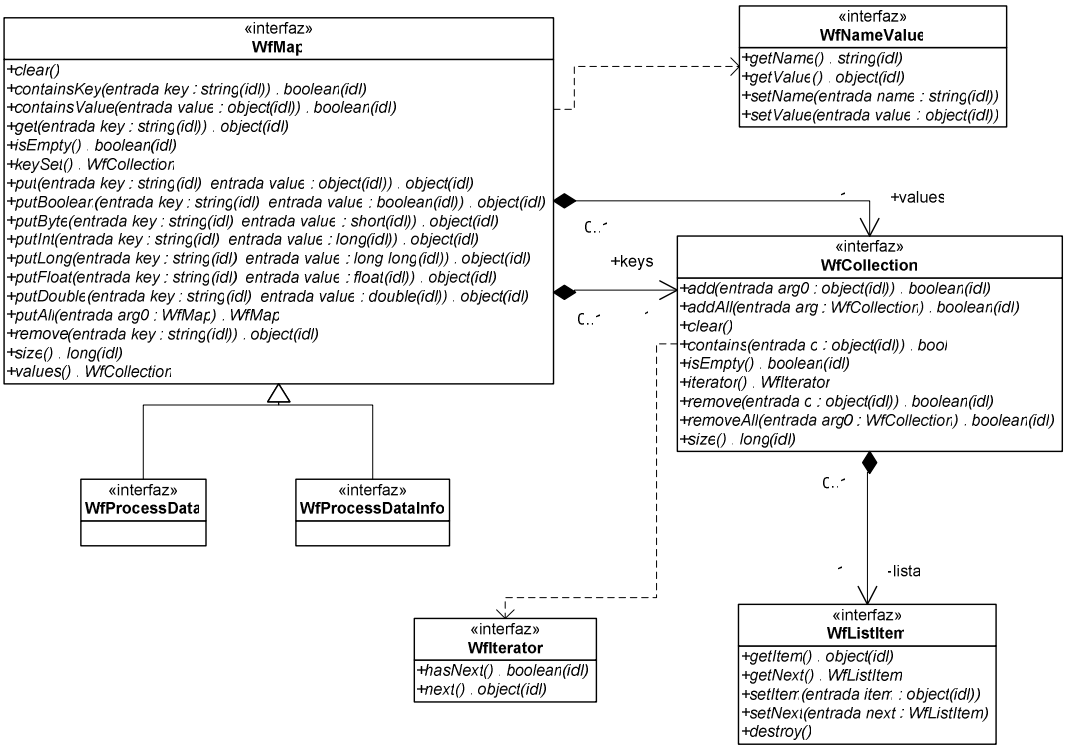


Figura 2-7 Diagrama de objetos para interfaces WfProcessData, WfProcessDataInfo, WfMap, WfNameValue, WfCollection, WfIterator y WfListItem

Es necesaria además la creación de interfaces que permitan el acceso a los objetos base. Estas interfaces el OMG no las incluyó dentro del alcance de su especificación, sino que las deja a criterio del desarrollador. Las interfaces diseñadas para este propósito son:

- WfProcessDirectory
- WfService
- WfServiceFactory

El directorio de procesos (WfProcessDirectory) permitirá la búsqueda de procesos (WfProcessMgr), de instancias de procesos (WfProcess) y de ítems de trabajo (WfActivity) que se encuentran funcionando en el WFMS.

La interfaz WfService proveerá las funciones necesarias para acceder al directorio de procesos (WfProcessDirectory) además de otras para obtención de referencias a recursos (WfResource).

La interfaz WfServiceFactory servirá para instanciar un nuevo servicio de flujo de trabajo WfService. Esta es la primera interfaz que la aplicación cliente verá cuando se conecte al servidor CORBA de workflow. Mediante una llamada a la función newWorkflowService(), con los parámetros correspondientes (nombre de usuario y contraseña), la aplicación cliente obtendrá una referencia a la interfaz WfService, desde la cual puede acceder a todas las demás interfaces.

El diagrama de objetos de WfProcessDirectory, WfService y WfServiceFactory se muestra a continuación:

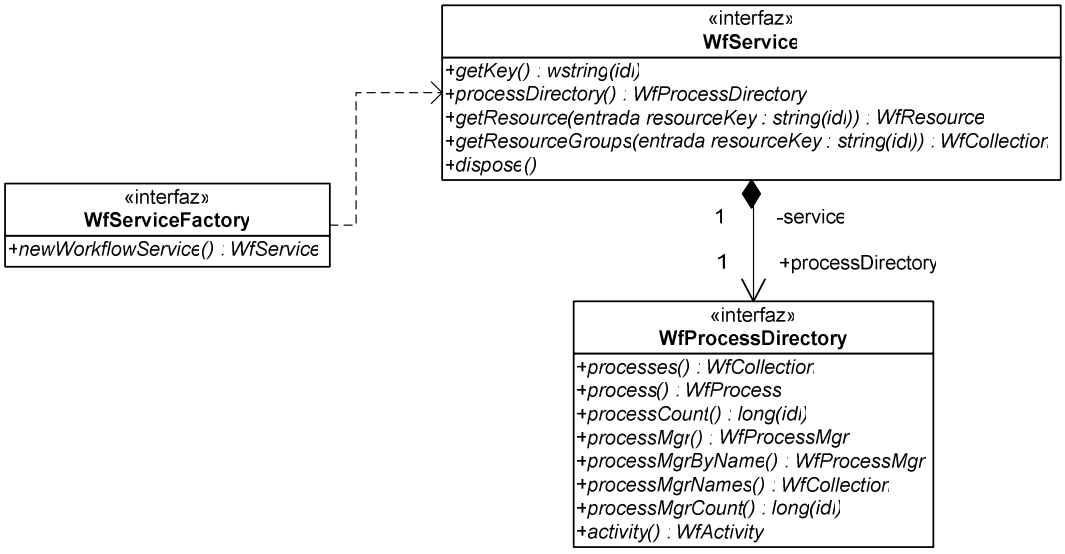


Figura 2-8 Diagrama de objetos de las interfaces WfServiceFactory, WfService y WfProcessDirectory

2.3.2.1.2 Clases que implementan las interfaces mencionadas

Para implementar las interfaces mencionadas anteriormente, se crearán las clases con anteponiendo la letra “C” de CORBA al nombre de la interfaz

correspondiente. Las clases a crearse son objetos que implementan los servicios de las interfaces.

Para crear los objetos de implementación de servicios, estos pueden heredar de la clase *javax.rmi.PortableRemoteObject* o pueden implementar una interfaz remota y luego usar el método *exportObject* para registrarse como objetos de servidor. Por simplicidad escogimos que los objetos extenderán la clase javax.rmi.PortableRemoteObject.

Para realizar el acceso mediante CORBA por parte del cliente a la clase principal de este sub-componente (CWfServiceFactory), esta clase debe encontrarse registrada en un ORB (Object Request Broker).

Según un manual de CORBA de Sun Microsystems: "El ORB es el servicio distribuido que implementa la petición al objeto remoto. Localiza el objeto remoto en la red, comunica la petición al objeto, espera los resultados y cuando están disponibles comunica los resultados de vuelta al cliente." [22]

Se creará la clase ejecutable CorbaWfServer, la cual realizará el registro de la clase CWfServiceFactory en un ORB de acuerdo a parámetros especificados en un archivo de texto. La clase CorbaWfServer se detalla a continuación:

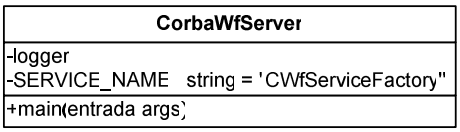


Figura 2-9 Diagrama de objetos de clase CorbaWfServer

Dentro de los atributos de esta clase,

---

[22] http://java.sun.com/developer/onlineTraining/corba/corba.html Sitio web de aprendizaje de CORBA para Java, de Sun Microsystems. Último acceso: Septiembre 2007.

- logger: Variable estática usada para gestión de logs.
- SERVICE_NAME: Constante que indica el nombre con el que se registrará el servicio.

Dado que la mensajería con el WFMS es manejada mediante JMS, es necesaria además la creación de una clase que permita la interacción con JMS. Esta clase se la llamará CJMSComm, la cual proveerá la función:

- sendMessage: Esta función recibirá como parámetro un mapa (colección de pares nombre-valor) con los valores que deben ser enviados a la cola JMS y luego de enviar la petición quedará esperando la respuesta del JMSWfServer y devolverá dicha respuesta.

2.3.2.1.3 Funcionamiento del subcomponente CORBAWfServer

Para escuchar peticiones mediante CORBA, la clase correspondiente (CWfServiceFactory) debe ser registrada en un ORB (Object Request Broker), para esto utilizamos la interfaz *Context* de java.

La clase CWfServer es una clase ejecutable, cuyo trabajo es instanciar una clase CWfServiceFactory, la cual escuchará las peticiones de los clientes, esta clase luego de ser instanciada es registrada mediante la interfaz *Context* en un ORB.

Los datos de cómo acceder al ORB son parametrizados en un archivo, como se verá más adelante.

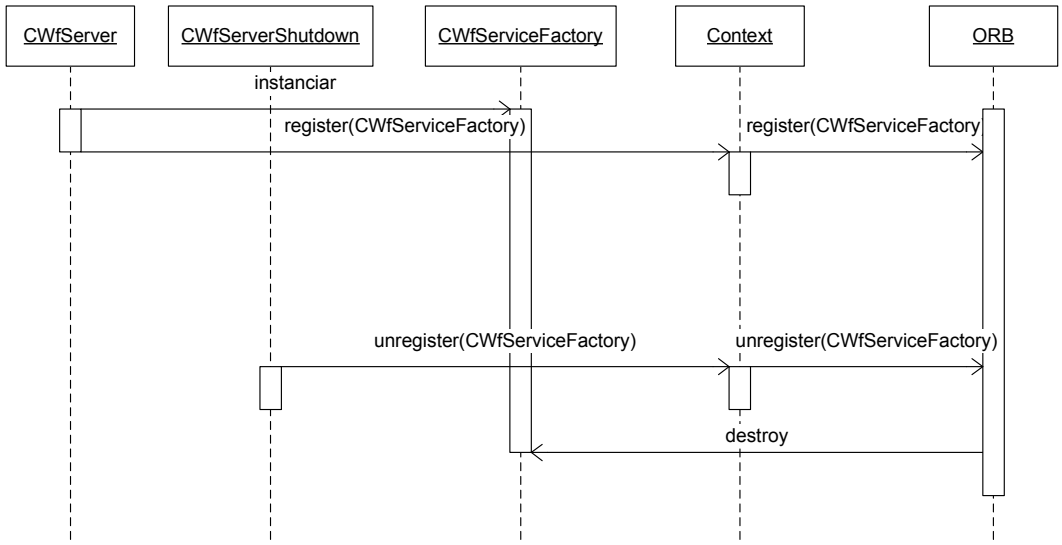


Figura 2-10 Diagrama de secuencia para creación y destrucción de servicio CWfServiceFactory

Luego de que la clase CWfServiceFactory haya sido instanciada y registrada en el ORB, está lista para escuchar las peticiones de los clientes mediante CORBA.

Para finalizar el servicio, es necesario ejecutar la clase CWfServerShutdown, la cual, mediante la interfaz *Context* envía un mensaje al ORB para que elimine el registro y destruya la instancia clase CWfServiceFactory que fue anteriormente creada.

2.3.2.1.4 Manejo de errores

Con el fin de realizar un manejo de los errores, el sistema deberá además lanzar las siguientes excepciones:

Nombre	**Descripción**
AlreadyRunningException	Esta excepción es lanzanda cuando se intenta iniciar un WfProcess que ya se está ejecutando.

AlreadySuspendedException	Esta excepción es lanzada por intentar suspender un WfExecutionObject que ya se encuentra suspendido.
CannotChangeRequesterException	Esta excepción es lanzada cuando un cambio de WfRequester es solicitado, pero no puede ser realizado.
CannotCompleteException	Esta excepción es lanzada al intentar completar la ejecución de un WfExecutionObject cuando no puede ser completado.
CannotConnectException	Esta excepción es lanzada cuando no se puede realizar la conexión al motor de workflow.
CannotResumeException	Esta excepción es lanzada por una operación en un WfExecutionObject que intenta continuar (Resume) un objeto que no está en una condición apropiada para hacerlo.
CannotStartException	Esta excepción es lanzada al intentar iniciar un WfProcess que no puede ser iniciado
CannotStopException	Esta excepción es lanzada por una operación en un WfExecutionObject que intenta parar un objeto que no se encuentra en la condición apropiada.
CannotSuspendException	Esta excepción es lanzada por una operación en un WfExecutionObject que intenta suspender un objeto que no se encuentra en la condición apropiada.
HistoryNotAvailableException	Esta excepción es lanzada por una petición del historial de eventos de auditoría de un WfExecutionObject cuando el historial no esta disponible.
InvalidControlOperationException	Esta excepción es lanzada por una operación sobre un WfExecutionObject que intenta realizar una operación de control inválida en dicho objeto.
InvalidDataException	Esta excepción es lanzada al intentar actualizar el contexto del resultado de un WfExecutionObject con datos que no corresponden (inválidos) para dicho objeto.

InvalidPerformerException	Esta excepción es lanzada al intentar enviar un WfAuditEvent una señal para WfRequester que no fue creado por un WfProcesses asociado con el WfRequester.
InvalidPriorityException	Esta excepción es lanzada al intentar asignar un valor de prioridad inválido a un WfExecutionObject.
InvalidRequesterException	Esta excepción es lanzada cuando un WfRequester es identificado que no puede ser un "parent" de instancias del proceso modelo. Cuando un WfRequester es rechazado, la aplicación puede decidir no registrar el WfRequester con el WfProcess.
InvalidResourceException	Esta excepción es lanzada al intentar asignar o remover un recurso inválido.
InvalidStateException	Esta excepción es lanzada al intentar cambiar el estado de un WfExecutionObject a un estado que no se ha definido para dicho objeto.
NotAssignedException	Esta excepción es lanzada al intentar liberar un WfResource de una asignación a la cual no se le ha asociado.
NotEnabledException	Esta excepción es lanzada al intentar crear un WfProcess usando un WfProcessMgr que está desactivado
NotRunningException	Esta excepción es lanzada por una operación en un WfExecutionObject que intenta realizar una operación de control en un objeto que necesita estar en estado "running" (corriendo), pero no lo está.
NotSuspendedException	Esta excepción es lanzada por una operación en un WfExecutionObject que intenta realizar una operación de control en un objeto que necesita estar en estado "suspended" (suspendido), pero no lo está.
RequesterRequiredException	Esta excepción es lanzada cuando un

	WfRequester valido es requerido por la definición del proceso, pero no ha sido suplido.
ResultNotAvailableException	Esta excepción es lanzada cuando el resultado pedido de un WfExecutionObject no está disponible (todavía).
SourceNotAvailableException	Esta excepción es lanzada al realizar una petición de la fuente de un WfAuditEvent cuando la fuente no está disponible.
TransitionNotAllowedException	Esta excepción es lanzada al intentar realizar una transición de estado inválida de un WfExecutionObject.
UpdateNotAllowedException	Esta excepción es lanzada cuando no es permitido actualizar el contexto de un proceso.

Tabla 2-1 Excepciones manejadas por el componente

2.3.2.2 JMSWfServer

Este sub-componente será el encargado de escuchar las peticiones del CORBAWfServer y crear los componentes de respuesta correspondientes, tomando los datos del WFMS.

Para el desarrollo de esta tesis, este sub-componente accede al WFMS WfmOpen, el cual fue escogido luego de realizar un análisis de varios WFMSs de código abierto. Dicho análisis se encuentra en el ANEXO 6.

Las clases implementadas por este componente serán:

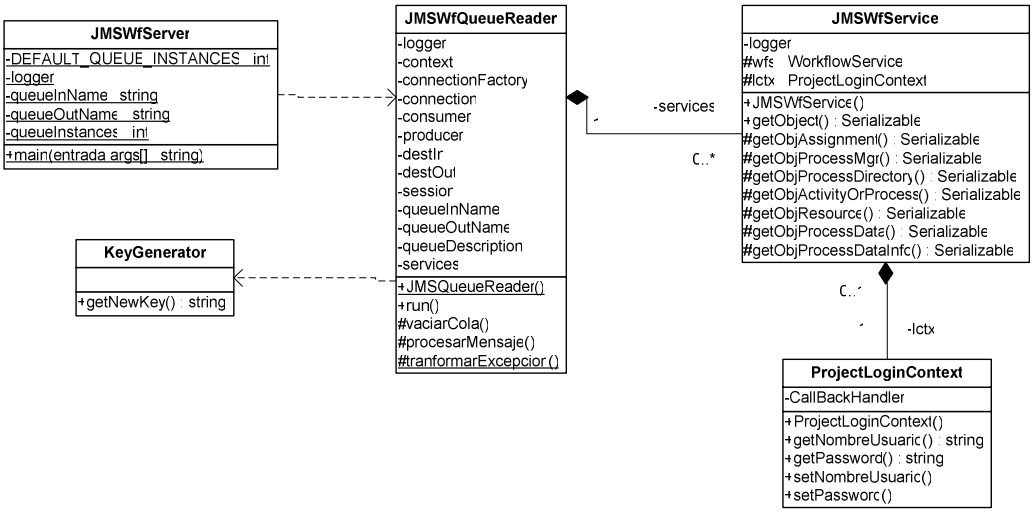


Figura 2-11 Diagrama de objetos de subcomponente JMSWfServer

Por defecto, la clase JMSWfServer buscará las colas con los nombres:

- JMSServiceQueue01
- JMSAnswerQueue01

Sobre las cuales la clase JMSWfQueueReader recibirá las peticiones y enviará los objetos de respuesta, respectivamente.

Puede ser que en ambientes de mayor concurrencia sea necesario más de un servidor JMSQueueReader, por tanto, se pueden configurar creando colas llamadas JMSServiceQueue02, 03, 04, …. y JMSAnswerQueue02, 03, 04…

La clase JMSWfServer creará tantas clases JMSQueueReader como colas JMSServiceQueue se encuentren configuradas en el servidor de aplicaciones.

Cada JMSQueueReader debe funcionar en un hilo separado, por tanto extenderá la clase *Thread* de java. Luego de ser instanciado, se debe llamar a la función run() para que dicho hilo comience su funcionamiento.

Cuando una instancia de JMSQueueReader comienza su funcionamiento lo primero que hace es vaciar las colas JMS correspondientes.

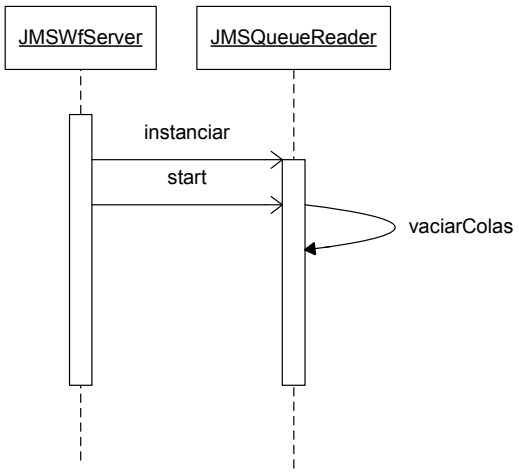


Figura 2-12 Diagrama de secuencias, instanciación de JMSQueueReader

La clase JMSQueueReader escuchará los datos escritos en una cola JMS mediante la interfaz *MessageConsumer* de java. Cada vez que reciba un mensaje llamará a la función *procesarMensaje* para procesar dicha petición.

Si la petición es de una nueva conexión de un cliente, la clase JMSQueueReader creará una nueva instancia de la clase JMSWfService, la cual se encargará de atender todas las peticiones posteriores de dicho cliente. Cada JMSService se guardará en un mapa nombre-valor, con una clave que identificará de manera única al JMSService, para que las siguientes peticiones del cliente accedan a la instancia correspondiente. Para la generación de la clave única se utiliza la clase KeyGenerator.

Cada instancia de JMSWfService está relacionada con un ProjectLoginContext, la cual maneja las funciones de acceso y autorización del cliente al servidor de aplicaciones y al WFMS.

Dado que el JMSWfService desarrollado fue diseñado para comunicación con WfmOpen, es necesaria la creación de una variable WorkflowService para la interacción con dicho WFMS.

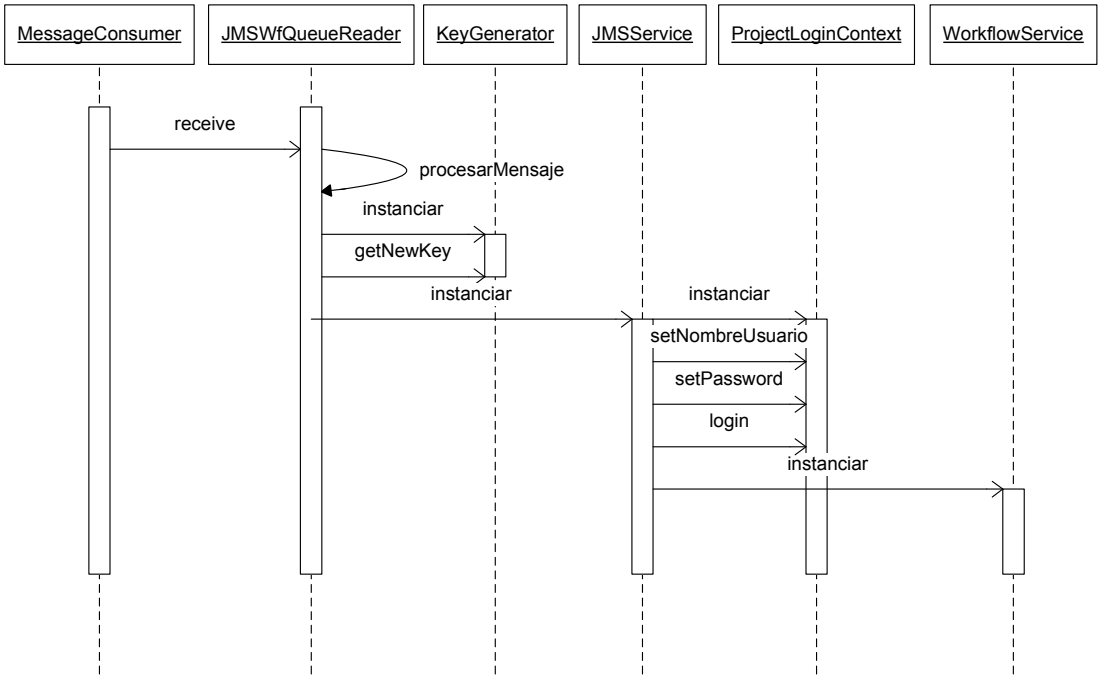


Figura 2-13Diagrama de secuencias, creación de servicio JMSWfService

Para las siguientes peticiones, JMSWfQueueReader buscará en el mapa de servicios, de acuerdo a una clave única, la instancia de JMSService correspondiente, y realizará una llamada a la función getObject para luego devolver la respuesta de la petición mediante la interfaz *MessageProducer* de java.

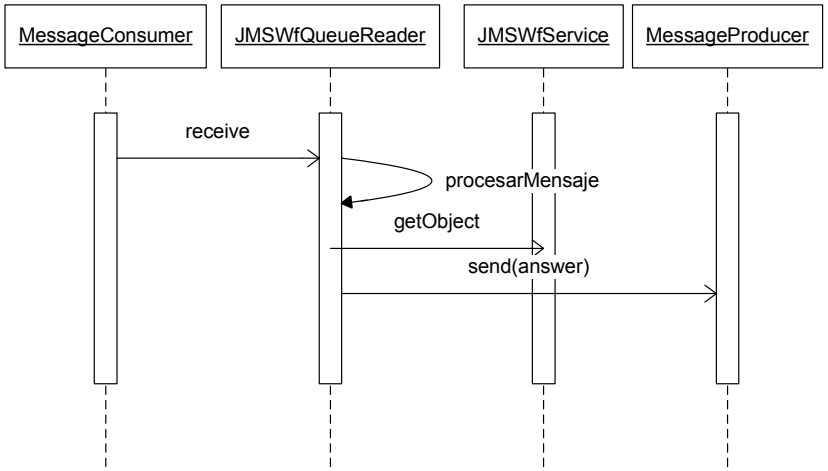


Figura 2-14 Diagrama de secuencias JMSWfService

2.3.2.3 CORBAWfClient

El sub-componente de lado del cliente será desarrollado en tecnología .NET.

Dado que Microsoft, no provee una implementación de un ORB para implementación de aplicaciones distribuidas .NET con CORBA, se hace necesaria la utilización de un ORB externo.

Se escogió el ORB llamado *IIOP.NET*, el cual se encuentra ampliamente difundido y utilizado, y que además, presenta la ventaja de ser gratuito de utilizar, al ser de código abierto.

La librería de cliente implementará una sola clase que depende de las librerías IIOPChannel y WfInterfaces.

En la librería IIOPChannel es en sí el ORB IIOP para .NET, mientras que la librería WfInterfaces es una librería que contiene los STUBS [23] generados para acceso al servicio CORBAWfServer.

La librería WfInterfaces debe ser generada en base a un archivo IDL [24] mediante la herramienta “IDL to CLS Compiler” provista por IIOP.NET. El archivo IDL debe ser generado en base a las interfaces escritas en java, usando la herramienta rmic.

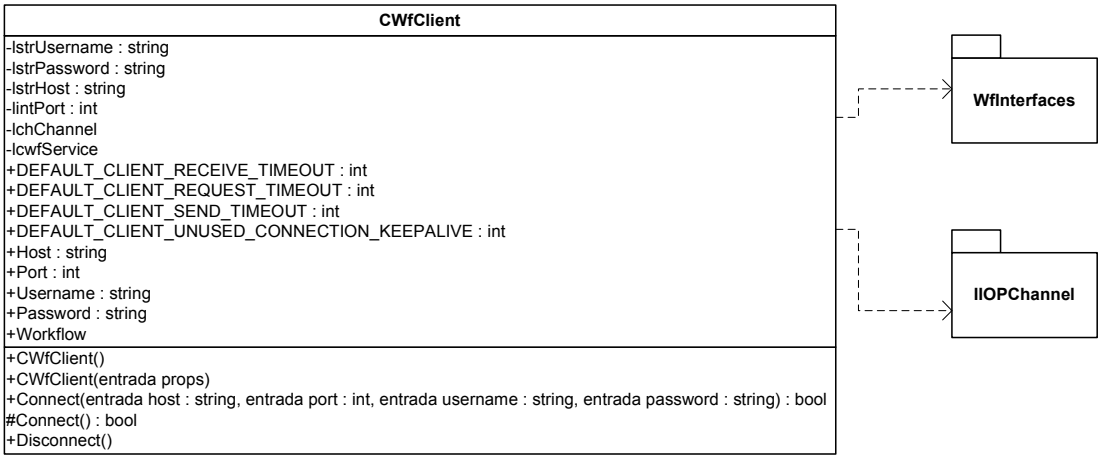


Figura 2-15 Diagrama de clases CORBAWorkflowClient

2.3.3 INTERACCIÓN DE LOS SUB-COMPONENTES

En el siguiente diagrama se muestra un ejemplo de interacción de los sub-componentes, para la función de conexión.

---

[23] En vez de hacer una copia de la implementación del objeto en el cliente que lo recibe, CORBA pasa un **stub** para un objeto remoto. El stub actúa como la representación local del objeto remoto y básicamente, para el llamador, es la referencia remota. Más información de la arquitectura CORBA en www.omg.org

[24] El lenguaje de definición de interfaces (IDL) es el estándar para definición de interfaces CORBA.

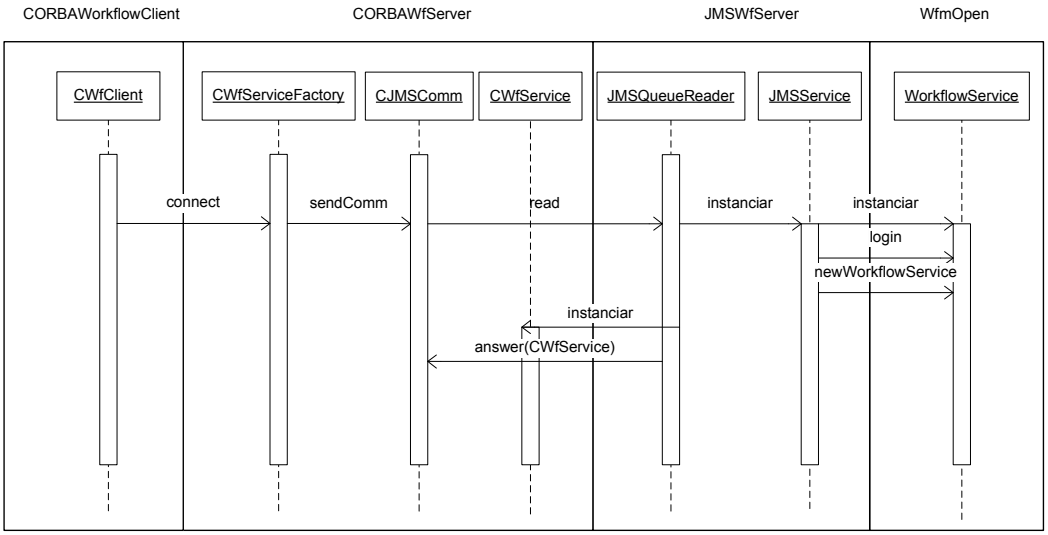


Figura 2-16 Diagrama de interacción para función de conexión

La librería cliente, hace una llamada a la función connect de la clase CWfServiceFactory que debe encontrarse registrada en un ORB.

Cuando la clase CWfServiceFactory fue instanciada, debe haber creado una clase CJMSComm, la cual le permitirá comunicarse con el JMSWfService mediante una cola JMS.

Cuando el cliente hace la llamada a la función *connect*, la clase CWfServiceFactory, ésta llama a la función sendComm de la clase CJMSComm. Esto hace que la clase JMSQueueReader, que estaba escuchando a dicha cola lea el mensaje y lo procese.

Dado que es un mensaje de creación de un nuevo servicio, crea una nueva instancia de la clase JMSService y se comunica con el WFMS para realizar las tareas correspondientes. Luego la misma clase JMSQueueReader crea una nueva instancia de la clase CWfService, la cual enviará como respuesta a la clase CJMSComm, la cual a su vez devolverá la misma respuesta a la clase CWfServiceFactory y finalmente ésta a la clase CORBAWorkflowClient.

Dado que la clase CORBAWorkflowClient ha obtenido una referencia al objecto CWfService instanciado, mediante esta referencia podrá acceder a todas las funciones de dicha clase, dentro del contexto correspondiente.

2.3.4 LOGGING

El componente deberá guardar el log de las actividades que realiza, para los siguientes niveles:

- FATAL: Un error irrecuperable, que termina la aplicación
- ERROR: Cuando se da un error en la aplicación, pero puede seguir funcionando.
- WARN: Mensajes de advertencia
- INFO: Mensajes informativos
- DEBUG: Mensajes utilizados durante el desarrollo y para verificar el correcto funcionamiento del componente.

La librería que se utilizará para realizar el logging es LOG4J, una librería de código abierto para java.

CAPITULO 3. ELABORACIÓN DEL COMPONENTE

3.1 CONSTRUCCIÓN

3.1.1 HERRAMIENTAS Y MARCO DE TRABAJO

3.1.1.1 Plataformas de ejecución

3.1.1.1.1 JDK

Para el funcionamiento del servidor de aplicaciones, el WFMS y los subcomponentes JMSWorkflowServer y CORBAWorkflowServer es necesario un Entorno de Ejecución Java ó JRE (Java Runtime Environment).

JDK significa Kit de Desarrollo Java (Java Development Kit). El JDK contiene las librerías necesarias para la ejecución de aplicaciones java (JRE) además de la documentación correspondiente y varias herramientas que se utilizarán en el presente desarrollo.

La versión de JDK utilizada es la J2SDK [25] 1.4.2_01, la cual es compatible con el servidor de aplicaciones y WFMS que se utilizarán.

3.1.1.1.2 .NET Framework

Para el funcionamiento del cliente, dado que la librería se desarrollará para .NET, es necesario el framework .NET para su funcionamiento.

Se utilizará la última versión de framework .NET disponible al momento de iniciar el desarrollo de la presente tesis: la versión 2.0.50727.42.

---

[25] J2SDK: “Java 2 Standard Development Kit”

3.1.1.2 Sistema de Gestión de Flujo de trabajo: WfmOpen

Como se mencionó en capítulos anteriores, luego de realizar un análisis entre varios WFMSs disponibles en el mercado actual, se escogió WfmOpen para el desarrollo de la presente tesis.

WfmOpen debe instalarse sobre un servidor de aplicaciones que cumpla con las especificaciones J2EE para un "contenedor de EJBs [26]". Por su alta aceptación en el mercado, por ser de uso gratuito, así como por su facilidad de configuración, se escogió Jboss como servidor de aplicaciones.

WfmOpen necesita también conectarse mediante JDBC [27] a un DBMS que provea los controladores apropiados. Se escogió MySQL como DBMS sobre el cual funcionaría el WFMS.

La versión de WfmOpen a utilizarse es la versión estable más reciente: 1.3.4., la cual viene con el instalador versión 2.1ea5. [28]

3.1.1.3 Servidor de aplicaciones: JBoss

Usando términos de la especificación J2EE, JBoss constituye un "contenedor de EJBs" y "contenedor de servlets", que cuenta además con muchas funcionalidades y librerías adicionales propias.

---

[26] EJB: "Enterprise Java Bean", no se hace una descripción completa de conceptos referentes a J2EE debido a que no forman parte del alcance de la presente tesis.

[27] JDBC: "Java Database Connection", estándar para conexión de aplicaciones java a bases de datos

[28] Estos instaladores pueden obtenerse del Internet, del sitio web del proyecto en SourceForge. La dirección es http://sourceforge.net/projects/wfmopen

El servidor de aplicaciones será utilizado de “contenedor” del WFMS, así como de los subcomponentes CORBAWfServer y JMSWfServer.

El hecho que el desarrollo se realice sobre un servidor de aplicaciones Jboss, no significa que el componente va a estar atado para su funcionamiento al mismo. Así como el WFMS, el componente podrá ser trasladado a otros servidores de aplicaciones sin ningún problema.

La versión de JBoss utilizada es la 1.4.0.5, que constituye la última versión estable disponible al momento de iniciar la presente tesis.

3.1.1.4 Servidor de bases de datos: MySQL

El componente en sí no necesita una capa de persistencia, por tanto no necesita un DBMS para su funcionamiento. Sin embargo, el WFMS sí lo necesita, por lo que se utilizará MySql para dicho propósito.

La creación de la base de datos y el llenado de datos iniciales son realizadas automáticamente por el instalador del WFMS.

No se hace un detalle de la estructura de la base de datos pues de ninguna manera es necesario que el componente acceda directamente a la base de datos, sino a través del WFMS. Además, entender cómo maneja los datos el WFMS no es necesario para el desarrollo del componente.

La versión de MySql utilizada es 5.0. [29]

[29] Disponible en Internet www.mysql.org

3.1.1.5 ORBs

Como se mencionó en capítulos anteriores, dentro de la arquitectura CORBA son necesarios los ORBs (object request brokers) [30]. Los ORBs escogidos son:

3.1.1.5.1 ORB java: Jacorb

JacORB viene incorporado por defecto en la instalación completa de JBoss. El es ORB de código abierto de mayor utilización para aplicaciones JAVA. La versión utilizada es la provista por Jboss 4.0.5.

3.1.1.5.2 ORB .NET: IIOP.NET

No existe ningún ORB provisto por Microsoft [31], se hizo por tanto necesario la utilización de un ORB de código abierto.

Se encontraron 2 opciones maduras de ORBs de código abierto para .NET:

- CorbaRemoting [32]
- IIOP .NET [33]

En pruebas realizadas se pudo utilizar ambos para comunicación con aplicaciones java mediante CORBA, sin embargo IIOP.NET presentó mayores facilidades por las herramientas que ofrece, logrando que el desarrollo sea más rápido.

---

[30] "El ORB es el servicio distribuido que implementa la petición al objeto remoto. Localiza el objeto remoto en la red, comunica la petición al objeto, espera los resultados y cuando están disponibles comunica los resultados de vuelta al cliente."
http://java.sun.com/developer/onlineTraining/corba/corba.html Sitio web de aprendizaje de CORBA para Java, de Sun Microsystems. Último acceso: Septiembre 2007.

[31] Por políticas de Microsoft, el apoyo directo de esta empresa para comunicación de aplicaciones de diferentes lenguajes es hacia la tecnología SOAP y no a CORBA.

[32] Disponible en Internet http://sourceforge.net/projects/corbaremoting

[33] Disponible en Internet, http://sourceforge.net/projects/iiopnet

Se escogió IIOP.NET como ORB para .NET. La versión a utilizarse es la 1.0.9.

3.1.2 HERRAMIENTAS DE PROGRAMACIÓN

3.1.2.1 Eclipse

Eclipse es un IDE (Entorno de Desarrollo Integrado por sus siglas en inglés: Integrated Development Environment) utilizado principalmente para desarrollo de aplicaciones java.

Se escogió este IDE por su facilidad de uso y su versatilidad. La versión utilizada es la 3.2.0.

3.1.2.2 ANT

ANT es una herramienta para automatizar las tareas repetitivas del proceso de construcción de una aplicación.

ANT puede realizar tareas como creación y borrado de directorios, copia de archivos, compilación de archivos, así como generación de código o llamada a herramientas externas.

ANT se utiliza junto con un archivo build.xml, en el cual se definen las tareas que esta herramienta debe ejecutar.

Se definió el siguiente archivo build.xml para su uso con ANT:

```
<project name="CORBAWorkflowService" default="run" basedir=".">
  <property file="build.properties" />
  <property name="src" value="src" />
  <property name="lib" value="lib" />
  <property name="dist.dir" value="dist" />
  <property name="build.dir" value="build" />
  <property name="build.src" value="${build.dir}/gensrc" />
  <property name="build.classes" value="${build.dir}/classes" />
  <property name="build.doc" value="${build.dir}/doc" />
```

```
  <property name="build.metainf.dir" value="${build.dir}/metainf" />
   <property  name="project.packageFolder"  value="edu/epn/workflow/CORBAWorkflowService"
       />
  <property name="project.package" value="edu.epn.workflow.CORBAWorkflowService" />
  <property name="project.service" value="WfServiceFactory" />
  <property name="project.netClientFolder" value="../CORBAWorkflowClient" />
  <property name="build.idl" value="../common/idl" />

 <property                                                                    name="IDLtoCLScompiler"
       value="../common/tools/IDLtoCLScompiler/IDLToCLSCompiler.exe" />

 <path id="build.classpath">
       <fileset dir="lib">
         <include name="*.jar" />
</fileset>
       <dirset dir="${build.classes}" />
 </path>

 <target name="clean">
  <delete dir="${build.dir}" />
  <delete dir="${build.idl}" />
  <delete dir="${dist.dir}" />
 </target>

 <target name="init">
  <mkdir dir="${build.dir}" />
  <mkdir dir="${dist.dir}" />
  <mkdir dir="${build.classes}" />
  <mkdir dir="${build.doc}" />
  <mkdir dir="${build.idl}" />
  <mkdir dir="${build.metainf.dir}" />
  <mkdir dir="${build.src}" />
 </target>

<target name="compile" depends="init">
 <javac srcdir="${src}" destdir="${build.classes}" classpathref="build.classpath">
  </javac>
          <rmic          base="${build.classes}"          includes="**/CWf*.class"          iiop="true"
       classpathref="build.classpath" />
 </target>

 <target name="compileNetDll" depends="init,compile">
    <rmic   base="${build.idl}"   classname="${project.package}.C${project.service}"   idl="true"
       classpathref="build.classpath" />
 <exec executable="${IDLtoCLScompiler}">
   <arg  line="o  ${project.netClientFolder}/bin  lib:${build.idl}  lib:${lib}  ${project.service}
       ${build.idl}/${project.packageFolder}/${project.service}.idl" />
  </exec>
</target>

<target name="run" depends="compile">
 <java          classname="${project.package}.CorbaWfServer"          classpathref="build.classpath"
       fork="true">
  <jvmarg value="Djava.naming.factory.initial=org.jnp.interfaces.NamingContextFactory" />
  <jvmarg value="Djava.naming.provider.url=jnp://localhost:1099" />
  <jvmarg value="Djava.security.auth.login.config=auth.conf" />
  </java>
  </target>

 <target name="javadoc" depends="init">
 <javadoc    sourcepath="${src}"    destdir="${build.doc}"    packagenames="edu.epn.workflow.*"
       windowtitle="Workflow Escuela Politécnica Nacional">
 <doctitle>
 <![CDATA[
<h1>Componente Workflow</h1>
  ]]>
  </doctitle>
 <bottom>
```

```
  <![CDATA[
 <i>Copyright &#169; 2000 EPN. Todos los derechos reservados.</i>
]]>
</bottom>
roup title="Group 1 Packages" packages="edu.epn.workflow.CORBAWorkflowService" />
</javadoc>
</target>
</Project>
```

Figura 3-1 Archivo build.xml utilizado para la construcción con Ant

Dentro del archivo anterior, creamos los siguientes objetivos (targets):

1. **init**: Crea los directorios necesarios.
2. **clean**: borra las carpetas de las clases compiladas y el código generado.
3. **compile**: compila las clases java y genera STUBS y SKELETONS[34] para JAVA mediante RMIC.
4. **compileNETDLL**: genera los archivos IDL y en base a estos genera los STUBS y SKELETONS en una librería WfInterfaces para uso con .NET mediante herramienta "IDL to CLS compiler"
5. **run**: Ejecuta la clase CORBAWfServer en el servidor de aplicaciones JBoss.
6. **javadoc**: Genera la documentación del componente mediante el uso de la herramienta javadoc.

Las herramientas de generación de código que se mencionan (RMIC y IDL to CLS Compiler" se detallan más adelante.

3.1.2.3 Microsoft Visual Studio .NET

Para la programación de la librería de cliente, así como la programación de la aplicación de demostración se utilizará Microsoft Visual Studio .NET 2005.

---

[34] STUB: Es un objeto que reside en el cliente, el cual lo ve como un objeto cualquiera, pero en sí consiste en una referencia al objeto remoto, para de esta manera evitar "tener una copia" del objeto remoto en el cliente. SKELETON: Objeto que reside en el servidor, es el objeto que recibe las peticiones hechas por el cliente y envía la respuesta correspondiente. Si se desea más información se debería consultar en un manual de CORBA, revisar: www.omg.org ó http://java.sun.com/developer/onlineTraining/corba/corba.html (Último acceso Septiembre 2007) para más información.

El lenguaje usado es C#.

3.1.3 HERRAMIENTAS DE GENERACIÓN DE CÓDIGO

3.1.3.1 RMIC

RMIC es una herramienta que viene incluida el kit de desarrollo de java (J2SDK), y es utilizada, en la presente tesis, para:

1. Generación de STUBS y SKELETONS java en base a interfaces escritas en lenguaje java.
2. Generación de archivos IDL (Interface Definition Language) en base a las mismas interfaces java.

3.1.3.2 IDL to CLS Compiler

Es una herramienta que viene con IIOP.NET, es utilizada para generar los STUBS y SKELETONS para .NET en base a un archivo IDL. Luego de la generación automática del código, la herramienta genera una librería DLL compatible con .NET la cual será utilizada para acceso las interfaces que fueron programadas en java.

3.1.3.3 Together Workflow Editor Community Edition

Esta herramienta no se utilizará para el desarrollo del componente en sí, sino para el desarrollo del aplicativo de demostración.

Consiste en una herramienta gráfica de código abierto que permite el diseño gráfico de procesos y genera el código XPDL[35] que se ingresa en el WFMS.

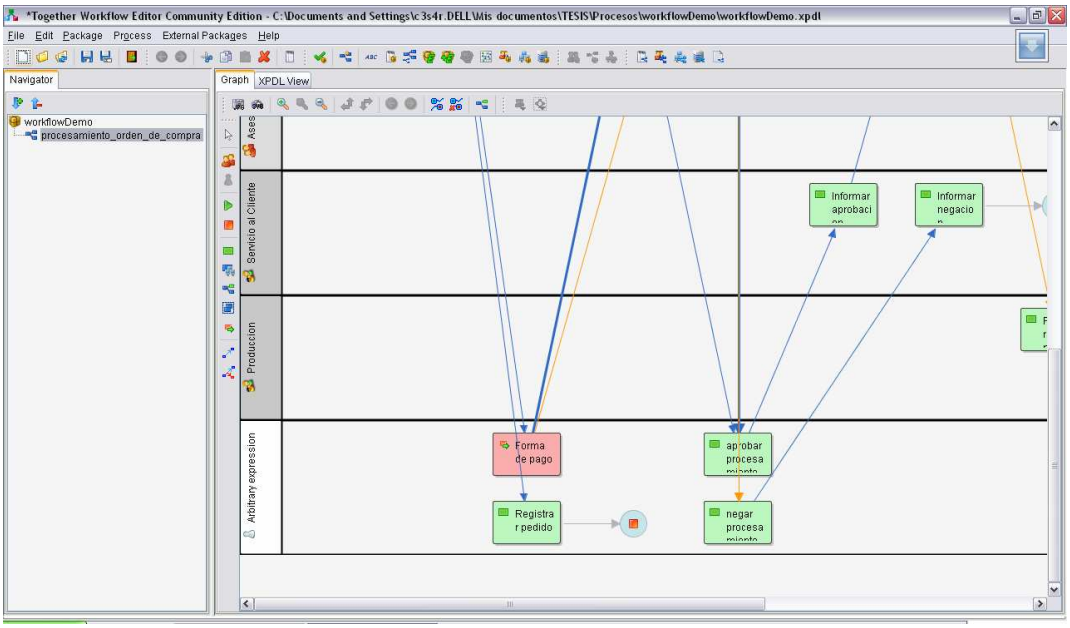


Figura 3-2 Herramienta Together Workflow Editor

3.1.4 ESTÁNDARES DE CODIFICACIÓN

3.1.4.1 Estándares para Java

Se seguirán los siguientes parámetros:

- Nombres de clases e interfaces:
  - Empezarán con mayúscula.
  - Si el nombre de la clase consta de varias palabras, cada palabra empezará con mayúscula, no se usan caracteres de espacio entre las palabras.

[35] XPDL: Lenguaje de Definición de Procesos, “XML Process Definition Language”, es el estándar de la WfMC para la definición de procesos de un WFMS (interfaz 1). Más información en el capítulo 1, en el tema “Modelo de referencia de WFMS propuesto por la WfMC”

- Constantes
    - Utilizarán los modificadores **static final** para indicar que son una constante.
    - Las constantes necesarias para las clases serán en mayúscula y si el nombre consta de varias letras utilizarán el símbolo "_" para separación entre palabras.

- Variables
    - Comenzarán con letra minúscula
    - Si el nombre consta de varias palabras, cada palabra empezará con mayúscula, excepto la primera.

- Excepciones:
    - En el nombre tendrán el sufijo **Exception**.

3.1.4.2 Estándares para C#

Parte del código de C# será generado por la herramienta "IDL to CLS Compiler", sobre la cual no tenemos un control sobre los estándares utilizados.

Sin embargo, para el resto de la codificación del componente se seguirán los siguientes parámetros:

- Constantes:
    - Las constantes necesarias para las clases serán en mayúscula y si el nombre consta de varias letras utlizarán el símbolo "_" para separación entre palabras.
    - Se usará el modificador *const* solo para constantes naturales (como por ejemplo el número de PI) y se usará el modificador *readonly* para el resto de constantes.
- Interfaces
    - El nombre será las primeras letras de cada palabra en mayúscula.

    - Tendrán el prefijo la letra “i” mayúscula.
- Variables privadas
    - Tendrán el prefijo la letra “l” minúscula seguido del tipo de dato.
- Atributos
    - Su nombre tendrá el sufijo “Attribute”
- Excepciones
    - Su nombre tendrá el sufijo “Exception”

Se intentará seguir las recomendaciones de “mejores prácticas” de Microsoft para la codificación, entre ellas podemos mencionar:

- Evitar programar varias clases en un mismo archivo. (Crear un archivo para cada clase).
- Evitar tener múltiples namespaces en un mismo archivo.
- Nombrar los métodos usando verbos.
- Usar nombres descriptivos de las variables y no nombres abreviados.

3.2 PRUEBAS

3.2.1 MARCO DE REALIZACIÓN PRUEBAS

Para la realización de las pruebas, dado que el componente por si mismo no provee ninguna funcionalidad, es necesaria la creación de todo un ambiente sobre el cual funcionará el componente.

Es por esto que las pruebas de sistema se realizarán a la par del desarrollo de la aplicación de demostración, la misma que servirá para comprobar el correcto funcionamiento del componente.

El ambiente de pruebas, con todos sus componentes se describe en el siguiente gráfico:

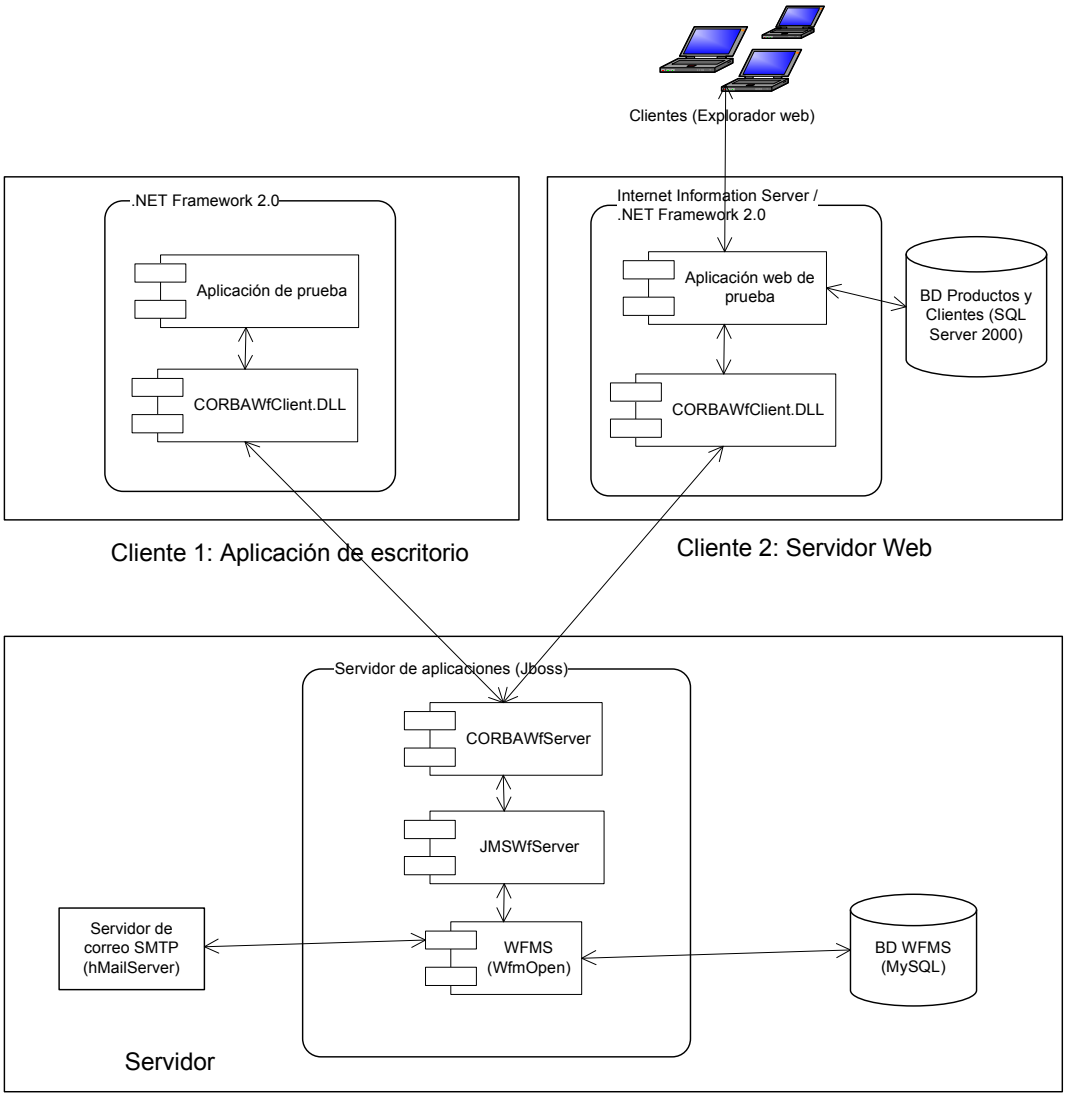


Figura 3-3 Marco para la realización de pruebas del componente

Como se verá más adelante, para la demostración del uso se crearán dos aplicaciones cliente:

- Una aplicación de escritorio
- Una aplicación web

Se necesita además en el lado del servidor:

- Los dos subcomponentes desarrollados:
  - CORBAWfServer

  - JMSWfServer
- El WFMS instalado y configurado apropiadamente (WfmOpen)
- Un servidor de base de datos para el WFMS (MySQL)
- Un servidor de correo saliente SMTP (hMailServer)

El detalle de las pruebas realizadas se mostrará luego del desarrollo de las aplicaciones de demostración (capítulo 4).

3.3 MANUAL DE UTILIZACIÓN

3.3.1 INSTALACIÓN Y CONFIGURACIÓN DE SERVIDOR DE APLICACIONES

Los subcomponentes CORBAWfServer y JMSWorkflowServer funcionan sobre un "contenedor de EJBs" dentro de la especificación J2EE.

La instalación y configuración de los servidores de aplicaciones de los diferentes proveedores es diferente, y por tanto a manera de ejemplo se mostrará la instalación sobre solo uno de ellos: JBoss.

Como ejemplo, otros servidores de aplicaciones que también se podrían usar son: IBM Websphere, Apache Gerónimo, Weblogic, entre otros.

3.3.1.1 Instalación de JBoss

El instalador de JBoss a usarse será la versión 1.2.0.G.A. [36], el mismo que contiene la versión JBoss 4.0.5, sobre la cual se desarrolló esta tesis.

[36] Disponible en Internet www.jboss.org

Al abrir este instalador, luego de escoger el idioma de instalación y aceptar los términos de licencia, escogemos el directorio de instalación. Se recomienda que el directorio de instalación NO contenga espacios ni caracteres especiales.

Luego de esto tenemos la siguiente interfaz, donde escogemos la configuración de instalación.

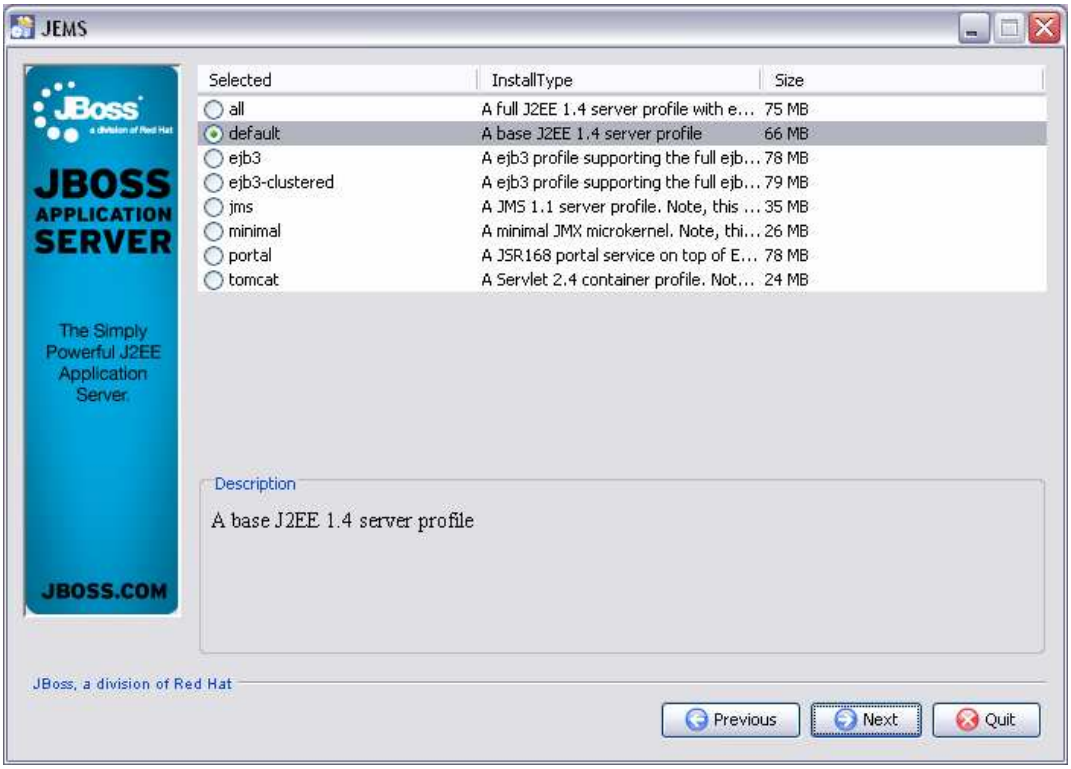


Figura 3-4 Pantalla de opciones de instalación de JBoss

La configuración completa asegura que todos los componentes necesarios serán instalados, esta es la instalación recomendada para ambientes de prueba, pero no para ambientes de producción, debido a que instala componentes innecesarios.

La otra opción es instalar la configuración por defecto o alguna de las otras configuraciones, pero debemos asegurarnos que los componentes necesarios se encuentren seleccionados en la siguiente pantalla. Entre los componentes indispensables tenemos:

- Librerías para CORBA (IIOP)
- Librerías para JMS

Se puede revisar en la documentación de JBoss más información acerca de estas librerías. [37]

3.3.1.2 Instalación de colas JMS

Dado que los dos sub-componentes del lado del servidor se comunican mediante JMS, es necesario configurar en el servidor de aplicaciones varias colas para comunicación entre los mismos.

Se deberán crear al menos las siguientes colas:

- JMSServiceQueue01
- JMSAnswerQueue01

Puede ser que en ambientes de mayor concurrencia sea necesario más de un servidor JMSWorkflowService, por tanto, se deberán crear colas con los nombres:

- JMSServiceQueueXX
- JMSAnswerQueueXX

Donde XX es un número entero cualquiera. El número de colas que deben crearse dependerá de la cantidad de usuarios y la intensidad de uso del sistema.

Se puede iniciar con un número bajo de colas y luego revisar los archivos de log del componente, en los cuales se escribirá un mensaje de advertencia (WARN),

---

[37] Documentación incluida en el instalador o en www.jboss.org

en caso de que el componente detecte que necesita aumentar el número de colas.

Para la configuración de colas, en JBoss, para cada cola se debe agregar el siguiente código en el archivo *server/deploy/jms/jboss-mq-destinations-service.xml:*

```
<mbean                                                    code="org.jboss.mq.server.jmx.Queue"
       name="jboss.mq.destination:service=Queue,name=JMSWfServiceQueue05">
       <depends
            optionalattributename="DestinationManager">jboss.mq:service=DestinationManager
            </depends>
       <depends
            optionalattributename="SecurityManager">jboss.mq:service=SecurityManager</depen
            ds>
       <attribute name="MessageCounterHistoryDayLimit">1</attribute>
       <attribute name="SecurityConf">
              <security>
                   <role name="guest" read="true" write="true" />
                   <role name="publisher" read="true" write="true" create="false" />
                   <role name="noacc" read="false" write="false" create="false" />
              </security>
       </attribute>
</mbean>
```

Figura 3-5 Código para configuración de colas JMS en JBoss

Se recomienda además revisar las seguridades para JMS cuando se vaya a instalar en un ambiente de producción.

3.3.2 INSTALACIÓN Y CONFIGURACIÓN DE WFMS

Los sub-componentes CORBAWorkflowClient y CORBAWorkflowServer deben funcionar sin problema con cualquier WFMS si existe el sub-componente JMSWorkflowServer correspondiente para dicho WFMS.

Dado que en el presente trabajo se desarrolló el sub-componente JMSWorkflowServer para WfmOpen, se mostrará la instalación y la configuración básica de WfmOpen.

3.3.2.1 Instalación de DBMS

Antes de instalar WfmOpen, es necesario instalar el DBMS que utilizará para almacenar los datos que necesita.

Puede instalarse cualquier DBMS que provea las librerías JDBC correspondientes. La versión usada para el desarrollo del componente es MySQL 5.0.

Dentro de las configuraciones necesarias en MySQL:

- Permitir el acceso mediante TCP-IP

Se recomienda además:

- Agregar contraseña al usuario administrador (root).
- Crear un usuario para el WFMS, puede llamarse por ejemplo wfmopen, con su respectiva contraseña.
- Asignar al usuario creado los permisos para creación de bases de datos.

Posteriormente, luego de la instalación de WfmOpen se recomienda:

- Quitar el permiso de creación de bases de datos al usuario creado.

3.3.2.2 Instalación y configuración de WfmOpen

Al iniciar el instalador wfmopen-2.1ea5-installer, luego de la pantalla de bienvenida, tenemos la siguiente pantalla:

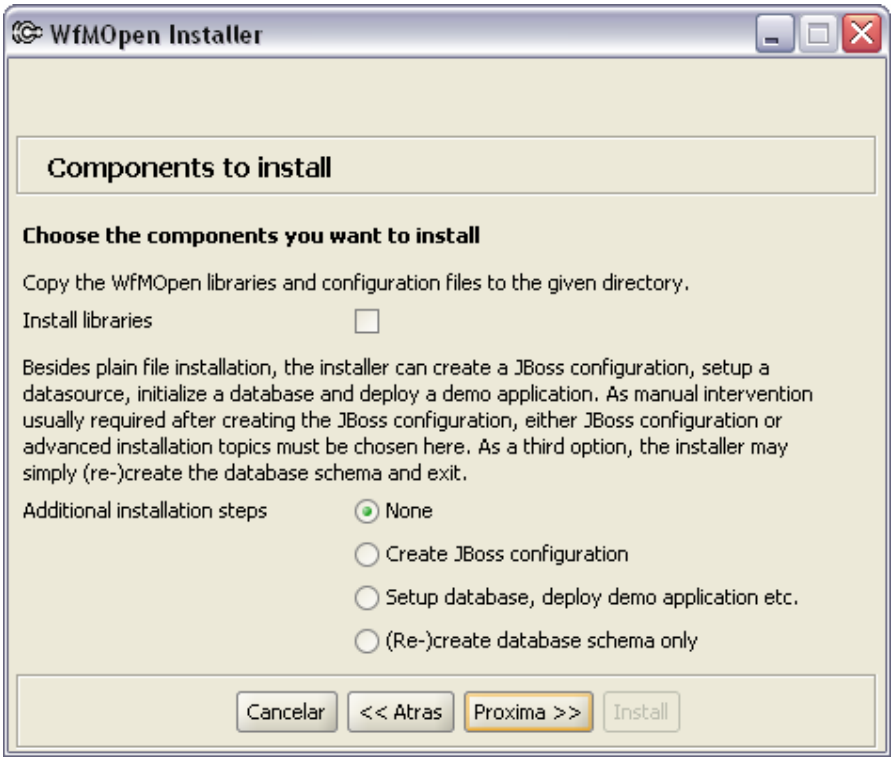


Figura 3-6 Opciones de instalación de WfmOpen

Dado que instalaremos sobre JBoss, escogemos esta opción. El instalador nos preguntará la ubicación de JBoss, así como los parámetros de conexión a la base de datos.

El instalador creará una carpeta dentro de la carpeta *server* de JBoss, la cual contendrá la aplicación y las configuraciones necesarias.

Terminado este proceso, debemos volver a ejecutar el instalador y esta vez seleccionar una de las dos opciones:

- “Setup database, deploy demo application”, si queremos que WfmOpen instale las aplicaciones y los datos de demostración.
- “(Re-)create database schema only”, si deseamos que WfmOpen se instale sin ninguna aplicación de demostración y sin datos de prueba.

3.3.3 INSTALACIÓN Y CONFIGURACIÓN DEL COMPONENTE (PARAMETRIZACIÓN)

En el lado del cliente del WFMS no es sino necesario copiar la librería CORBAWorkflowClient.DLL y sus dependencias correspondientes (ver más adelante) en la carpeta de la aplicación que desea conectarse al WFMS.

En el lado del servidor, debemos copiar los archivos CORBAWorkflowServer.jar y JMSWorkflowServer.jar junto con sus dependencias (se muestran más adelante), a cualquier carpeta del servidor.

Se podrían instalar dichos servicios en diferentes servidores, e incluso en otro servidor diferente al que se encuentra el WFMS, pero por lo general va a ser la mejor opción instalar estos componentes en un mismo servidor, junto con el WFMS.

Luego de copiar los archivos, es necesaria la configuración de los archivos:

- LOG4J.xml: Configuración de LOG4J. [38]
- CORBAWorkflowServer.xml

3.3.3.1 Configuración de CORBAWorkflowServer.xml

En este archivo se deben configurar los parámetros:

- CORBA.java.naming.factory.initial
- CORBA.java.provider.url

Los cuales sirven para acceder al ORB en el que se registrará el servicio CWfServiceFactory.

---

[38] Para información y ejemplos de configuración de LOG4J, revisar http://jakarta.apache.org/log4j.

3.3.4 LIBRERÍAS NECESARIAS EN EL SERVIDOR

Para el funcionamiento del componente, en el lado del servidor, deben existir al menos las siguientes librerías:

Para el sub-componente CORBAWorkflowServer:

- **commons-logging.jar** y **log4j.jar**: Librerías utilizadas para escribir los archivos de log.
- **jbossall-client.jar**: Librería para clientes de JBoss.

Para el sub-componente JMSWorkfowServer:

- **commons-logging.jar** y **log4j.jar**: Librerías utilizadas para escribir los archivos de log.
- **jbossall-client.jar**: Librería para clientes de JBoss.
- **de.danet.an.util.jar**: Librería de utilidades de WfmOpen.
- **de.danet.an.wfcore-apis.jar**: Librería de APIs[39] de WfmOpen.
- **de.danet.an.wfcore-client.jar**: Librería de cliente de WfmOpen.
- **jsr173_1.0_api-nostd.jar**: Librería utilizada por librería de cliente de WfmOpen.

3.3.5 LIBRERÍAS NECESARIAS EN EL CLIENTE

Para que una aplicación pueda acceder al servicio CORBAWorkflowServer, necesita la librería

- CORBAWorkflowClient.DLL

Dicha librería, necesita para su funcionamiento las siguientes:

[39] API: Interfaz de programador de aplicaciones, “Application Programmer Interface”

- **IIOPChannel.DLL**: Librería IIOP.NET para permitir el acceso a CORBA desde .NET.
- **WfInterfaces.DLL**: Librería de STUBS para acceso a interfaces provistas por el servicio CORBAWfServer.

Por tanto, las tres librerías son necesarias en el lado del cliente.

3.3.6 USO DEL COMPONENTE

Dentro de las referencias de la solución debemos agregar las tres librerías necesarias. (Menú Project → Add Reference…)

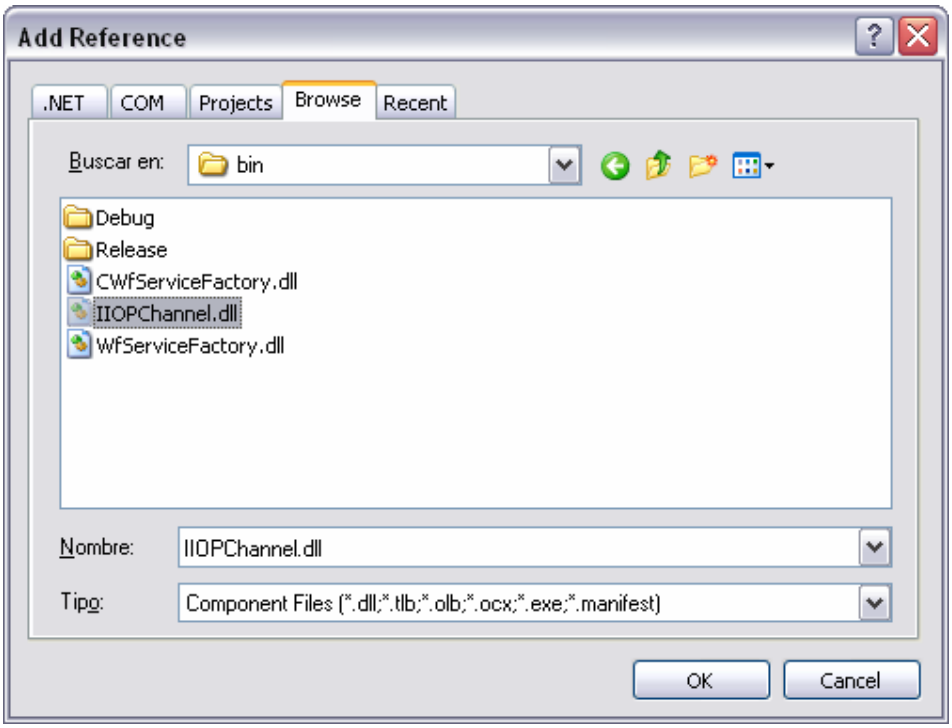


Figura 3-7 Pantalla para agregación de referencias, Visual Studio .NET 2005

En cada archivo de código en el que queramos utilizar el componente agregamos al inicio las líneas:

```
using edu.epn.workflow.CORBAWorkflowClient;
using edu.epn.workflow.CORBAWorkflowService;
```

Figura 3-8 Código de utilización del componente, referencia a librerías

Con el fin de acceder al servicio CORBAWfServer, debemos crear una instancia de la clase **CWfClient**.

```
CWfClient cwfc = new CWfClient();
```

Figura 3-9 Código de utilización del componente, instanciación de clase CWfClient

Para conectar al servicio de workflow utilizaremos la función *Connect*, con los parámetros correspondientes (Dirección IP, puerto, nombre de usuario y contraseña).

```
cwfc.Connect("localhost", 3528, "ML", "ML")
```

Figura 3-10 Código de utilización del componente, conexión

Para acceder a las interfaces debemos utilizar la propiedad *Workflow*.

```
cwfc.Workflow.processDirectory().processes().
```

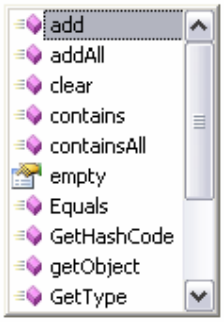


Figura 3-11 Código de utilización del componente, uso de propiedades y métodos

Finalmente, para desconectarse se usa la función D*isconnect*.

```
cwfc.Disconnect()
```

Figura 3-12 Código de utilización del componente, desconexión

3.3.7 ARCHIVOS DE LOG

Los diferentes mensajes de log del componente se guardan en dos archivos:

- CORBAWfServer.log
- JMSWfServer.log

Se puede revisar dichos archivos para verificar el funcionamiento del componente, para localizar errores, advertencias, etc.

CAPITULO 4. DESARROLLO DE APLICATIVO DE DEMOSTRACIÓN

4.1 DESCRIPCIÓN DEL PROBLEMA

El objetivo de este capítulo es, además de demostrar el uso del componente, mostrar las ventajas de utilizar un WFMS en un ambiente donde los procesos son cambiantes.

Históricamente, cuando se implementaba la automatización del flujo de trabajo en una empresa, un cambio en el proceso significaba un esfuerzo significativo ya que implicaba un cambio en el sistema y un impacto en la gente implicada.

En este contexto, si la empresa desea entrar en ciclos de mejora continua de procesos, esto le resultaría bastante costoso.

Creando el ambiente apropiado, y mediante el uso de este componente, se puede sin embargo desarrollar un sistema en el que el cambio de procesos puede ser mucho menos costoso y mucho más rápido, implementando lo que se llama un "Sistema de Gestión de Procesos de Negocio" ó "Sistema de BPM [40]".

El objetivo de los Sistemas de "Gestión de Procesos de Negocio", conocidos como sistemas de BPM, es entre otros, facilitar el cambio en los procesos. Se incluyó un pequeño análisis de BPM y su relación con los sistemas de flujo de trabajo en el ANEXO 5.

---

[40] BPM: "Gestión de Procesos de Negocio", por sus siglas en ingles: "Business Process Management"

Cabe recalcar que el objetivo no es crear un sistema de BPM, sino como se mencionó anteriormente, simplemente demostrar el uso y las ventajas de utilización del componente desarrollado.

Un proceso típico en una empresa de producción es el procesamiento de las órdenes de compra, para la demostración se buscará automatizar este proceso.

Para nuestro ejemplo, dicho proceso consistirá en la recepción de la petición, la validación de la información, el procesamiento de la petición de crédito si hubiere, la revisión de disponibilidad, la fabricación en caso de ser necesaria, el envío de artículos y la confirmación de recepción.

4.2 DISEÑO DE LA SOLUCIÓN

4.2.1 EL PROCESO: PROCESAMIENTO DE ÓRDENES DE COMPRA

Para el presente desarrollo, se harán dos definiciones del proceso: Una definición inicial, sobre la cual debe funcionar el sistema, y para la cual se desarrollarán las aplicaciones necesarias, y una segunda definición, que busca optimizar el proceso, y que deberá ser implementada sobre el sistema.

4.2.1.1 Actores

Para un proceso, podemos definir que una tarea puede ser realizada por:

- Una persona en específico
- Cualquier persona perteneciente a un rol en la empresa
- Una unidad organizacional
- Un sistema

Dado que este sistema no se implementará en un ambiente real, sino que tiene objetivos meramente demostrativos, no se ha buscado ser estricto en la definición del recurso que debe realizar cierta tarea, sino que se ha buscado más bien la simplicidad.

Por ejemplo, se verá que la tarea “Planificar Producción”, que debería ser realizada por un Jefe de Producción, se ha asignado al Departamento Producción, sin importar quien la realice.

En el sistema podemos identificar que participarán las siguientes unidades organizacionales:

- **Servicio al cliente**: Departamento encargado de contactarse con el cliente.
- **Bodega**: Departamento encargado del almacenamiento y envío de los artículos producidos.
- **Producción**: Departamento encargado de producir los artículos en la empresa.

Además los siguientes roles:

- **Asesor de crédito**: Persona encargada de realizar análisis de capacidad de pago y aprobar o negar un crédito.

Y además, habrá actividades que serían realizadas por ninguna persona, sino automáticamente, por el sistema.

4.2.1.2 Variables de flujo de trabajo

Para definir el flujo de cada caso, las decisiones se toman en base a las llamadas “variables de flujo de trabajo”. Para el ejemplo, se han identificado las siguientes variables de flujo de trabajo:

- **cliente_codigo**: Código único del cliente (String)
- **cliente_nombre**: Nombre del cliente (String)
- **cliente_email**: Correo electrónico del cliente (String)
- **cliente_tipo**: Tipo de cliente (String), para el ejemplo se han definido tres tipos:
    - "A": Cliente muy importante
    - "B": Cliente frecuente
    - "C": Cliente normal
- **cliente_credito_maximo**: Límite máximo de crédito que se le puede otorgar al cliente. (Float)
- **pedido_forma_pago**: Forma de pago del cliente (String). Se han definido las siguientes formas de pago:
    - "D": Debito
    - "C": Crédito
- **pedido_aprobado**: Indica si un pedido ha sido aprobado o negado (Boolean)
- **pedido_aprobado_envio_parcial**: Indica si se debe realizar un envío parcial antes de que los artículos faltantes sean producidos. (Boolean)
- **item_codigo**: Código del artículo solicitado por el cliente. [41] (String).
- **Item_precio:** Precio unitario del artículo (Float)
- **Item_cantidad**: Cantidad de artículos solicitados (Int)
- **item_stock**: Cantidad de artículos disponibles existentes en bodega. (Int).

4.2.1.3 Definición inicial

La definición inicial del proceso se muestra en el siguiente gráfico [42]:

---

[41] Por simplicidad, el sistema estará limitado a la petición de un solo tipo de artículos por orden de compra.

[42] Gráfico realizado en la herramienta "Together Workflow Editor", la cual genera archivos XPDL en base al gráfico del proceso.

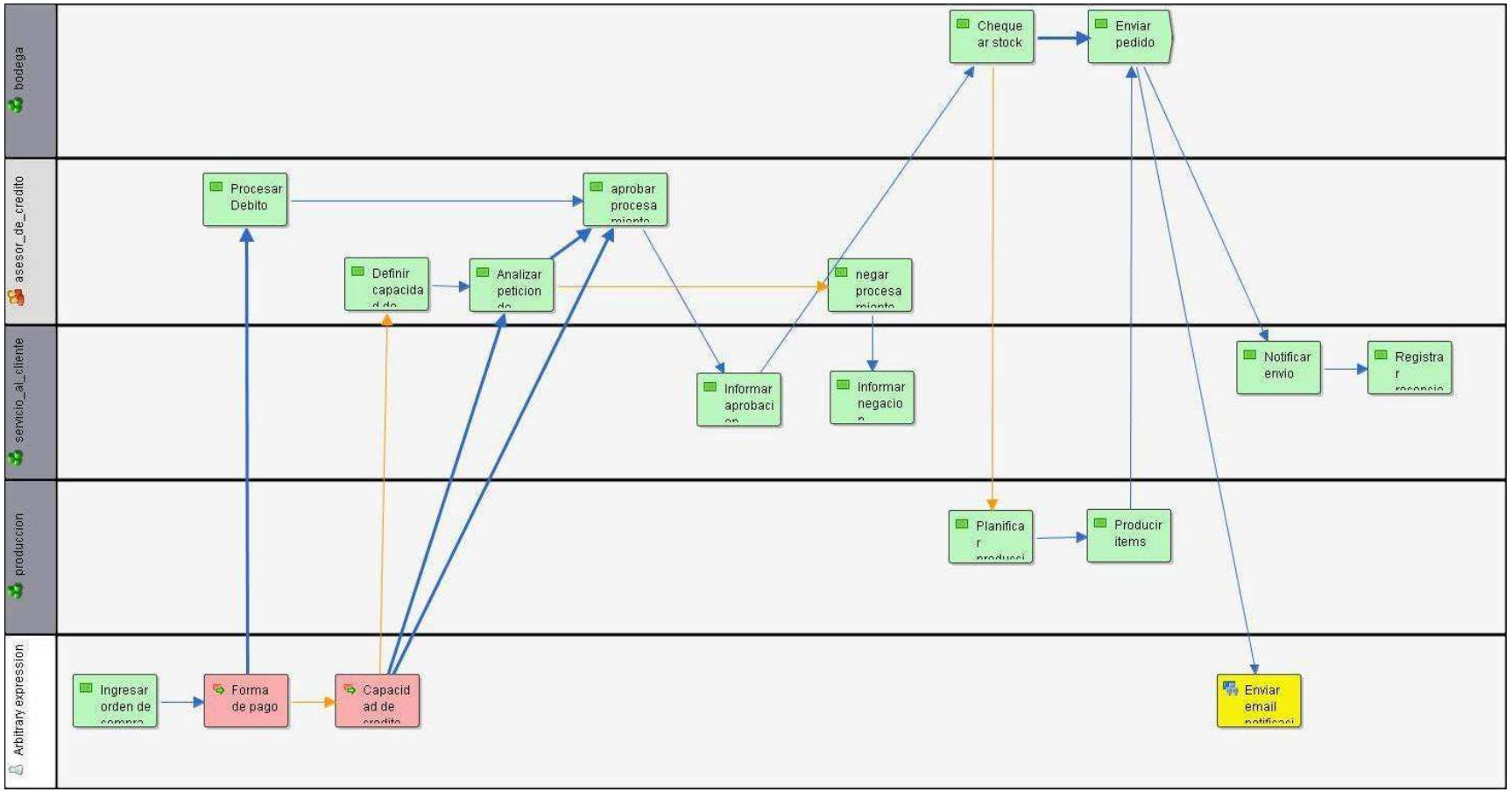


Figura 4-1 Proceso de orden de compra, primera definición

Las actividades de este proceso son:

Impersonales (realizadas por el sistema):

- Ingreso de orden de compra.
- Enrutar por forma de pago
- Enrutar por capacidad de crédito
- Enviar correo electrónico de notificación de envío

A cargo del rol “asesor de crédito”:

- Procesar débito bancario
- Definir capacidad de pago del cliente
- Analizar petición de crédito
- Aprobar procesamiento de la orden de compra
- Negar procesamiento de la orden de compra

A cargo del grupo de “servicio al cliente”:

- Informar aprobación de crédito
- Informar negación de crédito
- Notificar envío del pedido
- Registrar la recepción del pedido por parte del cliente

A cargo del grupo de “bodega”:

- Chequear stock real de productos
- Enviar pedido al cliente

A cargo del grupo de “producción”:

- Planificar producción
- Producir ítems

4.2.1.4 Segunda definición

La segunda definición se hizo en base a la primera, con los siguientes cambios:

- Se hace una diferenciación de los clientes, para dar un trato preferencial a los clientes más importantes.
- Se da al cliente la opción de un envío parcial si no existen en bodega todos los ítems solicitados, para luego enviar los ítems faltantes.
- Las notificaciones telefónicas (de aprobación, negación, envío) se hacen solamente a los clientes importantes, al resto se les enviará una notificación por correo electrónico.

La definición de proceso generada se muestra en el siguiente gráfico:

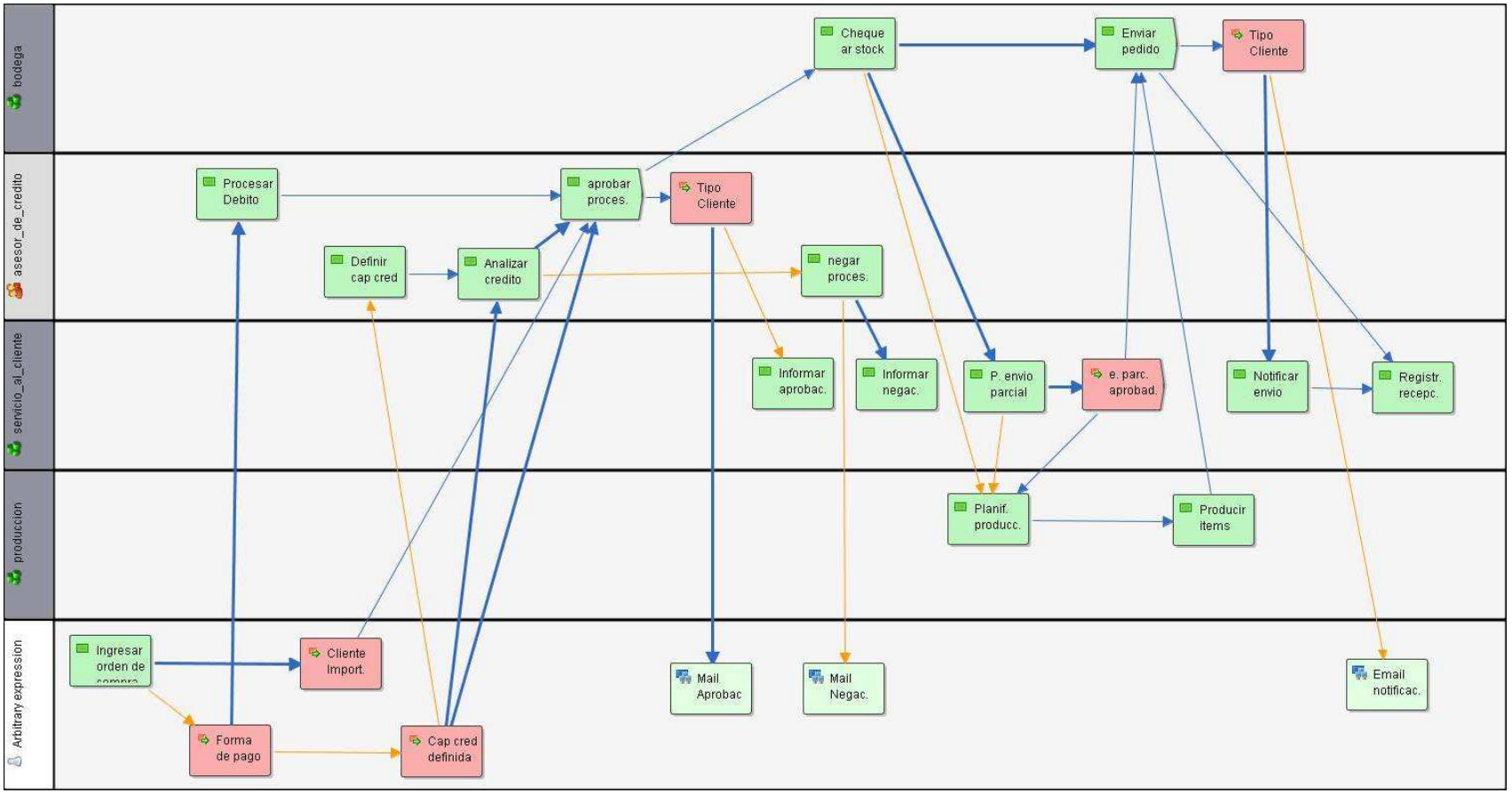


Figura 4-2 Proceso de orden de compra, segunda definición

4.2.2 DIAGRAMA DE COMPONENTES

Para la demostración, el sistema estará compuesto por los siguientes elementos:

- Dos aplicaciones “clientes de flujo de trabajo”:
  - **WfDemoApp**: Una aplicación de escritorio [43], para los empleados de la empresa
  - **WfDemoWebsite**: Una aplicación web, para los clientes de la empresa
    - Esta aplicación deberá mostrar una lista de productos y tener acceso a datos de clientes, necesitando tener datos persistentes, es necesario por tanto una base de datos en un DBMS. Se escogió el DBMS “SQL Server 2000” [44].
- Un servidor WFMS
  - El servidor WFMS utilizado será WfmOpen
  - Dado que el WFMS necesita una base de datos para su funcionamiento, esta será alojada en MySQL.
- Un servidor de correo saliente
  - Dado que en la definición de procesos revisada anteriormente hay tareas de envío de correo electrónico, es necesario un servidor de correo saliente. Se utilizará el servidor de correo saliente hMailServer [45].
- Explorador web
  - Dado que una de las aplicaciones es una aplicación web se necesita un explorador web para la visualización de dicha aplicación.

---

[43] La razón por la que se hizo esta aplicación de escritorio es solamente con fines demostrativos, en realidad en un ambiente real la mejor opción hubiera sido posiblemente una aplicación web.

[44] Se escogió SQL Server 2000 por 2 motivos: 1: facilidad de uso desde Microsoft Visual Studio .NET, y 2: Se desea mostrar en la demostración un ambiente totalmente mixto, en el que estarán funcionando múltiples tecnologías simultáneamente (el DBMS utilizado por el WFMS es MySQL y el DBMS utilizado por la aplicación web es SQLServer 2000).

[45] hMailServer es un servidor de correo saliente de código abierto para Windows. Más información en http://sourceforge.net/projects/hmailserver

En el siguiente gráfico se muestran los elementos mencionados.

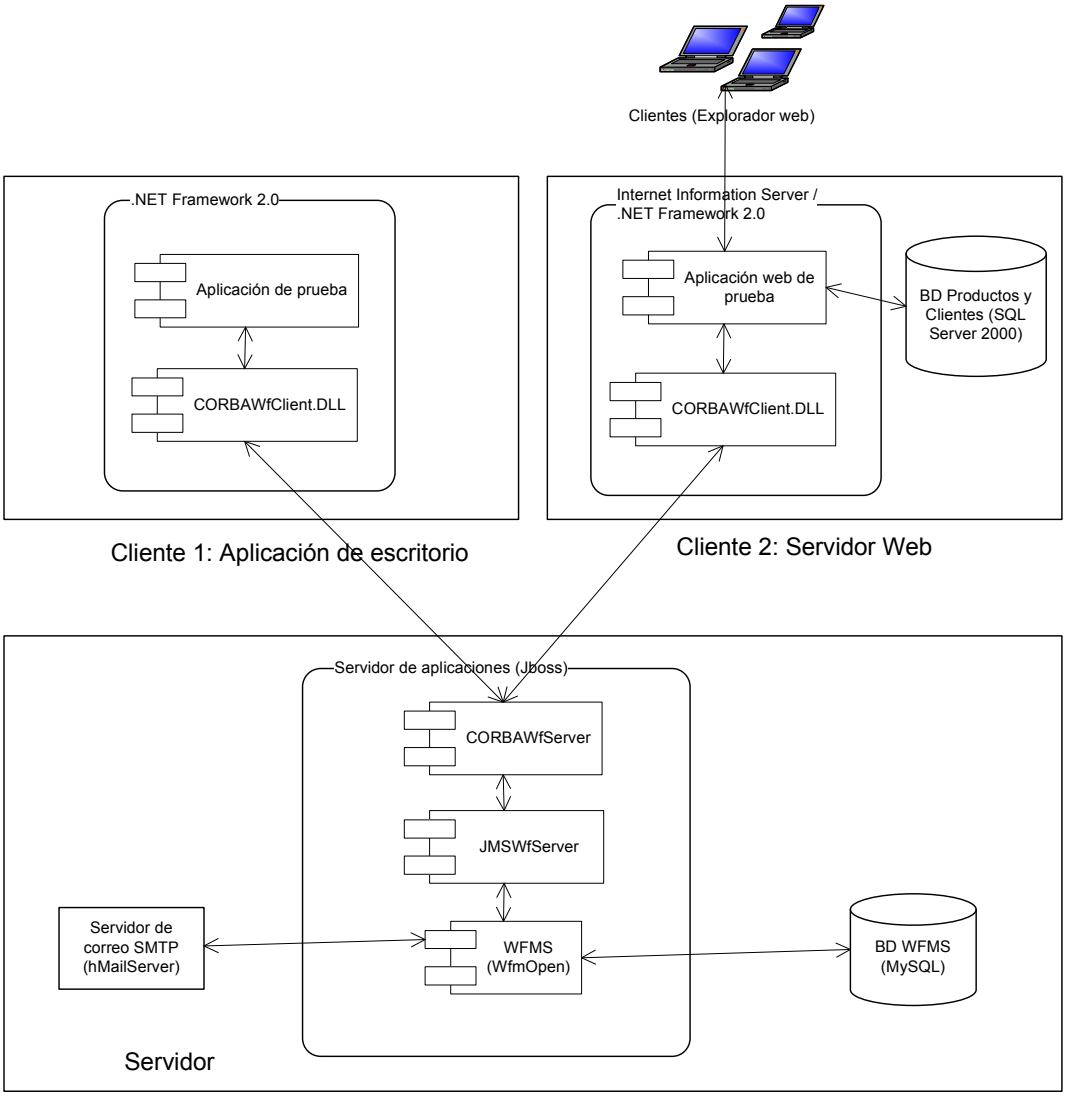


Figura 4-3 Elementos de la aplicación de demostración

Hay algunas librerías adicionales que son también necesarias, pero no se incluyeron en el gráfico con el fin de mantener la simplicidad. Se puede revisar la lista de librerías necesarias en el capítulo 3: (manual de utilización del componente: librerías necesarias en el servidor y librerías necesarias en el cliente).

4.2.3 APLICACIONES A DESARROLLARSE

Hay dos aplicaciones a desarrollarse:

- WfDemoWebsite
- WfDemoApp

4.2.3.1 WfDemoWebsite

Esta aplicación servirá para que los clientes de la empresa puedan ingresar una nueva orden de compra al sistema.

Los clientes deberán poder ingresar al sistema con su nombre de usuario y contraseña para luego escoger el producto, la cantidad y la forma de pago de su pedido.

El sistema deberá instanciar e iniciar un nuevo caso del proceso "orden_de_compra", y además deberá pasar los datos del cliente y los datos del pedido.

4.2.3.2 WfDemoApp

Esta aplicación será diseñada para los empleados de la empresa.

Cada usuario de esta aplicación deberá tener su nombre de usuario y contraseña, con la que la aplicación se conectará al WFMS.

Luego de ingresar, el usuario debe poder visualizar la lista de tareas asignadas:

- **Directamente**: cuando las tareas son asignadas específicamente para que él/ella la realice.

- **Indirectamente**: cuando la tarea ha sido asignada a un grupo o a un rol de la organización. Por ejemplo cuando la tarea asignada a cualquier “asesor de crédito”.

El usuario debe poder “aceptar” las tareas asignadas indirectamente. Por ejemplo, cuando una tarea este a cargo del rol “asesor_de_credito”, cuando el usuario acepte la tarea, esta será asignada directamente al usuario.

Esta aplicación deberá además tener una pantalla en la que permita visualizar todas las variables de flujo de trabajo correspondientes al caso.

Además deberá proveer la interfaz para el ingreso de los datos correspondientes a cada tarea y para dar la indicación al sistema de que la tarea ha sido completada.

Dado que es solamente una aplicación de demostración no se desarrollará interfaces para manejo de excepciones o para abortar casos o ítems de trabajo.

4.3 DESARROLLO

4.3.1 CONFIGURACIÓN DE WFMS Y BD

Con el fin de crear el ambiente necesario para el desarrollo de las aplicaciones de demostración se realizó los siguientes pasos [46]:

- Instalación y configuración del servidor de aplicaciones (JBoss)
- Instalación y configuración del WFMS (WfmOpen)
- Instalación del DBMS utilizado por el WFMS (MySQL)
- Instalación de la base de datos en MySQL
- Instalación del DBMS que será utilizado por la aplicación de demostración (SQLServer 2000).

[46] En el capítulo 3, en la sección “Manual de utilización del componente” se hizo una explicación más detallada de la instalación de la mayoría de estos elementos.

- Instalación de la base de datos en SQLServer 2000.
- Instalación y configuración del servidor de correo saliente (hMailServer)

4.3.2 INSTALACIÓN Y USO DEL COMPONENTE

Luego de que todo el ambiente esté preparado, es necesario configurar correctamente los parámetros del componente. Como hacerlo se detalla en el capítulo 3.

Luego de configurar el componente debemos ejecutar las clases:

- CORBAWfServer
- JMSWfServer

Con esto el ambiente está listo para el desarrollo y prueba de las aplicaciones cliente.

4.3.3 DESARROLLO DE APLICACIÓN

Lo primero que haremos antes de comenzar el desarrollo es cargar la definición de procesos en el WFMS.

Luego de que se ha diagramado el proceso, mediante una “Herramienta de definición de procesos”, se genera un código XPDL, el cual es una representación en XML del proceso en sí.

Dicha definición XPDL debe ser cargada en el WFMS, esto lo hacemos mediante las herramientas provistas por el mismo WFMS. En el caso de WfmOpen, este provee una interfaz web para la gestión de las definiciones de procesos. Dicha interfaz se instala automáticamente al instalar WfmOpen en la dirección: http://localhost:8080/wfportal

Al acceder a dicha dirección, luego de proporcionar el nombre de usuario y contraseña correspondiente, si tenemos los permisos necesarios, en la pestaña de administración de WfmOpen tenemos la siguiente pantalla:

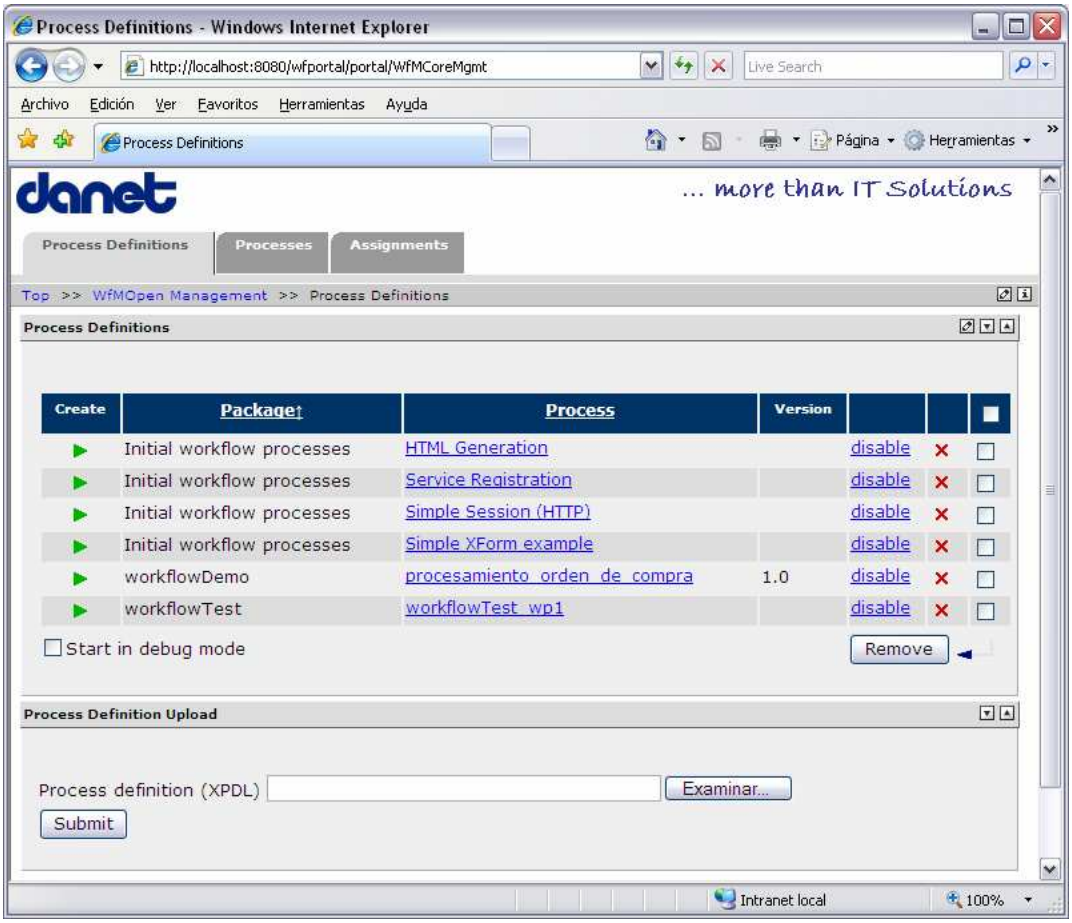


Figura 4-4 Herramienta de administración de WfmOpen

A través de esta pantalla cargamos la definición del proceso (archivo XPDL) al WFMS.

4.3.3.1 Desarrollo de aplicación WfDemoWebsite

Creamos en Visual Studio .NET una nueva aplicación de tipo “ASP.NET Website”.

A esta aplicación agregamos las referencias a las DLLs:

- CORBAWorkflowClient.dll
- IIOPChannel.dll
- WfInterfaces.dll

Y desarrollamos los formularios web:

- Default.aspx: Página de inicio, contiene el formulario de login.
- Dentro del directorio pages:
  - Cliente.aspx: Información del cliente, obtenida desde la base de datos
  - ordenCompra.aspx: Pantalla para ingreso de una nueva orden de compra.
  - Confirmacion.aspx: Pantalla de confirmación de ingreso de pedido.

Dado que el único formulario que accederá al WFMS es ordenCompra.aspx, solo en éste agregamos las líneas:

```
using edu.epn.workflow.CORBAWorkflowClient;
using edu.epn.workflow.CORBAWorkflowService;
```

4.3.3.2 Desarrollo de aplicación WfDemoApp

Creamos en Visual Studio .NET una nueva aplicación de tipo “Windows Application”.

A esta aplicación agregamos las referencias a las DLLs:

- CORBAWorkflowClient.dll
- IIOPChannel.dll
- WfInterfaces.dll

Y desarrollamos los formularios:

- frmLogin: Es la primera pantalla que se muestra, contiene el formulario de login.
- frmMain: Es el formulario principal, contiene 2 pestañas:
    - Lista de tareas: Muestra la lista de tareas asignadas directa e indirectamente al usuario.
    - Realización de tareas: Muestra tres elementos:
        - Lista de variables de flujo de trabajo del caso actual, con sus valores.
        - Instrucciones de realización de la tarea
        - Interfaz para envío de variables al WFMS y para indicar la finalización de la tarea asignada.

4.4 PRUEBAS

Lo primero que hacemos es subir el servidor de aplicaciones JBoss, en el que se encuentra instalado el WFMS. Para ello debemos ejecutar desde la línea de comandos:

```
C:\jboss\bin>run –c wfdemo
```

Figura 4-5 Inicialización de Jboss

Lo siguiente es subir la definición del proceso mediante la herramienta de administración provista por WfmOpen. Por defecto esta herramienta se configura en Jboss para su acceso mediante http://localhost:8080/wfportal.

Por defecto, WfmOpen crea varios usuarios para la configuración inicial:

- **ML/ML**: Usuario con los permisos para definición de procesos, visualización de procesos y asignación de tareas.

- **jetspeed/jetspeed**: Usuario con permisos para modificación de parámetros de herramienta de administración y permisos para gestión de seguridades (creación de usuarios, roles, grupos, etc.)

Dado que son necesarios usuarios para las pruebas del sistema, creamos mediante la herramienta de administración los siguientes roles, grupos y usuarios:

ROL	**NOMBRE USUARIO**
asesor_de_credito	maria_asesor
	carlos_asesor
	andrea_asesor
GRUPO	**NOMBRE USUARIO**
Bodega	juan_bodega
	pedro_bodega
Servicio_al_cliente	andres_servicio_cliente
	manuel_servicio_cliente
Producción	veronica_produccion
	carla_produccion

Tabla 4-1 Usuarios creados para las pruebas

Por facilidad se asignó como contraseña el mismo nombre de usuario para todos los usuarios creados.

Posteriormente, ejecutamos la aplicación web cliente, mediante la cual un cliente va a ingresar una nueva orden de compra al sistema.

Figura 4-6 Pantalla de inicio de sesión WfDemoWebsite

En esta pantalla ingresamos el nombre de usuario y contraseña correspondiente a un cliente (ej, **acme/acme**). Y escogemos en el menú de la izquierda la opción “Orden de Compra”.

Figura 4-7 Ingreso de orden de compra WfDemoWebsite

Luego de escoger los parámetros correspondientes, enviamos la petición y el sistema automáticamente comenzará el procesamiento del flujo de trabajo.

Por otro lado, para revisar los trabajos pendientes, los empleados usarán la otra aplicación de demostración (WfDemoApp).

Por ejemplo, un asesor de crédito ingresará con su nombre de usuario y su contraseña:

Figura 4-8 Pantalla de ingreso WfDemoApp

Y tendrá acceso a la lista de tareas asignadas directa e indirectamente a su usuario.

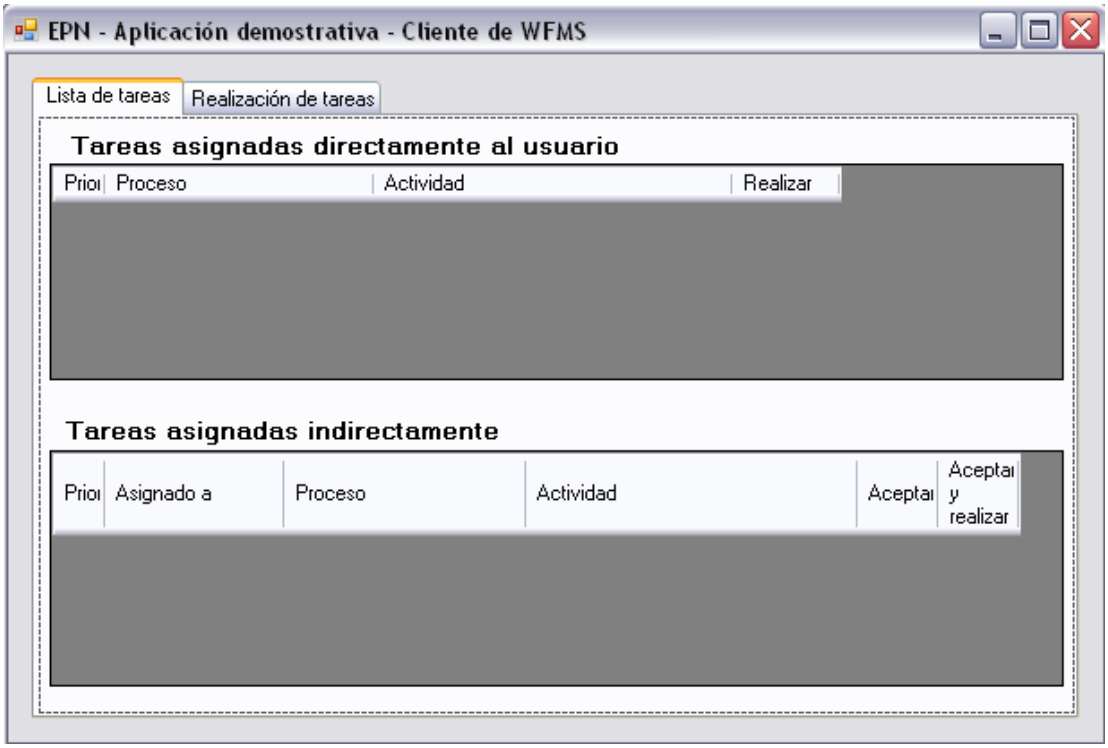


Figura 4-9 Lista de tareas WfDemoApp

Cuando el usuario acepte y haga clic en el botón de “Realizar” una tarea, la aplicación mostrará la siguiente pantalla, mediante la cual, por ejemplo, en este caso, la persona podrá acceder a los datos para “Definir la Capacidad de Crédito” del cliente, e informar el resultado de su trabajo al WFMS.

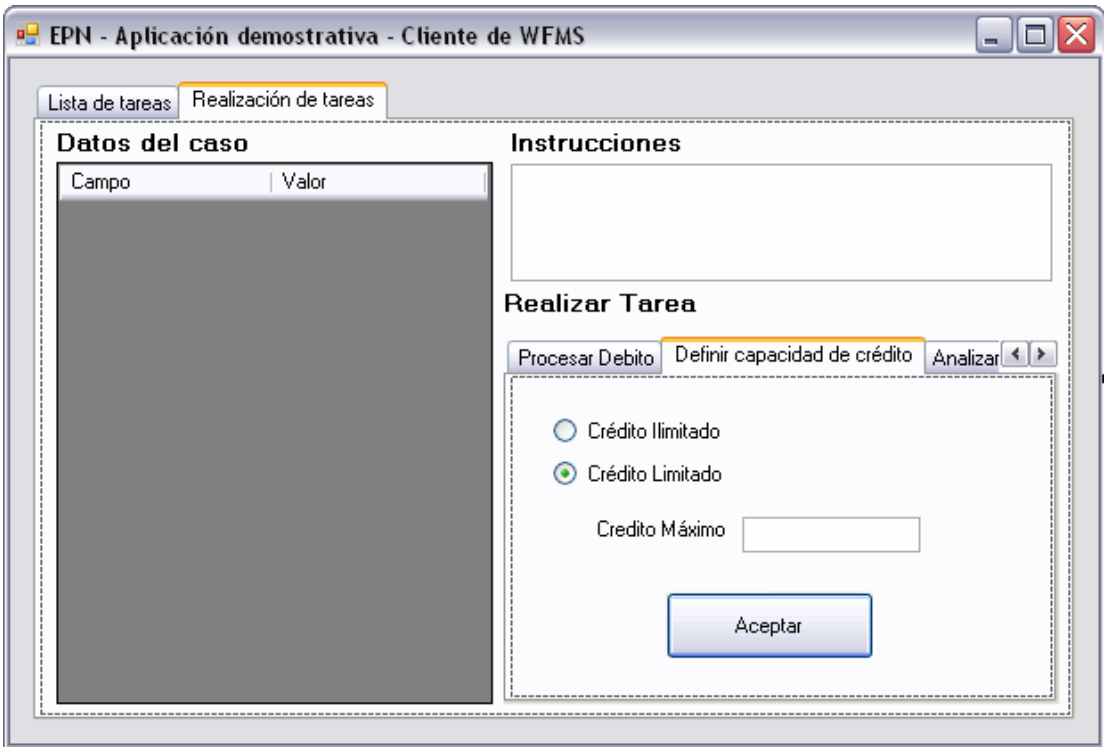


Figura 4-10 Pantalla de realización de tareas WfDemoApp

CAPITULO 5. CONCLUSIONES Y RECOMENDACIONES

5.1 CONCLUSIONES

- Los Sistemas de Gestión de Flujos de Trabajo (WFMS) facilitan la separación de la lógica de flujos de trabajo y la lógica propia de realización de las tareas del negocio.
- El uso de un WFMS y una buena estrategia de desarrollo rápido de aplicaciones facilitará en gran manera la implementación de ciclos de mejora continua de procesos en una organización.
- El uso del componente desarrollado facilita la interacción de “aplicaciones clientes de flujo de trabajo” con el WFMS, logrando así que el diseño y desarrollo de dichas aplicaciones pueda realizarse en un tiempo mucho menor.
- Aunque el componente desarrollado sigue estándares ampliamente aceptados para la interacción de los WFMS con las aplicaciones, estos estándares todavía están en una etapa inmadura y dejan varios puntos a criterio del desarrollador, por lo cual es imposible garantizar una integración total con otros productos de otros proveedores sin un desarrollo adicional.
- Aunque CORBA permite la interacción de aplicaciones escritas en diferentes lenguajes, todavía existen problemas de incompatibilidades entre los productos de los diferentes vendedores, estos problemas deberán ser resueltos en nuevas versiones de la especificación CORBA.
- El costo (tiempo y dinero) inicial necesario para el desarrollo de un sistema de información usando un WFMS generalmente va a ser mayor que si no se lo usase, sin embargo, luego de que el sistema se encuentre en producción generalmente va a resultar mucho menos costoso, especialmente si existen procesos cambiantes.

5.2 RECOMENDACIONES

- Se recomienda el uso de Sistemas de Gestión de Flujos de Trabajo (WFMS) para la automatización de procesos organizacionales cambiantes.
- Se recomienda el uso del componente desarrollado para el desarrollo de aplicaciones o componentes clientes de flujo de trabajo escritos en .NET y que deseen interactuar con un WFMS escrito en Java.
- Si se desea interactuar con un WFMS diferente al utilizado en el desarrollo de la presente tesis (WfmOpen) no se debe realizar un nuevo desarrollo de todos los subcomponentes, sino solo del sub-componente JMSWfServer, el cual debe ser implementado específicamente para el WFMS que se desee utilizar.
- Si se desea crear “aplicaciones cliente de flujo de trabajo” escritas en una plataforma diferente a .NET, se debe desarrollar una nueva librería CORBAWorkflowClient, en base a los archivos IDL generados en esta tesis.
- Para el uso del presente componente son necesarios conocimientos de las arquitecturas J2EE,.NET y CORBA, pero sobre todo un entendimiento de los conceptos de gestión de flujos de trabajo.
- Cuando se implemente un sistema usando el componente desarrollado, y en general para sistemas que utilicen WFMSs, se recomienda la participación de personas que cumplan con los roles de “Administrador de Procesos” y “Administrador de WFMS”. El primero debe tener un conocimiento amplio de los procesos organizacionales, mientras que el segundo un amplio conocimiento técnico del WFMS que se utilice.
- La capacidad computacional recomendada para el uso del componente en el lado del servidor viene dada no por el componente en sí, sino más bien por el WFMS y el servidor de aplicaciones que se utilicen.
- Para el lado del cliente, la capacidad computacional requerida viene dada prácticamente por los requerimientos propios de .NET.

BIBLIOGRAFÍA

1. AALST, Wil van der y HEE Kees van; “Workflow Management”. The MIT Press. Londres Inglaterra. 2002
2. AALST, Wil van der y HEE, Kees van, “Workflow Management: Models, Methods and Systems”, The MIT Press, Cambridge Massachussets, 2002.
3. AALST, Wil van der W.M.P, “The Application Of Petri Nets to Workflow Management”. Departmento de Matetemáticas y Ciencias de la Computación, Universidad Tecnológica Eindhoven, Holanda.
4. WfMC, “Workflow Management Application Programming Interface (Interface 2&3) Specification”, Documento Número WFMC-TC-1009, Julio 1998, Versión 2.0. Disponible en www.wfmc.org.
5. OMG, “Workflow Management Facility Specification v.1.2”, Versión 1.2. Abril 2000. Disponible en www.omg.org.
6. Sitio web de documentación formal del OMG (Object Management Group) http://www.omg.org/library/specindx.html, Ultimo acceso: Agosto 2007.
7. SOMERVILLE, Ian, “Ingeniería de Software”, Editorial Addison Wesley, 6ta Edición, Inglaterra, 2002.
8. WfMC, “The WfMC Glossary”, “The Workflow Handbook 2003”, Future Strategies Inc., Book Division, Lighthouse Point, Florida.
9. ALLEN, Rob, “Workflow, an Introduction”, Comité de Relaciones Externas de la WfMC, Reino Unido, Agosto 2005.
10. PIKE John, “BPM in Context, Now and in the future”, Ejecutivo de la WfMC, Reino Unido, Marzo 2006.
11. BOGGS Wendy y BOGGS Michael, “Mastering UML with Rational Rose 2002”, Sybex 2002, Alameda, California.
12. FISCHER, Layna, “Workflow Handbook 2003”, Future Strategies Inc. Books Division, LigthHouse Point, Florida, 2003.
13. HINOJOSA Gerardo, JÁCOME Verónica, “Utilización de una red Petri en el modelamiento de flujos de trabajo”, Tesis EPN, 2004.

14. PIND Lars, “Petri Nets in the Workflow Package”, http://www.project-open.org/product/modules/workflow/petri-nets.html Último acceso: Enero 2007.
15. LIPP Michael, DANET GmbH, “The Danet Workflow Component: User Manual for version 2.1ea5”, Boston Estados Unidos, Enero 2007.
16. FLATTERY Mark, “Workflow Systems”, Tesella Support Services, Oxfordshire Reino Unido, Abril 2005.
17. HUGHES Steve, ZHEN Changqing, GLOBUS Natasha, “Workflow Overview and Approach”, Julio 2004.
18. HOLLINSWORTH David, WFMC, “The Workflow Reference Model”, Documento número TC00-1003, Enero 1995.
19. ELLIS Clarence, “Workflow Technology”, Universidad de Colorado, Colorado Estados Unidos, Noviembre 2006.

INDICE DE FIGURAS

INDICE DE TABLAS

ANEXOS

ANEXO 1: DEFINICIONES DE TÉRMINOS UTILIZADOS

Con la finalidad de que no haya confusiones en los términos utilizados en el presente trabajo, se realizará una definición de los términos básicos utilizados.

Este capítulo no pretende ser un estudio completo de los conceptos de Flujo de Trabajo, sino que presenta una breve definición para evitar confusiones.

Desde que existen las organizaciones, estas han tenido en sus procesos, además de la planificación y la ejecución, el control de procesos.

Gran parte del funcionamiento de una empresa no es solo la ejecución de una tarea específica, sino la interacción entre recursos y procesos. De esta problemática nace el concepto de flujo de trabajo.

PROCESOS DE NEGOCIO

La definición de la WfMC, es: “Es un conjunto de uno o más procedimientos o actividades directamente ligadas, que colectivamente realizan un objetivo del negocio, normalmente dentro del contexto de una estructura organizacional que define roles funcionales y relaciones entre los mismos.” [47]

Un proceso de negocio puede o no ser automatizado o ser automatizado en parte.

Los procesos de negocio de una organización deberían estar documentados y corresponder a la realidad de cómo se ejecutan los procesos en la empresa, pero

---

[47] WfMC, “The WfMC Glossary”, www.wfmc.org

un problema que encontramos generalmente es que no existe una documentación o, aunque haya una documentación, ésta no siempre corresponde a la realidad.

FLUJO DE TRABAJO

La WfMC define a Flujo de Trabajo como:

"La automatización de un proceso de negocio, completo o en parte, durante el cual documentos, información o tareas son pasadas de un participante a otro para una acción, de acuerdo a un conjunto de reglas de procedimiento." [48]

Según Wil van der Aalst, "Un flujo de trabajo comprende casos, recursos y disparadores que relacionan a un proceso en particular." [49]

Algo que debemos entender claramente es que un Flujo de Trabajo no automatiza la ejecución de las tareas de un proceso (lo cual debe ser realizado por un recurso), sino que automatiza la ejecución y el control de la interacción entre los diferentes actores presentes en el proceso.

Se puede utilizar indistintamente el término "workflow" o flujo de trabajo pues hacen referencia al mismo concepto.

CASO

La manera más fácil de entender un caso es como un "producto en progreso". Un caso es una instancia particular dentro de un proceso. Ejemplos de casos son una orden de entrega, una petición de crédito, etc. Cada caso debe tener una identidad única, puede tener diferentes estados y siempre está en algún lugar del proceso.

---

[48] WfMC, "The WfMC Glossary", "The Workflow Handbook 2003", Future Strategies Inc., Book Division, Lighthouse Point, Florida, pag. 264

[49] AALST, Wil van der y HEE, Kees van, "Workflow Management: Models, Methods and Systems", The MIT Press, Cambridge Massachussets, 2002, pag. 356.

Algunos de los sinónimos que pueden utilizarse son: producto, instancia de Flujo de Trabajo, entre otros.

RECURSO

Un recurso hace referencia a actores como personas, máquinas, aplicaciones, departamentos, unidades de negocio, etc. Son actores que realizan tareas específicas con el fin de procesar cada caso.

DISPARADOR

Para que un caso pase de un estado a otro, debe no solamente tener las condiciones necesarias, sino que se requiere la acción de un actor para pasarlo a otro estado, esta acción es dada por un disparador.

Existen varios tipos de disparadores:

1. Iniciado por un recurso: Por petición de un usuario o una aplicación.
2. Generado externamente: Mensajes externos al proceso de Flujo de Trabajo.
3. Basado en tiempo: Se dispara a ciertas horas específicas o luego de haber transcurrido un tiempo definido.
4. Iniciados automáticamente: Cuando un caso cumple las condiciones necesarias se disparan inmediatamente.

GESTIÓN DE FLUJO DE TRABAJO

Hace referencia a las tareas realizadas, usando métodos, técnicas y herramientas con el fin de soportar los procesos de negocio estructurados en una organización.

El objetivo de gestionar un flujo de trabajo no es sino que la organización se asegure que las tareas sean ejecutadas por el recurso correcto en el tiempo correcto.

ANEXO 2: LOS SISTEMAS DE FLUJO DE TRABAJO: HISTORIA Y PERSPECTIVAS

DEFINICIÓN

"Un Sistema de flujo de trabajo es aquel que está diseñado para soportar los flujos de trabajo en una situación específica del negocio" [50]

CLASIFICACIÓN

1. Clasificación por el proceso a controlar

La presente clasificación ha sido aceptada y manejada históricamente por varios años, y está incluida a manera de referencia, pero creemos que no es la mejor manera de clasificar un Flujo de Trabajo, pues al separar la ejecución y el control de los procesos, podemos decir que un Flujo de Trabajo debería en realidad ser capaz de controlar cualquier tipo de proceso y no solamente un tipo en específico, por lo que esta clasificación no tendría razón de ser. De todas maneras se incluye pues puede ayudarnos a entender la diferencia entre varios sistemas de Flujo de Trabajo.

a. Flujo de Trabajo de producción

Es también llamado Flujo de Trabajo transaccional, y es aquel que se encarga de uno o varios procesos propios de producción de una empresa. Un flujo de trabajo de producción generalmente está diseñado para controlar una gran cantidad de información, es decir muchos casos que siguen un mismo proceso, con pocas excepciones.

---

[50] AALST, Wil van der, HEE kes van, "Workflow Management: Models, Methods and Systems", The MIT Press, Cambridge, Massachussets, 2002, pag. 357

Un ejemplo de Flujo de Trabajo de producción es aquel en que una recepcionista recibe una petición de crédito, esta es pasada a un agente de crédito y automáticamente envía una notificación al cliente, etc.

Un Flujo de Trabajo de producción puede ser tan simple como retirar un documento o puede llegar a ser muy complejo.

b. Flujo de trabajo colaborativo

La característica principal de Flujo de Trabajo colaborativo es la negociación y la existencia de menos casos en cada proceso de Flujo de Trabajo que un Flujo de Trabajo transaccional. En el centro del Flujo de Trabajo colaborativo están los documentos.

La diferenciación que se ha hecho tipicamente entre un Flujo de Trabajo de producción y uno colaborativo ha sido que el colaborativo puede involucrar varias iteraciones en una misma etapa, la cual finaliza cuando existe concordancia entre las partes.

Esta es una de las razones por las cuales mencionamos que realmente esta clasificación no es válida, pues un proceso de producción podría también incorporar la característica de varias iteraciones en un mismo estado hasta lograr un concenso de los actores.

Típicamente se han manejado herramientas como bases de datos documentales o sistemas de groupware para realizar Flujo de Trabajo colaborativo. Una de las herramientas también más utilizadas para Flujo de Trabajo colaborativo ha sido el correo electrónico.

c. Flujo de Trabajo Administrativo

Es una mezcla de los dos tipos anteriores, y está principalmente orientado a los procesos administrativos de la empresa.

Ejemplos típicos de este tipo de Flujo de Trabajo son los pedidos de vacaciones, aprobación de presupuestos, pedidos de compras, etc.

2. Clasificación por el modo de funcionamiento

Al igual que la clasificación anterior, esta clasificación es una manera antigua de clasificar al Flujo de Trabajo, pues la concepción actual debería ser que un Flujo de Trabajo no tenga un solo modo de funcionamiento, sino que pueda funcionar de varias maneras de acuerdo a la necesidad propia del proceso.

a. Flujo de Trabajo documental

Es aquel utiliza documentos para su funcionamiento. El centro de todo el Flujo de Trabajo son los documentos.

Un Flujo de Trabajo documental puede ser llevado a cabo manualmente en una empresa por supervisores y mensajeros y por los mismos actores, bajo un procedimiento definido. También puede realizarse Flujo de Trabajo documental utilizando bases de datos documentales y sistemas de groupware.

b. Flujo de Trabajo de mensajería

Es aquel que utiliza correo electrónico o mensajería de cualquier tipo como centro de su funcionamiento.

Por mucho tiempo, algunos sistemas de Flujo de Trabajo han implementado el correo electrónico como una plataforma base para la realización de su trabajo. Las empresas utilizan mucho este tipo de Flujo de Trabajo para procesos en los que hayan pocos casos manejables, y no debería usarse para cuando el número de casos es grande.

c. Flujo de Trabajo usando bases de datos

Es aquel que realiza la interacción de tareas y actores usando como centro una base de datos, en la que se almacenan las tareas a realizarse.

HISTORIA

Cuando se inició la automatización de procesos con la computación moderna, en la época de los años 60, las soluciones eran totalmente personalizadas para una situación específica. Si se deseaba desarrollar un sistema para control de procesos, lo que ahora conocemos como un sistema de Flujo de Trabajo, toda la programación, incluyendo el acceso a datos, la lógica del negocio, la interfaz de usuario y toda la lógica en sí de Flujo de Trabajo estaban incluidas dentro de la misma aplicación.

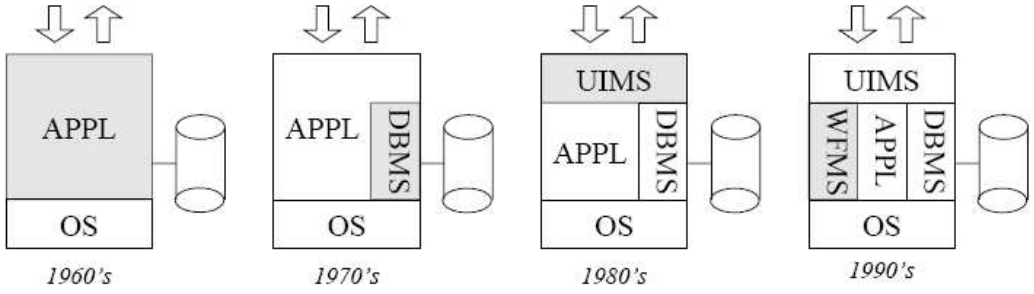
Esta arquitectura de desarrollo de sistemas no permitía la reutilización de código de tareas comunes como el acceso a datos, y hacía demasiado difícil el mantenimiento de dichos sistemas.

En los años 70's comienzan a surgir de manera fuerte los DBMS o Sistemas de Administración de Bases de Datos. Todos los sistemas comienzan a utilizarlos con el fin de separar la lógica de acceso a datos de la lógica propia de la aplicación; los sistemas de Flujo de Trabajo no son la excepción, y es así como un sistema hecho para solucionar problemas de Flujo de Trabajo de los años 70, estaba compuesto por la aplicación y un DBMS, los cuales funcionaban sobre algún sistema operativo.

Para los años 80's la tendencia es además, separar la capa de presentación, mediante los llamados UIMS (Sistemas de Administración de Interfaces de Usuario), y creando así la arquitectura de n-capas. En la actualidad los UIMS son realmente entornos de desarrollo de interfaces de usuario.

En los años 90's comienza a utilizarse el concepto de WFMS o Sistema de Gestión de Flujo de Trabajo. El concepto de WFMS, aunque lleva ya más de 10 años, todavía no se encuentra en una etapa madura como los DBMS o los UIMS.

Un esquema gráfico de esta evolución lo encontramos en un libre de van der Aalst, como se muestra a continuación:



Los sistemas de Flujo de Trabajo en una perspectiva histórica

Fuente: W.M.P. van der Aalst, "The Application Of Petri Nets to Workflow Management", pag. 2

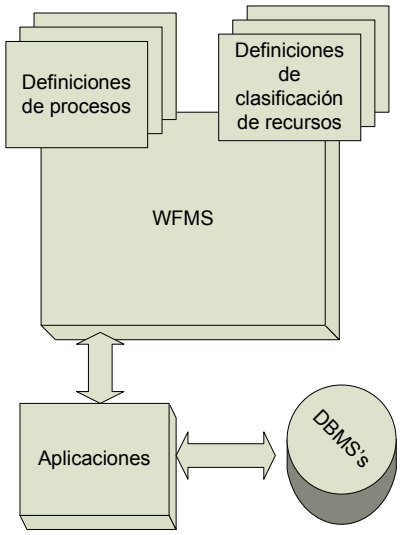
Históricamente, hasta los años 80's, los sistemas de Flujo de Trabajo incluían toda la lógica de flujos de trabajo, la definición de los procesos y las tareas automatizadas del proceso dentro de la misma aplicación. A partir de los años 90's se empieza a separar la lógica de flujos de trabajo genérica de la lógica propia de las aplicaciones de la empresa, llegando así a la creación de los WFMS's (Sistemas de Gestión de Flujo de Trabajo).

ESTADO ACTUAL

Actualmente, un sistema de Flujo de Trabajo consta de varias partes, una de las cuales es el sistema de Gestión de Flujo de Trabajo, que se encarga de la lógica propia de flujos de trabajo.

Las partes de un sistema de Flujo de Trabajo son por lo general:

1 Uno o varios WFMS's
2 Definiciones de procesos
3 Definiciones de la estructura empresarial, que contiene datos como recursos, clasificación de recursos, etc.
4 Aplicaciones
5 Uno o varios DBMS's



Componentes de un sistema de Flujo de Trabajo

PERSPECTIVAS

Las empresas tienen a cambiar y mejorar continuamente sus procesos, y se requiere cada vez más automatizar el control de los mismos. Los sistemas de Flujo de Trabajo son en sí sistemas para automatizar el control de los procesos.

Los Sistemas de Gestión de Flujo de Trabajo (WFMS) tienden a fortalecerse, debido a que permiten la automatización del control de procesos cambiantes, sin tener que cambiar todo el sistema de Flujo de Trabajo.

Los WFMS requieren para la interacción con usuarios el desarrollo de aplicaciones cliente, las cuales deben ser lo más simples posible, y su desarrollo debe ser también lo más rápido posible, debido al cambio continuo.

Es así que la tendencia para empresas que requieran sistemas de Flujo de Trabajo es levantar un marco de trabajo sobre un WFMS y utilizar la metodología RAD (Rapid Application Development) para la creación de interfaces de usuario.

La tendencia de las empresas que tienen procesos cambiantes es implementar sistemas para BPM (Administración de Procesos de Negocio). Un BPMS (Sistema de Administración de Procesos de Negocio) consta básicamente de uno o varios WFMS que interactúan con uno o varios DBMS y muchas aplicaciones de cliente.

ANEXO 3: SISTEMA DE GESTIÓN DE FLUJO DE TRABAJO (WFMS)

DEFINICIÓN DE WFMS

Hasta hace poco, la lógica de los procesos del negocio estaban implícitas en las aplicaciones, lo que dificultaba el cambio rápido de procesos en una organización pues significaba necesariamente un cambio en las aplicaciones.

Es por esto que se volvió necesario desarrollar los llamados Sistemas de Gestión de Flujo de Trabajo, que son básicamente "una herramienta de software genérica que nos permite la definición, ejecución, registro y control de flujos de trabajo." [51]

Usando las palabras de van der Aalst, "en escencia, el sistema de Gestión de Flujo de Trabajo es un bloque de construcción genérico para soportar los procesos de negocio".

Usando un WFMS separamos la lógica transaccional propia de la empresa y la lógica común de flujos de trabajo.

Las aplicaciones a desarrollarse para el sistema de Flujo de Trabajo serán mucho más sencillas, pues no deberán sino interactuar con el WFMS y ejecutar tareas específicas del negocio.

---

[51] AALST, Wil van der y HEE, Kees van, "Workflow Management: Models, Methods and Systems", The MIT Press, Cambridge Massachussets, 2002, pag. 1

PROPÓSITO DE UN WFMS

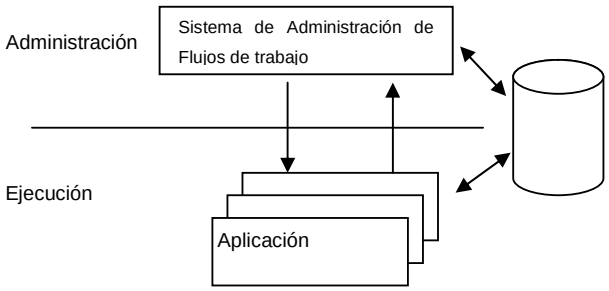
1. **Separar la ejecución y el control de los procesos**

Un Sistema de Gestión de Flujo de Trabajo, como ya se vio anteriormente, es un sistema especializado en el manejo de la lógica de flujos de trabajo.

Tradicionalmente la administración de procesos ha estado implícita dentro de las aplicaciones, de tal manera que cambios en los procesos de un negocio implicaban generalmente cambios en los sistemas.

Un WFMS nace de la necesidad de separar la administración de procesos y la ejecución de las tareas de los procesos de negocio de una empresa.

La separación de administración y la ejecución de los procesos se harían de la siguiente manera:



Separación entre la lógica y la ejecución

FUENTE: AALST, Wil van der HEE, Kees van; Workflow Management. The MIT Press. Londres Inglaterra. 2002

2. Propósito funcional

"El propósito principal de un WFMS es soportar la definición, ejecución, registro y control de procesos de una organización." [52]

a. Definición de procesos:

El WFMS debe permitir la definición de nuevos procesos, así como la redefinición de procesos que cambien en la organización, mediante un lenguaje estándar.

Haciendo una analogía con el DBMS, el cual debe tener un lenguaje llamado DDL o Lenguaje de Definición de Datos, el WFMS debe tener un lenguaje de definición de procesos.

b. Ejecución y control de procesos:

Este es el propósito principal del WFMS. La parte central de un WFMS debe ser el sistema de activación de flujos de trabajo, el cual realice efectivamente el control y la ejecución del flujo de trabajo, en base a las definiciones que se hayan hecho y en base a la interacción con los usuarios o aplicaciones.

Como se verá más adelante, en las partes de un WFMS, este sistema de activación está compuesto por uno o más motores de Flujo de Trabajo.

c. Registro de procesos:

Finalmente el propósito de un WFMS es realizar un registro de las definiciones y de la ejecución de los procesos, con el fin de realizar un monitoreo de los mismos. Debe proveer para esto una interfaz para la comunicación con las herramientas de Monitoreo.

---

[52] Idem, pag. 10

SISTEMAS DE WFMS AUTÓNOMOS Y EMBEBIDOS

1. Flujo de Trabajo autónomo

Son los WFMS's que tienen una funcionalidad completa sin necesidad de ningún software adicional, talvez con excepción de un DBMS y/o aplicaciones invocadas para envío de mensajes.

Los WFMS's autónomos se comunican con las aplicaciones que proveen la interfaz al usuario a través de interfaces de programación de aplicaciones (API's). Un WFMS autónomo es comparable a un DBMS, el cual no depende de aplicaciones externas para su funcionamiento.

2. Flujo de Trabajo embebido

Son los WFMS's que tienen su lógica incluida dentro de un sistema más grande. Un ejemplo típico son algunos sistemas de ERP (Enterprise Resource Planning ó Planeación de Recursos Empresariales), los cuales incorporan las funcionalidades de un WFMS dentro de sí.

Algunas soluciones Flujo de Trabajo embebido proveen interfaces solamente para incorporar aplicaciones propietarias. Otras en cambio sí proveen interfaces para la interacción con aplicaciones externas y aún con otros motores de Flujo de Trabajo.

ANEXO 4: ¿POR QUÉ USAR UN WFMS?

La función del WFMS es coordinar el trabajo de múltiples recursos (humanos y computacionales) de procesos definidos de la organización.

El propósito principal de usar un WFMS es lograr que los procesos de la organización sean mejores, más baratos y más rápidos. El WFMS logra esto porque:

MEJORA LA EFICIENCIA

1. **Evita esperas generadas por un flujo de trabajo no automatizado.**

Un ejemplo de esto lo podemos ver cuando el flujo de trabajo es mediante papelería. Siempre hay un tiempo de retardo para pasar los documentos de un puesto de trabajo a otro, no solo el tiempo de transporte, sino sobre todo el tiempo de espera a que haya un recurso disponible para hacerlo.

En un proceso por ejemplo, puede ser que los documentos generados por una actividad, necesarios para iniciar la siguiente actividad, sean transportados por una persona cada 2 horas o cuando se hayan acumulado una cantidad definida de casos. Esto genera un retardo innecesario en el flujo del trabajo.

2. **Disminuye errores humanos**

Existen errores humanos que pueden disminuir la eficiencia de un proceso de la organización. Los errores que se disminuyen están relacionados sobre todo con el direccionamiento equivocado de trabajo, pérdida de documentos, entre otros.

3. Disminución de esfuerzo humano

Las tareas de asignación/reasignación y direccionamiento de trabajo pueden suponer una cantidad de esfuerzo humano muy grande, sobre todo si es grande el número de casos que se manejan en un proceso.

En ciertos casos, el transporte físico de documentos puede suponer también esfuerzo humano. Un ejemplo de esto es cuando se tiene personal exclusivamente dedicado a llevar la papelería de un puesto de trabajo a otro. Este trabajo puede ser realizado por el WFMS, disminuyendo así el costo de la realización del proceso en la organización.

4. Mejor direccionamiento

El direccionamiento hecho por un WFMS es hecho en base a indicadores que permiten determinar cuál es el mejor recurso disponible para la realización de la tarea en cada caso específico.

Cuando el direccionamiento de trabajo es hecho por una persona, éste puede no ser el mejor, primero, debido a que aunque se haga de una manera ordenada y procedimental, es muy difícil determinar sin mucho esfuerzo el mejor recurso disponible para la realización de una tarea.

Por ejemplo, no siempre una distribución igual de trabajo a todos los recursos es la mejor opción, ya que hay recursos que pueden presentar una mayor productividad que otros.

Además, debido a la naturaleza implícita del ser humano, en las decisiones de direccionamiento de trabajo pueden influir factores personales.

MEJOR CONTROL

En base a indicadores generados por el WFMS, se puede mejorar el control de los procesos.

Se puede controlar casos específicos, saber en qué estado o que etapa dentro del proceso se encuentra un caso específico, entre otras cosas.

PERMITE MEDICIONES REALES DE COSTOS, EFICIENCIA, EFICACIA, ENTRE OTROS

Al ser el WFMS un componente central, en el que se registra la realización de cada actividad de un proceso, éste registra información como: quién la hizo, cuánto tiempo se tomó realizarla, cuándo comenzó, cuándo terminó, cuál fue el resultado, etc…

Toda esta información es útil, entre otras cosas, para:

- Determinar costos de procesos
- Determinar cuellos de botella
- Determinar eficiencia y eficacia de los recursos

Claro está, si se tienen las herramientas y el conocimiento necesario para utilizar la información provista por el WFMS.

Una analogía de esto podemos hacerla con los datos estadísticos generados por un DBMS: Pueden ser muy útiles, pero solo si se sabe interpretarlos.

Ya que los procesos están definidos, y existe la información generada por el WFMS, se pueden utilizar modelos matemáticos basados por ejemplo en redes Petri para realizar cuantificaciones de varios indicadores de los procesos.

FLEXIBILIDAD PARA CAMBIO Y MEJORA DE PROCESOS

Una de las grandes ventajas del uso de un WFMS es que permite la creación de nuevas versiones de procesos, así como la eliminación o creación de nuevos procesos. Es decir, permite el rediseño de procesos “en línea”, de acuerdo a necesidades de la empresa.

En los sistemas tradicionales, sin WFMS, cuando un proceso cambia, es necesario el cambio en los sistemas, para que se adapten a la nueva realidad de la organización. Al usar un WFMS, el esfuerzo necesario para adaptar los sistemas cuando los procesos son cambiados es mucho menor.

Esto permite implementar un ciclo de mejora continua de procesos, sin que esto produzca un impacto mayor en los sistemas o en las personas.

EJEMPLOS DE MEJORA AL USAR UN WFMS

Básicamente, habíamos dicho que el uso de un WFMS tiene como objetivo que los procesos en la empresa sean mejores, más baratos y más rápidos. Podemos ver que se ha logrado este objetivo en varios casos reales:

Caso 1: Krafts Food

Kraft Foods decidió consolidar todas las actividades de cuentas por cobrar y servicios a consumidor en San Antonio. Esto significaba manejar alrededor de 2.5 millones de facturas por año, de sus 60 plantas de operación.

“La nueva tecnología ha logrado una reducción del 37% en costos de transacción. El proyecto ha mostrado una tasa interna de retorno del 46% y el costo por transacción es alrededor del 54% del costo anterior. Además, el sistema de workflow regula las políticas de operación de Kraft, proveyendo productividad para el trabajo de auditoría. Esto es realmente un logro significativo cuando estas

emitiendo 2’500.000 facturas por año junto con la documentación asociada como notas de crédito y enmiendas” [53].

En resumen, las mejoras obtenidas por el sistema tenemos:

- “Reducción del personal en un 31%
- Ahorro de $300.000 en impuestos y penalizaciones
- Reducción del tiempo de generación y seguimiento de correspondencia en un 80%
- Reducción del tiempo para completar la petición de un consumidor en un 18%
- Reducción del tiempo de recuperación de un documento en un 99%” [54]

Dentro de los beneficios tangibles tenemos:

- Reducción en costos de operación
- Mejora en la productividad
- Mejores tiempos de procesamiento

Dentro de los beneficios intangibles:

- Mejores servicios
- Mejor administración del cambio
- Calidad: Menos errores
- Mejor comunicación
- Es un soporte para toma de decisiones
- Mejor capacidad de planeamiento
- Mejor capacidad de implementación
- Permite mejor comunicación inter-organizaciones

---

[53] ALLEN, Rob, “Workflow, an Introduction”, WfMC External Relations Comitee, United Kingdom, pag. 5

[54] Idem, pág. 5

ANEXO 5: BPM Y FLUJOS DE TRABAJO

DEFINICIÓN DE BPM

BPM o Administración de Procesos de Negocio es un término que se viene manejando en los últimos años, una definición de Dave McCoy dice:

BPM "… es una mezcla de administración de procesos / Flujo de Trabajo con tecnología de integración de aplicaciones… para soportar rica interacción humana e intensa conectividad de aplicaciones" [55]

Otra definición un poco más amplia, de John Pike, dice BPM es:

"Software de desarrollo rápido de aplicaciones para:

- La automatización de procesos basados en reglas
- Enrutamiento de documentos, información y tareas
- Dentro y entre organizaciones
- De manera oportuna
- Totalmente integrada con tecnologías complementarias y sistemas heredados

Para obtener beneficios significantes y medibles"[56].

SISTEMAS DE FLUJO DE TRABAJO Y SISTEMAS DE BPM

Los sistemas de BPM son diseñados para establecer una plataforma sobre la cual la empresa puede, de una manera eficiente, administrar los procesos del negocio.

---

[55] PIKE John, "BPM in Context, Now and in the future", WfMC Chair, United Kingdom, Marzo 2006, pag. 3

[56] Idem, pag. 3

La diferencia básica entre un sistema de Flujo de Trabajo y un sistema de BPM es que el sistema de BPM, además de tener todas las características de un sistema de Flujo de Trabajo, debe proveer una plataforma para el desarrollo e integración rápida de aplicaciones.

Un sistema de BPM es un sistema de Flujo de Trabajo, pero un sistema de Flujo de Trabajo no necesariamente es un sistema de BPM.

WFMS Y SISTEMAS DE BPM

Una parte del Sistema de BPM va a ser necesariamente un WFMS, el cual sería el centro del funcionamiento del sistema de BPM.

Un sistema de BPM puede además incorporar dentro de sí no solo uno, sino varios sistemas de Gestión de Flujo de Trabajo, los cuales interactúan entre sí.

ANEXO 6: BREVE ANÁLISIS DE LOS WFMS DISPONIBLES EN EL MERCADO ACTUAL

En el presente análisis se tratará de abarcar los productos más conocidos dentro del mercado actual, de todas maneras, debido al rápido crecimiento del sector, es imposible asegurar que sea un análisis completo o totalmente válido.

SOLUCIONES PROPIETARIAS

- Fuego
- Staffware
- COSA
- Action Workflow
- Adobe Live Cycle
- QFlow
- Together Workflow Server

Los WFMSs mencionados son productos que se encuentran en producción en varios sistemas alrededor del mundo.

WFMS DE CÓDIGO ABIERTO

Dentro de los WFMSs de código abierto, se vieron muchas opciones, pero las soluciones más maduras que se encontraron son:

- Enhydra Shark
- WFMOpen
- JBPM
- OBE (Open Business Engine)

BREVE ANÁLISIS

Dado que, a excepción de Together Workflow Server, fue imposible conseguir versiones de prueba de las soluciones propietarias, dichas soluciones no se tomaron en cuenta para el análisis de cuál WFMS se utilizaría para el desarrollo de la presente tesis.

1. **Together Workflow Server**

Es una solución basada en Enhydra Shark, que incorpora un servidor CORBA para la interacción de clientes. Fácil de instalar y configurar, pero no proveía el componente para desarrollo de aplicaciones cliente CORBA si no se compra el software completo.

2. **Enhydra Shark**

Solución de código abierto, bastante madura. Basado en el modelo de referencia de un WFMS de la WfMC. Provee API's para interacción de clientes basadas en una especificación propia, además de API's basadas en el estándar definido por la WfMC versión 1.0. La comunicación de clientes se realiza mediante RMI, por lo que los clientes pueden ser solo Java. Desarrollado en base a estándares de la WfMC.

3. **WFMOpen**

Basado en el modelo de referencia de un WFMS de la WfMC. Desarrollado en base a estándares del OMG y la WfMC, pero con modificaciones y ampliaciones propias. No provee interfaz CORBA para comunicación con clientes, sino una API java basada en RMI, por tanto los clientes pueden ser solo java.

4. **JBPM**

Bastante maduro y estable, pero no está basado en el modelo de referencia de un WFMS de la WfMC. Tiene la desventaja que no permite la definición de procesos

mediante XPDL (otro estándar de la WfMC), sino mediante BPDL (Business Process Definition Language). La interacción de clientes, igualmente se da solamente en un entorno java sobre el servidor de aplicaciones JBoss.

5. **OBE (Open Business Engine).**

Basado también en el modelo de referencia de un WFMS de la WfMC. Provee un API para la interacción de clientes java mediante RMI y XML-RPC.

ANEXO 7: DESCRIPCIÓN BREVE DE LAS POSIBLES METODOLOGÍAS A UTILIZARSE

PUDS (PROCESO UNIFICADO DE DESARROLLO DE SOFTWARE)

Es un marco de desarrollo de software:

- Iterativo e incremental, compuesto de 4 fases: Inicio, Elaboración, Construcción y Transición. Cada fase está dividida en iteraciones.
- Dirigido por los casos de uso: Se utilizan los casos de uso para capturar los requisitos funcionales y para definir los contenidos de las iteraciones.
- Centrado en la arquitectura
- Enfocado en los riesgos

Es una metodología ampliamente probada y aceptada. Pero dada su complejidad, no siempre es la mejor opción para proyectos pequeños.

XP (PROGRAMACIÓN EXTREMA)

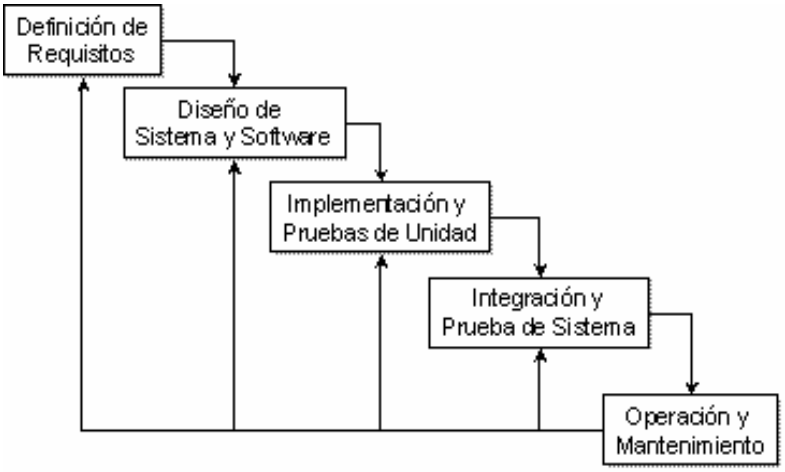
Es una metodología que pone más énfasis en la adaptabilidad que en la previsibilidad. Es la metodología más destacada dentro de los llamados “procesos ágiles” de desarrollo de software.

Esta metodología propone un desarrollo rápido e incremental, con pruebas unitarias contínuas, programación en parejas y entregas rápidas e incrementales.

Es una metodología que no presenta mucha confiabilidad sino solo para proyectos pequeños. Se utiliza generalmente cuando los requerimientos son cambiantes o no pueden ser definidos claramente al inicio del proyecto.

METODOLOGÍA DE DESARROLLO EN CASCADA

La metodología consta de las siguientes fases:



Modelos de proceso de software: Desarrollo en cascada

FUENTE: Ingeniería de Software, Somerville, 2002

La metodología de desarrollo en cascada es un enfoque riguroso de las etapas de desarrollo de software, proponiendo que no se debe comenzar una fase sino hasta haber terminado la anterior.

De esta manera, cualquier error en el diseño en la etapa de pruebas conduce necesariamente al rediseño y a la remodificación. Esta metodología por tanto, debe ser utilizada cuando los requerimientos de usuario son fácilmente identificables, y no son cambiantes.

Dentro de las ventajas de esta metodología es que tiende a ser la base para producir software altamente confiable.

Dentro de las desventajas tenemos que: presenta problemas de desempeño cuando los requisitos y la arquitectura no están bien definidos, es difícil implementar correcciones sobre la marcha y el cliente no va a ver progresos durante el desarrollo, sino solamente al final verá el producto terminado.

ANEXO 8: BREVE ANÁLISIS DE VARIOS MECANISMOS DE COMUNICACIÓN REMOTA

Dado que las aplicaciones clientes deben permitir la interacción con el WFMS de manera remota, debemos considerar, para el desarrollo del componente, los mecanismos para interacción de aplicaciones distribuidas.

La definición de la WfMC para las interfaces 2 y 3, es en sí una especificación de una API (Interfaz de programación de aplicación), pero no define específicamente un mecanismo de acceso remoto de aplicaciones; tienen por tanto los diferentes WFMSs diferentes implementaciones de sus interfaces, basadas en la definición de la WfMC.

La especificación estudiada, provista por el OMG, determina en cambio el estándar CORBA, como mecanismo de comunicación remota.

RESUMEN DE VARIOS MECANISMOS DE COMUNICACIÓN REMOTA

Dentro de los posibles mecanismos de comunicación remota utilizados para la interacción de las aplicaciones cliente con los WFMSs tenemos los siguientes:

1. **RPC**

Una de las primeras tecnologías utilizadas ampliamente para el desarrollo de aplicaciones distribuidas es RPC, que significa “Llamada a procedimiento remoto” por sus siglas en inglés. Fue desarrollado por Sun Microsystems.

2. **COM**

La tecnología RPC fue posteriormente redefinida por DEC (Digital Equipment Corp.), generando la tecnología llamada COM (Component Object Model). COM era una adaptación a objetos de RPC.

3. DDE y OLE

Microsoft adoptó a COM como base para sus futuros desarrollos, incluyendo DDE (Dinamyc Data Exchange o Intercambio de datos dinámicos), y posteriormente OLE (Object Linking and Embedding ó Enlace e Inmersión de Objetos), esta tecnología tenía muchos problemas, debido a que usaba intensamente interfaces gráficas de usuario y en su arquitectura en sí, estaba formada por demasiadas capas que funcionaban sobre COM.

4. DCOM

Debido a que era necesaria una alternativa menos pesada, Microsoft rehizo una nueva alternativa basada en COM, llamada DCOM (Distributed Component Object Model ó Modelo de Objetos de Componentes Distribuidos).

5. XML-RPC

Es un mecanismo de llamada a procedimiento remoto que utiliza XML para codificar las llamadas y HTTP como protocolo de transporte. Fue creada por UserLand Software en asociación con Microsoft en 1998, y se caracteriza por su simplicidad.

6. SOAP

Al considerar Microsoft que XML-RPC era muy simple, añadió varias funcionalidades, y después de varias etapas de desarrollo, creó un mecanismo más completo pero también más complicado, llamado SOAP (Simple Object Access Protocol), el cual constituye uno de los estándares más ampliamente utilizados en la actualidad, sobre todo para proporcionar los “Servicios Web”.

7. RMI

Significa Remote Method Invocation o Invocación de Métodos Remotos es un mecanismo que es dependiente del lenguaje, y que solo funciona en Java.

Es uno de los mecanismos más ampliamente utilizados, y es una de las mejores alternativas si se desea interactuar solamente con componentes programados en Java.

8. CORBA

La arquitectura de CORBA es bastante parecida a la de RMI, con la diferencia de que CORBA no es dependiente del lenguaje Java, sino que puede ser implementada desde cualquier otro lenguaje que lo permita.

Sun Microsystems implementó CORBA desde la versión JDK1.2, y aunque CORBA está hecho para que permita la interacción de sistemas programados en diferentes lenguajes, por razón de sus políticas, Microsoft no incorpora librerías para el desarrollo de aplicaciones CORBA en sus entornos de desarrollo.

RESUMEN

Dentro de las tecnologías que se usan ampliamente actualmente tenemos: RMI, CORBA, DCOM y SOAP.

RMI es útil cuando queremos interacción solamente de aplicaciones Java, así como DCOM es una buena opción si solamente queremos interacción de aplicaciones de Microsoft.

Tanto CORBA como SOAP permiten interacción de varias tecnologías. Ambas tecnologías sin embargo presentan la desventaja de su dificultad de aprendizaje e implementación.

CORBA y SOAP usualmente no son una buena opción si se va a realizar la interacción de aplicaciones en un solo lenguaje de programación, pues presentan la desventaja de ser menos eficientes.

TECNOLOGÍAS DE COMUNICACIÓN REMOTA IMPLEMENTADAS EN LOS WFMSS ACTUALES

La mayoría de WFMSs de código abierto utilizan las tecnologías basadas en Java, es decir RMI y CORBA.

Los WFMSs propietarios, dependiendo de la tecnología en la que han sido desarrollados implementan interfaces basadas en Java (RMI y CORBA) o basadas en tecnología Microsoft (DCOM y SOAP).

Algunos WFMSs incorporan además interfaces con tecnología XML-RPC.

TECNOLOGÍA ESCOGIDA PARA INTERACCIÓN DEL COMPONENTE CON EL WFMS

Dado que:

- Un gran número de WFMSs tanto de código abierto como propietarios proveen interfaces java para la interfaz 2 definida por la WfMC
- Java provee soporte para CORBA
- Se desarrollarán aplicaciones cliente en otros lenguajes fuera de Java
- El estándar propuesto por el OMG propone CORBA como tecnología para interacción de los componentes.

Definimos que el mecanismo de comunicación utilizado para la interacción del componente con el WFMS debe ser CORBA, pues CORBA permite la interacción de sistemas desarrollados bajo diferentes plataformas.

Tendremos sin embargo la desventaja de un rendimiento menor, cuando el cliente sea también escrito en Java. (RMI es la mejor opción para aplicaciones Java-Java).